

Ingres 10.0

Connectivity Guide

INGRES

ING-10-CN-03

This Documentation is for the end user's informational purposes only and may be subject to change or withdrawal by Ingres Corporation ("Ingres") at any time. This Documentation is the proprietary information of Ingres and is protected by the copyright laws of the United States and international treaties. It is not distributed under a GPL license. You may make printed or electronic copies of this Documentation provided that such copies are for your own internal use and all Ingres copyright notices and legends are affixed to each reproduced copy.

You may publish or distribute this document, in whole or in part, so long as the document remains unchanged and is disseminated with the applicable Ingres software. Any such publication or distribution must be in the same manner and medium as that used by Ingres, e.g., electronic download via website with the software or on a CD-ROM. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of Ingres.

To the extent permitted by applicable law, INGRES PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IN NO EVENT WILL INGRES BE LIABLE TO THE END USER OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USER OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF INGRES IS EXPRESSLY ADVISED OF SUCH LOSS OR DAMAGE.

The manufacturer of this Documentation is Ingres Corporation.

For government users, the Documentation is delivered with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227-7013 or applicable successor provisions.

Copyright © 2010 Ingres Corporation. All Rights Reserved.

Ingres, OpenROAD, and EDBC are registered trademarks of Ingres Corporation. All other trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Contents

Chapter 1: Introducing Ingres Connectivity **13**

In This Guide.....	13
Connectivity Solutions Not in This Guide.....	14
Basic Networking Concepts.....	14
Ingres Components and Tools.....	16
Ingres Instance	17
System-specific Text in This Guide.....	18
Terminology Used in This Guide	19
Syntax Conventions Used in This Guide	19

Chapter 2: Exploring Net **21**

Ingres Net	21
General Communication Facility	22
Net Security	23
Installation Configurations That Require Net.....	24
Net and Other Ingres-related Products	24
Net and Enterprise Access and EDBC Products	25
Net and Ingres Star	25
Net Product Integration Summary	26
Benefits of Net.....	27
Net Concepts.....	28
Virtual Nodes	28
Connection Data.....	29
Remote User Authorizations	30
Global and Private Definitions.....	31
Net Management Tools	32
Net and Bridge Users.....	33
System Administrator and Ingres Net	34
Database Administrator and Ingres Net.....	34
End Users and Ingres Net	34
GCA Privileges	35

Chapter 3: Installing and Configuring Net **37**

Net Installation Components.....	37
How You Prepare for Installation	37
Network Installation and Testing.....	38
Setup Parameters for Net	41

How Net Setup Works on an Existing Installation	43
How Communications Are Enabled	43
How You Install Net.....	44
invalidpw Program	45
Create Password Validation Program (UNIX)	46
Net Configuration Parameters—Customize Ingres Net.....	47

Chapter 4: Establishing Communications 49

How User Access Is Established.....	49
Requirements for Accessing Remote Instances.....	50
Requirements for Accessing Distributed Databases	51
Access Tools for Defining Vnodes.....	51
Netutil (Net Management Utility)	53
Netutil Startup Screen	53
Virtual Node Name Table in Netutil.....	54
Login and Password Data Table in Netutil	55
Connection Data Table in Netutil	57
Other Attribute Data Table in Netutil	59
Netutil Operations.....	60
Prerequisites to Establish and Test a Remote Connection	61
Establish and Test a Remote Connection Using Netutil.....	62
Delete an Entry	68
Change an Entry.....	70
Define an Installation Password for the Local Instance	74
Netutil Non-Interactive Mode	75
Command Line Flags in Netutil Non-interactive Mode	76
Create Function—Create a Remote User Authorization.....	78
Destroy Function—Destroy a Remote User Authorization	80
Show Function—Display Remote User Authorizations	81
Create Function—Define an Installation Password for the Local Instance	82
Create Function—Create a Connection Data Entry.....	83
Destroy Function—Destroy a Connection Data Entry	84
Show Function—Display Connection Data Entries.....	86
Stop and Quiesce Commands—Stop or Quiesce One or More Communications Servers.....	88
Network Utility and Visual DBA.....	89
Virtual Nodes Toolbar.....	89
Simple and Advanced Vnodes	90
Advanced Vnode Parameters	90
Installation Password Definitions for the Local Instance	93
Changing Installation Passwords	93
Additional Vnode-Related Tasks	93
Server-related Tasks	95

Chapter 5: Using Net **97**

Connection to Remote Databases	97
Database Access Syntax—Connect to Remote Database	98
Use of the SQL Connect Statement with Net	101
Commands and Net	102
User Identity on Remote Instance	103
-u Command Flag—Impersonate User	103
Verify Your Identity	104

Chapter 6: Maintaining Connectivity **105**

Start Communications Server	105
Stop Communications Server	106
Network Server Control Screen in Netutil	106
Stop or Quiesce a Communications Server Using Netutil	108
Inbound and Outbound Session Limits	110
How You Set Inbound and Outbound Session Limits	111
Logging Levels	111
How You Change the Logging Level	111
How You Direct Logging Output to a File	112
GCF Server Management Using iimonitor	113
Default Remote Nodes	114
How You Set Default Remote Nodes	114
Start Data Access Server (DAS)	115
Stop Data Access Server (DAS)	115

Chapter 7: Troubleshooting Connectivity **117**

How Connection Between the Application and DBMS Server Is Established	117
Where Ingres Net Information Is Stored	118
config.dat—Store Net Configuration Values	119
Name Server Database—Store Remote Access Information	120
Causes of Connectivity Problems	122
How You Diagnose Connectivity Problems	122
General Net Installation Check	123
Connection Errors	127
How You Resolve Net Registration Problems	129
Security and Permission Errors	129

Chapter 8: Exploring Bridge **131**

Ingres Bridge	131
How the Bridge Server Works	131

Tools for Configuring Bridge	132
Installation Configurations That Require Bridge	132
How Bridge Is Installed	134
How Bridge Is Started.....	134
config.dat File—Store Bridge Configuration	135
ingstart Command—Start the Bridge Server	135
iigcb Command—Start the Bridge Server.....	136
How the Client Is Set Up	136
vnode Definition—Enable Client Access to Remote Servers Through Bridge.....	137
Bridge Server Monitoring.....	137
Stop the Bridge Server	138
How a Connection Is Established Through Bridge	138
Bridge Troubleshooting	139
Sample Bridge Server Configuration	140

Chapter 9: Configuring the Data Access Server 143

Data Access Server	143
Data Access Server Parameters—Configure DAS	144
How You Enable Data Access Server Tracing	146
Tracing Levels	146

Chapter 10: Understanding ODBC Connectivity 147

ODBC Driver	147
ODBC Call-level Interface	148
Unsupported ODBC Features.....	149
Read-Only Driver Option	149
ODBC Driver Requirements.....	149
ODBC Driver Manager Programs	150
Support for Previously Released ODBC Drivers	152
ODBC Driver Names.....	152
Backward Compatibility Issues for ODBC DSN Definitions	152
Access to a Remote Instance	153
Configure a Data Source (Windows).....	153
Configure a Data Source (UNIX and VMS).....	155
iiiodbcadmin Utility	156
Connection String Keywords	156
ODBC CLI Implementation Considerations	159
Configuration on UNIX, Linux, and VMS	159
Optional Data Source Definitions.....	160
Supported Applications	160
ODBC Programming	161

ODBC Handles.....	161
How ODBC Applications Connect to a Database.....	162
SQLConnect()—Connect Using a Data Source Name.....	162
SQLDriverConnect()—Connect without Using a Data Source Name.....	165
Connect Using Dynamic Vnode Definitions.....	167
ODBC User Authentication	167
Query Execution.....	175
Database Procedures Execution	178
Fetches Data	181
Scrollable Cursors.....	190
Large Objects (Blobs) Support.....	195
Transactions Handling	202
Date/Time Columns and Values	204
Boolean Columns.....	208
National Character Set (Unicode) Columns	208
Metadata (Catalog) Queries.....	209
Termination and Clean-up.....	211
ODBC CLI Connection Pooling.....	212
Ingres ODBC and Distributed Transactions (Windows)	214
How You Enable the Use of Distributed Transactions through the Ingres ODBC Driver.....	215
Vnode Definitions When Using Distributed Transactions through ODBC.....	215
Troubleshooting Distributed Transactions through ODBC	216
ODBC Trace Diagnostics.....	217
Standard ODBC Tracing	217
Windows Environments	218
UNIX, Linux and VMS Environments	218

Chapter 11: Understanding JDBC Connectivity **223**

JDBC Components.....	223
JDBC Driver	223
JDBC Information Utility—Load the JDBC Driver	225
Unsupported JDBC Features.....	227
JDBC Driver Interface	228
JDBC Driver and Data Source Classes.....	228
JDBC Implementation Considerations.....	239
JDBC User Authentication	239
How Transactions Are Autocommitted.....	240
Cursors and Result Set Characteristics.....	242
Cursors and Select Loops.....	244
Batch Statement Execution	245
Database Procedures	246
BLOB Column Handling.....	249

Date/Time Columns and Values	253
National Character Set Columns.....	255
Data Type Compatibility	256
JDBC Tracing	259
Tracing Levels.....	261

Chapter 12: Understanding .NET Data Provider Connectivity 263

.NET Data Provider.....	263
.NET Data Provider Architecture	264
Data Provider Data Flow	264
Data Provider Assembly.....	265
Data Provider Namespace	265
Data Retrieval Strategies	266
Connection Pooling	267
Code Access Security.....	268
.NET Data Provider Classes.....	268
IngresCommand Class.....	269
Sample Program Constructed with .NET Data Provider	273
IngresCommandBuilder Class	275
IngresConnection Class	277
IngresConnectionStringBuilder Class	298
IngresDataReader Class	303
IngresDataAdapter Class	310
IngresError Class.....	313
IngresErrorCollection Class	314
IngresException Class	316
IngresFactory Class	317
IngresInfoMessageEventArgs Class	318
IngresInfoMessageEventHandler Class.....	319
IngresMetaDataCollectionNames Class.....	321
IngresParameter Class	321
IngresParameterCollection Class	327
IngresPermission Class.....	329
IngresRowUpdatedEventArgs Class	329
IngresRowUpdatedEventHandler Class.....	330
IngresRowUpdatingEventArgs Class.....	331
IngresRowUpdatingEventHandler Class	332
IngresTransaction Class.....	332
Data Types Mapping	334
DbType Mapping.....	336
Coercion of Unicode Strings	337
IngresDataReader Object—Retrieve Data from the Database	337

Build the IngresDataReader	337
IngresDataReader Methods	338
Example: Using the IngresDataReader.....	338
ExecuteScalar Method—Obtain a Single Value from a Database	339
GetBytes Method—Obtain BLOB Values from a Database	339
GetSchemaTable Method—Obtain Schema Information from a Database.....	340
ExecuteNonQuery Method—Modify and Update Database	340
IngresDataAdapter Object—Manage Data	341
How Database Procedures Are Called	342
Row Producing Procedures	343
Integration with Visual Studio	344
Install the Data Provider into the Toolbox.....	345
Start the Ingres Data Adapter Configuration Wizard.....	346
Design a Query Using the Query Builder.....	350
Server Explorer Integration	352
Application Configuration File—Troubleshoot Applications.....	354
Obtain File Version of the Data Provider	355

Appendix A: TCP/IP Protocol **357**

Listen Address Format	357
Network Address Format	358
Connection Data Entry Information.....	359
Windows	359
UNIX	359
VMS	359
MVS	360

Appendix B: SNA LU0 Protocol **361**

Listen Address Format	361
MVS	361

Appendix C: SNA LU62 Protocol **363**

Listen Address Format	363
MVS	364
Solaris	366
HP-UX.....	367
RS/6000	369

Appendix D: SPX/IPX Protocol	371
Listen Address Format	371
Windows	372
UNIX and VMS	373
 Appendix E: DECnet Protocol	 375
Listen Address Format	375
VMS	376
 Appendix F: LAN Manager Protocol	 377
LAN Manager Listen Address—Enable Communications	377
 Appendix G: SunLink Gateway Configuration Files	 379
SunLink Gateway Configuration File	379
Solaris Independent LUs	380
Solaris Dependent LUs	382
SunOS (or Sun-4) Independent LUs	384
SunOS (or Sun-4) Dependent LUs	386
 Appendix H: AIX SNA Services/6000 Configuration Profiles	 389
Sample Configuration Profiles	389
CONNECTION Profile for Independent LUs	390
CONNECTION Profile for Dependent LUs	390
LOCALLU Profile for Independent LU	391
LOCALLU Profile for Dependent LU	392
MODE Profile for Independent LUs	393
MODE Profile for Dependent LUs	393
 Appendix I: HP-UX SNAplus Configuration	 395
Sample Configuration File Excerpts	395
Independent LUs	396
Dependent LUs	397
Dynamically Loadable TP	398
 Appendix J: Netu Procedures	 399
Netu (Deprecated)	399
Start Netu	399
Netu User Interface	400

Stop the Communications Server	400
Modify Node Entry	401
Modify Remote Authorization Entry.....	401
Exit Netu	402
Remote Node Definition Operations	402
Add or Merge Remote Node Definitions	403
Delete Remote Node Definitions.....	405
How You Change Remote Node Definitions	406
Retrieve Remote Node Definition Information	408
Remote User Authorization Operations	410
Define Remote User Authorizations	411
Delete Remote User Authorizations	412
Change Remote User Authorizations.....	413
Retrieve Remote User Authorizations.....	416
Netu Options for Stopping the Communications Server	417
Obtain GCF Address	418
Stop Communications Server	419

Appendix K: IPv6 Configuration **421**

IPv6 Configuration Overview.....	421
TCP/IP and Ingres Communications	422
Parameters for Controlling IPv6 Support.....	422
II_TCPIP_VERSION Environment Variable—Specify Version of TCP/IP to Use	423
II_GC_PROT Environment Variable—Set IPC Protocol	424
ii.hostname.gcX.*.protocol.status Resource—Set Network Communications Protocol	425
Options for Disabling IPv6 Support	425
Use IPv4 Addresses Only	426
Back Out IPv6 Support	427
iicvtwintcp Command—Convert wintcp to tcp_ip Protocol Setting	428
IPv6 in the JDBC Driver and Ingres .NET Data Provider.....	430
Examples of Disabling IPv6 Support.....	431

Index **433**

Chapter 1: Introducing Ingres Connectivity

This section contains the following topics:

[In This Guide](#) (see page 13)

[Connectivity Solutions Not in This Guide](#) (see page 14)

[Basic Networking Concepts](#) (see page 14)

[Ingres Components and Tools](#) (see page 16)

[System-specific Text in This Guide](#) (see page 18)

[Terminology Used in This Guide](#) (see page 19)

[Syntax Conventions Used in This Guide](#) (see page 19)

In This Guide

The *Connectivity Guide* describes how to establish and maintain communications between Ingres® installations. The connectivity information presented in this guide for accessing Ingres databases also applies to Enterprise Access and ODBC products and the databases they support.

This guide includes the following information:

- How to install, configure, use, and maintain Ingres® Net and Ingres Protocol Bridge.
- Using JDBC, ODBC, and .NET Data Provider connectivity components in the Ingres environment.
- Configuration and troubleshooting tips for each of the network protocols supported by Ingres.

This chapter briefly describes networking concepts, Ingres components and tools, and conventions used in this guide.

Connectivity Solutions Not in This Guide

Ingres provides a variety of connectivity drivers, data adapters, and dialects, including the following:

- Ingres Python DBI Driver
- Ingres PHP Driver
- Ingres Perl DBI Extension
- Ingres Torque Database Adapter
- Ingres Hibernate Dialect

For a list of latest solutions and details on each, see the downloads page of the Ingres web site.

Basic Networking Concepts

To use this guide effectively, you should be familiar with the following basic networking terms and concepts.

A *network* is a collection of connected computers, software, and communication links.

A *heterogeneous environment* is a computing environment that includes a variety of machines, operating systems, software, and protocols.

A *homogeneous environment* is a computing environment in which all machines are the same, and use the same operating system, software, and protocols.

A *protocol* is a standard that defines a set of rules for the transference of data between computers. A protocol specifies how the data is represented, how the transfer occurs, and how errors are detected and transmissions are acknowledged.

A *node* is a computer that is connected to a network. Each network node has a unique address within the network.

The term *local* refers to the instance or node on which you are working.

The term *remote* refers to all non-local instances or nodes on the network. For example, assume that your network has three instances, "napoleon," "eugenie," and "josephine," and that you are working on "napoleon." From your perspective, "napoleon" is the local instance and "eugenie" and "josephine" are the remote instances. If a co-worker is working on "josephine," for that person, "josephine" is the local instance and "napoleon" and "eugenie" are remote instances.

JDBC (Java Database Connectivity) is a standardized API (Application Programming Interface) that allows database connectivity. It defines a set of function calls, error codes and data types that can be used to develop database independent applications using Java.

ODBC (Open Database Connectivity) is a standardized API (Application Programming Interface) that allows database connectivity. It defines a set of function calls, error codes and data types that can be used to develop database independent applications using Structured Query Language (SQL).

ODBC permits maximum interoperability—a single application can access many different database management systems. This enables an ODBC developer to develop, compile, and deploy an application without targeting a specific type of data source. Users can add the database drivers that link the application to the database management systems of their choice.

Ingres Components and Tools

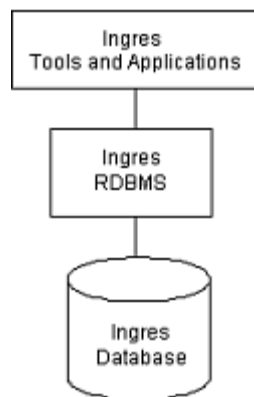
To use this guide effectively, you should be familiar with the basic components of Ingres, client/server concepts, and the Ingres tools required to configure, maintain, and view data.

The basic components of Ingres are as follows:

- The Relational Database Management System (RDBMS)—The Relational Database Management System is a set of Ingres processes. This set includes the processes that make up the Ingres DBMS Server and those that make up the logging and locking system. All of these processes work together to process queries from users running applications or using Ingres tools.
- The database—The database is the structure in which the RDBMS stores the data.

Client/Server—The Ingres database management system is the *server* that processes requests from clients. The Ingres tools and database applications are the *clients*.

Relationships among Ingres components and tools are illustrated here:



The Ingres tools used to configure, maintain, and view data include the following (commands to invoke these tools are shown in parentheses):

- Configuration Manager (vcbf)
- Configuration-By-Forms (cbf)
- Ingres Visual Manager (ivm)
- Visual Performance Monitor (vdbamon)
- Journal Analyzer (ija)
- Import Assistant (iia)
- Export Assistant (iea)
- Visual Configuration Differences Analyzer (vcda)
- Visual Database Objects Differences Analyzer (vdda)
- Visual SQL (vdbasql)
- Visual DBA (vdba)
- Net Management Utility (netutil)
- Network Utility (ingnet)
- Terminal Monitor (isql)
- Report-By-Forms (rbf)
- Query-By-Forms (qbf)
- Applications-By-Forms (abf)

For a description of each tool, see the *System Administrator Guide*.

The application development tools used to write customized applications include:

- Vision
- Ingres 4GL

For instructions on using these tools, see the *Forms-based Application Development Tools User Guide*.

Ingres Instance

An Ingres *instance* consists of a set of installed products that share a unique system-file location, ownership, and installation code, together with any data files created by these products. An instance is classified as either a server installation or a client installation.

Server Installation

An Ingres *server installation* consists of a DBMS server process (iidsbms), a Name Server process (iigcn), a set of Ingres tools, and the files and logs necessary to run the DBMS Server. For a detailed description of DBMS servers, see the *System Administrator Guide*.

If the server installation allows remote clients to access its DBMS servers, the server installation also includes the Ingres Net Communications Server process (iigcc).

Client Installation

An Ingres *client installation* contains a Name server process (iigcn), a Communications server process (iigcc), a Data Access Server process (iigcd), the API components that support client applications (Ingres JDBC Driver, ODBC Driver and .NET Data Provider) and the Ingres tools. A client installation *does not* run a DBMS server or store any data.

System-specific Text in This Guide

This guide provides information that is specific to your operating system, as in these examples:

Windows: This information is specific to the Windows operation system.

UNIX: This information is specific to the UNIX operation system.

VMS: This information is specific to VMS operating system.

When necessary for clarity, the symbol ■ is used to indicate the end of the system-specific text.

For sections that pertain to one system only, the system is indicated in the section title.

Terminology Used in This Guide

This guide uses the following terminology:

- A *command* is an operation that you execute at the operating system level. An extended operation invoked by a command is often referred to as a *utility*.
- A *statement* is an operation that you embed within a program or execute interactively from a terminal monitor.

Note: A statement can be written in Ingres 4GL, a host programming language (such as C), or a database query language (SQL or QUEL).

Syntax Conventions Used in This Guide

This guide uses the following conventions to describe syntax:

Convention	Usage
Monospace	Indicates key words, symbols, or punctuation that you must enter as shown
Italics	Represent a variable name for which you must supply an actual value
[] (brackets)	Indicate an optional item
{ } (braces)	Indicate an optional item that you can repeat as many times as appropriate
(vertical bar)	Separates items in a list and indicates that you must choose one item

Chapter 2: Exploring Net

This section contains the following topics:

[Ingres Net](#) (see page 21)

[Installation Configurations That Require Net](#) (see page 24)

[Net and Other Ingres-related Products](#) (see page 24)

[Benefits of Net](#) (see page 27)

[Net Concepts](#) (see page 28)

[Net Management Tools](#) (see page 32)

[Net and Bridge Users](#) (see page 33)

[GCA Privileges](#) (see page 35)

Ingres Net

Ingres Net is a server process that allows you to work on one Ingres instance and access databases on another instance. Both instances can reside on the same machine or they can reside on different machines. For example, with Ingres Net on each instance in your network, you can access Ingres databases on remote nodes as well as on your own local node. Similarly, with Ingres Net in a cluster, you can access Ingres databases on any node in the cluster.

Ingres Net connects multiple computer architectures, operating systems, and network protocols. This capability broadens the range and number of machines that can offer solutions to problems requiring Ingres-based distributed processing. Ingres Net automatically handles all low-level details of data format conversion required in such heterogeneous environments.

Ingres Net is implemented on industry-standard networking protocols. It is designed to be independent of underlying communications hardware and networking software. Subject to the appropriate security checks, Ingres Net lets you treat any remote database as a local database.

General Communication Facility

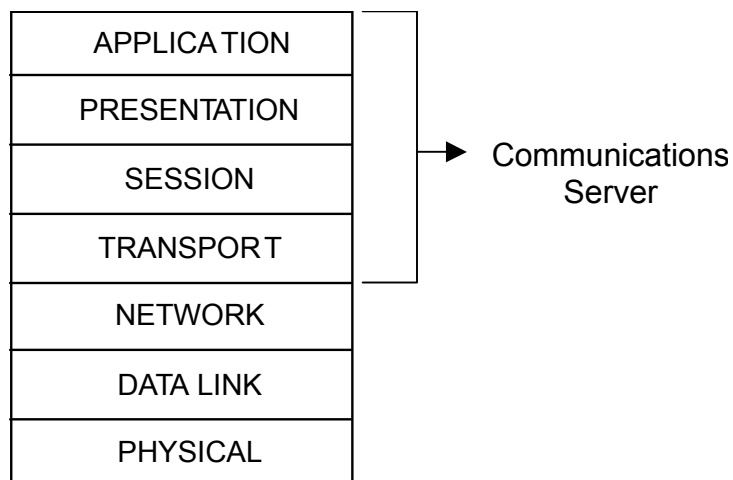
Ingres Net works with the basic Ingres components to enable connectivity between client and server instances. It also uses the General Communication Facility to manage communication among various components of Ingres.

The General Communication Facility (GCF) manages communication among all the components of Ingres. The GCF consists of five parts:

- The **General Communications Architecture** (GCA), which provides communication connections between Ingres processes on the same instance.
- The **Name Server** (iigcn) maintains a list of all registered, active servers. The Name Server provides information to user processes that enables a connection to a local DBMS Server. When a process wants to connect to a remote DBMS Server, the Name Server provides information that allows the process to first connect to a Communications Server. The Communications Server establishes communication with the remote DBMS Server. An instance has only one Name Server process.
- The **Communications Server** (iigcc) is the main process component of Ingres Net. It monitors outgoing communication from local applications to remote DBMS servers and incoming communication from remote applications to local DBMS servers. An instance can have multiple Communications server processes.
- The **Data Access Server** (iigcd) translates requests from the Ingres JDBC Driver and the .NET Data Provider into Ingres internal format and forwards the request to the appropriate DBMS Server. The Data Access Server (DAS) accesses DBMS servers on remote machines using Net.
- The **Protocol Bridge Server** (iigcb) provides services for Ingres Bridge, a product that enables a client application running on one type of local area network to access a DBMS server running on a different type of network. Ingres Bridge “bridges” a client using one network protocol to a server using another. (This component is functional only if you are using Ingres Bridge.)

Communications Server

As the main server process of Ingres Net, the Communications Server (iigcc), also referred to as the Net Server, provides access to standard network protocols. It is modeled on the top four layers of the network layering structure and communication protocols known as the Open Systems Interconnection (OSI) standards. These standards are specified by the International Standards Organization (ISO). The following figure displays these layers.



Net Security

Ingres Net supports the Ingres security system; users can access only the data for which they are authorized.

Installation Configurations That Require Net

With one exception, any installation configuration in which the client and server processes do not reside on the same machine or in the same instance must use Ingres Net.

The exception occurs when Ingres is configured in the cluster mode on nodes that are part of a cluster. In this case, the processes can reside on separate machines without using Net. If Ingres is configured in its normal (rather than cluster) mode on a node that is part of a cluster, Ingres Net is required to connect client and server processes on separate nodes.

For example, an Ingres 4GL client application using Net accesses a remote DBMS server. In this configuration, the Java application does not use Ingres Net because the Data Access Server (DAS) is local to the DBMS Server. If the DAS is remote to the DBMS Server, Ingres Net is required to enable the client/server connection.

Net and Other Ingres-related Products

Ingres Net, along with any of the following products, can fit into a variety of installation configurations to provide enhanced access and communication capabilities in more complex installations. Using these products with the basic Ingres components provides simultaneous, distributed access to databases and applications in a heterogeneous environment.

Ingres Bridge

Enables client applications running on one type of network LAN to access an Ingres server running on a different type of network.

Enterprise Access and EDBC products

Provide access to non-Ingres databases.

Ingres Star

Allows access to multiple databases transparently and simultaneously.

Net and Enterprise Access and EDBC Products

Enterprise Access and EDBC products allow you to use Ingres tools, interfaces, and applications to access data stored in non-Ingres databases by translating queries into forms that are understood by the non-Ingres databases. Consequently, you can perform operations and queries on files stored in these non-Ingres databases as if they were Ingres tables.

A sample installation configuration uses Ingres Net and EDBC to database 2 (DB2). The installation is on node_a, a VMS implementation of the Ingres tools.

An installation on node_b is EDBC to DB2 on an MVS environment. The two nodes communicate using the SNA LU0 protocol. Ingres Net is present on both nodes. Users on node_a can access DB2 data on node_b as if the DB2 tables were Ingres tables stored on node_a.

Note: MVS refers to all IBM MVS-based operating systems, including OS/390 and z/OS.

For a detailed discussion of Enterprise Access or EDBC architecture, see the guides for your specific Enterprise Access or EDBC product.

Net and Ingres Star

In Ingres, one application can have multiple sessions, with each session accessing one database. However, in many applications, the ability to access multiple databases in a single session is also very useful. In Ingres, this expanded functionality is available by using Ingres Star.

Ingres Star lets you create a distributed database. A distributed database is composed of some or all of the tables from a number of databases. When you access a distributed database, these tables appear to reside in a single database. The composition and operation of the distributed database is completely transparent to the user.

Combining Ingres Net and Ingres Star gives you simultaneous access to any number of databases residing on separate instances. Ingres Net gives you the ability to query a database on a different instance, and Ingres Star allows you to simultaneously query more than one database. You need *both* Ingres Net and Ingres Star to simultaneously access more than one database *if the databases are in separate instances*.

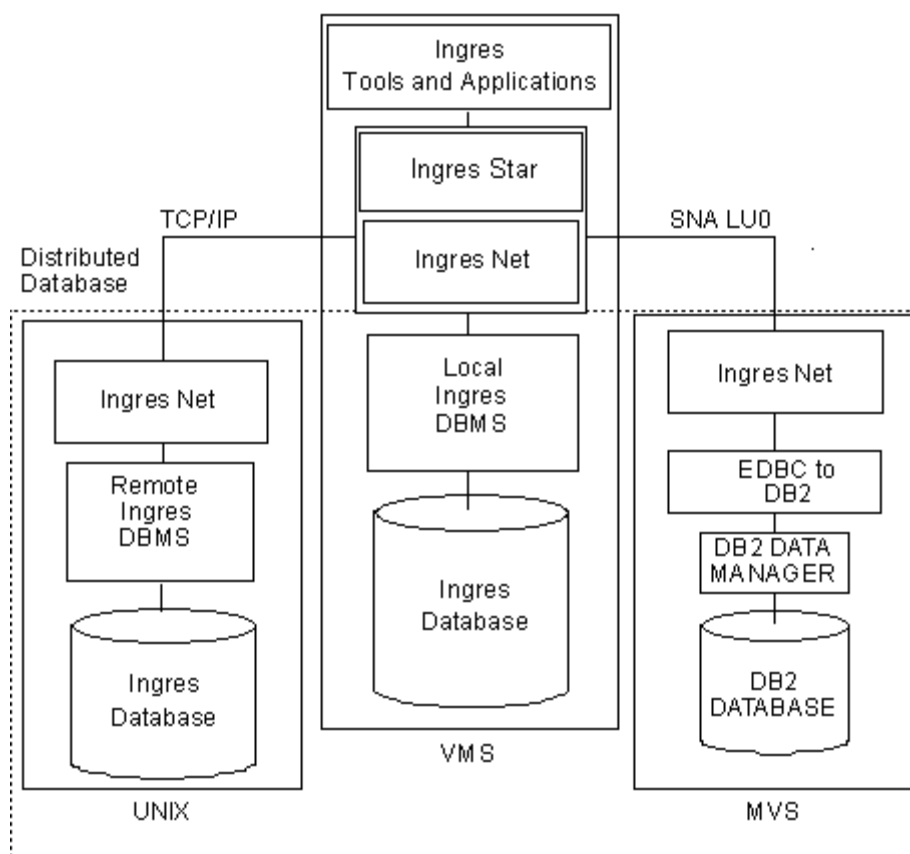
For a full explanation of how to use Ingres Star, see the *Ingres Star User Guide*.

Net Product Integration Summary

By making an Enterprise Access or EDBC product part of an Ingres Star database, you can transparently and simultaneously access both Ingres and non-Ingres databases.

The following figure illustrates a distributed database environment that contains an Enterprise Access or EDBC product. In this illustration, the local DBMS Server resides on the VMS operating system. The remote DBMS Server resides on UNIX. EDBC to DB2 resides on MVS.

Ingres applications on VMS are linked by Ingres Net to the remote DBMS Server on UNIX with the TCP/IP network protocol. Ingres applications on VMS are linked by Ingres Net to the remote EDBC to DB2 on MVS with the SNA LU0 network protocol.



Benefits of Net

Ingres Net provides many benefits in your computing environment. With Ingres Net, you can do the following:

- Improve total system performance

Ingres Net lets you dedicate each node in a network to specific applications, tools, or databases. By doing so, you can optimize each node for a primary function and avoid the problem of designing for conflicting requirements.

- Build larger applications

When you are working with Ingres installed on just one machine, the number of users and applications you can support is limited by the capacity of the machine. With Ingres Net you can connect multiple machines and *instances* in a network to support larger applications and to handle more users.

- Simplify expansion

With Ingres Net, the network serves as a vehicle for expanding the computer environment. When more computing power is needed, add smaller, less expensive machines instead of replacing existing computers with larger, more costly machines. This preserves the existing hardware investment and allows expansion without disrupting productivity.

- Minimize data communication costs

Ingres Net uses the network efficiently. Together, the following features minimize the number of messages and the amount of data sent, thus minimizing data communication costs.

- The local Ingres program communicates with its remote data manager using the SQL language. Built on the relational model, SQL manipulates sets of records rather than a single record at a time. This allows the use of more compact commands and queries.
- The remote data manager carries out database access functions entirely on the remote node. Only data requested by the user is transmitted over the network to the local application. For example, in update operations, users are not requesting to see any data; they simply want to change some existing data. The remote data manager, therefore, carries out all the work.

- Improve resource sharing

Consider a company headquarters that must support a number of sales offices across the country. Although each sales office needs only a small on-site computer to support its few users, it must have access to the data stored in the large corporate database.

With Ingres Net, local data can remain on the local nodes, providing fast response to users in the local sales offices. These same users also have the advantage of sharing the large database maintained on the central computer at corporate headquarters when necessary without being required to house a copy of it on their local machine.

By connecting databases and applications on different machines, you can balance computer resources, promote data sharing, and improve access to an organization's information.

Net Concepts

Concepts related to Ingres Net include the following:

- Virtual nodes
- Connection data
- Remote user authorizations
- Global and private definitions

Virtual Nodes

A virtual node (vnode) is a name defined on the local instance to identify a particular remote instance. Assigning a vnode name is typically the first step in the process of establishing connection and authorization data for a remote instance.

Whenever local users connect to a database on a remote instance or run an application that accesses a database on a remote instance, they must do *one* of the following:

- Use the virtual node name assigned to that instance
- Specify all of the required information in the connection string using the "dynamic vnode" format

Using vnodes is generally simpler for users because they only have to enter a single, user-friendly vnode name when they run an application, rather than detailed network-specific connection information. Another advantage of vnodes is that network changes can be updated for a vnode without notifying the user or changing the application.

Connection Data

Connection data refers to the information that the Communications Server in the local instance requires to locate and connect to the Communications Server on a remote instance. Connection data includes the following:

- The network address or node name of the remote instance's host machine
- The listen address of the remote instance's Communications Server
- The network protocol by which the local and remote instances communicate

Connection data is defined for each vnode, but can also be specified when using the dynamic vnode format.

It is possible to have more than one connection data entry for the same vnode. For example, if the remote instance has more than one Communications Server or can be accessed through more than one network protocol, this information can be included in the vnode definition by adding extra connection data entries.

Listen Address

A listen address is a unique identifier used for interprocess communications. The format of a listen address is dependent on the network protocol and the hardware.

For descriptions of listen address formats for the protocols supported by Ingres Net, see the appendixes in this guide.

Remote User Authorizations

Connection data alone is not sufficient to access a remote Ingres instance. You must authorize users to access the remote instance.

A remote user authorization consists of either of the following:

- An Installation Password

An Installation Password enables the user to access the remote instance directly. Users retain their identity as defined on the local instance.

If an Installation Password is defined, a login account is optional.

- Login account and password

A login account (set up by the system administrator) on the host machine of the remote instance. Users take on the identity of the login account through which they access the remote instance.

The main advantage of using an Installation Password is that users on the local node do not require a login account on the remote instance's host machine. They can access the remote instance directly provided they are recognized as valid Ingres users on the remote instance.

Note: Installation passwords must be used only when user privileges are the same on both local and remote machines. Using installation passwords between machines with different access privileges can lead to security access violations. For example, if a user is able to access the Ingres administrator account on a client machine but not on the server machine, use of installation passwords allows the user to bypass standard security and access the database as the Ingres administrator through Ingres Net.

Ingres Net requires the following remote user authorization information:

- Name of remote login account (if applicable)
- Password (either a login account password or an Installation Password)

For more information, see the chapter "Establishing Communications."

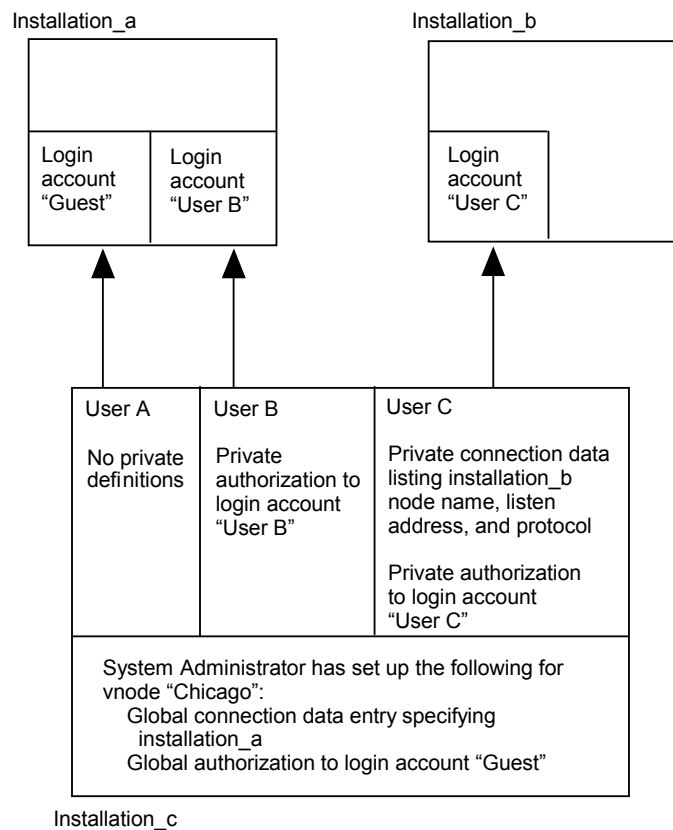
Global and Private Definitions

Both connection data entries and remote user authorization entries can be defined for vnodes as either global or private. A global entry is available to all users on the local instance. A private entry is available only to the user who creates it.

Each user can create a private entry. Only a user with the GCA privilege NET_ADMIN (typically a system administrator) can create a global entry.

If both a private and a global entry exist for a given vnode, the private entry takes precedence when the user who created the private entry invokes the vnode.

The following figure shows how connections are made when both private and global entries are defined for a given vnode.



On installation_c, the system administrator has created a vnode ("Chicago") with a global connection data entry specifying installation_a and a global remote user authorization specifying a login account ("Guest") on that instance. User A has not defined any private definitions for vnode "Chicago" that takes precedence over the global definitions.

When User A invokes vnode "Chicago," a connection is made to installation_a through login account "Guest." User B has added a private remote user authorization to vnode "Chicago," specifying the login account "User B." When User B invokes vnode "Chicago," the private authorization takes precedence over the global authorization, and a connection is made to installation_a through the login account "User B."

User C has added a private connection data entry to vnode "Chicago." The private connection data entry contains the listen address, node name, and network protocol of installation_b.

User C has also added a private authorization to login account "User C" on installation_b. When User C invokes vnode "Chicago," the private definitions take precedence over the global definitions, and a connection is made to installation_b through the login account "User C."

Net Management Tools

The Ingres Net management tools allow you to store and manage the vnode information (connection data and remote user authorizations) used by the Communications Server and the Bridge Server to connect to remote Ingres instances. These tools are:

- The forms-based Net Management Utility, netutil
- The GUI-based Network Utility and Visual DBA

For information on using these tools, see the chapter "Establishing Communications."

Net and Bridge Users

At a site, several levels of users are often defined by the tasks and responsibilities they have within the installation. For installations with Ingres Net and Ingres Bridge, these levels are:

- Operating system administrator

The operating system administrator sets up the operating system environment in which Ingres is installed. This person is the owner of the operating system account (for example, root in UNIX, system in VMS), which provides all permissions and privileges available from the operating system.

- Network administrator

The network administrator is responsible for the physical installation and maintenance of the network. These responsibilities include designing a network configuration that provides optimal user and database access, and installing and maintaining the necessary hardware and software.

- System administrator

The system administrator is the owner of the user account that provides permissions in the Ingres environment. Responsibilities include installing and maintaining Ingres and Ingres Net, authorizing user access, and maintaining and troubleshooting the installation.

- Database administrator (DBA)

Each database in an installation has a DBA who is responsible for maintaining and tuning the database, granting permission to access objects in the database (such as tables and views), and backup and recovery of the database.

- End users

An end user is anyone who uses the instance and is not an operating system administrator, system administrator, DBA, or user with special privileges.

A person may have responsibilities at more than one level. For example, a user can be the database administrator of one database and simply an end user of another.

Users at the system administrator, database administrator, and end user levels have specific Ingres Net responsibilities.

System Administrator and Ingres Net

The system administrator often performs the following Ingres Net-specific tasks, however any user with the appropriate privileges can perform these tasks:

- Defining global connection data entries and remote user authorizations. This task requires the GCA privilege NET_ADMIN.
- Starting, stopping, configuring, and monitoring Ingres servers, including the Name, Communications, and Bridge servers. These tasks require the GCA privilege SERVER_CONTROL.

The NET_ADMIN and SERVER_CONTROL privileges are assigned by default to the installation owner user ID, system (on VMS), and root (on UNIX). To assign these privileges to another user, the system administrator must manually add the following line to the config.dat file:

```
ii.node_name.privileges.user.user:  SERVER_CONTROL,NET_ADMIN
```

For example:

```
ii.panther.privileges.user.joan:  SERVER_CONTROL,NET ADMIN
```

Database Administrator and Ingres Net

The DBA must ensure that users who remotely access an Ingres instance have a user profile that permits access.

End Users and Ingres Net

End users are responsible for the following Ingres Net and Ingres Bridge-specific tasks:

- Defining their private connection data entries, if any
- Defining their private remote user authorizations, if any

GCA Privileges

GCF Servers, APIs, and utilities permit certain operations based on the following privileges:

SERVER_CONTROL

Stop GCF servers. Run iimonitor against GCF servers. Add and delete server registrations.

NET_ADMIN

Add and delete global vnode entries. View, add, and delete the vnode entries of another user.

MONITOR

Access server statistics (IMA internal access through GCM).

TRUSTED

Installation administrator privileges, including those above.

The Ingres DBMS and Enterprise Access Gateways can also reference these privileges.

Chapter 3: Installing and Configuring Net

This section contains the following topics:

[Net Installation Components](#) (see page 37)

[How You Prepare for Installation](#) (see page 37)

[How Net Setup Works on an Existing Installation](#) (see page 43)

[How Communications Are Enabled](#) (see page 43)

[How You Install Net](#) (see page 44)

[Net Configuration Parameters—Customize Ingres Net](#) (see page 47)

Net Installation Components

When you install Ingres Net, the following components are automatically installed with it:

- Data Access Server (provides network access to the DBMS Server for Ingres JDBC drivers and .NET Data Providers)
- Ingres JDBC Driver
- Ingres ODBC Driver
- Ingres Bridge

How You Prepare for Installation

Before you install Net, do the following:

1. Make sure you have met the installation prerequisites for the network protocol you are using, and that the physical network is installed and working.

Note: If you choose a Complete install when installing Ingres, all supplied network protocols are installed with default port values, including Novell Netware SPX/IPX protocol (NVLSPX). You will receive an error at startup for any protocols not configured in the network. For example, error "Network open failed for protocol NVLSPX" will be issued if the network is not configured to use the NVLSPX protocol (or configured with an incorrect port value).

2. Understand the configuration parameters you must supply values for during the setup phase of the installation process.

Network Installation and Testing

Before you install Ingres Net, the network administrator must make sure the network is properly installed and operating.

TCP/IP Installation (Windows)

To install Ingres Net for Windows with TCP/IP as its network protocol, you must first install the TCP/IP network software for Windows. From the Network applet in the Control Panel, choose Add Software. From the list of choices, choose TCP/IP Protocol and Related Components and follow the installation instructions.

To use symbolic node names (host names) of a remote host instead of its numeric IP address, you must either configure a Domain Name Server in the DNS section of the TCP/IP configuration or add a list of IP addresses and corresponding symbolic names in a file called %windir%\system32\drivers\etc\hosts. For information on the format of this file, see Windows documentation.

SPX/IPX Installation (Windows)

To install Ingres Net for Windows with SPX/IPX as its network protocol, you must first install the SPX/IPX network software for Windows.

From the Network applet in the Control Panel, choose Add Software. From the list of choices, choose NWLink IPX/SPX Compatible Transport and follow the installation instructions.

For information on installing the SPX/IPX protocol, see Windows documentation.

TCP/IP Installation (UNIX)

To install Ingres Net for UNIX with TCP/IP as its network protocol, you must first configure TCP/IP for UNIX.

To use symbolic node names (host names) of a remote host instead of its numeric IP address, you must either configure TCP/IP to use a Domain Name Server configuration or add a list of IP addresses and corresponding symbolic node names (host names) of all remote hosts that are referred to by host name in a file called the `/etc/hosts` file (or other list of network host addresses).

Establish aliases for node names in the `/etc/hosts` file. This is useful if the node name contains characters that are not accepted by Ingres Net. For information about how to establish aliases, see UNIX documentation for your machine.

Fully test TCP/IP before installing Ingres Net. You must be able to connect to other nodes on the network using `telnet` and `ftp` commands.

TCP/IP Services Installation (VMS)

To install Ingres Net for VMS with TCP/IP as its network protocol, you must first install TCP/IP Services on VMS. To use symbolic node names (host names) of a remote host instead of its numeric IP address, you must either configure TCP/IP to utilize a Domain Name Server configuration or add a list of IP addresses and corresponding symbolic node names (host names) of all remote hosts that are referred to by host name in the `TCP$HOST` file.

Establish aliases for node names in the `TCP$HOST` file. This is useful if the node name contains characters that are not accepted by Ingres Net. For information about how to establish aliases, see VMS documentation.

Test TCP/IP fully before installing Ingres Net. You must be able to connect to other nodes on the network using `telnet` and `ftp` commands. The connection can be tested with a `TCPIP PING` command, or use the `telnet` utility to connect to the node. If the connection succeeds, you are ready to add the nodes to the Ingres installation using `netutil`.

For more information, see the chapter *Establishing Communications* and the VMS documentation on TCP/IP services.

Note: TCP/IP Services for OpenVMS, formerly UCX, is often still referred to as UCX.

DECnet Installation (VMS)

Installing Ingres Net using DECnet as a network protocol requires no additional procedures in the DECnet installation. Simply install DECnet and test it fully before installing Ingres Net. Be sure that all nodes that use Ingres Net are defined and accessible through DECnet. You must be able to use the set host command to connect to any node on the network that uses Ingres Net. For details, see *DECnet-Plus for OpenVMS Installation and Basic Configuration* or *DECnet-Plus Introduction and User's Guide*.

MultiNet TCP/IP Installation (VMS)

When installing MultiNet TCP/IP on a network that uses Ingres Net, you must make it emulate either Wollongong TCP/IP or TCP/IP Services for OpenVMS. For details on enabling TCP/IP Services emulation (using the MultiNet SET LOAD-UC_DRIVER command) or WIN/TCP emulation (using the SET WINS-COMPATIBILITY command), see the *MultiNet for OpenVMS System Administrator's Guide*.

Depending on the selected emulation mode, you must follow the guide's instructions for Wollongong TCP/IP or TCP/IP Services for OpenVMS.

SunLink SNA Peer-to-Peer Installation (Solaris and Sun-4)

When using SunLink SNA Peer-to-Peer (LU 62) as the network protocol, you must set up the appc Gateway configuration files to define the SNA Logical Unit (LU) and Physical Unit (PU) resources associated with Ingres Net connections. For information on setting up this configuration file, see the *SunLink SNA Peer-to-Peer Administrator's Guide*.

The "SunLink Gateway Configuration Files" appendix contains sample excerpts from configuration files. An experienced SNA communications specialist must perform the configuration.

If using independent LUs, make sure (in SunLink SNA Peer-to-Peer Release 7.x) that you start the cnos_local and cnos_remote processes in addition to starting and configuring the appc Gateway process.

Test SunLink SNA Peer-to-Peer fully before installing Ingres Net. Use the SunLink SNA test_p2p program to perform testing.

Ingres Net does not currently support the Physical Unit Management Services (PUMS) that SunLink SNA provides.

HP-UX SNAplus (HP-UX 9.0)

When using HP-UX SNAplus (LU6.2) as the network protocol, you must configure the links, connections, Modes, Remote APPC LUs, Local APPC LUs, and Invocable TPs associated with Ingres Net connections. For information on the configuration procedure, see the *HP SNA Products Remote System Configuration Guide*, the *HP-UX SNAplusLink Administrator's Guide*, and the *HP-UX SNAplusAPI Administrator's Guide*.

For more detailed information on configuration, see the appendix "Netu Procedures" in this guide. An experienced SNA communications specialist must perform the configuration.

You must test HP-UX SNAplus fully before installing Ingres Net. For guidance, see the sample programs in `/usr/lib/sna/samples`.

If the SNAplus control daemon is restarted, any Communications Servers that are using the protocol must also be restarted.

AIX SNA Services/6000 (IBM RS/6000)

When using AIX SNA Services/6000 as the network protocol, you must create a set of configuration profiles that describe the hardware and software that are used for communications. For information on defining these profiles, see *Using AIX SNA Services/6000* and *AIX SNA Services/6000 Reference*. An experienced SNA communications specialist must perform the configuration.

Ensure that SNA Services/6000 is fully functional before installing Ingres Net. In particular, make sure that the SNA subsystem and the network attachment that is to be used can be started using the `startsrc` console command. *Using AIX SNA Services/6000* contains details on the use of this command.

The appendix "AIX SNA Services/6000 Configuration Profiles" contains sample excerpts from configuration profiles showing examples of those profiles that must be specifically tailored for Ingres Net. Once these profiles are defined and Ingres Net is installed on both the local and remote machines, use the `startsrc` command to start up the connection that you have configured. This is not necessary for Ingres Net operations, but it helps to verify that the configuration profiles are correct before attempting to actually run Ingres Net.

Setup Parameters for Net

The parameters that must be specified when installing and setting up Ingres Net depend on whether it is a server or client installation. They also depend on whether you choose to use an Installation Password to authorize access to a server installation from a remote client installation.

Installation Password and Remote User Authorization

Installation Passwords and their corresponding remote user authorizations can be wholly or partially set up during the Ingres Net installation procedure.

You can set up Installation Passwords and remote user authorizations at any time *after* the installation procedure using Network Utility, Visual DBA, or netutil. For more information about these procedures, see the chapter "Establishing Communications."

Setup Parameters for a Server Installation

If Ingres Net is part of a server installation, you are asked to supply the following information:

Installation Password

Is an alphanumeric string that can be used to authorize remote users to access the DBMS Server on this installation.

The first eight characters of the string must be unique on the installation.

Default: None

VMS: If you are installing Ingres Net on a VMS system, you are not prompted to define an Installation Password. To define one, use netutil after completing the installation procedure.

Setup Parameters for a Client Installation

If you are installing Ingres Net on a non-NFS client installation, you are asked for the following information.

Installation Code

An installer-defined, two-character code that identifies this installation.

Default: II

Windows and UNIX: The first character must be a letter; the second character can be a letter or numeral. If there is more than one installation on the same node, each installation must have a unique installation code. ■

VMS: This parameter applies only to group-level installations. System level installations are assigned an internal code of "aa".

Make sure that the first letter of your group-level installation code is not "a" and not in use by another group-level installation in the node. ■

Region and Time Zone

The region of the world and the time zone in which this client installation is located. You must enter these values even if they are the same as for this client's DBMS Server (host) node.

Default on some systems: NA-PACIFIC.

NFS clients are prompted only for the Installation Password. For detailed information on running Setup for NFS clients, see the *Installation Guide*.

Note: If you are setting up NFS clients *from the server installation*, you are not prompted to set up a remote user authorization entry. You must set up your remote user authorization entries using netutil after you have completed the installation procedure. For instructions on setting up a remote user authorization using netutil, see the chapter "Establishing Communications."

How Net Setup Works on an Existing Installation

When Ingres Net is added to an existing installation, the procedures differ slightly but the fundamentals remain the same. You must rerun the Ingres Setup program to install and configure Ingres Net. The prompts that you must answer remain the same, regardless of whether Ingres Net is being installed as part of an initial installation or as an addition to an existing installation.

How Communications Are Enabled

A server and client are able to communicate through Ingres Net as soon as the installation procedure is complete if an Installation Password and corresponding remote user authorization entry are set up on that server and client, respectively.

Otherwise, before Ingres Net can be used to connect installations, the necessary remote user authorizations, connection data, and Installation Passwords or Login Account Passwords must be defined.

How You Install Net

To install Ingres Net as part of a new or existing installation, follow this process:

Note: On VMS, make sure any user account that is using Net has the NETMBX privilege.

1. Make sure you have met the installation prerequisites for the network protocol you are using, and that the physical network is installed and working. For more information, see Network Installation and Testing (see page 38) and the appendix specific to your network protocol.
2. Be ready with the necessary information to set up the Net installation parameters.

Windows:

For server installations:

- Installation Password, if you are defining one at this time

For client installations:

- Installation Code
- Region and Time Zone

If access is authorized to a remote server using an Installation Password:

- Installation Password defined on server installation
- Name of the host machine on which the server installation resides
- Server installation's listen address

UNIX:

- Host Name: The name of the host machine on which the remote installation resides
- Listen Address: The listen address for the remote installation's Communications server. The default is the server installation code
- Installation Password: The Installation Password defined on the remote installation

VMS:

- Installation Code
- Time Zone

3. Shut down your installation if you are adding Ingres Net to an existing installation.
4. Perform the appropriate installation procedure documented in the *Installation Guide*.
5. Start your installation.

6. Authorize users to use Ingres by using the create user statement (or Visual DBA if available). For details, see the *Database Administrator Guide*.
7. Define connection data for the remote installations you plan to access. For detailed procedures, see the chapter "Establishing Communications."
8. Define remote user authorizations for the remote installations you plan to access, if you did not do so during the installation procedure. For detailed procedures, see the chapter "Establishing Communications."
9. Define an Installation Password for the local installation to enable remote users to access this installation

Note: This step is not necessary if you defined a password during the installation procedure.

For detailed procedures, see the chapter "Establishing Communications."

10. **UNIX:** On UNIX and Linux systems, make sure the password validation program `ingvalidpw` is installed. For details, see Create Password Validation Program (UNIX) (see page 46).

Note: The `ingvalidpam` program can be used instead of `ingvalidpw` to validate passwords through pluggable authentication modules (PAM). For more information, see the *Security Guide*.

Note: If you are upgrading an existing Ingres Net installation, your existing `netutil` (or `netu`) definitions remain in effect.

ingvalidpw Program

In some environments, Ingres uses the `ingvalidpw` program to validate user passwords. It is not strictly a part of Ingres Net. The passwords may originate from any local application or from a remote application coming through Ingres Net or the Data Access Server.

`Ingvalidpw` is used depending on the requirements of the platform where the password is validated. For example, the Ingres DBMS Server uses the `ingvalidpw` program to validate shadow passwords on UNIX or to enforce C2 security in some UNIX environments.

Create Password Validation Program (UNIX)

On UNIX, the Ingres DBMS Server uses the `ingvalidpw` program to validate shadow passwords. This executable is created at installation time or loaded from the distribution media.

The `mkvalidpw` script tries to recompile the `ingvalidpw` program if your machine has a C compiler available; otherwise it copies the supplied `ingvalidpw` program to the `$II_SYSTEM/ingres/bin` directory. The `mkvalidpw` script also sets the `II_SHADOW_PWD` variable in the Ingres `symbol.tbl` to enable shadow password validation.

To create the `ingvalidpw` program

1. Log in as root.
2. Set the `II_SYSTEM` and `PATH` variables to the same values as those for the user account that owns the installation.
3. Run the `mkvalidpw` script, located in the directory `$II_SYSTEM/ingres/bin`, as follows:

```
mkvalidpw
```

The `ingvalidpw` executable is created.

4. Shut down and restart the Name Server.

The `ingvalidpw` program is ready for operation.

Net Configuration Parameters—Customize Ingres Net

Your Net installation can be customized by changing the values of the Ingres Net configuration parameters. Default values are assigned during installation.

You view or change the configuration parameters using Configuration-By-Forms (or Configuration Manager, if available).

The Net configuration parameters are as follows:

inbound_limit and outbound_limit

Defines inbound and outbound session limits.

Default: 64 inbound sessions and 64 outbound sessions

log_level

Defines the logging level.

Default: 4

Protocol

Specifies the name of the network protocol.

Default: Any protocol present at installation is indicated as active.

Listen Address

Specifies the GCC listen addresses for this installation.

Default: A GCC listen address is assigned for any protocol present at installation. The format depends on the protocol.

default_server_class

Specifies the server class assumed as the default when a server class is specified.

Default: INGRES

remote_vnode

(Optional) Specifies the vnode assumed as the default when a vnode is not specified.

Default: None.

local_vnode

Specifies the vnode name configured for the local installation.

Default: Name of host machine.

For more information, see the following chapters:

- [Using Net](#)
- [Maintaining Connectivity](#)
- [Troubleshooting Connectivity](#)

Chapter 4: Establishing Communications

This section contains the following topics:

[How User Access Is Established](#) (see page 49)

[Access Tools for Defining Vnodes](#) (see page 51)

[Netutil \(Net Management Utility\)](#) (see page 53)

[Netutil Non-Interactive Mode](#) (see page 75)

[Network Utility and Visual DBA](#) (see page 89)

How User Access Is Established

For users to be able to access Ingres, the following two steps are required:

1. The system administrator sets up accounts for local users, and for those remote users who access the local instance through a local account. This step is optional if an Installation Password is defined, in which case users access Ingres directly, without going through a local account. The system administrator can do this either before or after the installation procedure.

Note: During installation, the installation owner user ID is defined. This account belongs to the Ingres administrator and is automatically authorized with maximum Ingres privileges that allow this user to perform all operations. Other user accounts can be set up after Ingres is installed.

2. After Ingres is running and the accounts are set up, the system administrator or database administrator uses Visual DBA or the **create user** statement to authorize the accounts that use this Ingres instance. For more information about this procedure, see the *Database Administrator Guide*.

Requirements for Accessing Remote Instances

When the instance includes Ingres Net, users can connect to databases on remote instances as well as those on their local instance. To connect to remote instances, the following requirements must be met:

- A virtual node (vnode) name must be defined for each remote instance that is accessed, unless you use the dynamic vnode format to connect.

A virtual node (vnode) is a name defined on the local instance that points to the connection data and authorization data necessary to access a particular remote instance. When a user on the local node wants to access a database on a remote instance or run an application that accesses a database on a remote instance, the user must specify the vnode name for the instance in addition to the name of the database.

The vnode name can be the same as the node's real address or node name. However, because the real names or addresses are often difficult to remember, and because there can be more than one instance on the node, other names are typically chosen for vnode names.

- A connection data entry must be defined for each remote instance that is accessed.

A connection data entry contains the information necessary for Ingres Net to locate and connect to an instance on a remote node. A connection data entry is typically associated with a particular, locally defined vnode, but can also be specified dynamically with the dynamic vnode format. It includes the name or address of the node on which the remote instance resides, the listen address of the remote instance, and an Ingres keyword for the network protocol used between the local and remote nodes.

- Remote user authorizations must be defined for each remote instance that is accessed.

A remote user authorization contains the login and password information necessary to gain access to a remote instance. It is typically associated with a particular, locally defined vnode, but can also be specified dynamically with the dynamic vnode format.

Note: It is not necessary for a particular user ID to be defined as an operating system account on an instance's host machine to be a valid Ingres user on that instance. An account on a remote node can be authorized in exactly the same way as an account on the local node. Provided an Installation Password has been defined locally, the remote account can then access the instance directly without having to go through a local account.

Instructions for defining an Installation Password for the local instance are provided later in this chapter. For the differences between these two types of passwords, see Remote User Authorizations (see page 30).

Requirements for Accessing Distributed Databases

Just as a local DBMS Server accesses a local database, a Star Server (part of the Ingres Star product) accesses a distributed database. When one or more of the databases that make up the distributed database reside on separate instances, Ingres Star uses Ingres Net. To use Ingres Star across Ingres Net, vnodes (with their corresponding connection data entries and remote user authorizations) must be defined on the Star server instance for each of the remote instances containing the databases that make up the distributed database.

If the connection data entries and the remote user authorizations are defined as private, connection data entries and remote user authorizations must also be defined for the installation owner (or the system administrator). These definitions are used if a distributed transaction fails. The Star Server attempts to recover the transaction as the owner of the Ingres Star instance.

Access Tools for Defining Vnodes

You define vnode names and their corresponding connection data entries and remote user authorizations using one of these tools:

- Net Management Utility (netutil)
- Network Utility (ingnet)
- Visual DBA

Not all tools are available on all platforms.

Note: The Network Utility (if supported on your platform) is the preferred means of creating vnodes in Ingres.

To access the Network Utility


Windows and UNIX Environments that Support Ingres Visual Tools:

Use one of the following ways:

- Choose Start, Programs, Ingres, Network Utility.
- From Ingres Visual Manager, choose File, Run, Ingres Network Utility or select the Network Utility toolbar button.
- Enter **ingnet** on the command line.

To access Visual DBA


Use one of the following ways:

- Choose Start, Programs, Ingres, Visual DBA, or right-click the Ingres installation icon in the Windows status bar and choose Visual DBA.
- From Ingres Visual Manager, choose File, Run, Ingres Visual DBA or select the Visual DBA toolbar button.
- Enter `vdba` on the command line. For the required command syntax, see the *Command Reference Guide*. 

To access `netutil`

Enter **`netutil`** on the command line. For the required command syntax, see the *Command Reference Guide*.

VMS and UNIX Environments that Do Not Support Ingres Visual Tools:

Enter **`netutil`** on the command line. For the required command syntax, see the *Command Reference Guide*. 

When Ingres and Ingres tools are installed as a cluster installation, it is necessary to run **`netutil`** from only one node to set up connection data entries and remote user authorizations for all of the nodes in the cluster.

Netutil (Net Management Utility)

The forms-based Net Management Utility, netutil, is used to define the connection and authorization data used by the Communications Server to access remote instances.

System administrators (or any user with the appropriate Ingres privileges) can use netutil to perform the following tasks:

- Add, change, or delete global remote user authorizations or connection data entries.

These tasks require the GCA privilege NET_ADMIN.

- Add, change, or delete any user's private remote user authorizations or connection data entries using the **-u** command flag.

These tasks require the NET_ADMIN privilege. For more information about the **-u** flag, which allows a user to perform operations on behalf of other users, see Command Line Flags in Netutil Non-interactive Mode (see page 76).

- Stop the Communications Server.

This task requires the GCA privilege SERVER_CONTROL. For instructions on stopping the Communications Server using the forms-based netutil, see the chapter "Maintenance Procedures."

End users can use netutil to:

- Add, change, or delete their private connection data entries.
- Add, change, or delete their private remote user authorizations.

Netutil Startup Screen

The netutil user interface consists of four tables and a menu of operations that can be performed on entries in these tables.

The four tables on the netutil startup screen are:

- Virtual Node Name (vnode) table
- Login/password data table
- Connection data table
- Other attribute data table. On the Startup screen, choose Attributes.

Virtual Node Name Table in Netutil

The Virtual Node Name table on the netutil startup screen determines what information is displayed in the Connection data and Login/password tables. These tables display information about the vnode highlighted in the Virtual Node Name table.

Naming Rules for Vnodes

Valid vnode names cannot include:

- Double colons (::)
- Slashes (/)
- Commas (,)

Vnode names are not case-sensitive, except on Star Server installations.

Login and Password Data Table in Netutil

The Login and password data table on the netutil startup screen is used for the following tasks:

- To record a remote user authorization using a login account and password on the remote instance's host machine
- To record a remote user authorization using the Installation Password defined on the remote instance
- To define an Installation Password for the *local* instance

The information you enter into the fields in the Login/password data table depends on which of the above tasks you are performing. For more information, see Task-Specific Values for the Login/Password Data Fields (see page 56).

The Login/password data columns are as follows:

Type

Is the type of definition, either Global or Private. For details, see Global and Private Definitions (see page 31).

Scope

Is a read-only message, supplied automatically, that briefly describes the scope of the connection. The message depends on the value that you enter in the Type field.

If you enter Private, netutil displays the following message:

User *user_id* only

If you enter Global, netutil displays the following message:

Any user on *node_name*

Login

Specifies the name of the account to be used on the remote instance's host machine.

Note: If you are authorizing access to a remote instance using an Installation Password or defining an Installation Password for the local instance, enter an asterisk (*) into this field.

After you fill in this field, netutil prompts you for a password.

Task-Specific Values for the Login/Password Data Fields

The following table shows the required values for the Type, Login, and Password fields according to the kind of record you are entering:

		Type	Login	Password
Remote User Authorization	Using login account password	Global or Private	Name of remote login account	Password of remote login account
	Using Installation Password	Global or Private	* (asterisk)	Installation Password of remote instance
Local Installation Password		Global	* (asterisk)	Local Installation Password

Note: When creating a local installation password, the vnode name used must be identical to the name that has been configured as LOCAL_VNODE on the Configure Name Server screen of the Configuration-By-Forms (cbf) utility and is generally the same as the local machine name.

Connection Data Table in Netutil

The Connection data table on the netutil startup screen specifies the network address of the remote node, the listen address of the remote instance's Communications Server, and the network protocol that is used to make the connection.

The Connection data table has the following columns:

Type

Specifies type of connection, either global or private.

Net Address

Identifies the network address or name of the remote node.

Your network administrator specifies this address or name when the network software is installed. Normally, the node name as defined at the remote node is sufficient for the node address.

The format of the net address depends on the type of network software used by the node. For protocol-specific information, see the appendixes in this guide.

Protocol

Specifies the Ingres keyword for the protocol used by the local node to connect to the remote node.

Protocol availability varies by platform. For a list of protocols and their associated Ingres keywords, see Network Protocol Keywords (see page 58).

Listen Address

Identifies the unique identifier used by the remote Communications Server for interprocess communication.

Just as the vnode name identifies an instance on the network, the listen address identifies a process (the Communications Server) in the remote instance.

The format of a remote node listen address depends on the type of network software that the node is using. For protocol-specific information, see the appendixes in this guide.

Network Protocol Keywords

When entering connection data for a remote instance, you are prompted for the name of the network protocol that is used to make the connection. You must respond with one of the following keywords:

tcp_ip

TCP/IP Internet protocol for UNIX and Windows (using WinSock 2.2 API). This is the default TCP/IP protocol and supersedes wintcp.

wintcp

(Deprecated) TCP/IP Internet protocol for Windows (using WinSock 1.1 API).

lanman

Microsoft NetBIOS protocol for Windows (using WinSock 1.1 API)

nvlspx

Novell Network SPX/IPX protocol for Windows (using WinSock 1.1 API)

decnet

DECnet protocol for VMS

sna_lu0

SNA LU0 protocol for MVS and VMS

sna_lu62

SNA LU62 protocol for RS/6000, HP/UX, Solaris, and MVS

tcp_dec

TCP/IP Services for OpenVMS and Multinet TCP/IP when running in TCP/IP Services emulation

tcp_ibm

IBM TCP/IP for MVS

tcp_knet

KNET TCP/IP for MVS

tcp_sns

TCP/IP protocol for SNS TCP/IP

tcp_wol

Wollongong TCI/IP Internet protocol for VMS and Multinet TCP/IP when running in Wollongong emulation

spx

Novell Network SPX/IPX for UNIX and VMS

For additional information on the protocols supported for your environment, see the Ingres Readme file for your operating system.

Other Attribute Data Table in Netutil

The Other attribute data table in netutil specifies additional connection, encryption and authentication attributes for a vnode. For a description of each attribute and its associated values, see Configure Vnode Attributes (see page 64).

The Other attribute data columns are as follows:

Type

Is the type of connection, either Global or Private.

Attr_Name

Is the name of the attribute.

Attr_Value

Is the value of the attribute.

Netutil Operations

The following operations are available from the netutil startup screen:

Create

Creates a new record in the highlighted table.

In the vnode table, this operation allows you to create a new vnode name and define its user authorization and connection data.

In the Connection data table or Login/password data table, this operation allows you to create an additional entry for an existing vnode.

Destroy

Deletes the highlighted record.

Note: Deleting a record in the Virtual Node Name (vnode) table automatically deletes the Login/password and Connection data table records associated with that vnode.

Attributes/Login

Toggles the display to show attribute or login information for the highlighted node. The initial display shows login information, and the Attributes menu option appears on the menu. Choosing Attributes displays attribute information, and the Attributes menu option is replaced by the Login menu option. Choosing Login brings back the original display.

Edit

Modifies the highlighted record.

Control

Stops or quiesces the local Communications Server. This menu item takes you to the Network Server Control screen.

Test

Tests a vnode after all of the user authorization and connection data has been defined.

Netutil tests to see if a connection can be made to the remote instance using any of the connection data entries and remote user authorizations defined for the vnode. Note that individual connection data entries and remote user authorizations cannot be tested.

Help

Displays help screens.

End

Exits netutil.

Prerequisites to Establish and Test a Remote Connection

To establish and test a remote connection, the following information is required:

- The network address or name of the node on which the remote instance resides.
- The listen address of the remote instance's Communications Server.
- The keyword for the network protocol that is used to make the connection. For more information, see Network Protocol Keywords (see page 58).
- The name of the login account that is used to access the remote instance. (This information is not applicable when using an Installation Password to authorize access.)
- The password of the remote login account or the remote instance's Installation Password.

Establish and Test a Remote Connection Using Netutil

You use netutil to establish and maintain access to remote instances. Defining a virtual node name is the first step in the process of establishing a connection.

To define a virtual node (vnode) and use it to test a connection to a remote instance

1. Enter the following command at the operating system prompt:

```
netutil
```

The netutil startup screen appears.

2. Make sure that the cursor is in the Virtual Node Name table; then choose Create from the menu.

A pop-up window appears, displaying the following prompt:

Enter new virtual node name.

3. Enter a virtual node name of your choosing and select OK from the menu.

A pop-up window appears, displaying the following prompt:

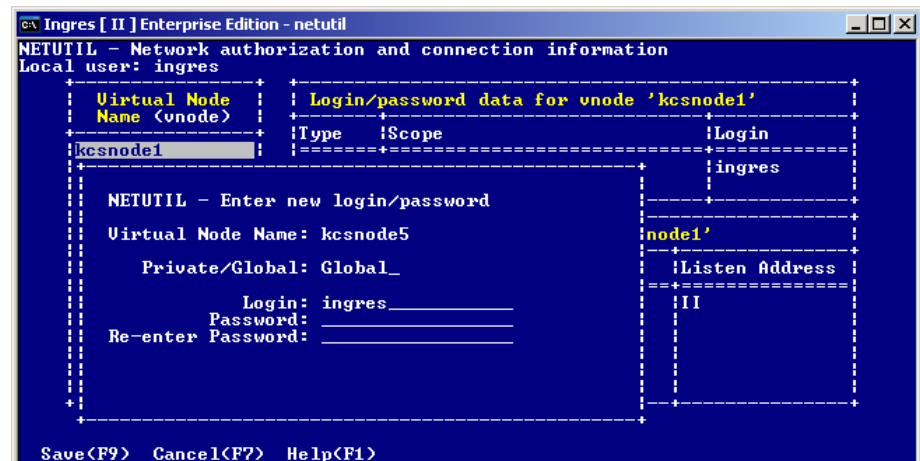
Choose type of login to be created

Global—Any user on [local node]

Private—User [user name] only

4. Use the arrow keys to highlight Global or Private; then choose Select from the menu.

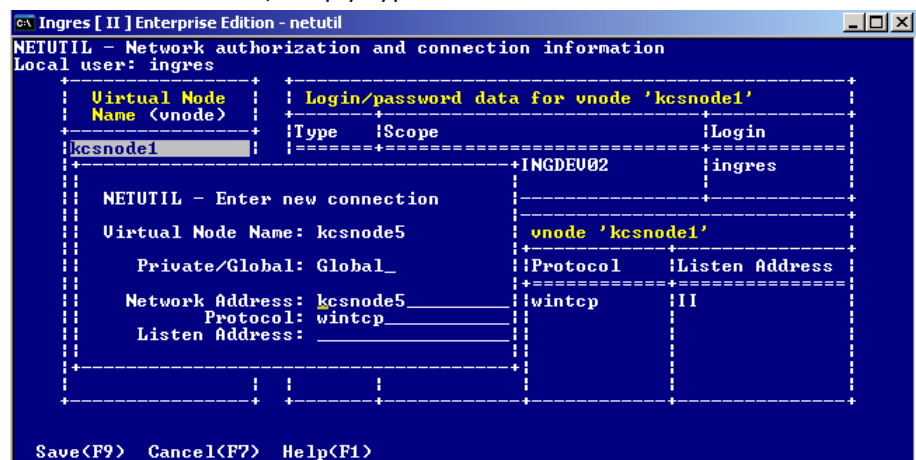
The Enter new login/password pop-up window appears. It displays prompts for the login of the account that is used on the remote node, the password of that account, and verification of the password.



- Enter the login and the password, then re-enter the password as prompted. (Notice that for security purposes neither the password nor the verification appears on screen.) When you have entered the login and password information, choose Save from the menu.

Note: If you are using an Installation Password to authorize access, enter an asterisk (*) in the Login field, and then enter the remote instance's Installation Password in the Password field.

The Enter new connection pop-up window appears. It displays prompts for the connection type (private or global), the network address, the network protocol to be used, and the Listen address of the remote instance. For your convenience, netutil supplies default values for the first three fields. To enter a new value, simply type over the default value.



- Enter the connection data, and then choose Save from the menu.

Netutil returns to the startup screen. The data you entered in this and the previous steps is displayed in the Vnode, Login/password data, and Connection data tables.

- Choose Test from the menu.

Netutil attempts to establish a connection to the remote instance using authorization and connection data you have entered.

A message is displayed in a pop-up window indicating whether the test is successful.

If the connection is not successful, the error message indicates the nature of the error or where to look for further information.

- Press Return.

You are returned to the startup screen.

Configure Vnode Attributes

In addition to defining login and connection data, you can use netutil to configure vnode attributes. Attributes define additional connection, encryption, and authentication information for the vnode.

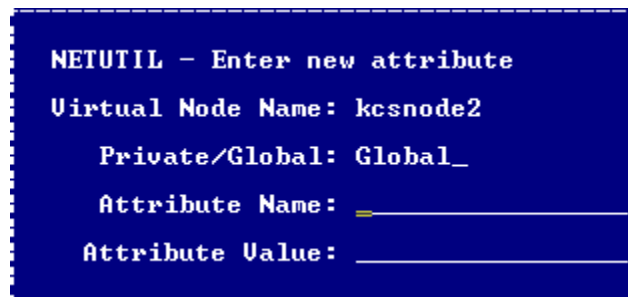
To configure one or more attributes for a vnode

1. From the netutil startup screen, select the Attributes menu option.

Attribute information for the first vnode in the Virtual Node Name table is displayed.

2. Use the arrow keys to select the vnode for which you want to configure an attribute. Tab to the Other attribute data for vnode table and select the Create menu option.

The Enter new attribute pop-up window appears.



```
NETUTIL - Enter new attribute
Virtual Node Name: kcsnode2
Private/Global: Global_
Attribute Name: _____
Attribute Value: _____
```

3. Enter an attribute name and value (see page 65).
4. Select the Save menu option.

Netutil returns to the startup screen. The attribute you configured is now displayed in the Other attribute data for vnode table.

Vnode Attribute Names and Values

Valid vnode attribute names and values are as follows:

connection_type

Indicates the connection type. The only valid value is **direct**, which indicates that a direct connection with the remote instance must be established without using Net. This attribute improves performance because data goes directly from the application process on the client machine to the Ingres DBMS process on the server machine, thus bypassing Communications Server processing, except for initial connection setup.

For direct access to occur, the following conditions must be met:

- The client and server machines must be the same platform type (for example, both Windows, or both Solaris, or Windows to Linux on Intel-based machines) and the environment variable `II_CHARSET` must be identical on both the client and server machines.
- The same network protocol must be used for the local IPC on the client and server machines.
- On Windows, the client and server machines must be logged into the same Windows network domain if using (default) named pipes for local IPC.
- Ingres II 2.5 or higher must be installed on both the client and server machines.
- On any platform which uses the IPC protocol for local and network connections, the environment variable, `II_GC_REMOTE` must be set (for the DBMS Server instance) to `ON` or `ENABLED` to allow direct connections. For important information on network security and `II_GC_REMOTE`, see the *System Administrator Guide*.
- Between Windows and UNIX or Linux, `II_GC_PROT` is set to `tcp_ip` on Windows, and the UNIX or Linux system is running TCP/IP for its local protocol.
- An Ingres client installation should not be at a higher major release level than the Ingres server. While this is not a requirement across all releases of Ingres, Ingres 10 clients specifically are not supported for direct access to pre-Ingres 10 servers; connection attempts will likely hang. The reverse configuration works: An older release Ingres client can directly access a newer release Ingres server.

encryption_mode

Specifies the encryption mode for the connection. If set, this value overrides the Communications Server's `ob_encrypt_mode` parameter value configured using Configuration Manager or the Configuration-By-Forms utility. The local and remote Communications Servers must be able to negotiate a common mechanism to perform the encryption. Valid values are:

- `off` – No encryption. The connection fails if the remote Communications Server has an encryption mode of **required**.
- `optional` – Encryption occurs if the remote Communications Server has an encryption mode of **on** or **required**.
- `on` – Encryption occurs if the remote Communications Server has an encryption mode other than **off**.
- `required` – Encryption occurs if the remote Communications Server has an encryption mode other than **off**. If `off`, the connection fails.

encryption_mechanism

Specifies the mechanism to be used to encrypt the remote connection. If set, this value overrides the Communications Server's `ob_encrypt_mech` parameter value configured using Configuration Manager or the Configuration-By-Forms utility. Valid values are:

- `none` – Disables encryption
- `*` – Allows all encryption mechanisms to be considered during Communications Server negotiations
- *mechanism_name* – Indicates a specific mechanism to be considered during Communications Server negotiations

authentication_mechanism

Specifies the mechanism to be used for remote authentication in a distributed security environment. This setting replaces the need for a user ID and password. If set, this value overrides the Communications Server's `remote_mechanism` parameter value configured using Configuration Manager or the Configuration-By-Forms utility. The only valid value is **kerberos**.

Multiple Connection Data Entries

If a remote instance has more than one Communications Server or can be accessed by more than one network protocol, that information can be included in the vnode definition by adding another connection data entry. This allows you to distribute the load of communications processing and increase fault tolerance.

Note: When more than one Communications Server listen address is defined for a given vnode, Ingres Net automatically tries each server in random order until it finds one that is available. Similarly, when a connection fails over one network protocol, Ingres Net automatically attempts the connection over any other protocol that has been defined.

End users can create private connection data for an existing vnode by adding an entry to the Connection Data table. For the user who creates it, a private connection data entry overrides a global connection data entry defined to the same vnode. In other words, Ingres Net uses the private connection data entry whenever the user who created the entry uses the vnode.

You need the following information before adding a connection data entry:

- The network address of the node on which the remote instance resides
- The listen address of the remote instance's Communications Server. (For the correct listen address format for your network protocol, see the appropriate appendix or check the Configure Net Server Protocols screen in the Configuration-By-Forms utility on the remote instance.)
- The keyword for the network protocol (see page 58) that is used to make the connection.

To define an additional connection data entry in netutil

1. Select the desired vnode in the Virtual Node Name table.

The connection data for the highlighted vnode appears in the Connection Data table.

2. Move the cursor to the Connection Data table, and then choose Create from the menu.

The Enter new connection pop-up window appears displaying prompts for the connection type (private or global), the network address, the network protocol to be used, and the Listen address of the remote instance. For your convenience, netutil supplies default values for the first three fields; to enter a new value, simply type over the default value.

3. Enter the connection data; then choose Save from the menu.

Netutil returns to the startup screen. The data you entered is now displayed in the Connection Data table.

Additional Remote User Authorization

End users can create a private remote user authorization for an existing vnode.

For the user who creates it, a private authorization overrides a global authorization defined to the same vnode. Ingres Net uses the private authorization whenever that user uses the vnode.

You need the following information before adding a private remote user authorization:

- The name of the remote account that is used to access the remote instance
(This information is not applicable when using an Installation Password to authorize access.)
- The password of the remote login account or the remote instance's Installation Password.

To define and test a new remote user authorization in netutil

1. Select the desired vnode in the Virtual Node Name table.
The remote user authorization for the highlighted vnode appears in the "Login/password data" table.
2. Move the cursor to the "Login/password data" table, and then choose Create from the menu.
The Enter New Login/Password pop-up window appears. It displays prompts for the login of the account that is used on the remote node, the password of that account, and verification of the password.
3. Enter the login and the password, and then re-enter the password as prompted. (For security purposes neither the password nor the verification appears on screen.)

Note: If you are using an Installation Password to authorize access, enter an asterisk (*) in the Login field, and then enter the remote instance's Installation Password in the Password field.

Choose Save from the menu.

Netutil returns to the startup screen. The data you entered is now displayed in the Login/password data table.

Delete an Entry

To delete a virtual node entry or one of its connection data entries, remote user authorizations or attributes, place the cursor on the desired record and choose Destroy from the menu.

Delete All Vnode Information

To delete all information for a specific vnode

1. Highlight the desired entry in the Virtual Node Name table and choose Destroy from the menu.

A pop-up window appears with the following prompt:

Really destroy all data for vnode [vnode name]?

No—Do not destroy all data for vnode

Yes—Destroy all data for vnode

2. Use the arrow keys to highlight No or Yes (No is the default); then choose Select from the menu.

Netutil removes the vnode from the Virtual Node Name table and all associated information from the Login/password data and Connection data tables.

Delete a Connection Entry for a Vnode

To delete one of the connection data entries associated with a particular vnode

1. Highlight the desired entry in the Connection Data table and choose Destroy from the menu.

A pop-up window appears with the following prompt:

Really destroy connection entry?

No—Do not destroy connection entry

Yes—Destroy connection entry

2. Use the arrow keys to highlight No or Yes (No is the default), and then choose Select from the menu.

Netutil removes the entry from the Connection Data table.

Delete a Remote User Authorization for a Vnode

To delete one of the remote user authorizations associated with a particular vnode

1. Highlight the desired entry in the Login/password data table and choose Destroy from the menu.

A pop-up window appears with the following prompt:

Really destroy [private/global] login/password entry '[Login name]'?

No—Do not destroy [private/global] login/password entry

Yes—Destroy [private/global] login/password entry

2. Use the arrow keys to highlight No or Yes (No is the default); then choose Select from the menu.

Netutil removes the entry from the Login/password data table.

Delete an Attribute Associated with a Vnode

To delete an attribute associated with a particular vnode

1. From the netutil startup screen, select the Attributes menu option.
The "Other attribute data" table is displayed.
2. Select the desired vnode from the Virtual Node Name table. Tab to the Other attribute data for vnode table, highlight the attribute that you want to delete, and choose Destroy from the menu.

A pop-up window appears with the following prompt:

Really destroy attribute entry?
No—Do not destroy attribute entry
Yes—Destroy attribute entry

3. Use the arrow keys to highlight No or Yes (No is the default); then choose Select from the menu.

Netutil removes the attribute from the Other attribute data for vnode table.

Change an Entry

To modify a virtual node entry or one of its Connection data entries, remote user authorizations, or attributes, place the cursor on the desired record and select Edit from the menu.

Modify a Vnode Name

To modify a vnode name

1. Select the desired entry in the Virtual Node Name (vnode) table ,and then choose Edit from the menu.

A pop-up window appears, displaying the following prompt:

Enter the new name for ['vnode name']
New name:

2. Enter the new virtual node name and choose OK from the menu.

The Enter Global/Private Password pop-up window appears and prompts you to re-enter the remote account password or Installation Password associated with this vnode. For security reasons, any time a vnode name is modified, you must re-enter the associated passwords.

3. Enter the password, re-enter the password as prompted, and then choose Save from the menu.

If there is a second remote user authorization associated with this vnode, a second pop-up window appears. Repeat this step with the password of the second authorization.

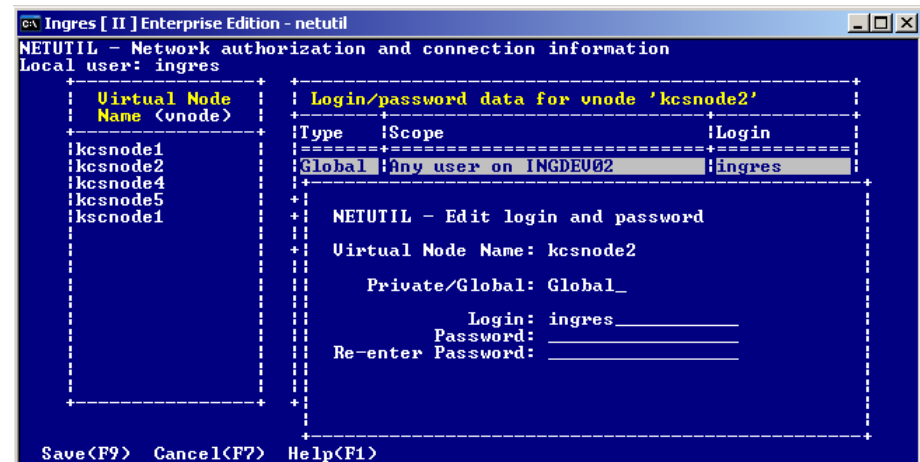
After you have saved all password information, netutil returns to the startup screen. The edited vnode name is displayed in the Virtual Node Name (vnode) table.

Edit a Remote User Authorization

To edit a remote user authorization

1. Select the desired entry in the Login/password data table, and choose Edit from the menu.

The Edit login and password pop-up window appears and prompts you to enter new login and password data.



2. Enter the login and password for the remote account; then re-enter the password as prompted.

Note: If you are using an Installation Password to authorize access, enter an asterisk (*) in the Login field, and then enter the remote instance's Installation Password in the Password field.

3. Choose Save from the menu.

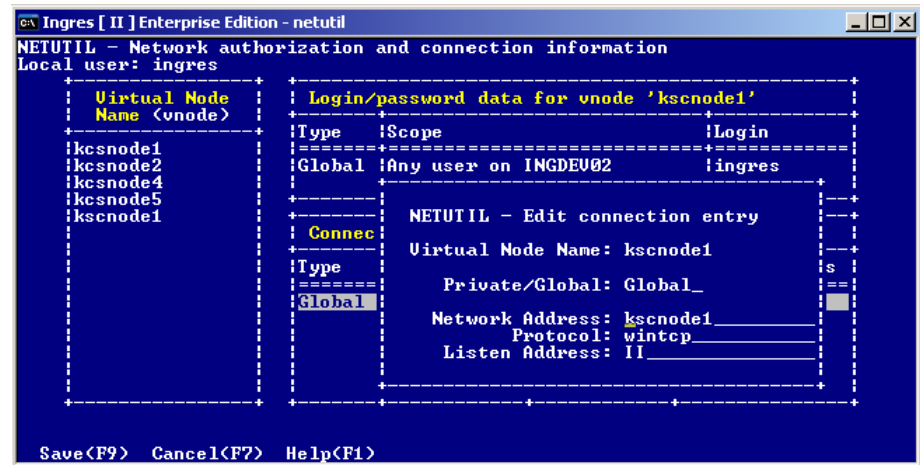
Netutil returns to the startup screen. The edited remote user authorization is displayed in the Login/password data table.

Edit a Connection Data Entry

To edit a connection data entry

1. Select the desired entry in the Connection Data table, and choose Edit from the menu.

The Edit connection entry pop-up window appears, which displays the connection type, network address, protocol, and listen address for the selected entry.



2. Tab to the fields to be changed and enter the new values. Choose Save from the menu.

Netutil returns to the startup screen. The edited connection data entry is displayed in the Connection Data table.

Edit Vnode Attribute

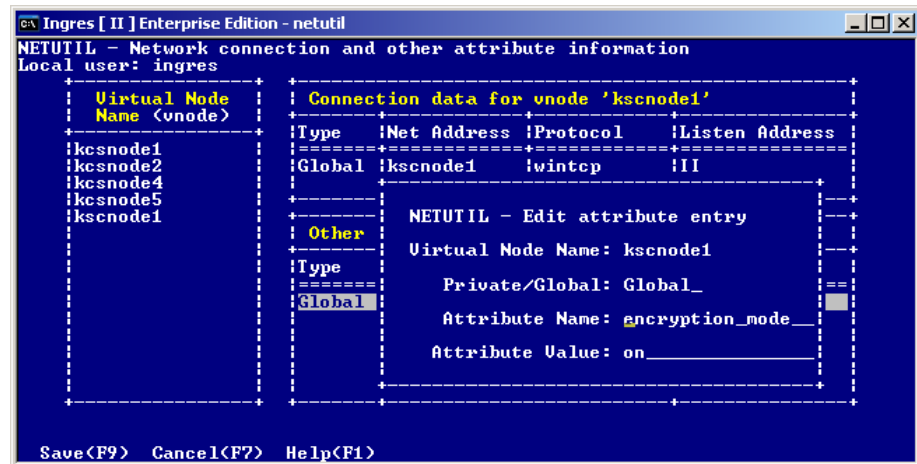
To edit attribute data for a particular vnode

1. From the netutil startup screen, select the Attribute menu option from the netutil startup screen.

The "Network connection and other attribute information screen" appears.

2. Select the desired vnode from the vnode list. Tab to the Other attribute data for vnode table and select the attribute that you want to edit. Choose Edit from the menu.

The Edit attribute entry pop-up window appears.



3. Edit the attribute by typing over the displayed data with the desired changes. For a list of valid attribute names and values, see Configure Vnode Attributes (see page 64). Choose Save from the menu.

Netutil returns to the startup screen. The attribute you edited is now displayed in the Other attribute data for vnode table.

Define an Installation Password for the Local Instance

To define an Installation Password for the local instance

1. Enter the following command at the operating system prompt:
`netutil`

The netutil startup screen appears.
2. Make sure that the cursor is in the Virtual Node Name table, and then choose Create from the menu.

A pop-up window appears, displaying the following prompt:
`Enter new virtual node name:`
3. Enter the virtual node name for the local instance and choose OK from the menu.

Note: The virtual node name must be identical to the name that has been configured as LOCAL_VNODE on the Configure Name Server screen of the Configuration-By-Forms (cbf) utility and is typically the same as the local machine name.

A pop-up window appears, displaying the following prompt:
`Choose type of login to be created`
`Global—Any user on [local node]`
`Private—User [user name] only`
4. Highlight Global by using the arrow keys, and then choose Select from the menu.

The Enter new login/password pop-up window appears. It displays prompts for the global login, the password, and verification of the password.
5. Enter an asterisk in the Global Login field, enter an Installation Password in the Password field, and finally re-enter the password as prompted. (Notice that for security purposes neither the password nor the verification appears on screen.) When you have entered this information, choose Save from the menu.

The Enter new connection pop-up window appears. It displays prompts for the connection type (private or global), the network address, the network protocol to be used, and the Listen address of the instance.
6. Choose Cancel from the menu. (It is not necessary to enter connection data for the local instance.)

Netutil returns to the startup screen. The data you entered in the previous steps is displayed in the vnode and Login/password data tables.

Netutil Non-Interactive Mode

Netutil supports a non-interactive mode of operation controlled by command line flags and an input control file. You can use this mode if you want to write your own system administration utility programs or authorize large numbers of users using a batch file.

The following functions are available through this interface:

Create

Creates a new connection data entry or remote user authorization.

Destroy

Destroys a connection data entry or remote user authorization.

Show

Displays information to the terminal. This function does not correspond to a menu item in the forms-based interface.

Stop

Stops all Communications Servers.

For example, this command stops a specific Communications Server:

```
stop 2937
```

Quiesce

Stops all Communications Servers after the sessions currently in progress on those servers have terminated.

For example, this command quiesces a specific Communications Server:

```
quiesce 2116
```

Note: The Edit and Test functions found in the forms-based netutil interface are not supported in non-interactive mode.

Command Line Flags in Netutil Non-interactive Mode

The following command line flags are supported in netutil's non-interactive mode:

-u user

Impersonate the specified user for the purpose of managing private authorization and connection entries. Only a user with the NET_ADMIN privilege (generally a system administrator) can impersonate another user.

-file filename

When this flag is used, netutil processes commands specified in the indicated input control file.

The format of the input control file is described in the following section.

-file-

If the input file is specified as "-" (a single dash character), input is taken from the standard input channel. This allows the user to enter commands directly from the keyboard or to run netutil as part of a UNIX pipeline. To exit, press Ctrl+Z.

-vnode vnode

Connect to the Name server on the remote instance specified by the vnode name.

The vnode name must be defined on the local host's Name server; that is, connection and authorization information must exist locally for that vnode name. This information can be defined by invoking netutil on the local Name server.

Input Control File

The input control file is an ASCII file that stores instructions about operations to be performed on the Name Server database. Each line of the file represents either a create, destroy, or show operation. These lines are called "input lines" in the remainder of this section.

The following conventions are observed:

- Blank lines are ignored in the control file.
- Case is insignificant except where significance is imposed by the usage of the data. For example, the login name on a UNIX system has significant case.
- The character "#" indicates a comment; all text following a "#" character on any line is ignored.
- Input lines are divided into fields, which are separated by a blank space. For example, the following input line contains four fields:

```
show private login paulj
```

Invariant Fields

The first four fields of an input line describe the action to be performed and the vnode with which the action is associated. These four fields appear in the order given below in every input line (except stop and quiesce server commands).

The following table defines these fields and their potential values:

Field	Parameter	Value	Description
1	Function	Create, Destroy, or Show	The task that is performed.
2	Type	Global or Private	The registration type of the object. A global object is available to all users on the local node. A private object is available to a single user.
3	Object	Login or Connection Attribute	The object to be created, destroyed, or shown. "Connection" refers to a connection data entry. "Login" refers to a remote user authorization. "Attribute" refers to a vnode attribute entry.
4	Virtual Node Name	Vnode name	The virtual node name. Each line in the input control file must contain a vnode identifier.

Note: Values in any of the first three fields (Function, Type, and Object) can be abbreviated to a unique left substring. In practice, this means that a single-letter abbreviation is sufficient for any of these fields.

Values in the Virtual Node Name field cannot be abbreviated.

In addition to the four fields discussed above, other fields are required depending on the task to be accomplished by the input line. For example, an input line creating a remote user authorization requires an additional two fields: a login field and a password field. An input line creating or destroying a connection data entry requires an additional three fields: a network address field, a protocol field, and a listen address field.

For detailed information about additional fields, see the examples that follow.

Wildcards

On input lines that specify either the Destroy or Show function, the asterisk character (*) can be entered as a wildcard in any field other than the Function, Type, and Object fields.

The asterisk character (*) indicates that the field is not to be used in selecting the data records to which the function is applied. Therefore, it is possible to destroy or display a number of records with a single input line.

Note: Wildcards cannot be used with the Create function.

Create Function—Create a Remote User Authorization

In netutil non-interactive mode, you can use the create function to create a remote user authorization.

This function has the following format when used to create a remote user authorization:

```
create type login vnode login password
```

type

Specifies the type of entry. Valid values are:

global

Indicates that the object is available to all users on the local node.

private

Indicates that the object is available to a single user.

vnode

Identifies the virtual node name associated with this authorization.

login

Identifies the name of the account to be used on the remote instance's host machine.

If you are authorizing access to the remote instance using an Installation Password, an asterisk (*) must be entered into this field.

password

Identifies the password of the remote account or the remote instance's Installation Password, depending on which method of authorization you are using.

Examples: Create a Remote User Authorization

This command creates a private authorization for vnode "payroll" for user Jane:

```
C P L Payroll jane jpassword
```

This command creates a global authorization for vnode "accounting" using an Installation Password:

```
cr gl login accounting * acctpassword
```

Note: Any previously existing authorization of the specified type is replaced by the execution of this line.

Note: Private authorizations are created for the currently logged-in user or for the user identified by the -u flag. Only a user with the GCA privilege NET_ADMIN can create a global authorization.

Destroy Function—Destroy a Remote User Authorization

In netutil non-interactive mode, you can use the destroy function to destroy a remote user authorization.

This function has the following syntax when destroying a remote user authorization:

```
destroy type login vnode
```

type

Specifies the type of entry. Valid values are:

global

Indicates that the object is available to all users on the local node.

private

Indicates that the object is available to a single user.

vnode

Identifies the virtual node name associated with this authorization.

Examples: Destroy a Remote User Authorization

This command destroys a private login on vnode "payroll." The entry to be destroyed is uniquely identified by its type and the vnode name. No additional fields are necessary.

```
DE PR L payroll # Current user now uses global login
```

This command destroys a private login on all vnodes where it occurs. Using a wildcard in the vnode field lets you destroy all instances of a particular login with a single input line:

```
DE PR L *
```

Note: Private authorizations are destroyed for the currently logged-in user or for the user identified by the -u flag. Only a user with the GCA privilege NET_ADMIN can destroy a global authorization.

Show Function—Display Remote User Authorizations

In netutil non-interactive mode, you can use the show function to display remote user authorizations. The login information for the specified vnodes is displayed on the terminal, or written to standard output (Windows and UNIX) or SYS\$OUTPUT (VMS). The password is not displayed.

The information is displayed in a format similar to that of control file input lines for ease of use in programs that edit and re-use the information.

The show function has following format for displaying remote user authorizations:

```
show type login vnode
```

type

Specifies the type of entry. Valid values are:

global

Indicates that the object is available to all users on the local node.

private

Indicates that the object is available to a single user.

vnode

Identifies the virtual node name associated with this authorization. An asterisk (*) can be used as a wildcard in the vnode, field.

Example: Display Remote User Authorizations

The following command displays the global login of vnode "accounting:"

```
S GL login accounting
```

The following line is displayed:

```
global login accounting ingres
```

Create Function—Define an Installation Password for the Local Instance

In netutil non-interactive mode, you can use the create function to create an Installation Password for the local instance.

This function has the following format:

```
create global login local_vnode * password
```

local_vnode

Identifies the name that has been configured as LOCAL_VNODE on this instance. This name can be found on the Configure Name Server screen of the CBF utility.

password

Defines the Installation Password you have chosen for this instance.

Example: Define an Installation Password

This command defines an Installation Password for the local instance, which has a local_vnode name of "payroll:"

```
create gl login payroll * payroll_password
```

Create Function—Create a Connection Data Entry

In netutil non-interactive mode, you can use the create function to create a connection data entry.

This function has the following format:

```
create type connection vnode network_address protocol listen_address
```

type

Specifies the type of entry. Valid values are:

global

Indicates that the object is available to all users on the local node.

private

Indicates that the object is available to a single user.

vnode

Identifies the virtual node name associated with this connection entry.

network_address

Identifies the address or name of the remote node. Your network administrator specifies this address or name when the network software is installed. Normally, the node name as defined at the remote node is sufficient for this field.

The format of a net address depends on the type of network software that the node is using.

protocol

Specifies the keyword for the protocol used to connect to the remote instance. For a list of protocols and their associated keywords, see Network Protocol Keywords (see page 58).

listen_address

Is the unique identifier used by the remote Communications Server for interprocess communication. The format of a listen address depends on the network protocol.

Example: Create a Connection Data Entry

The following command creates a global connection data entry on vnode "payroll," where:

Network address = payroll

Protocol = TCP/IP

Listen address = fe0

```
C G C payroll payroll tcp_ip fe0 # payroll comsvr 1
```

Note: The virtual node name and the network address are different objects, although it is common for them to have the same value.

If a connection entry already exists that matches the specified one in all respects, the operation has no effect and no error is reported.

Note: Private connection data entries are created for the currently logged-in user or for the user identified by the -u flag. Only a user with the GCA privilege NET_ADMIN can create a global connection data entry.

Destroy Function—Destroy a Connection Data Entry

In netutil non-interactive mode, you can use the destroy function to destroy a connection data entry. To obtain the network address, protocol, and listen address of the connection data entry you want to destroy, use the show command.

This function has the following format:

```
destroy type connection vnode network_address protocol listen_address
```

type

Specifies the type of entry. Valid values are:

global

Indicates that the object is available to all users on the local node.

private

Indicates that the object is available to a single user.

vnode

Identifies the virtual node name associated with this input line. An asterisk (*) can be used as a wildcard to select a range of records.

network_address

Identifies the address or name of the remote node. An asterisk (*) can be used as a wildcard to select a range of records.

protocol

Specifies the keyword for the protocol used to connect to the remote instance. For a list of protocols and their associated keywords, see Network Protocol Keywords (see page 58). An asterisk (*) can be used as a wildcard to select a range of records.

listen_address

Is the unique identifier used by the remote Communications Server for interprocess communication. An asterisk (*) can be used as a wildcard to select a range of records.

Examples: Destroy a Connection Data Entry

The following command destroys a private connection data entry on vnode "payroll", where:

Network address = payroll

Protocol = TCP/IP

Listen address = fe2

```
D p c payroll payroll tcp_ip fe2 # No comm server on fe2
```

The following command destroys all global connection data entries for vnode "accounting" that include the TCP/IP protocol:

```
d gl c accounting * tcp_ip *
```

Note: Private connection data entries are destroyed for the currently logged-in user or for the user identified by the -u flag. Only a user with the GCA privilege NET_ADMIN can destroy a global connection data entry.

Show Function—Display Connection Data Entries

In netutil non-interactive mode, you can use the show function to display connection data entries. The connection information for the specified vnode is displayed on the terminal, or written to standard output (Windows and UNIX) or SYS\$OUTPUT (VMS). The information is displayed in a format similar to the format of control file input lines, for ease of use in programs that edit and re-use the information. The password is not displayed.

This function has the following format:

```
show type connection vnode network_address protocol listen_address
```

type

Specifies the type of entry. Valid values are:

global

Indicates that the object is available to all users on the local node.

private

Indicates that the object is available to a single user.

vnode

Identifies the virtual node name associated with this input line. An asterisk (*) can be used as a wildcard to select a range of records.

network_address

Identifies the address or name of the remote node. An asterisk (*) can be used as a wildcard to select a range of records.

protocol

Specifies the keyword for the protocol used to connect to the remote instance. For a list of protocols and their associated keywords, see Network Protocol Keywords (see page 58). An asterisk (*) can be used as a wildcard to select a range of records.

listen_address

Is the unique identifier used by the remote Communications Server for interprocess communication. An asterisk (*) can be used as a wildcard to select a range of records.

Example: Display Connection Data Entries

The following displays global connection data entries on vnode "payroll," where:

Network address = payroll

Protocol = * (This field is not to be used in selecting records.)

Listen address = * (This field is not to be used in selecting records.)

```
S GL conn payroll payroll * *
```

The following line is displayed:

```
global connection payroll payroll tcp_ip fe2
```

Stop and Quiesce Commands—Stop or Quiesce One or More Communications Servers

In netutil non-interactive mode, to stop all Communications Servers on the instance, enter the following commands at the system prompt:

```
netutil -file-  
stop
```

To stop a single Communications Server, enter the following commands at the system prompt:

```
netutil -file-  
stop server_id
```

server_id

Is a unique string that identifies a particular Communications Server on the instance. To find the *server_id*, use the iinamu utility.

Examples: Quiesce One or More Communications Servers

The following commands entered at the system prompt quiesce all Communications Servers on the instance (that is, stops the Communications Servers after all current sessions have terminated):

```
netutil -file-  
quiesce
```

The following commands entered at the system prompt quiesce Communications Server 2937 (that is, stop the server after all current sessions have terminated):

```
netutil -file-  
quiesce 2937
```

Note: Only a user with the GCA privilege SERVER_CONTROL can stop a Communications Server.

Network Utility and Visual DBA

The GUI-based tools Network Utility (ingnet) and Visual DBA can both be used to define vnodes containing the connection and authorization data used by the Communications Server to access remote instances.

Because Network Utility is a tool dedicated to defining and managing vnodes, it is preferred over netutil or Visual DBA for performing these tasks in Ingres.

System administrators (or any user with the appropriate Ingres privileges) can use these visual tools to perform the following tasks:

- Add, change, or delete global remote user authorizations or connection data entries.
- Add, change, or delete any user's private remote user authorizations or connection data entries.
- Define an Installation Password for the *local* instance

End users can use these visual tools to:

- Add, change, or delete their private connection data entries.
- Add, change, or delete their private remote user authorizations.

In Network Utility and Visual DBA, vnodes are defined using vnode objects. A vnode object specifies a virtual node name, and login and connection information.

Using the Nodes branch in the Virtual Nodes toolbar/window, you can create and alter vnodes, view vnode objects, and drop vnode objects.

Detailed steps for these procedures can be found in the Procedures section of online help for Network Utility and Visual DBA.

Virtual Nodes Toolbar

The Virtual Nodes toolbar in both Network Utility and Visual DBA provides the same functionality. Use the toolbar to create, alter, drop, and disconnect vnodes. The toolbar also provides features that allow you to connect to database servers and open various types of database administration utility windows.

For example, connect to the Database Object Manager and open a DOM Scratchpad. The toolbar also allows you to use an SQL Scratchpad, monitor your system performance, and display a list of Dbevents.

Simple and Advanced Vnodes

Virtual nodes are classified as *simple* or *advanced*. A simple vnode is one in which there is only one set of login and connection parameters associated with it. An advanced vnode is one in which there is more than one connection data definition and/or up to two login data definitions.

Advanced Vnode Parameters

You can maintain up to two login data definitions for each vnode. If the first definition has been defined as private, the other login must be global (and vice versa). A global entry is available to all users on the local instance. A private entry is available only to the user who creates it.

If one of the login data definitions is private and the other is global, the vnode is considered to be an *advanced* vnode. A vnode is also considered to be an advanced vnode if it has more than one connection data definition and/or has one or more vnode attribute definitions.

Use the Advanced Node Parameters branch in Network Utility and Visual DBA to:

- Add, alter, or drop connection data
- Add private and global logins
- Add, alter, or drop vnode attributes

For additional information on performing these tasks, see the following sections.

Multiple Connection Data Entries

If a remote instance has more than one Communications Server or can be accessed by more than one network protocol, that information can be included in the vnode definition by adding another connection data entry. This allows you to distribute the load of communications processing and increase fault tolerance.

Note: When more than one Communications Server listen address is defined for a given vnode, Ingres Net automatically tries each server in random order until it finds one that is available. Similarly, when a connection fails over one network protocol, Ingres Net automatically attempts the connection over any other protocol that has been defined.

End users can create private connection data for an existing vnode by adding an entry to the Connection Data table. For the user who creates it, a private connection data entry overrides a global connection data entry defined to the same vnode. In other words, Ingres Net uses the private connection data entry whenever the user who created the entry uses the vnode.

You need the following information before adding a connection data entry:

- The network address of the node on which the remote instance resides
- The listen address of the remote instance's Communications Server. (For the correct listen address format for your network protocol, see the appropriate appendix or check the Configure Net Server Protocols screen in the Configuration-By-Forms utility on the remote instance.)
- The keyword for the network protocol (see page 58) that is used to make the connection.

To define an additional connection data entry using Network Utility

See the topics under Defining Connection Data in the Procedures section of the Network Utility online help.

Additional Remote User Authorization

End users can create a private remote user authorization for an existing vnode.

For the user who creates it, a private authorization overrides a global authorization defined to the same vnode. Ingres Net uses the private authorization whenever that user uses the vnode.

You need the following information before adding a private remote user authorization:

- The name of the remote account that is used to access the remote instance
(This information is not applicable when using an Installation Password to authorize access.)
- The password of the remote login account or the remote instance's Installation Password.

To define and test a new remote user authorization using Network Utility

See the following topics in the Procedures section of the Network Utility online help:

- Adding a Login Database Definition
- Altering a Login Database Definition
- Dropping a Login Database Definition

Vnode Attributes Configuration

In addition to login and connection data, you can use Network Utility or Visual DBA to configure the following vnode attributes:

- connection_type
- encryption_mode
- encryption_mechanism
- authentication_mechanism

For a description of each attribute and its associated values, and for detailed steps for adding, altering and dropping these attributes, see the following topics in the Procedures section of online help for Network Utility and Visual DBA:

- Adding a Vnode Attribute
- Altering a Vnode Attribute
- Dropping a Vnode Attribute

Installation Password Definitions for the Local Instance

As previously explained, users can access a remote Ingres instance using a login account set up on the remote instance's host machine, or through an Installation Password, which allows users direct access to the instance. The Installation Password is configured for the local instance containing the databases that remote users want to access.

Detailed steps for this procedure can be found in the Procedures section of online help for Network Utility and Visual DBA. See the following online topic:

- Adding an Installation Password

Changing Installation Passwords

Changing installation passwords requires special care. Because of caching of information on the client and server, installation passwords must be changed at least 30 minutes after the last use of Ingres Net. Failure to do this can cause connections to fail with "E_GC0141_GCN_INPW_INVALID."

Additional Vnode-Related Tasks

In addition to connection data entries and user authorization tasks, you can use Network Utility and Visual DBA to perform vnode-related tasks, such as refreshing a vnode, testing a vnode, disconnecting from a vnode, and opening utility windows.

Refresh Vnodes

Refreshing a vnode updates loaded vnode data.

To refresh particular vnodes

Click the Force Refresh button on the toolbar.

or

Choose a node, Force Refresh.

You can configure your background refresh settings by choosing File, Preferences.

Test Vnodes

In Network Utility and Visual DBA, you can test if a connection to a specific node can be established.

To test a connection

From the Virtual Nodes toolbar, choose Node, Test Node.

If the connection fails, an error message is returned.

Disconnect from a Vnode

Closing a window does not end communications with the servers on that window. Network Utility and Visual DBA continue to request data refreshes from the vnode until you disconnect from it.

Detailed steps for this procedure can be found in the Procedures section of Network Utility and Visual DBA online help. See the topic Disconnecting From a Vnode.

Opening Utility Windows

By choosing a virtual node, you can both establish a physical connection between the database server and your client workstation, and also open one of the four types of Visual DBA database administration windows.

The detailed steps for these procedures can be found in the Procedures section of online help for Network Utility and Visual DBA. See the following topics:

- Opening a Utility Window
- Close Window
- Activate Window

Server-related Tasks

In Network Utility and Visual DBA, Servers represent server classes associated with a particular vnode. You can view a list of the servers that exist for a given vnode using the Servers branch in the Virtual Nodes toolbar/window.

From this branch, you can:

- Access a list of users on a particular server
- Connect to a local server
- Establish a connection at the user level (under a different user name)
- Disconnect at the vnode level
- Disconnect at the user level

The detailed steps for these procedures can be found in the Procedures section of online help for Network Utility and Visual DBA. See the following topics:

- Connecting to a Server
- Impersonating Another User
- Disconnecting from a Server
- Disconnecting a User from a Server

Chapter 5: Using Net

This section contains the following topics:

[Connection to Remote Databases](#) (see page 97)

[Commands and Net](#) (see page 102)

[User Identity on Remote Instance](#) (see page 103)

Connection to Remote Databases

In general, Ingres Net provides users with transparent access to remote databases. Only when the connection is first established must you specify the node on which the database resides and, in some circumstances, the type of server. After you are connected, you can work in the database as if it were local; no further reference to its location is necessary.

Note: If a default remote node is defined on the local instance, you do not have to specify a vnode name when you make a connection to that node. For information about using this feature, see Default Remote Nodes (see page 114).

Database Access Syntax—Connect to Remote Database

The syntax for accessing a remote database through an operating system-level command is:

```
command vnode::dbname[/server_class]
```

where:

command

Is any command used to invoke an Ingres tool, such as cbf, vcbf, sql, qbf or rbf.

vnode::

Is the remote node on which the database is located. The two colons are required.

The remote node can be specified as either of the following:

vnode_name

Is the virtual node name that points to the connection data and authorization data necessary to access a particular remote instance.

@host+

Is a "dynamic vnode" connection string that includes the connection data, user authorization, and attributes that are associated with a remote node. For the format of @host+, see Dynamic Vnode Specification (see page 99).

dbname

Is the name of the database.

server_class

Is the type of server being accessed at the remote site. For a list of server classes, see Server Classes (see page 100).

Example:

This command runs the terminal monitor (sql) and connects using vnode "production" to the customerdb database:

```
sql production::customerdb
```

Dynamic Vnode Specification—Connect to Remote Node

When connecting to a remote node (using the *vnode::dbname* syntax), you can specify a dynamic vnode instead of a vnode name. The dynamic vnode specification includes the connection data, user authorization, and attributes that are associated with a remote node.

Note: A dynamic vnode can be used wherever a vnode is allowed, unless otherwise stated.

A dynamic vnode specification has the following format:

```
@host,protocol,port[;attribute=value{;attribute=value}][[user,password]]
```

@host

Identifies the network name or address of the node on which the remote database is located. The @ character is required because it identifies this specification as a dynamic vnode rather than a vnode name.

protocol

Identifies the network protocol to be used by the local node to connect to the remote node. For a list of protocols and their associated keywords, see Network Protocol Keywords (see page 58).

port

Identifies the listen address of the Ingres instance on the remote node.

attribute=value

(Optional) Is one or more additional connection, encryption, and authentication attributes for the connection. Vnode attributes are described in Configure Vnode Attributes (see page 64).

user

Identifies the user (login) name on the remote system.

password

Is the password for the user on the remote system.

Note: The user and password are optional for a dynamic vnode, but must be enclosed in brackets if used.

Examples of dynamic vnode specification:

This command runs the terminal monitor (sql) and connects to node *hosta* using protocol *tcp_ip* to remote Ingres symbolic port *II*. The login and password are *Johnny* and *secretpwd*. The remote database name is *customerdb*:

```
sql @hosta,tcp_ip,II;[Johnny,secretpwd]::customerdb
```

This command establishes a direct connection by using the `connection_type` attribute:

```
sql @hosta,tcp_ip,II;connection_type=direct[Johnny,secretpwd]::customerdb
```

Server Classes

If you do not specify a server class when connecting to a database, Ingres assumes a default. The default is the value in `default_server_class` on the remote instance (ingres, unless defined otherwise).

Valid Ingres server classes are as follows:

ingres

Indicates DBMS Server

star

Indicates Star Server (Ingres Ingres Star)

db2

Indicates EDBC for DB2

db2udb

Indicates Enterprise Access for DB2 UDB

rdb

Indicates Enterprise Access for Rdb

ims

Indicates EDBC for IMS

rms

Indicates Ingres RMS Access

vsam

Indicates EDBC for VSAM

mssql

Indicates Enterprise Access for MS SQL

oracle

Indicates Enterprise Access for Oracle

informix

Indicates Enterprise Access for Informix

sybase

Indicates Enterprise Access for Sybase

To view or change the default server class value, use the Configure Name Server screen of the Configuration-By-Forms (cbf) utility, or the Parameters Page, Name Server Component in Configuration Manager (vcbf).

The server class for the DBMS Server (default is ingres) and Star Server (default is star) can also be changed. This is typically done to distinguish between multiple DBMS or Star servers that have different sets of parameters, so that users can connect to a specific server using an assigned server class name.

Additional server types are added to this list as additional Enterprise Access or ODBC products are developed. Check the Readme file for the most up-to-date set of products.

Use of the SQL Connect Statement with Net

If you are using the **connect** statement in an application, connect to a database on a remote instance using the following syntax:

```
exec sql connect 'vnode::dbname[/server_class]'
```

Note: The *vnode* can be either a vnode name or a dynamic vnode specification (@host+).

You must use the single quotes around the designation of the vnode and database names (and server class, if applicable). For example, assume that you have an application residing on "napoleon" that wants to open a session with the database "advertisers" on "eugenie." The following statement performs this task (assuming also that "lady" is a valid vnode name for "eugenie"):

```
exec sql connect 'lady::advertisers';
```

Note that a server class is not specified in this statement; therefore the default server class defined on "eugenie" is used.

If the target database is accessed through an Enterprise Access or ODBC products, be sure to include the appropriate keyword for the server class. For example:

```
exec sql connect 'lady::advertisers/db2';
```

If the target database is accessed through Ingres Star, be sure to include the appropriate keyword for the server class. For example:

```
exec sql connect 'lady::advertisers/star';
```

When you are working over Ingres Net, you can use the **-u** flag with a command to imitate another user provided the User ID that you are working under on the remote node has the SECURITY privilege.

Commands and Net

You can run any of the following Ingres commands against a remote database:

abf	imageapp	report
accessdb	ingmenu	sql
compform	isql	sreport
copyapp	netutil	unloaddb
copydb	printform	upgradedb
copyform	qfb	upgradefe
copyrep	query	vifred
dclgen	rbf	vision

Note: The `optimizedb` command works across Ingres Net only if the client and server machines have identical architectures. Do *not* use this command across Ingres Net if the client and server have different architectures.

You cannot run the following Ingres commands against a remote database:

createdb	destroydb	iiimonitor
iinamu	lockstat	logstat
statdump	sysmod	usermod
verifydb		

For additional information on commands, see the *Command Reference Guide*.

User Identity on Remote Instance

A user's identity when working on a remote instance depends upon the type of access authorized.

- When access to a remote instance is authorized using an Installation Password, users retain their local identities (User IDs) when working on the remote instance.
- When access is authorized through a login account on the remote instance's host machine, users take on the identity (User ID) of this account when working on the remote instance.

In either case, the user's privileges and permissions on the remote instance can differ from those on the local instance. For example, a user can have system administrator privileges on the local instance but only very general, low-level privileges and permissions on the remote instance. It is important to make sure that the privileges and permissions assigned to you on the remote instance are adequate for the work that you intend to perform.

User privileges and permissions are set up individually for each instance using Visual DBA or **create user** statement. They apply only to the instance on which they are set up. For more information about this procedure, see the *Security Guide*.

Note: Using Network Utility or Visual DBA, you can access a list of users on Ingres server nodes and establish a connection at the user level under a different user name. For more information, see Impersonating Another User in online help for either of these visual tools.

-u Command Flag—Impersonate User

You can use the -u command flag on a remote instance to impersonate another user provided your user ID on the remote instance has the SECURITY permission. (Typically, a system administrator has this privilege.)

This command flag has the following format:

```
-u user_ID
```

where *user_ID* is the user ID of the user you are impersonating.

Verify Your Identity

When impersonating a user using the `-u` command flag, you may need to verify your identity.

To verify your identity

Use the following command:

```
dbmsinfo ('username')
```

The user ID that you are working under is displayed.

Chapter 6: Maintaining Connectivity

This section contains the following topics:

[Start Communications Server](#) (see page 105)
[Stop Communications Server](#) (see page 106)
[Network Server Control Screen in Netutil](#) (see page 106)
[Stop or Quiesce a Communications Server Using Netutil](#) (see page 108)
[Inbound and Outbound Session Limits](#) (see page 110)
[Logging Levels](#) (see page 111)
[How You Direct Logging Output to a File](#) (see page 112)
[GCF Server Management Using iimonitor](#) (see page 113)
[Default Remote Nodes](#) (see page 114)
[Start Data Access Server \(DAS\)](#) (see page 115)
[Stop Data Access Server \(DAS\)](#) (see page 115)

Start Communications Server

The Communications Server starts automatically when you start your Ingres instance. Sometimes, however, it is necessary to stop the Communications Server.

You can start the local instance's Communications Server using Ingres Visual Manager (IVM). For specific instructions, see IVM online help.

You can also start the Communications Server using the command line utilities.

To start the Communications Server at the command line

1. Log in as the installation owner.
2. Enter the following command at the operating system prompt:

```
ingstart -iigcc
```

The configured number of Communications Servers (set during installation) is started.

Stop Communications Server

You can stop the local instance's Communications Server using Ingres Visual Manager (IVM). For specific instructions, see IVM online help.

You can also stop the Communications Server using the command line utilities.

You can stop the local instance's Communications Servers using Visual Performance Monitor. This "soft" shutdown operation waits for all sessions to end before stopping the server. Close the sessions (or ask users of those sessions to close them) before shutting down a server. For specific instructions, see Visual Performance Monitor online help.

To stop the communications server at the command line

1. Log in as the installation owner.
2. Enter the following command at the operating system prompt:

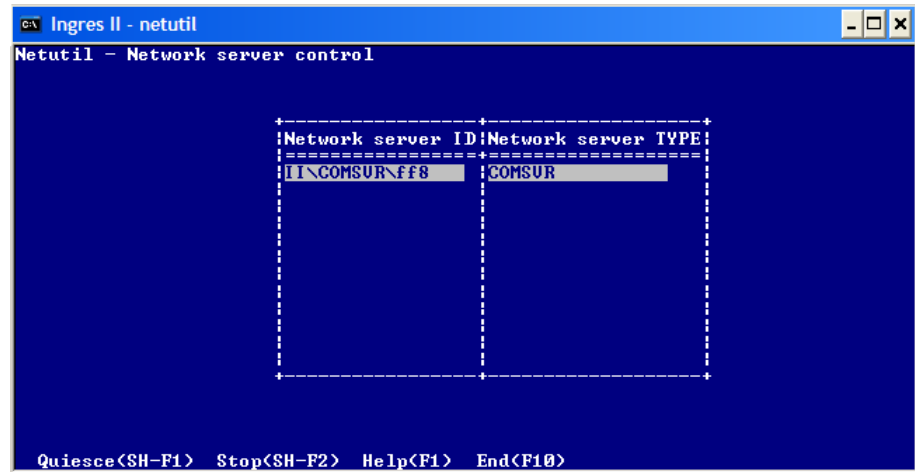
```
ingstop -iigcc
```

The configured number of Communications Servers is stopped.

Note: To increase the configured number of servers, you must reconfigure Ingres Net using Configuration Manager (vcbf) or the Configuration-By-Forms (cbf) utility.

Network Server Control Screen in Netutil

You can stop or quiesce the local instance's Communications Server using the Network Server Control screen in netutil.



The Network Server Control screen contains the following tables:

Network Server ID table

Lists the server IDs of the Communications Servers and the Bridge Servers on the local instance.

Network server TYPE table

Lists server TYPES (COMSVR, BRIDGE).

Note: You cannot obtain information about remote Communications Servers from this screen.

If the local instance has only one Communications Server, the Network Server Control screen menu selections are:

Quiesce

Stops the highlighted Communications Server after all sessions currently in progress terminate

Stop

Stops the highlighted Communications Server immediately, disconnecting any open sessions

Help

Displays help screens

End

Returns you to the netutil startup screen

If the local instance has more than one Communications Server, the menu contains two additional selections:

Quiesce All

Stops all Communications Servers after the sessions currently in progress terminate.

Stop All

Stops all Communications Servers immediately, disconnecting any open sessions

Stop or Quiesce a Communications Server Using Netutil

You can stop or quiesce a Communications Server using the Network Server Control screen in netutil.

To stop a single Communications Server

1. Tab to the Login/password table on the netutil startup screen, and select Control from the function menu.

The Network Server Control screen appears.

2. Highlight the desired entry in the Network Server ID table, and select Stop from the menu.

A pop-up screen appears with the following prompt:

Really stop network server [server ID]?

Yes — Stop

No — Don't Stop

3. Use your arrow keys to highlight No or Yes (Yes is the default), and then choose Select from the menu.

Netutil stops the Communications Server and returns to the Network Server Control screen.

4. Select End.

You are returned to the netutil startup screen.

To stop multiple Communications Servers

1. Select Stop All from the Network Server Control screen menu in netutil.

A pop-up screen appears with the following prompt:

Really stop all network servers?

Yes - Stop

No - Don't Stop

2. Use your arrow keys to highlight No or Yes (Yes is the default), and choose Select from the menu.

Netutil stops all local Communications Servers and returns to the Network Server Control screen.

3. Select End.

You are returned to the netutil startup screen.

To quiesce a single Communications Server

1. Highlight the desired entry in the Network Server ID table in the Network Server Control screen in netutil, and select Quiesce from the function menu.

A pop-up screen appears with the following prompt:

```
Really quiesce network server [server ID]?  
Yes - Quiesce  
No  - Don't quiesce
```

2. Use your arrow keys to highlight No or Yes (Yes is the default), and choose Select from the menu.

Netutil quiesces the Communications Server and returns to the Network Server Control screen.

3. Select End.

You are returned to the netutil startup screen.

To quiesce multiple Communications Servers

1. Select Quiesce All from the Network Server Control screen menu.

A pop-up screen appears with the following prompt:

```
Really quiesce all network servers?  
Yes - Quiesce  
No  - Don't quiesce
```

2. Use your arrow keys to highlight No or Yes (Yes is the default), and choose Select from the menu.

Netutil quiesces all local Communications Servers and returns to the Network Server Control screen.

3. Select End

You are returned to the netutil startup screen.

Note: For instructions on stopping or quiescing Communications Servers using netutil non-interactive mode, see the chapter "Establishing Communications."

Inbound and Outbound Session Limits

A default number of 64 inbound and 64 outbound sessions are configured during the installation process. After the Communications Server has been running, you can change these default limits.

Resetting the maximum number of inbound and outbound sessions does not affect a currently running Communications Server. If you alter these figures after you start a Communications Server, you must stop and restart the server for the new limits to take effect.

The Ingres Net configuration parameters that determine the maximum number of allowed inbound and outbound sessions for a Communications Server are `inbound_limit` and `outbound_limit`.

The maximum values that you can assign to `inbound_limit` and `outbound_limit` are operating system dependent.

UNIX: Bear in mind when setting inbound and outbound session limits that the maximum number of sessions that can be concurrently supported cannot exceed 14 less than the number of file descriptors allocated to each process. The following formula expresses the maximum number of connections that can be supported at any given time:

$$\text{inbound_limit} + \text{outbound_limit} \leq (\text{per_process_open_file_limit} - 14) / 2$$

The number of file descriptors allocated to a process is a UNIX kernel parameter (`NOFILES` on most platforms). ■

Windows and VMS: Ingres uses the `inbound_limit` and `outbound_limit` values when it allocates resources. Consequently, if the sum of the new values is greater than the sum of the current values, you must shut down the instance (instead of only the Communications Server) and restart it so that the system can allocate the appropriate level of resources. If the sum of the new values is equal to or less than the sum of the current values, you can simply stop the Communications Server and restart it after you have reset the values. ■

How You Set Inbound and Outbound Session Limits

The `inbound_limit` and `outbound_limit` parameters determine the maximum number of allowed inbound and outbound sessions for a Communications Server.

You can view or change the values for these parameters using the Parameters page for the selected Net Server in Configuration Manager (vcbf) or the Configure Net Server Definition screen in the Configuration-By-Forms (cbf) utility.

Logging Levels

By default, Ingres logs DBMS error messages, Ingres Net error messages, and Communications Server startup and shutdown messages to the `errlog.log` file.

The following logging level values are available:

0

Logs no error messages (silent)

1

Logs startup messages only

4

(Default) Logs GCC START/STOP status messages, fatal GCC errors that cause the GCC process to stop, connection-specific errors that cause a specific connection to be broken, as well as logging level 1 messages

6

Logs connection setup and termination messages for all connections, as well as logging levels 4 and 1 messages.

How You Change the Logging Level

Logging level is defined by the Ingres Net configuration parameter `log_level`.

To change the value of the `log_level` parameter, use the Parameters Page for the selected Net Server in Configuration Manager (vcbf) or the Configure Net Server Definition screen in the Configuration-By-Forms (cbf) utility.


How You Direct Logging Output to a File

During an Ingres session, each process or program queries the value of `II_GCA_LOG` when it starts up. If this environment variable/logical is set to a file name, the program sends its trace output to the specified file in addition to sending the output to the `errlog.log`. If you want to see the GCC trace output for a Communications Server, set `II_GCA_LOG` and stop and restart the Communications Server.


After you restart the Communications Server, unset `II_GCA_LOG`. You can leave `II_GCA_LOG` set, but you receive trace output for any Ingres process that starts after it was set.

To send the GCC information that Ingres logs in `errlog.log` to another file in addition to the log file, follow this process:

Windows and UNIX:

1. Log in as the installation owner.
2. Set the Ingres environment variable `II_GCA_LOG` to a file name.
3. Stop and restart the Communications Server. 

VMS:

1. Log in as the installation owner.
2. Define the logical `II_GCA_LOG` to a file name. If this is a production instance, define the logical at the system level. If this is a test instance, define the logical at the group level.
3. Stop and restart the Communications Server. 

GCF Server Management Using iimonitor

GCF servers (Name Server, Communications Server, and Data Access Server) can be monitored using the iimonitor utility. The iimonitor utility can be used to:

- Show and remove sessions
- Display server status
- Enable, disable, and register servers
- Turn tracing on and off dynamically
- Remove tickets
- Remove pooled sessions

For more information, see the iimonitor command in the *Command Reference Guide*.

Default Remote Nodes

A system administrator can define a default remote node for the local node. When this parameter is set, users are automatically connected to the default node whenever they request a connection without specifying a vnode name. If users want to access a database on their local node, they must specify the name configured as `local_vnode` on the Parameters Page, Name Server Component in Configuration Manager (vcbf) or the Configure Name Server screen in the Configuration-By-Forms (cbf) utility.

To illustrate, assume that the system administrator has set up the node "eugenie" as the default remote node for users at the node "josephine." The node "eugenie" has the database "advertisers" and "josephine" has the database "employees." Whenever users on "josephine" issue database connection requests that do not specify a vnode name, they are automatically connected to "eugenie" because "eugenie" is the default remote node for "josephine." For example, look at the following statement:

```
isql advertisers
```

If users on "josephine" issue this statement, Ingres Net automatically connects them to the "advertisers" database on "eugenie." If the users on "josephine" want to query a *local* database, they must specify josephine's `local_vnode` name. For example, if the `local_vnode` name for "josephine" is "royal," users on "josephine" issue the following statement to query the local database "employees":

```
isql royal::employees
```

Note: Do not set the default remote node name to point to a vnode that is in fact a loopback to the local instance. If you do so, your local connections loop through Ingres Net until all resources are exhausted and the connection fails.

How You Set Default Remote Nodes

To define a default remote node for the local node, set the configuration parameter `remote_vnode` on the Parameters Page, Name Server Component in Configuration Manager (vcbf) or the Configure Name Server screen in the Configuration-By-Forms (cbf) utility.

Start Data Access Server (DAS)

The DAS (iigcd) starts up automatically when you start up your Ingres instance. Sometimes, however, it is necessary to stop the DAS. In such instances, you can use the following procedures to restart the server.

You can start the DAS using one of these methods:

- Using Ingres Visual Manager (IVM) to start the local instance's DAS. For specific instructions, see IVM online help.
- Using a command line utility. This procedure uses the configuration values set during installation to start the DAS.

To start the DAS using the command line utility

1. Log in as the installation owner.
2. Enter the following command at the operating system prompt:

```
ingstart -iigcd
```

The DAS is started using the configuration values set during installation.

Stop Data Access Server (DAS)

You can stop the DAS using one of these methods:

- Using Ingres Visual Manager (IVM) to stop the local instance's DAS. For specific instructions, see IVM online help.
- Using a command line utility.

To stop the DAS at the command line with ingstop

1. Log in as the installation owner.
2. Enter the following command at the operating system prompt:

```
ingstop -iigcd
```

To stop the DAS at the command line with iigcdstop

1. Log in as the installation owner.
2. Enter the following command at the operating system prompt:

```
iigcdstop addr
```

where *addr* is the server address, which can be obtained using the iinamu utility and issuing the request **show dasvr**.

Chapter 7: Troubleshooting Connectivity

This section contains the following topics:

[How Connection Between the Application and DBMS Server Is Established](#)

(see page 117)

[Where Ingres Net Information Is Stored](#) (see page 118)

[Causes of Connectivity Problems](#) (see page 122)

[How You Diagnose Connectivity Problems](#) (see page 122)

How Connection Between the Application and DBMS Server Is Established

For queries to be processed, applications must establish a connection to the DBMS Server through Ingres Net. When an application issues a query, the query is sent to the DBMS Server for execution. The server executes the query and returns data to the application.

For a client to connect to a server through Ingres Net, many internal connections must be made.

When the application, the DBMS Server, and the database reside on the same instance, establishing a connection is a short process:

1. The application program connects to the local Name Server (iigcn) and requests the listen address of the local DBMS Server process.
2. The Name Server returns this information to the application, which thereafter communicates directly with the DBMS Server through that address.

When the application and DBMS Server are on separate instances, the process has more steps:

1. The application finds the local Name Server (iigcn) listen address and talks to the local Name Server to request remote access.
2. The *local* Name Server passes the listen address of the *local* Communications Server (iigcc) and the listen address of the *remote* Communications Server (iigcc) back to the application. (The *local* Name Server (iigcn) stored the *remote* Communications server's listen address when you ran netutil on the local instance.)
3. The application connects to the *local* Communications Server, passing it the *remote* Communications Server's listen address as part of the remote access request.
4. The *local* Communications Server connects to the *remote* Communications Server and requests a connection to a DBMS Server on that remote instance.
5. The Communications Server on the remote instance finds the listen address of the Name Server on the remote instance. The Communications Server requests connection information from the Name Server, passing the name of the database for which a connection is requested.
6. The remote Name Server returns the listen address of a DBMS Server that is capable of servicing a request for connection to the target database.
7. The remote Communications Server connects to the DBMS Server on the remote instance.

When these steps are completed, a "virtual connection" has been established between the application and the DBMS Server.

Where Ingres Net Information Is Stored

Ingres Net configuration values are stored in *either* of the following:

config.dat file

Stores Ingres Net configuration parameters, which can be changed using Configuration-By-Forms or Configuration Manager.

Name Server database

Stores remote access information, which can be entered using the netutil utility.

config.dat—Store Net Configuration Values

The following Ingres Net configuration parameters, stored in config.dat, can be viewed and changed using the Configuration Manager (vcbf) or Configuration-By-Forms (cbf) utility. The default values are assigned during installation.

Note: The Net Server is the Communications Server.

Parameter	Default	vcbf page	cbf screen
inbound_limit	64 inbound sessions	Parameters Page, Net Server Component	Configure Net Server Definition screen
outbound_limit	64 outbound sessions		
log_level	Level 4	Parameters Page, Net Server Component	Configure Net Server Definition screen
Protocol	Any protocol present at installation is indicated as active	Protocols Page, Net Server Component	Configure Net Server Protocols screen
Listen Address	A GCC listen address is assigned for any protocol present at installation. (The format depends on the protocol.)	Protocols Page, Net Server Component	Configure Net Server Protocols screen
default_server_class	INGRES	Parameters Page, Name Server Component	Configure Name Server screen
remote_vnode	No default value	Parameters Page, Name Server Component	Configure Name Server screen
local_vnode	Name of host machine	Parameters Page, Name Server Component	Configure Name Server screen

Name Server Database—Store Remote Access Information

Ingres maintains an internal database called the Name Server database, which is used by the Name Server (iigcn). The Name Server database contains the information required for remote access, such as remote node names, listen addresses, login accounts and passwords, virtual node names, and local and remote Installation Passwords. This information can be entered in the Name Server database using the netutil utility.

In a client/server configuration, this database contains one file called iiname.all and one or more "nodename" files for each client node and for the local node. For example:

```
iiname.all
IIINGRES_nodename1
IICOMSVR_nodename1
IISTAR_nodename1
IINODE_nodename1
IILogin_nodename1
IIUSVR_nodename1
IIDB2UDB_nodename1
IIORACLE_nodename1
IIRDB_nodename1
IIRMS_nodename1
IILTICKET_nodename1
IIRTICKET_nodename1
IIINGRES_nodename2
IICOMSVR_nodename2
IISTAR_nodename2
IINODE_nodename2
IILogin_nodename2
IIUSVR_nodename2
IIDB2UDB_nodename2
IIORACLE_nodename2
IIRDB_nodename2
IIRMS_nodename2
IILTICKET_nodename2
IIRTICKET_nodename2
```

A unique set of files is created for *each* node registered as an Ingres Net client.

In the cluster environment, the Name Server database has only one file of each type. For example:

```
iiname.all
INGRES
COMSVR
STAR
NODE
LOGIN
IUSVR
DB2 UDB
ORACLE
RDB
```


RMS
LTICKET
RTICKET

All the nodes in an Ingres Cluster Solution instance share the same files.

The files in the Name Server Database are as follows:

iiname.all

Contains a list of all of the types of servers that the instance is expected to manage. The possibilities are DBMS servers, Communications servers, and Star servers.

IIINGRES_nodename

Contains the GCA listen addresses of all the DBMS servers registered with the Name Server (iigcn) on the specified node.

The file is written when the DBMS Server starts and is cached when the node's (identified by *nodename*) Name Server starts.

IICOMSVR_nodename

Contains the GCA listen address of the Communications Server (iigcc) on the specified node (identified by *nodename*). The file is written when the local Communications Server starts and is cached when the local Name Server starts.

IISTAR_nodename

Contains the GCA listen address of the Star Server on the specified node (identified by *nodename*). The file is written when the Star Server starts and is cached when the *nodename*'s Name Server starts.

IINODE_nodename

Contains the connection data entries established for the specified node (identified by *nodename*) by running netutil, Network Utility (ingnet) or Visual DBA from that node. The file is written whenever you select the "create" option to add a connection data entry for an existing vnode. The file is cached when the *nodename*'s Name Server starts.

IILOGIN_nodename

Contains the remote user authorizations set up at the specified node (identified by *nodename*) by running netutil, Network Utility (ingnet) or Visual DBA from that node. The file is written whenever you add a remote user authorization. It is cached when the *nodename*'s Name Server starts.

Causes of Connectivity Problems

You can trace most network connectivity problems to one of the following causes:

- The network or the network protocol is not properly installed
- The Name Server (iigcn) or Communications Server (iigcc) process is not running
- Vnode entries are incorrect
- There are port connection problems
- There are problems with the Ingres Net files

How You Diagnose Connectivity Problems

Often, the most difficult task in problem solving is determining the origin of the problem. Sometimes the circumstances of the problem point to a particular cause. For example, if only one user on a node is experiencing an Ingres Net connection problem, that user's vnode entries are probably incorrect. Some problems, however, leave more ambiguous clues.

To determine the origin of a problem, follow these steps:

1. Examine the Ingres error file, `errlog.log`. Ingres logs DBMS error messages, Ingres Net error messages, and Communications Server startup and shutdown messages to this log.

The default location for the `errlog.log` file is:

Windows: `%II_SYSTEM%\ingres\files\errlog.log`

UNIX: `$II_SYSTEM/ingres/files/errlog.log`

VMS: `II_SYSTEM:[INGRES.FILES]ERRLOG.LOG`

Often the error message provides sufficient information to determine the origin of the problem.

2. Examine any of the optional logs or tracing facilities, if set up in your installation.
3. Perform the General Ingres Net Installation Check described next if examining the error messages does not pinpoint the origin of the problem.

If you are having password or other security or permission problems with Ingres Net, use the procedure in Security and Permission Errors (UNIX) to resolve them.

General Net Installation Check

The Ingres Net installation check is a diagnostic procedure that checks your installation to determine the following:

- Whether the problem is Ingres Net-related
- Whether the iigcn and iigcc processes are running
- Whether the network protocol software is working

How You Check Net Installation on Windows

If you are experiencing a problem and cannot determine its source, use this diagnostic procedure as a starting point:

1. Verify that your network protocol is functioning.
 - a. Use the ping command to connect between machines to verify that basic TCP/IP networking is working.
 - b. On both the client and the server, verify that TCP/IP is properly installed and configured. Do this by attempting to connect to the default localhost (or loopback) listen address from each machine. Type one of the following commands to loop back to your own machine using the network:
 - ping localhost
 - ping 127.0.0.1
 - ping ::1 (if TCP/IP version 6 enabled)

If either “a” or “b” fails, the problem is with the underlying network. Contact your network administrator.

2. If the *remote* node is a UNIX machine, verify that you can connect to the target database on the remote node when you are logged in directly to the remote node.
 - a. Use telnet to log in to the remote node from your local node.
 - b. Enter a command that connects you to the database. For example:

```
sql database_name
```

If you cannot connect to the database even when logged in directly to the remote node, the problem is something other than Ingres Net.

If you can connect this way, but cannot connect when you are Using Net to log into the remote node and connect (through the syntax `sql vnode_name::database_name`), it is an Ingres Net problem. Proceed with Step 3.

3. Check that the `iigcc` process is registered with the Name Server:

- a. Enter **iinamu** at the operating system prompt.
- b. Type **show comsvr**.

If you receive no output from the `show comsvr` command, this means that no Communications Server is registered with the Name Server.

4. Check that configuration parameters such as `local_vnode` and the Communications Server listen address are correctly set. These parameters can be viewed and, if necessary, changed using the Configuration Manager (`vcbf`) or Configuration-By-Forms (`cbf`) utility.
5. Check the `II_GCNxx_PORT` environment variable where `xx` is the installation ID. It must be visible only when using the `ingprenv` utility. It must never be visible when using the UNIX commands `env` or `printenv`. `II_GCNxx_PORT` must not be part of your local operating system environment. If it is set in the local environment, it overrides their proper settings in the Ingres symbol table.

You must be the installation owner (who by default has Ingres user privileges) to take corrective action.

How You Check Net Installation on Linux and UNIX

If you are experiencing a problem and cannot determine its source, use this diagnostic procedure as a starting point:

1. Verify that your network protocol is functioning.
 - a. Use the `rlogin` and/or `telnet` commands to connect between machines to verify that basic TCP/IP networking is working.
 - b. On both the client and the server, verify that TCP/IP is properly installed and configured. Do this by attempting to connect to the default localhost (or loopback) listen address from each machine. Type one of the following commands to loop back to your own machine using the network:
 - `telnet localhost`
 - `telnet 127.0.0.1`
 - `ping ::1` (if TCP/IP version 6 enabled)

The login messages that follow the command reveal whether you are connected to your own machine (the name of the machine can be embedded in the messages). If they do not, you can log in and issue the `hostname` command to display the name of the machine to which you are connected.

If either "a" or "b" fails, the problem is with the underlying network. Contact your network administrator.

2. Verify that you can connect to the target database on the remote node when you are logged in directly to the remote node.
 - a. Use telnet, rlogin, or your site's network server bridge software to log in to the remote node from your local node.
 - b. Enter a command that connects you to the database. For example:

```
$ sql database_name
```

If you cannot connect to the database even when logged in directly to the remote node, the problem is something other than Ingres Net.

If you can connect this way, but cannot connect when you are Using Net to log into the remote node and connect (through the syntax `sql vnode_name::database_name`), it is an Ingres Net problem. Proceed with Step 3.

3. To verify that the Communications Server (iigcc) and Name Server (iigcn) processes are running on your local node, use the `ps` command. This command shows the status of all currently running processes. Also check the processes on the remote node.
4. Check that the iigcc process is registered with the Name Server:
 - a. Enter **iinamu** at the operating system prompt.
 - b. Type **show comsvr**.

If you receive no output from the `show comsvr` command, this means that no Communications Server is registered with the Name Server.

5. Check that configuration parameters such as `local_vnode` and the Communications Server listen address are correctly set. These parameters can be viewed and, if necessary, changed using the Configuration Manager (`vcbf`) or Configuration-By-Forms (`cbf`) utility.
6. Check the `II_GC�xx_PORT` environment variable where `xx` is the installation ID. It must only be visible using the `ingprenv` utility. It must never be visible using the UNIX commands `env` or `printenv`. `II_GC�xx_PORT` must not be part of your local UNIX shell environment. If it is set in the local environment, it overrides their proper settings in the Ingres symbol table.

You must be the installation owner (who by default has Ingres user privileges) to take corrective action.

How You Check Installation on VMS

If you are experiencing a problem and cannot determine its source, use this diagnostic procedure as a starting point:

1. Verify that your network protocol is functioning.

You must be able to connect to another node on the network. If you cannot, your network software is not working. Contact your network administrator to correct the networking problem.

2. Verify that you can connect to the database on the remote node when you are logged in directly to the remote node.

- a. Log directly into the remote node.

- b. Enter a command that connects you to the database. For example:

```
$ sql database_name
```

If you cannot connect when logged in directly to the remote node, the problem is something other than Ingres Net.

If you can connect this way, but cannot connect when you are Using Net to log into the remote node and make the connection (through the syntax `sql vnode_name::database_name` for example), it is an Ingres Net problem. Proceed with Step 3.

3. To verify that the `iigcc` and `iigcn` processes are running properly on your local node:

Check the error log (`errlog.log`) for any error messages indicating a startup failure on the part of either `iigcc` or `iigcn`. Check the `iigcc` process on the remote node also.

Alternatively, at the operating system prompt, type **show system**.

This command displays a list of the processes currently active. Check for the following processes:

```
II_GCC
II_GCN
II_DBMS
II_IUSV (dmfrpc)
DMFACP
```

4. Check that the `iigcc` process is registered with the Name Server:

- a. Enter **iinamu** at the operating system prompt.

- b. Type **show comsvr**.

If you receive no output from the `show comsvr` command, this means that there is no Communications Server registered with the Name Server.

5. Check that configuration parameters such as `local_vnode` and the Communications Server listen address are correctly set. These parameters can be viewed and, if necessary, changed using the Configuration Manager (`vcbf`) or Configuration-By-Forms (`cbf`) utility.

Connection Errors

Connection errors can occur for a variety of reasons. For example, a failure in any of the internal connections described in *How Connection Between the Application and DBMS Server Is Established* (see page 117) results in a connection error.

How connection errors are reported depends on where the failure occurs. If failure occurs:

- At the local instance, errors are reported directly to the user interface program or the application.
- Between the local and remote instances, for example, when attempting to connect from the local Communications Server to the remote Communications Server, errors go to the local `errlog.log` file as well as to the application.
- At the server installation, errors are reported to both the local and remote `errlog.log` file and to the application.

Local Connection Errors

Each Communications Server has a GCA and GCC listen address. The GCA listen address is the server's connection to local processes and is known only to the local Name Server (`iigcn`). The GCC listen address is the server's connection to the network and is known to all nodes in the network. These listen addresses are stored separately.

The GCA address is stored at runtime in an `IICOMSVR` file in the Name Server database. You can obtain this address using the `iinamu` utility. Do not attempt to view these files directly. For more information about `iinamu`, see the *Command Reference Guide*.

The GCC address is stored in the `config.dat` file when the installation is configured. To view or change the GCC address, use the Net Server Protocol Configuration screen in the Configuration-By-Forms (`cbf`) utility, or the Net Server Protocols page in Configuration Manager (`vcbf`).

When the Communications Server starts up, it must be able to obtain the use of the network (GCC) listen address. If the Communications Server cannot use this listen address because the operating system has allocated the address to another process, the Communications Server cannot listen on that protocol. This problem can occasionally arise if the installation is not started from the machine boot file.

How You Resolve Remote Connection Errors

When you cannot establish a remote connection, use this procedure to diagnose the problem:

1. Check the `errlog.log` for error messages.
2. If that does not identify the problem, follow the procedure for your protocol in the General Net Installation Check section of this chapter. This procedure tells you if your network and protocol are working properly and if the Name Server (`iigcn`) and Communications Server (`iigcc`) processes are working properly.
3. If the problem remains unidentified after you have looked at the error messages and performed the installation check, use the following procedure to verify that your `netutil` connection data entry contains the correct listen address.
 - a. From the local instance, check the connection data for the remote instance. Note the listen address specified in the `netutil` Connection Data table.
 - b. From the remote instance, check to see which GCC listen address the remote instance's Communications Server is using. You can find this information in the Net Server Protocol Configuration screen in the Configuration-By-Forms (`cbf`) utility, or the Net Server Protocols page in Configuration Manager (`vcbf`).
 - c. If the listen address found Step a does not match the listen address found in Step b, correct the problem by re-registering the remote instance's GCC listen address. Do this from the local instance, using `netutil` to edit the incorrect entry. For procedures for adding, deleting, and changing a `vnode` definition, see the chapter "Establishing Communications."

How You Resolve Net Registration Problems

To resolve net registration problems, use this procedure:

1. Use the General Net Installation Check to verify that your installation is properly installed and working.
2. Check that your connection data entries and remote user authorizations are correct.

The utilities used to set up connection data and remote user authorizations (Network Utility, Visual DBA, or netutil) can test a connection, but you must explicitly choose the Test operation from a menu. If you did not test the connection after entering, adding, or editing connection data or remote user authorizations, the information can be incorrect.

3. Check that the required connection data and remote user authorizations for the target installation exist. If they are present, check the following:
 - That all vnode names and user (account) names are spelled correctly
 - That the proper network protocol has been specified
 - That listen addresses and network addresses are correct

Note: End users check their private entries. A user with the SECURITY privilege (typically a system administrator) checks another user's private entries by using the -u command flag in netutil to impersonate that user. Users can also perform this task using Network Utility and Visual DBA.

Any user can check global entries, however if corrections are required, they must be made by a user with the GCA privilege NET_ADMIN (typically the system administrator).

4. If you are experiencing problems connecting to a distributed database, make sure that the connection data and remote user authorizations required by Ingres Star have been entered on the Star Server installation. For more information, see the *Ingres Star User Guide*.

Security and Permission Errors

Ingres Net encrypts the password entered in netutil and compares it with the encrypted password in "/etc/passwd" (or your machine's similar password file). If the two do not match, an error is returned.

How You Resolve Ingres Security Problems (UNIX)

If you are having password or other security/permission problems in Ingres Net, use the following procedure:

1. Verify that you can log in to the remote machine directly. If you cannot, you do not have the right password.
2. Using netutil, re-enter the remote user authorization.
3. If you are running NIS ("yellow pages"), the account's correct password will be in the yellow pages password file (/etc/yppasswd) rather than in /etc/passwd. Add the following string to the end of /etc/passwd file to tell Ingres Net to look in /etc/yppaswd for the encrypted password:

```
+::0:0:::
```

4. If you have additional security such as C2 security enabled on the target machine, you must verify that the ingvaldpw executable exists in \$II_SYSTEM/ingres/bin by typing:

```
$ ls -l $II_SYSTEM/ingres/bin/ingvaldpw
```

This executable is required to make the password in the secure area readable by Ingres.

Note: Not all Ingres UNIX releases use ingvaldpw to enforce C2 security. If the ingvaldpw executable is required for your release, it will be documented in the Readme file for your platform.

5. If the ingvaldpw executable exists:
 - a. Verify that it is owned by root. If not, log in as root and issue the command:

```
$ chown root ingvaldpw
```
 - b. Verify that it has the "set uid" bit set. If not, issue the command:

```
$ chmod 4711 ingvaldpw
```
 - c. Verify that the Ingres variable II_SHADOW_PWD is set to the full path to the ingvaldpw executable. Type:

```
$ ingprenv | grep II_SHADOW_PWD
```

The ingprenv utility displays the II_SHADOW_PWD variable.

6. If the ingvaldpw executable is not installed, create it using the mkvaldpw script. For details, see Create Password Validation Program (UNIX) (see page 46).

Chapter 8: Exploring Bridge

This section contains the following topics:

[Ingres Bridge](#) (see page 131)
[How Bridge Is Installed](#) (see page 134)
[How Bridge Is Started](#) (see page 134)
[How the Client Is Set Up](#) (see page 136)
[Bridge Server Monitoring](#) (see page 137)
[Stop the Bridge Server](#) (see page 138)
[How a Connection Is Established Through Bridge](#) (see page 138)
[Bridge Troubleshooting](#) (see page 139)
[Sample Bridge Server Configuration](#) (see page 140)

Ingres Bridge

Ingres Bridge is a component of Ingres that enables a client application running on one type of local area network to access an Ingres server running on a different type of network. The client and server do not have to communicate over the same network protocol (such as TCP/IP, SNA LU62); Ingres Bridge “bridges” a client using one network protocol to a server using another.

For example, a PC on a TCP/IP network communicates through Ingres Bridge to an EDBC server (such as DB2, IMS, or Datacom/DB) on an SNA network.

Ingres Bridge does not provide any security checking but simply passes the messages through. Security is handled on the server in the usual way.

How the Bridge Server Works

Ingres Bridge consists of the Bridge Server.

The Bridge Server (iigcb) process connects a client application on one type of network to a server on a different type of network. Modeled on the transport layer of the Ingres Net architecture, the Bridge Server does the following:

- Listens for and accepts incoming connection requests and establishes corresponding connections to a local or remote Communications Server
- Allows bi-directional data transfer over the established connections
- Terminates the connections in an orderly way

Tools for Configuring Bridge

You configure the Bridge Server using one of these utilities (based on your environment):

- Netutil
- Visual DBA
- Ingres Visual Manager

Installation Configurations That Require Bridge

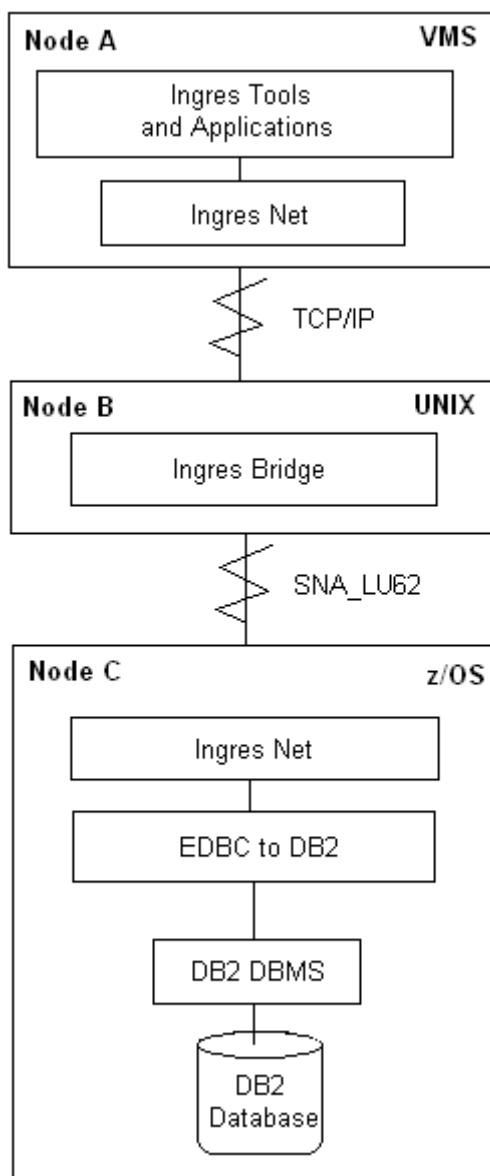
Ingres Bridge is required in any installation configuration where the client and server processes do not reside on the same machine *and* the client machine is on one type of local area network and the server machine is on another type of network.

Ingres Bridge runs on an intermediate platform between the client and the server; the intermediate platform must support both the client and the server network protocols. Ingres Bridge runs as a stand-alone installation or as a part of an Ingres client or server installation.

Ingres Star provides a similar network bridging capability. Ingres Star is required when the user views different physical databases as a single logical database. Ingres Bridge must be used when this is not the case, and the user wants to connect a client and server that run on different network protocols. Ingres Bridge has a fairly small “footprint” and has little impact on response time.

Sample Installation Configuration Using Bridge

The following figure shows a sample installation configuration that uses Ingres Net, Ingres Bridge, and EDBC to DB2. Ingres Bridge runs on a separate installation on an intermediate platform.



Node A is an Ingres for VMS installation. Node B is an Ingres Bridge installation in a UNIX environment using TCP/IP. Node C is an EDBC to DB2 installation in a z/OS environment using SNA_LU62. Node A and Node C are not directly connected to each other.

Ingres Net is present on Node A and Node C. Users on Node A can access DB2 data on Node C as if the DB2 tables were Ingres tables stored on Node A.

How Bridge Is Installed

Ingres Bridge is installed as a component of Ingres. It uses the same installation procedure as Ingres and Ingres tools.

The component appears in the install utility as Ingres Protocol Bridge.

Note: Ingres Bridge is installed as the only component in an installation or with other components such as Ingres Net.

How Bridge Is Started

Ingres Bridge is started by reading configuration parameter values from one of the following:

- The config.dat file

This method gives you have the flexibility of routing the client connections dynamically, and allows multiple routes.

To use values from config.dat, you can start Ingres Bridge using the `ingstart` command or Visual Manager (if available in your environment).

- The `iigcb` command line options

This method requires you to stop and start Ingres Bridge if you want to route the client connections to a different installation, and allows only a single "from-to" route.

config.dat File—Store Bridge Configuration

After the installation and setup phases of Ingres Bridge, default configuration entries are defined in the config.dat file in the Ingres Bridge installation. You can change some of the configuration parameters values by using Configuration-By-Forms (cbf) or Configuration Manager (vcbf). These values are then stored in the config.dat file.

Here is an example configuration entry in config.dat:

```
ii.<hostname>.gcb.*.tcp_ip.port.<vnode>:<listen address>
```

This entry means that Ingres Bridge accepts the incoming client connections from TCP/IP on the port specified by the listen address and route them to the DBMS Server installation defined by the vnode. The vnode name matches a vnode name defined for the DBMS Server installation.

The vnode name must be set in config.dat before starting the server. Only the connection information can be changed for the vnode name, which enables you to change the routing information without stopping and starting Ingres Bridge.

Login/password (remote authorization) data for the vnode is not required because the login data is obtained from the connecting client; only the connection data for the server is required.

ingstart Command—Start the Bridge Server

The ingstart command starts the Bridge Server using the values in config.dat.

If Ingres Bridge is installed with other components such as Ingres Net, or has been configured using the Configuration-By-Forms utility, use the following ingstart command to start the Bridge Server:

```
ingstart -iigcb
```

Or start the Name Server first and the Bridge Server next, using the following :

```
ingstart -iigcn
```

```
ingstart -iigcb
```

iigcb Command—Start the Bridge Server

The Bridge Server process (iigcb) can be executed from the system prompt.

This command has the following format:

```
iigcb -from prot -to dest_prot hostname listen_addr
```

prot

Is the local protocol (for example, tcp_ip).

dest_prot

Is the destination protocol (for example, SNA LU62).

hostname

Is the network name or address where the target DBMS Server and Communications Server are located (format dependent on protocol).

listen_addr

Is the unique identifier for the Communications Server that is used for Ingres Net connections with the destination protocol.

For example, the following command starts the Bridge Server process:

```
iigcb -from tcp_ip -to sna_lu62 hostname listen_addr
```

The following lines are displayed in the errlog.log file:

```
Network open complete for protocol TCP_IP, port <xx>  
Network open complete for protocol SNA_LU62, port <xx>  
Protocol Bridge normal startup: rev. level 1.1/02
```

Ingres Bridge is now ready for clients to make connections to it on the TCP/IP port specified by the listen address in the following line in the config.dat file in the Ingres Bridge installation:

```
ii.hostname.gcb.*.tcp_ip.port: listen address
```

How the Client Is Set Up

To enable the client machine to access remote servers through Ingres Bridge, you must first create a vnode entry for the host machine on which Ingres Bridge is running.

vnode Definition—Enable Client Access to Remote Servers Through Bridge

The following information defines a vnode. You enter this information using any of the Net Management tools.

Note: The Network Utility (if supported on your platform) is the preferred means of creating vnodes in Ingres.

Virtual Node

Defines the Ingres Bridge node.

Remote Node

Identifies the network name or address of the machine on which the Bridge Server is running.

Protocol

Specifies the Ingres keyword for the protocol used by the local client node to connect to the remote node. For details, see Network Protocol Keywords (see page 58).

Listen Address

Is the listen address of the Bridge Server. This address varies by protocol. For more information, see the appropriate appendix in this guide.

Username

Is the login ID for the host machine on which the target DBMS Server is running.

Password

Is the password associated with the login ID for the host machine on which the target DBMS Server is running.

Bridge Server Monitoring

To determine if the Bridge Server is running, use either of the following:

- Ingres Visual Manager
- The `linamu` utility's **show bridge** command

Stop the Bridge Server

You can use either the `ingstop` command or Ingres Visual Manager to stop the Bridge Server.

To stop the Bridge Server using `ingstop`

Issue the following command at the operating system prompt:

```
ingstop -iigcb
```

How a Connection Is Established Through Bridge

When an application on one type of local area network attempts to establish a connection to a server on a different type of network, the following sequence of events establishes the connection:

- The application gets the local Name Server (`iigcn`) listen address and connects to the local Name Server to request remote access.
- The local Name Server passes the listen address of the local Communications Server (`iigcc`) and the listen address of the remote Bridge Server (`iigcb`) back to the application. (The local Name Server (`iigcn`) stored the remote Bridge Server's listen address when you defined a vnode for the remote node on which the Bridge Server is running.)
- The application connects to the local Communication Server, passing it the remote Bridge Server's listen address as part of the remote access request.
- The local Communications Server connects to the remote Bridge Server. The remote Bridge Server gets the connection data entries from the Name Server on that instance and re-directs the connection to the Communications Server (`iigcc`) on the target database's network using the connection data that it received from the Name Server.
- The Communications Server on the target database's network (a different network than that of the requesting application) finds the listen address of the Name Server on that network's installation. The Communications Server requests connection information from the Name Server by passing the name of the database for which the connection is requested.
- The Name Server returns the listen address of a DBMS Server on that instance that is capable of servicing a request for connection to the target database.
- The Communications Server (`iigcc`) connects to the DBMS Server on the remote instance.

When these steps are completed, a virtual connection has been established between the application and the DBMS Server through the Bridge Server.

Bridge Troubleshooting

Most problems with Ingres Bridge are related to one of the following situations:

- Network or protocol not properly installed
- The Name Server (iigcn), Communications Server (iigcc), or Bridge Server (iigcb) process not running
- Incorrect netutil entries
- Port connection problems

To determine the origin of a problem, begin by examining the Ingres error file, `errlog.log`. The Bridge Server's startup and shutdown messages and Ingres Bridge error messages are logged to this file. The error log is maintained in the following file:

Windows:

`%II_SYSTEM%\INGRES\FILES\ERRLOG.LOG`

UNIX:

`$II_SYSTEM/ingres/files/errlog.log`

VMS:

`II_SYSTEM: [INGRES.FILES]ERRLOG.LOG`

For additional information on problems related to the Bridge Server process, see the chapter "Troubleshooting Connectivity."

Sample Bridge Server Configuration

The following is a sample Bridge Server setup for a client on Windows to an EDBC for DB2 server on z/OS by means of Ingres Bridge on Solaris. The client supports TCP/IP and the DB2 server supports SNA LU62. Ingres Bridge supports both network protocols.

The following examples show pertinent excerpts from the files.

Client on Windows—This connection between the client and Ingres Bridge is supported by TCP/IP. The following excerpt is for the client:

VNODE Definition:

Virtual Node	= db2gw
Remote Node	= abc
Protocol	= tcp_ip
Listen Address	= CC7 (matches Bridge listen address below)
Username	= johnm (userid in DB2 Gateway)
Password	= xxxxxx

User invokes terminal monitor:

```
SQL db2gw::db23/db2
```

Bridge on Solaris—This connection between Ingres Bridge and the EDBC for DB2 server is supported by SNA LU62. The following excerpt is for Ingres Bridge:

```

hostname = abc
Ingres Variables:
  II_INSTALLATION = CC
config.dat file:
  ii.abc.gcb.*.inbound_limit:      50(max concurrent sessions)
  ii.abc.gcb.*.tcp_ip.port:        CC
  ii.abc.gcb.*.tcp_ip.port.bvdb2gw CC7 (Bridge listen address)
    ("bvdb2gw" is vnode for DB2 Gateway in netutil below)
  ii.abc.gcb.*.tcp_ip.status:      ON
  ii.abc.gcb.*.sna_lu62.poll:      4000
  ii.abc.gcb.*.sna_lu62.port:      abcgw0.sunlu62
    ("abcgw0" is gateway name in /etc/appcs below,
     "sunlu62" can be anything in this case)
  ii.abc.gcb.*.sna_lu62.status:    ON
netutil entry:
  Virtual Node      = bvdb2gw
  Net Address       = s2 (matches unique_session_name below)
  Protocol          = sna_lu62
  Listen Address    = sunlu62 (anything OK here)

/etc/appcs file: (Sun SNA server config file)
  abcgw0 abc:abcgw0
Sun SNA network config file:
:DEFINE_PARTNER_LU
fql_plu_name  = A04IS2G2 (VTAM applid for DB2 Gateway)
u_plu_name    = A04IS2G2 (VTAM applid for DB2 Gateway)
DEFINE MODE
mode_name     = INGLU62
unique_session_name = s2
System Administrator starts the Name Server and Bridge Server
  ingstart -iigcn
  ingstart -iigcb

```

Server on z/OS—The following excerpt is for the server:

```

VTAM Config:
  Applid for DB2 Gateway  = A04IS2G2
  Acbname for DB2 Gateway = IIS2GWS2

DB2 Gateway IIPSERV file:
  IIPSERV TYPE=SNA_LU62,
  INSTALL=S2,
  ACB=IIS2GWS2,
  LOGMODE=INGLU62,

DB2 Gateway IIPARM file:
  II_PROTOCOL_SNA_LU62    = YES

```


Chapter 9: Configuring the Data Access Server

This section contains the following topics:

[Data Access Server](#) (see page 143)

[Data Access Server Parameters—Configure DAS](#) (see page 144)

[How You Enable Data Access Server Tracing](#) (see page 146)

Data Access Server

The Data Access Server (DAS) process (iigcd) is a component of the General Communications Architecture (GCA) and runs as part of a standard Ingres instance.

The server translates JDBC or .NET Data Provider requests from the Ingres JDBC Driver or the .NET Data Provider into Ingres internal format and forwards the request to the appropriate DBMS server. The DAS supports the same network protocols and port designations as the Communications Server.

Through the DAS, a JDBC or .NET Data Provider client has full access to Ingres, Enterprise Access, and EDBC databases. Using Net, the DAS can also provide JDBC or .NET Data Provider clients with access to these databases on remote machines.

Data Access Server Parameters—Configure DAS

To configure the DAS, use the Data Access Server Parameters page in Configuration-By-Forms (cbf) or Configuration Manager (vcbf).

The DAS has the following configurable parameters:

client_max

Defines the maximum number of concurrent client connections permitted.
Set to -1 for no limit.

client_timeout

Defines the time, in minutes, to wait for client requests. If the time expires with no request from the client, the client and DBMS Server connections are aborted.
Set to 0 for no timeout.

connect_pool_expire

Defines the time, in minutes, for which a DBMS Server connection remains in the connection pool. The connection is aborted if a pooled connection is not used in this amount of time.
Set to 0 for no expiration.

connect_pool_size

Defines the maximum number of DBMS Server connections held in the connection pool.
Set to -1 for no limit.

connect_pool_status

Specifies the operational mode of the connection pool. Modes are:

on

Enables pooling unless explicitly disabled by the client. The DAS saves and reuses DBMS Server connections when connection pooling is enabled.

off

Disables pooling.

optional

Enables pooling but only when requested by the client.

<protocol>.port

Identifies the listen address for the network protocol port. This can be a numeric port identifier or an Ingres symbolic port identifier such as II7. This port must not be used by any other network server on the platform.

The symbolic port ID has the following syntax:

`XY{n}{+}`

where

- X is an alphabetic character
- Y is an alphanumeric character
- n is a subport in the range 0 to 15. A leading 0 is permitted in the subport (for example: II09 is interpreted as II9).
- + indicates that a port ID can be rolled up.

If the DAS startup count is specified as greater than 1, CBF automatically adds a + designator to the port identifier. When listen attempts are made on such a port identifier, the address is incremented (rolled up) and a listen attempt is made on the next address. This allows for connection attempts to multiple Data Access Servers.

<protocol>.status

Specifies the status of the network protocol. Options are:

on

Indicates that the DAS must listen to (accept) connection requests on the protocol.

off

Disables the protocol.

How You Enable Data Access Server Tracing

As a GCA-based server, the DAS (iigcd) is a companion to the Ingres Name Server (iigcn) and the Communications Server (iigcc), and supports GCA tracing and other similar module tracing.

To enable DAS tracing, use any of these methods:

- Add entries to the Ingres configuration file (config.dat) in the gcd section. This method is preferred because it allows trace output from multiple servers to be logged in the same file.
- Set the environment variables prior to starting the server.
- Enable tracing dynamically by using the iimonitor set trace command. This command overrides trace settings in the config.dat and environment variables. For details, see the iimonitor set trace command in the *Command Reference Guide*.

As a general rule, use the config.dat file for server tracing and the environment variables for client tracing.

The entries or values you must supply are as follows:

Configuration File Entry	Environment Variable	Value	Description
gcd_trace_log	II_GCD_LOG	log	Path and file name of the trace log
gcd_trace_level	II_GCD_TRACE	0 – 5	Tracing level for the DAS

Tracing Levels

The tracing level determines the type of information that is logged. The following levels are currently defined:

- 1 – Errors and exceptions
- 2 – High level method invocation
- 3 – High level method details
- 4 – Low level method invocation
- 5 – Low level method details

Chapter 10: Understanding ODBC Connectivity

This section contains the following topics:

- [ODBC Driver](#) (see page 147)
- [ODBC Call-level Interface](#) (see page 148)
- [Unsupported ODBC Features](#) (see page 149)
- [Read-Only Driver Option](#) (see page 149)
- [ODBC Driver Requirements](#) (see page 149)
- [Access to a Remote Instance](#) (see page 153)
- [Configure a Data Source \(Windows\)](#) (see page 153)
- [Configure a Data Source \(UNIX and VMS\)](#) (see page 155)
- [Connection String Keywords](#) (see page 156)
- [ODBC CLI Implementation Considerations](#) (see page 159)
- [Supported Applications](#) (see page 160)
- [ODBC Programming](#) (see page 161)
- [Ingres ODBC and Distributed Transactions \(Windows\)](#) (see page 214)
- [ODBC Trace Diagnostics](#) (see page 217)

This chapter introduces the Ingres ODBC components that enable ODBC connectivity to Ingres data sources. It provides a description of each component, a list of supported API features, data source configuration instructions, connection string keyword definitions, and guidelines for implementing ODBC-enabled applications in the Ingres environment.

ODBC Driver

The Ingres ODBC driver (subsequently referred to as the ODBC driver) enables ODBC-enabled applications to access Ingres, Enterprise Access, and EDBC databases. The driver is installed as part of a standard Ingres client installation or as a stand-alone product.

ODBC Call-level Interface

The Ingres ODBC Call-level Interface (ODBC CLI) provides access to the ODBC application environment without the need to use third-party software. It is installed when you install the Ingres ODBC Driver and is supported on all platforms on which Ingres runs.

The Ingres ODBC CLI performs the following functions:

- Optionally determines driver characteristics from ODBC configuration files
- Loads and unloads the ODBC driver into and from application memory
- Maps the driver manager API to the driver API
- Performs basic error checking
- Provides thread safety
- Provides ODBC tracing
- Provides function templates, type definitions, and constant definitions for ODBC applications
- Provides connection pooling, which allows connections to be shared in ODBC applications.

Note: The ODBC CLI is not a generic ODBC driver manager. While it does provide functions similar to other ODBC driver managers, it is designed specifically to support ODBC-based application access to the Ingres 3.5 ODBC driver. It does not support Ingres ODBC drivers provided by third-party vendors.

The ODBC CLI is available on all supported platforms except Windows. The `iiodbadmin` utility configures ODBC Data Source Definitions for the CLI. For more information, see [Configure a Data Source \(UNIX and VMS\)](#) (see page 155).

Unsupported ODBC Features

The ODBC driver does not currently support the following features:

- Executing functions asynchronously
- Translation DLL (Ingres handles this requirement through the `II_CHARSETxx` environment variable.)
- The GUID (Globally Unique Identifier) data type, which is specific to Microsoft Access databases.
- Installer DLL

On Windows, the Microsoft installer DLL can be used to install the Ingres ODBC driver, if required. The Ingres ODBC driver can be installed from the Ingres installer software or the Ingres ODBC Standalone Patch Installer.

On non-Windows platforms, the `odbcinst` and `iisuodbc` utilities use the Ingres ODBC Configuration API to configure driver information.

- `SQLBulkOperations()`

Read-Only Driver Option

To support the release of a non-configurable read-only driver into production environments, the ODBC driver can optionally be installed as a read-only driver. This driver allows SQL statements such as `SELECT`, `EXECUTE PROCEDURE`, and `ODBC CALL`, but does not allow update statements (for example, `INSERT`, `DELETE`, `UPDATE`, `CREATE`, and so on).

Both ODBC drivers (read-only and read/update) are installed during the standard Ingres installation. Selection of the driver type is performed during configuration of an ODBC data source. For more information, see *Configure a Data Source (Windows)* (see page 153) and *Configure a Data Source (UNIX and VMS)* (see page 155).

ODBC Driver Requirements

The following sections list the ODBC driver software, platform, and protocol requirements. For additional information relating to the ODBC driver, see <http://www.ingres.com>. The Ingres ODBC CLI and Driver Manager are packaged with every Ingres release and service pack. ODBC patches are also available from Ingres Support.

ODBC Driver Manager Programs

The following are the installation requirements for the ODBC driver.

Windows: Microsoft's ODBC Driver Manager must be installed to use the ODBC driver (release 2.5 or above of the ODBC Driver Manager is acceptable). If the existing Windows installation has no ODBC Administrator or ODBC driver, these items can be downloaded as part of the Microsoft Data Access SDK (MDAC) from <http://www.microsoft.com>. 📄

UNIX and VMS: The Ingres ODBC CLI is the preferred ODBC driver manager if no other ODBC drivers are required. No additional download is required. The only requirement for installation is to execute the utility `iisuodbc`. The `iisuodbc` utility provides configuration information to Ingres and creates an ODBC configuration file.

If the ODBC application requires non-Ingres ODBC drivers, unixODBC Driver Manager can be installed to use the Ingres ODBC driver. The unixODBC Driver Manager is often included with Linux or UNIX installations or can be downloaded from <http://www.unixodbc.org>. The download includes a Readme file with instructions for UNIX, Linux, and VMS. 📄

UnixODBC Implementation Considerations

The Ingres ODBC driver can be used with the unixODBC Driver Manager on non-Windows platforms. Unlike the Ingres ODBC CLI, the unixODBC Driver Manager allows other ODBC drivers to be used in addition to the Ingres ODBC driver. To build the application, the include files `sql.h` and `sqlx.h` files in `$II_SYSTEM/ingres/files` can be used; alternatively, the include files `sql.h`, `sqlx.h`, `sqltypes.h`, and `sqlcode.h` provided with the unixODBC installation can be used.

UnixODBC data sources can be configured using the Ingres ODBC Administrator, just as for the Ingres ODBC CLI. Unlike the CLI, the `ODBCINI` and `ODBCSYSINI` environment variables must be specified if the locations for the `odbc.ini` and `odbcinst.ini` files differ from the default locations. For more information about `ODBCINI` and `ODBCSYSINI` in the unixODBC environment, see the unixODBC documentation.

Linux: Here is an example for building a unixODBC application on Linux:

```
cc -c myOdbcApp.c /I/disk1/unixODBC-2.2.12/include
ld -o myOdbcApp myOdbcApp.o -L/disk1/unixODBC-2.2.12/DriverManager -lodbc
```

VMS: Here is an example for building a unixODBC application on VMS:

```
$ cc/nowarn/name=as_is/nodeb/float=ieee/opt/include=DKA0:[TESTENV.UNIXODBC-
2.2.3.INCLUDE]-
  odbctest.c
$ ln/nodeb/exe=odbctest_unixodbc.exe odbctest.obj, sys$input/opt
CASE_SENSITIVE=YES
sys$share:ODBC.EXE/share
SYS$LIBRARY:PTHREAD$RTL.EXE/SHARE
```

Notes:

- The “as_is” qualifier is required for compiling and the “CASE_SENSITIVE=YES” qualifier is used for unixODBC. This means that the ODBC function entry points are case-sensitive, which is the default for unixODBC on VMS.
- For successful loading of the Ingres ODBC Driver, `SYS$LIBRARY:PTHREAD$RTL.EXE` must be included when the unixODBC image is linked. Multi-threading, however, is not currently supported for the Ingres ODBC driver on VMS.

Support for Previously Released ODBC Drivers

Each release of Ingres includes an ODBC driver. Each ODBC driver is designed to be compatible with the major and minor release numbers of Ingres. Thus, the ODBC driver released with Ingres 9.1.1 is designed to be compatible with Ingres 9.1.0, since the major and minor release numbers are 9 and 1, respectively. Compatibility beyond that scope is not supported. Thus, an ODBC driver released with Ingres 10.0 is not compatible with an Ingres 9.2 environment, or vice versa.

For Ingres 9.0 (also known as Ingres 2006) and higher, the Ingres ODBC Driver 3.5 is required. For previous releases, either ODBC 3.5 or ODBC 2.8 is supported. Prior to Ingres 9.0, there is no Ingres ODBC CLI on UNIX; the unixODBC Driver Manager is required.

ODBC Driver Names

The Ingres ODBC driver is installed with the driver names "Ingres" and "Ingres xx", where xx is the value of II_INSTALLATION. Thus, regardless of the Ingres version or location, the driver name "Ingres" always references the most recent installation of the Ingres ODBC driver, and "Ingres xx" is the name of the driver associated with an Ingres installation with an instance ID of "xx". Read-only drivers are named "Ingres Read Only" and "Ingres xx Read Only."

If multiple versions of Ingres are installed on the same machine, we recommend to always reference the "Ingres xx" driver in connection strings or when creating an ODBC data source name; otherwise, subsequent installations or upgrades may corrupt the driver path defined by the driver name "Ingres". If the machine has only one Ingres installation, "Ingres" or "Ingres xx" can be used.

Backward Compatibility Issues for ODBC DSN Definitions

On UNIX, Linux, and VMS, the ODBC driver is loaded automatically from the "library" directory on Ingres, regardless of the ODBC DSN definition.

On Windows, the path of the ODBC driver remains hard-coded in the registry. Thus, the ODBC DSN definitions must be refreshed if Ingres is installed in a different location than the previous installation.

Access to a Remote Instance

Before connecting to a remote Ingres DBMS, Enterprise Access, or EDBC instance, a vnode connection must be defined.

More information

Requirements for Accessing Remote Instances (see page 50)
Access Tools for Defining Vnodes (see page 51)

Configure a Data Source (Windows)

A data source configuration is a collection of information that identifies the database you want to access using the ODBC driver. You can configure as many data sources as you require. Once defined, a data source is available for use by any application that uses ODBC.

ODBC data sources are a convenient way of connecting to a database. You can, however, connect to a database without them by using only a connection string. For details, see Connection String Keywords (see page 156).

Note: On 64-bit Windows, when creating a new data source using the ODBC Data Source Administrator, the Ingres ODBC driver does not appear in the list of drivers on the User DSN or System DSN tab. The Ingres ODBC driver is a 32-bit driver and appears only in the 32-bit ODBC Data Source Administrator. The ODBC Data Source Administrator shortcut in the Control Panel is a 64-bit shortcut and will bring up 64-bit ODBC Data Source Administrator and only display the 64-bit drivers.

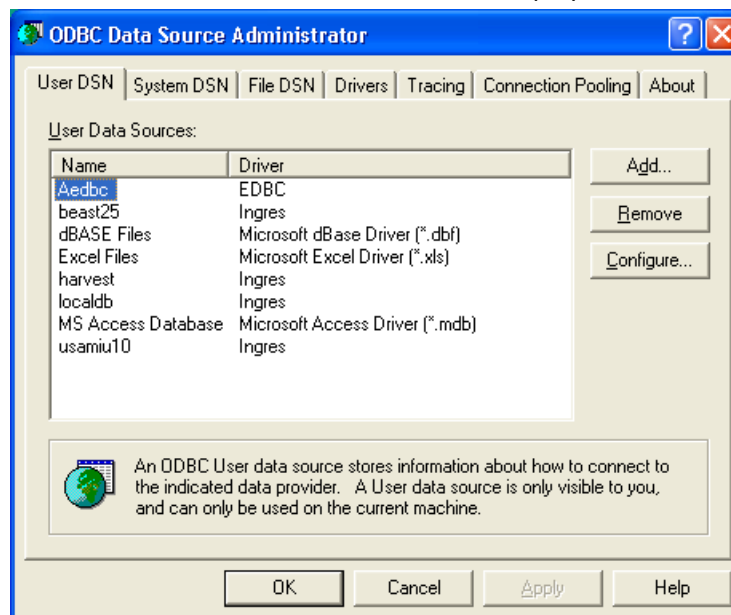
To bring up the 32-bit ODBC Data Source Administrator, which will list the Ingres ODBC driver, run `c:\windows\SysWow64\odbcad32.exe`.

To configure a new data source on Windows

1. Run the ODBC Data Source Administrator provided on Windows.

To do this on Windows XP, click Start, Control Panel, Administrative Tools, Data Sources (ODBC).

The ODBC Data Source Administrator is displayed:



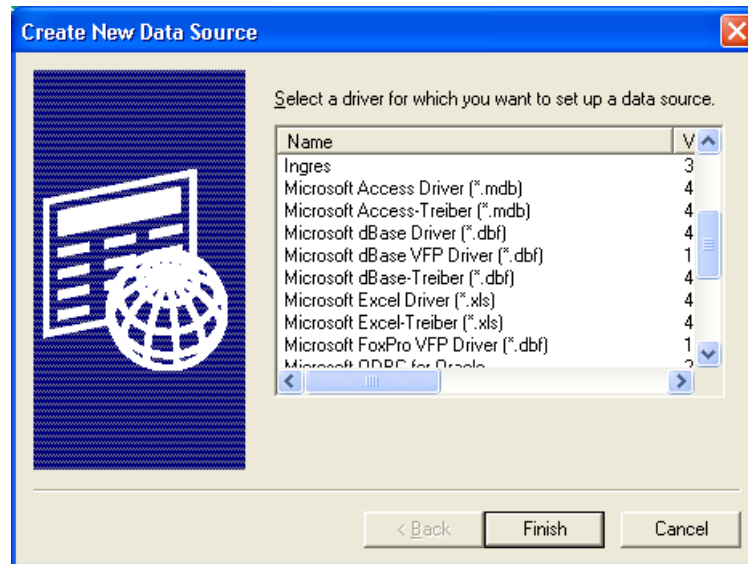
You can define one or more data sources for each installed driver. The data source name must provide a unique description of the data; for example, Payroll or Accounts Payable.

A data source can be defined as system or user, depending on whether it must be visible to all users (and services) or only the current user.

2. Select the User DSN or the System DSN tab, depending on your requirements, and click Add.

Note: A system DSN pointing to a public server definition is required for Microsoft Internet Information Server (IIS) and Microsoft Transaction Server (MTS).

The Create New Data Source dialog opens, which lists all the ODBC drivers installed on your system.



Note: To switch ODBC DSNs defined previously for the ODBC 2.8 driver to the new ODBC 3.5 driver, remove the DSN by selecting it in the ODBC Data Source Administrator Data Sources list, and clicking Remove. Add the DSN again using the new ODBC driver.

3. Select the Ingres driver and click Finish.

The Ingres ODBC Administrator dialog opens.

Fill in the necessary fields on this dialog. For details, see the online help.

Configure a Data Source (UNIX and VMS)

A data source configuration is a collection of information that identifies the database you want to access using the ODBC driver. You must configure a data source before connecting to a database through ODBC.

To configure a new data source on UNIX and VMS

Run the Ingres ODBC Administrator utility, **iiodbcadmin**.

iiodbcadmin Utility

The Ingres ODBC Administrator (iiodbcadmin) utility lets you perform the following tasks:

- Create, edit, and remove data sources
- Perform configuration tasks for each installed driver, including:
 - Turn on ODBC tracing
 - Define the path of the driver
 - View details about the configuration
 - Set configuration options, such as connection pooling timeout
- Test a data source connection

For details, see the Ingres ODBC Administrator online help.

Connection String Keywords

If your application requires a connection string to connect to a data source, you must specify the data source name. Optionally, you can specify *attribute=value* pairs to override certain data source and vnode definitions. The ODBC function SQLDriverConnect() or SQLDriverConnectW() is required for connection strings.

The connection string has the form:

```
DSN=data_source_name[:attribute=value[:attribute=value]...]
```

Alternatively, you can bypass data source definitions entirely if you include sufficient information in the connection string. The minimum attributes in this case are the SERVER, SERVERTYPE, and DATABASE *attribute=value* pairs.

Connection strings that bypass the DSN definition have the form:

```
CONNECTSTR=SERVER=server_name; SERVERTYPE=server_type;  
DATABASE=database[:attribute=value]...]
```

The following table provides the keywords for each connection string attribute.

Keyword	Attribute Value Description
DSN	Data source name.
DRIVER	Driver description as returned by SQLDrivers().

Keyword	Attribute Value Description
UID	User ID to override vnode definition.
PWD	Password to override vnode definition. If specified, UID must also be specified.
DBMS_PWD	Database password. Database passwords are defined by the accessdb or VDBA utilities or by the CREATE USER SQL command. Although supported as a connection string attribute, DBMS_PWD is not supported as a DSN configuration attribute.
SERVER	Vnode name (see page 51).
SERVERTYPE	Server type (for example, INGRES, IDMS, or DB2).
DATABASE	Database name as defined on the server.
DB	A synonym for DATABASE.
ROLENAME	Role name to override vnode definition.
ROLEPWD	Role password to override vnode definition.
GROUP	Group identifier for the session. Equivalent to the -G flag of the Ingres command-line flags.
BLANKDATE	=NULL Indicates that the driver must return empty string DATE values as NULL.
DATE1582	=NULL Indicates that the driver must return values of '1582-01-01' as NULL.
DATE	Same as DATE1582 keyword.
SELECTLOOPS	=N Indicates that Cursor Loops must be used.
CATCONNECT	=Y Indicates that a second separate Ingres session must be used for catalog functions (SQLTables, and so on).
NUMERIC_ OVERFLOW	=IGNORE Indicates that no error is issued if an arithmetic error of numeric overflow, underflow, or divide by zero occur. Equivalent to "-numeric_overflow=ignore" command line flag.

Keyword	Attribute Value Description
CATSCHEMANULL	=Y Returns NULL for schema names from ODBC catalog functions.
CONVERTINT8TOINT4	=Y Coerces eight-byte (i8) integer values from the DBMS to four-byte (i4).
ALLOWUPDATE	=Y Allows updates on sessions set to read-only.
DEFAULTTOCHAR	=Y Treats Unicode strings as "standard" (multi-byte) strings.
DISABLEUNDERSCORE	=Y Disables underscore wildcard search characters in catalog function.

ODBC CLI Implementation Considerations

The ODBC CLI includes two include files for compiling applications:

- `sql.h`
- `sqlext.h`

These files can be found at `$II_SYSTEM/ingres/files`.

Other standard ODBC includes libraries, such as `sqlcode.h` or `sqltypes.h`, are already included within the ODBC CLI version of `sql.h` and `sqlext.h`.

UNIX and Linux: The ODBC CLI is installed as the shared library `libiiodbc.[ext]`. Depending on the UNIX or Linux implementation, the library extension (`[ext]`) varies. The library resides in `$II_SYSTEM/ingres/lib`.

Here is an example for building an ODBC CLI application on Linux:

```
cc -c myOdbcApp.c -I$II_SYSTEM/ingres/files
ld -o myOdbcApp myOdbcApp.o -L$II_SYSTEM/ingres/lib -liiodbc.1
```

VMS: The library is named `ODBCCLIFELIB.EXE` and resides in `II_SYSTEM:[INGRES.LIBRARY]`.

ODBC CLI applications link against the appropriate shared library. The logical `II_ODBC_CLILIB` references the Ingres ODBC CLI library. The `II_ODBC_CLILIB` library is defined by the `iisuodbc` utility.

ODBC applications can be coded using the ODBC Command Line Interface exactly as if coded with the Microsoft ODBC Driver Manager library or `unixODBC` Driver Manager library.

Here is an example for building an ODBC CLI application on VMS:

```
$ CC/nowarn/include=ii_system:[ingres.files] myApp.c
$ LINK myApp.obj, sys$input/opt
ii_odbc_clilib/share
```

Configuration on UNIX, Linux, and VMS

Before using the ODBC CLI, the utility `iisuodbc` must be executed. `Iisuodbc` configures Ingres with the appropriate name of the ODBC driver library and creates the ODBC configuration file, `odbcinst.ini`. For more information, see [Configure a Data Source \(UNIX and VMS\)](#) (see page 155).

Optional Data Source Definitions

The use of `odbcinst.ini`, `odbc.ini` or the ODBC configuration registry is optional for the ODBC CLI. An application can invoke `SQLDriverConnect()` and specify a connection string that omits an ODBC Data Source (DSN) specification.

If the connection string has sufficient information to connect to the database, the location and name of the ODBC driver library are automatically determined. The use of the `iiodbcadmin` utility (UNIX, Linux, and VMS) or Microsoft ODBC Administrator (Windows) is optional.

Supported Applications

The Ingres ODBC driver is compatible with many database applications that use ODBC. Little or no knowledge of ODBC is required to use these applications.

The following list is a sample of applications and application languages that can be used with the Ingres ODBC Driver:

- Microsoft Access
- Microsoft Excel
- Open Office Base
- Microsoft ADO using OLE DB and ODBC
- Microsoft .NET Data Provider for ODBC (ODBC.NET)
- Microsoft .Net Data Provider for ADO using OLE DB and ODBC
- Microsoft ASP Web Pages
- Ingres Python DBI Driver
- JDBC/ODBC Bridge
- PHP ODBC Interface
- Business Objects Crystal Reports
- Microsoft Query

For detailed information on using these technologies, see the Ingres Community Wiki ODBC section.

ODBC Programming

If you are proficient in the C programming language, you can call ODBC routines directly. The Ingres ODBC driver supports all platforms (operating systems) that Ingres supports, so this is an option if you want to run your ODBC application on more than one platform.

Low-level ODBC applications can be more efficient than higher-level applications, since the layers of software between your application and the data are reduced. The disadvantage of low-level ODBC applications is that less of the work is automated, and the interface is less intuitive than a higher-level application interface such as ADO or the .NET Data Provider for ODBC.

The following functional overview examines some of the ODBC function calls and discusses issues specific to the Ingres environment.

ODBC Handles

ODBC handles store the environment of ODBC execution components. The components that the handles describe are not visible to the ODBC application, but they are visible internally in the ODBC driver.

There are four types of ODBC handles:

- Environment handles describe the general ODBC runtime environment.
- Connection handles describe the ODBC environment specific to individual connections.
- Statement (query) handles describe the ODBC environment for queries.
- Descriptor handles describe metadata information related to result sets.

`SQLAllocHandle()` allocates ODBC driver resources for these four types of handles. The format of `SQLAllocHandle()` is:

```
SQLAllocHandle( HandleType, ParentHandle, Handle)
```

HandleType

Specifies the type of handle to be allocated.

ParentHandle

Specifies the handle with which the allocated handle is associated.

Handle

Identifies the handle itself.

How ODBC Applications Connect to a Database

ODBC applications are never linked directly against the ODBC driver DLL or shared library. Instead, the applications are linked against the ODBC Driver Manager, which can be the Microsoft ODBC Driver Manager, the unixODBC Driver Manager, or the Ingres ODBC CLI.

The ODBC application has no knowledge of the driver until a connection is made. During a successful connection, the Ingres ODBC driver is dynamically loaded into the program image. After loading, the functions called in the ODBC application are passed down to the driver implementation of the function. In this way, the same ODBC application can be used with different ODBC drivers, and thus can be used with a variety of different databases. This portability is a principal aim of ODBC.

The Ingres ODBC driver is designed to work from a driver manager such as unixODBC or the Ingres ODBC CLI. Applications link with the driver manager. The Ingres ODBC driver is loaded into the driver manager when the first connection is made to a database. One can theoretically link directly with the Ingres ODBC driver, leaving out the driver manager, but this is not supported and may yield unpredictable results.

The Ingres ODBC Driver, the Ingres ODBC CLI, and the unixODBC Driver Manager are implemented as shared libraries on UNIX, but the names vary according to platform conventions. For example, on Linux, for 32-bit Ingres: the Ingres ODBC Driver is implemented as `$II_SYSTEM/ingres/lib/libiodbcdriver.1.so` and `libiodbcdriverro.1.so`, and the Ingres ODBC CLI is implemented as `libiodbc.1.so`.

SQLConnect()—Connect Using a Data Source Name

SQLConnect() allows you to connect to the database using connection attributes stored in the ODBC Data Source Name (DSN) definition. See [Configure a Data Source \(Windows\)](#) (see page 153) and [Configure a Data Source \(UNIX and VMS\)](#) (see page 155) for information about creating an ODBC DSN definition.

The following program is the minimum code required for:

- Initializing ODBC driver
- Setting the ODBC version to version 3
- Connecting to the database
- Disconnecting from the database
- Cleaning up and exiting

Example: SQLConnect() Function

```

# ifdef _WIN32
# include <windows.h>
# else
# include <stdio.h>
# include <stdlib.h>
# endif
# include <sql.h>
# include <sqlext.h>

main( )
{
    HENV henv = NULL;
    HDBC hdbc = NULL;

    /* Initialize the ODBC environment handle. */
    SQLAllocHandle( SQL_HANDLE_ENV, NULL, &henv );

    /* Set the ODBC version to version 3 (the highest version) */
    SQLSetEnvAttr( henv, SQL_ATTR_ODBC_VERSION,
        (void *)SQL_OV_ODBC3, 0 );

    /* Allocate the connection handle. */
    SQLAllocHandle( SQL_HANDLE_DBC, henv, &hdbc );

    /* Connect to the database using the ODBC DSN definition. */
    SQLConnect( hdbc,          /* Connection handle */
        (SQLCHAR *)"myDSN",   /* The ODBC DSN definition */
        SQL_NTS,              /* This is a null-terminated string */
        (SQLCHAR *)NULL,      /* No username required */
        SQL_NTS,              /* This is a null-terminated string */
        (SQLCHAR *)NULL,      /* No password required */
        SQL_NTS );            /* This is a null-terminated string */

    /* Disconnect from the database. */
    SQLDisconnect( hdbc );

    /* Free the connection handle. */
    SQLFreeHandle( SQL_HANDLE_DBC, hdbc );

    /* Free the environment handle. */
    SQLFreeHandle( SQL_HANDLE_ENV, henv );

    /* Exit this program. */
    return( 0 );
}

```

If your login name is a valid Ingres user for a local Ingres database, the above code is all you need to connect through SQLConnect(). See the ODBC User Authorization section for a discussion on authorizing yourself as another user.

Other details of the connection, such as the database name and type of database, are pre-defined in the ODBC DSN definition referenced by the string "myDSN".

Note that `SQLAllocHandle()` was invoked to initialize the environment handle and that `SQLSetEnvAttr()` sets the ODBC version to `SQL_OV_ODBC3`. These two calls are necessary to initialize your ODBC application as an ODBC level 3 driver. The default is ODBC level 2. It is advisable to initialize as version 3 because some ODBC level 3 functions, such as the treatment of date and time values, depend on the ODBC version being set at level 3.

SQLDriverConnect()—Connect without Using a Data Source Name

You can bypass ODBC DSN information completely if you use the `SQLDriverConnect()` function instead of `SQLConnect()`. `SQLDriverConnect()` uses a connection string instead of an ODBC DSN definition; the `odbc.ini` file is not used unless the DSN connection string attribute is specified in the connection string.

Connection strings are typically used in higher-level languages that use ODBC, such as:

- Microsoft ADO
- Microsoft OLE DB Provider for ODBC
- Microsoft .NET Data Provider for ODBC
- PHP ODBC interface
- Python DBI driver
- JDBC/ODBC Bridge

An ODBC connection string is structured as a set of “attribute, value” pairs in the following abstract:

```
connectStr = attribute=value[;attribute=value[;...]]
```

Each set of “attribute, value” pairs is separated by a semicolon (“;”). No spaces or other separators, such as <TAB>, are allowed between semicolons and the next attribute pair.

The attribute pair “DSN=myDSN” causes `SQLDriverConnect()` to search for an ODBC DSN specification just as in `SQLConnect()`. However, other connection attributes specified in the connection string override any similar specification in the ODBC DSN definition.

`SQLDriverConnect()` requires you to provide the following minimum connection attributes:

- Database
- Type of driver
- Type of database (Ingres, SQLServer, Oracle, etc.)
- Server. In Ingres terms, the server is the name of a vnode definition in `neutil`, `ingnet`, or `VDBA`. If you are connecting locally, the default string name “(local)” indicates a local connection.

The following program using `SQLDriverConnect()` performs exactly the same function as the previous code example.

Example: SQLDriverConnect() Function

```
# ifdef _WIN32
# include <windows.h>
# else
# include <stdio.h>
# include <stdlib.h>
# endif
# include <sql.h>
# include <sqltext.h>

main()
{
    HENV henv=NULL;
    HDBC hdbc=NULL;
    SQLCHAR connectString[256];

    /* Initialize the ODBC environment handle. */
    SQLAllocHandle( SQL_HANDLE_ENV, NULL, &henv );

    /* Set the ODBC version to version 3 (the highest version) */
    SQLSetEnvAttr( henv, SQL_ATTR_ODBC_VERSION,
        (void *)SQL_OV_ODBC3, 0 );

    /* Allocate the connection handle. */
    SQLAllocHandle( SQL_HANDLE_DBC, henv, &hdbc );

    /*
    ** Fill the connection string with the minimum
    ** connection information.
    */
    strcpy( connectString, "driver={Ingres};servertype=Ingres;" \
        "server=(local);database=myDB" );

    /* Connect to the database using the connection string. */
    SQLDriverConnect( hdbc, /* Connection handle */
        0, /* Window handle */
        connectString, /* Connection string */
        SQL_NTS, /* This is a null-terminated string */
        (SQLCHAR *)NULL, /* Output (result) connection string */
        SQL_NTS, /* This is a null-terminated string */
        0, /* Length of output connect string */
        SQL_DRIVER_NOPROMPT ); /* Don't display a prompt window */

    /* Disconnect from the database. */
    SQLDisconnect( hdbc );

    /* Free the connection handle. */
    SQLFreeHandle( SQL_HANDLE_DBC, hdbc );

    /* Free the environment handle. */
    SQLFreeHandle( SQL_HANDLE_ENV, henv );

    /* Exit this program. */
    return(0);
}
```

The example above applies to Windows programs. On UNIX, Linux, and VMS, if you are using the Ingres ODBC CLI, only the database name is required as the minimum connection attribute:

```
strcpy( connectString, "database=myDB" );
```

The Ingres ODBC CLI assumes that the driver type is "Ingres", the server type is "Ingres", and that the server is "(local)" if you do not so specify.

A complete list of ODBC connection attributes and values is available in the Connection String Keywords (ODBC) section.

Connect Using Dynamic Vnode Definitions

Vnode definitions allow applications to connect to databases over the network. The "server" connection string attribute equates to the vnode name.

Normally, you use VDBA, ingnet or netutil to define the vnode name and network connections, similar to ODBC DSN definitions. However, both DSN and vnode definitions may be bypassed as shown in the following example code snippet:

```
strcpy( connectString, "driver=Ingres;servertype=ingres;" \
    "server=@myHostname.myDomain.com,tcp_ip,II;" \
    "uid=myUserName;pwd=myPassword;database=myDatabase" );
```

The above connection string is an example of dynamic vnode definitions discussed in the Using Net chapter of this guide. The example shows that the ODBC can connect to a database without pre-defining any connection information.

Note: When you use dynamic vnode definitions in your connection string, you must specify the "uid" (user name) and "pwd" (password) connection attributes.

ODBC User Authentication

There are several methods for authorizing users from within ODBC applications.

User Name and Password

SQLConnect() and SQLDriverConnect() allow your application to specify a user (login) name and password. For example:

```
SQLConnect(hdbc,                /* Connection handle */
           (SQLCHAR *) "myDSN", /* The ODBC DSN definition */
           SQL_NTS,             /* This is a null-terminated string */
           (SQLCHAR *) "myUserName", /* Local user name */
           SQL_NTS,             /* This is a null-terminated string */
           (SQLCHAR *) "myPassword", /* Local password */
           SQL_NTS);            /* This is a null-terminated string */
```

In SQLDriverConnect(), the "uid" and "pwd" attributes specify the local user name and local password, respectively.

Specification of the user name and password authorizes the ODBC application to act as it were logged in to the local machine as that user. This is true even if the ODBC DSN definition specifies a network connection to another machine. If you specify an invalid user name or password, the connection fails regardless of whether the connection target is local or over the network.

Ingres Super Users

An Ingres user with security administrator privileges can assume the identity of other users without having to provide a password. Such users are called Ingres "super" users, as in Linux and Unix notation. An example in SQLConnect() follows:

```
SQLConnect( hdbc,                /* Connection handle */
           (SQLCHAR *) "myDSN", /* The ODBC DSN definition */
           SQL_NTS,             /* This is a null-terminated string */
           (SQLCHAR *) "altUserName", /* Alternate username */
           SQL_NTS,             /* This is a null-terminated string */
           (SQLCHAR *) NULL,     /* No password required */
           SQL_NTS );           /* This is a null-terminated string */
```

The example code allows the ODBC application to authorize as "altUserName" without a password. In functional terms, this is the equivalent to the -u flag in Terminal Monitor.

Note: The local user name must be an Ingres super user when connecting locally. If connecting remotely, the user name defined in netutil for the target vnode must be an Ingres super user.

Installation Passwords

SQLConnect() and SQLDriverConnect() may be used with installation passwords. The following code example demonstrates the use of installation passwords with SQLDriverConnect(). Note that the user name and password are not specified.

```
strcpy( connectString, "driver={Ingres};servertype=Ingres;" \
    "server=vnodeDef;database=remoteDB" );

SQLDriverConnect( hdbc,          /* Connection handle */
    0,                          /* Window handle */
    connectString,              /* Connection string */
    SQL_NTS,                    /* This is a null-terminated string */
    (SQLCHAR *)NULL,           /* Output (result) connection string */
    SQL_NTS,                    /* This is a null-terminated string */
    0,                          /* Length of output connect string */
    SQL_DRIVER_NOPROMPT ); /* Don't display a prompt window */
```

If the ODBC application executes in the local "ingres" account, the target Name Server authenticates "ingres" using the installation password for the target database. You may not use installation passwords for a local database unless the connection is done via a vnode definition.

The Ingres super user concept applies to installation passwords. Ingres super users can specify alternate user names, but do not need to supply passwords, as shown in the following SQLConnect() example:

```
SQLConnect( hdbc,                /* Connection handle */
    (SQLCHAR *)"myDSN",          /* The ODBC DSN definition */
    SQL_NTS,                      /* This is a null-terminated string */
    (SQLCHAR *)"altUserName",    /* Alternate username */
    SQL_NTS,                      /* This is a null-terminated string */
    (SQLCHAR *)NULL,             /* No password required */
    SQL_NTS );                  /* This is a null-terminated string */
```

The example above assumes "myDSN" references a vnode definition.

DBMS Passwords

DBMS passwords are extra passwords defined for Ingres users and maintained in the Ingres database. When an Ingres user account is associated with a DBMS password, the correct DBMS password must be supplied as a connection parameter, otherwise the connection attempt will be rejected.

DBMS passwords in the DSN definition in `odbc.ini` are not supported. The Ingres ODBC driver supports DBMS passwords only through `SQLDriverConnect()`, as shown in the following example:

```
/*
** The "dbms_pwd" connection attribute specifies the DBMS password.
*/
strcpy( connectString, "driver={Ingres};servertype=Ingres;" \
    "server=(local);database=myDB;uid=myUserName; \
    "pwd=myPassword;dbms_pwd=myDBMS_password );

SQLDriverConnect( hdbc,          /* Connection handle */
    0,                          /* Window handle */
    connectString,               /* Connection string */
    SQL_NTS,                     /* Nll-terminated string */
    (SQLCHAR *)NULL,            /* Output connection string */
    SQL_NTS,                     /* N=ull-terminated string */
    0,                           /* Length of output connect string */
    SQL_DRIVER_NOPROMPT ); /* No prompt window */
```

Kerberos

User name or password specifications are not required in a Kerberos authentication environment, as shown in the following example:

```
SQLConnect( hdbc,          /* Connection handle */
            (SQLCHAR *) "myDSN", /* The ODBC DSN definition */
            SQL_NTS,          /* This is a null-terminated string */
            (SQLCHAR *) NULL, /* No username required */
            SQL_NTS,          /* This is a null-terminated string */
            (SQLCHAR *) NULL, /* No password required */
            SQL_NTS );        /* This is a null-terminated string */
```

For Kerberos authentication to succeed, the following requirements must be met:

- Kerberos must be installed and configured on the Kerberos client and Kerberos KDC (Kerberos Domain Controller).
- The current login session must have valid Kerberos tickets obtained via the "kinit" command or from a Kerberos authorization utility such as Leash on Windows.
- An Ingres service principal name must be defined with keytabs defined in a Kerberos keytab file.
- Kerberos must be configured for Ingres via the "iisukerberos" command and possibly the CBF utility.
- The ODBC DSN definition "myDSN" must reference a vnode definition, regardless of whether the connection is to a local or remote database.
- The vnode definition must define the "authentication_mechanism" attribute as "kerberos".

For a full discussion of Kerberos authentication in an Ingres environment, see the *Security Guide*.

SQLConnect() authenticates the application according to the Kerberos principal. The user name of the Kerberos principal is passed to the DBMS as the owner of the connection; otherwise, no authentication information is passed between the client and server when SQLConnect() is invoked. Authentication is performed by the Ingres Service Principal of the server-side Name Server.

Dynamic (Run-Time) Authentication (Windows Only)

Some ODBC applications require that the user name and password are not hard-coded in the code. To address these requirements, ODBC applications can prompt the user for more information.

When prompted for connection information, the Ingres ODBC driver displays the following input page:



The screenshot shows a Windows-style dialog box titled "Ingres 2006 DriverConnect". It contains several input fields and two buttons. The fields are: "Database:" with the text "var", "ServerName:" with a dropdown menu showing "INGNET02", "ServerType:" with a dropdown menu showing "INGRES", "User Id:" (empty), "Password:" (empty), "Role Name:" (empty), "Role PWD:" (empty), "Group:" (empty), and "DBMS PWD:" (empty). The "OK" and "Cancel" buttons are located to the right of the "Database:" and "ServerName:" fields respectively.

Users can then enter the missing login information in the prompt window, and the application will attempt to connect using the supplied information.

Connection Prompt Window—Using SQLDriverConnect()

The Ingres ODBC driver supports the DriverCompletion argument in SQLDriverConnect(). The DriverCompletion argument is the eighth and last argument in SQLDriverConnect(). The value SQL_DRIVER_PROMPT directs the driver to display a prompt window. The value SQL_DRIVER_NOPROMPT directs the driver to not display a prompt window.

Example: DriverCompletion Argument

```
/*
** Fill the connection string with the minimum
** connection information.
*/
strcpy( connectString, "driver={Ingres};servertype=Ingres;" \
    "server=(local);database=myDB" );

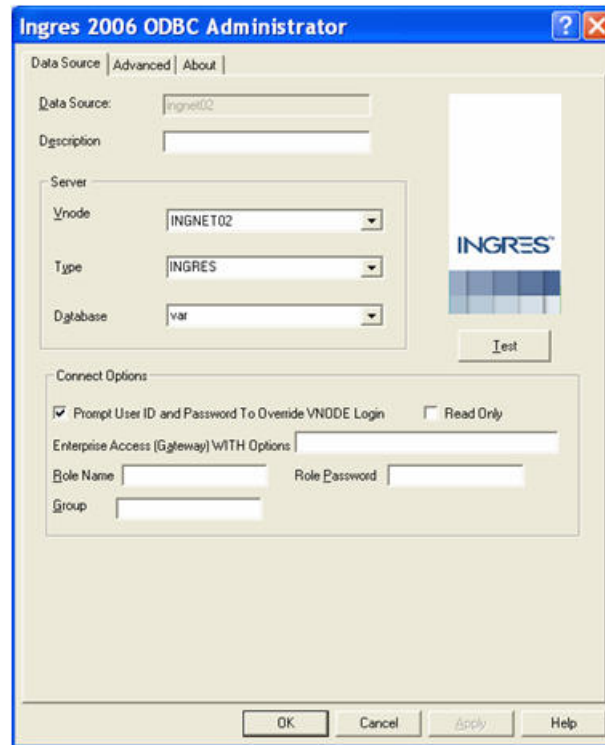
/* Connect to the database using the ODBC DSN definition. */
SQLDriverConnect( hdbc,      /* Connection handle */
    winHandle,              /* Window handle */
    connectString,          /* Connection string */
    SQL_NTS,                /* This is a null-terminated string */
    (SQLCHAR *)NULL,        /* Output (result) connection string */
    SQL_NTS,                /* This is a null-terminated string */
    0,                      /* Length of output connect string */
    SQL_DRIVER_PROMPT );    /* Display a prompt window */
```

The example above displays a prompt window in a program that has a valid handle to a Windows application. Windows-sensitive applications such as ADO or the .NET Data Provider for ODBC are usually able to display prompts.

Note: ODBC programs that make no calls to the Windows API cannot display connection prompt windows. ADO programs executed as ASP pages from IIS cannot display prompt windows.

Connection Prompt Window—Using DSN Definition

SQLDriverConnect() is not visible in Windows applications such as ADO or the .NET Data Provider for ODBC. It is not possible to modify the DriverCompletion argument in these cases. Ingres ODBC DSN definition pages include a Prompt User ID and Password check box that forces the application to prompt for more information, as in the following example:



If the application references the ODBC DSN definition at connect time, the prompt window is displayed.

Note: IIS applications, such as ASP pages, will not display prompt windows.

Specification of User Names and Passwords in ODBC

For remote connections, the specification of user names and passwords overrides the user names or passwords in the vnode definition.

For local connections, the specification of user names and passwords causes the ODBC application to behave as if it was logged in as the alternate user. This behavior is true regardless of whether or not the current login is an Ingres "super" user.

It does not matter if user names and passwords are provided interactively or are hard-coded, except in the case of Kerberos. For Kerberos, if you specify user names and passwords, the ODBC application authenticates against the local installation in addition to the Kerberos authentication. The Kerberos user specified in "kinit" must match the user name specified in `SQLConnect()` or `SQLDriverConnect()`.

SQLBrowseConnect()—Prompt for Connection Information

`SQLBrowseConnect()` is another function that can be used to prompt for connection information. `SQLBrowseConnect()` returns missing connection information in a formatted output string that is used as a guide in the next connection attempt. When `SQLBrowseConnect()` receives the minimum connection information, a connection is made to the database.

Query Execution

This section explores methods for executing queries in ODBC applications.

SQLExecDirect()—Execute Queries Directly

`SQLExecDirect()` is commonly used for executing queries. The following code snippet creates a table and inserts a row of data.

Example: SQLExecDirect() Function

```
/* Allocate a statement handle. */
SQLAllocHandle( SQL_HANDLE_STMT, hdbc, &hstmt );

/* Execute a "create table" query. */
SQLExecDirect(hstmt, "create table cars( model varchar(20) )",
              SQL_NTS);

/* Insert one row of data. */
SQLExecDirect(hstmt, "insert into cars ( model ) values
( 'Hummer' )", SQL_NTS);
```

SQLPrepare() and SQLExecute()—Prepare and Execute Queries

For queries that are executed multiple times, preparing the queries before execution may improve performance because the DBMS stores the query plan of prepared queries. Rather than calculating a query plan each time a query is executed, the DBMS references the query plan for all subsequent iterations of the query.

SQLPrepare() and SQLExecute() perform the prepare and execute functions, respectively.

Unless the ODBC application explicitly prepares a query via SQLPrepare(), the ODBC always executes queries directly; there is no provision for automatically preparing queries in Ingres.

The following code snippet is the equivalent of the SQLExecDirect() code example in the Execute Queries Directly section.

Example: SQLPrepare() and SQLExecute() Functions

```
/* Prepare a "create table" query. */
SQLPrepare( hstmt, "create table cars( model varchar(20) )",
            SQL_NTS );

/* Execute the prepared query */
SQLExecute(hstmt);
```

Prepared queries persist in the default autocommit state. However, if you turn off autocommit via SQLSetConnectAttr(), the query plan associated with a prepared query is destroyed whenever the query is rolled back or committed. You must then re-prepare the query via SQLPrepare().

Queries with Dynamic Parameters

ODBC applications frequently require data to be sent to the database dynamically. Such data cannot be hard-coded in the query itself.

For comparison, the following set of queries inserts three rows with hard-coded parameters:

```
insert into cars( model ) values( 'Hummer' );
insert into cars( model ) values( 'Mustang' );
insert into cars( model ) values( 'Camray' );
```

The above set of queries would be adequate if it was known in advance that the target table would always receive this data. Often this is not the case, so the data must be sent dynamically. The ODBC uses question marks ("?",) as placeholders for dynamic queries:

```
insert into cars( model ) values ( ? );
```

A character string can be defined and *bound* to the query. Binding a parameter to a query means that the data is sent as a parameter along with the query. The DBMS reconstructs the parameter data with the query as if the data were hard-coded in the original query.

In order to send data to the DBMS, the ODBC must tell the DBMS what the data looks like. The ODBC function SQLBindParameter() serves that purpose. The following example allows an ODBC application to send three rows of data dynamically to a table consisting of one column with a varchar of length 20:

```
SQLCHAR model[3][21] = { "Hummer", "Mustang ", "Camray " };
int i;
SQLINTEGER orind = SQL_NTS;

[ Allocate handles and connect. ]

SQLExecDirect( hstmt, "insert into cars( model ) values( ? )",
               SQL_NTS );

for ( i = 0; i < 3; i++ )
{
    SQLBindParameter( hstmt,          /* Statement handle */
                      1,              /* Column number 1 */
                      SQL_PARAM_INPUT, /* This is an input parameter */
                      SQL_C_CHAR,     /* This is a string in C */
                      SQL_VARCHAR,    /* Destination column is varchar */
                      strlen( model[i] ), /* Length of the parameter */
                      0,               /* No precision specifier */
                      model[i],        /* The data itself */
                      0,               /* Maximum length (default 0) */
                      &orind );        /* Null-terminated string */
}
```

Dynamic parameters may be used in "where" clauses. The following example selects from the cars table, using a dynamic parameter. This query is known as a searched query:

```
SQLCHAR model[21] = "Hummer";
int i;
SQLINTEGER orind = SQL_NTS;

[ Allocate handles and connect. ]

SQLExecDirect( hstmt, "select model from cars where model =
( ? )", SQL_NTS );

SQLBindParameter( hstmt, /* Statement handle */
1, /* Column number 1 */
SQL_PARAM_INPUT, /* This is an input parameter */
SQL_C_CHAR, /* This is a string in C */
SQL_VARCHAR, /* Destination column is varchar */
strlen( model ), /* Length of the parameter */
0, /* No precision specifier */
model, /* The data itself */
0, /* Maximum length (default 0) */
&orind ); /* Null-terminated string */
```

Database Procedures Execution

If the database procedure has no parameters, database procedures can be executed in a straightforward manner using `SQLExecDirect()`:

```
SQLExecDirect( hstmt, "execute procedure myDbProc", SQL_NTS );
```

If the database procedure requires parameters, ODBC "escape sequence" syntax must be used. The ODBC uses escape sequence syntax to signify to the ODBC Driver Manager that implementation of the syntax in question is to be performed in a way that is specific to the driver.

The general form of escape syntax for database procedures is:

```
{ retcode = call dbproc [ ( ? ) [ , ( ? ) ... ] }
```

The Ingres ODBC driver supports the following Ingres database procedures:

- Input parameters
- BYREF parameters
- Returned rows
- Procedure return values

Database Procedures that Return Values

SQLBindParameter() binds parameters for database procedures, just as for other types of queries. The following example executes a procedure that has no input parameters and returns an integer value:

```
SQLINTEGER retval = 500;
SQLINTEGER orind = 0;

SQLBindParameter( hstmt, /* Statement handle */
                  1,      /* Parameter number */
                  SQL_PARAM_OUTPUT, /* It's an output parameter */
                  SQL_C_LONG,      /* Source data is an integer */
                  SQL_INTEGER,      /* Target column is an integer */
                  0,                /* Length not required */
                  0,                /* Precision not required */
                  &retval,          /* The data itself */
                  0,                /* Max length not required */
                  &orind1);         /* Indicator can be zero */

SQLExecDirect( hstmt, "{ ? = call myDbProc () }", SQL_NTS );
```

The value returned from the procedure "myDbProc" is returned in the integer "retval" after the procedure is executed. Note that the third argument, ParameterType, is designated as SQL_PARAM_OUTPUT.

Database Procedures with Input Parameters

Input parameters are sent to the database procedure but not returned to the application. The following example shows how input parameters are used:

```
SQLINTEGER retval = 500;
SQLINTEGER orind = 0;

SQLBindParameter( hstmt, /* Statement handle */
                  1,      /* Parameter number */
                  SQL_PARAM_INPUT, /* It's an input parameter */
                  SQL_C_LONG,      /* Source data is an integer */
                  SQL_INTEGER,      /* Target column is an integer */
                  0,                /* Length not required */
                  0,                /* Precision not required */
                  &inputVal,        /* The data itself */
                  0,                /* Max length not required */
                  &orind1);         /* Indicator can be zero */

SQLExecDirect( hstmt, "{ call myDbProc ( ? ) }", SQL_NTS );
```

Note that the ParameterType argument for "inputVal" is now SQL_PARAM_INPUT. The parameter marker "?" is now designated as an input parameter to myDbProc by placing it within the parentheses after myDbProc.

Database Procedures with BYREF Parameters

BYREF parameters can be used for both input and output. The following example is almost the same as the Input Parameters example, but with one exception:

```
SQLBindParameter( hstmt,      /* Statement handle */
                  1,          /* Parameter number */
                  SQL_PARAM_INPUT_OUTPUT, /* It's an BYREF parameter */
                  SQL_C_LONG, /* Source data is an integer */
                  SQL_INTEGER, /* Target column is an integer */
                  0,          /* Length not required */
                  0,          /* Precision not required */
                  &byRefval,  /* The data itself */
                  0,          /* Max length not required */
                  &orind1);   /* Indicator can be zero */

SQLExecDirect( hstmt, "{ call myDbProc ( ? ) }", SQL_NTS );
```

Since this procedure handles BYREF parameters, the call to `SQLExecDirect()` may begin with a value of 500 for the "byRef" variable, but return with any valid integer value, such as -1.

Database Procedures that Return Rows

No special parameter treatment is required for database procedures that return rows. `SQLBindParameter()` can be used for return values, input parameters, and BYREF parameters as before, regardless of whether rows are to be returned.

The following example shows a procedure that returns rows but has no input parameters:

```
/* Create the row-returning procedure. */
SQLExecDirect( hstmt, "create procedure retRow result row " \
    "( varchar(20) ) as declare pmodel = varchar(20) not null; " \
    "begin for select model into pmodel from cars do " \
    "return row( pmodel ); endfor; end" ), SQL_NTS );

/* Execute the procedure. */
SQLExecDirect( hstmt, "{ call retRow () }", SQL_NTS );

/* Fetch the result data. */
SQLFetch( hstmt );
```

Fetch Data

As with dynamic parameters, ODBC applications must bind the fetched data to variables in the ODBC application. Data can be fetched one row at a time, or the application can declare an array to serve as a record set.

It is not necessary for the type of variable to be similar to the type of column in the DBMS. For instance, one can read a table containing an integer and bind it to a character string. The ODBC performs the conversion internally.

SQLFetch()—Fetch Single Rows

SQLFetch() fetches a single row of data after a select query has been executed. As with all ODBC functions, SQLFetch() returns a status that can be analyzed to determine whether the end of the result set has been reached.

The following example shows how SQLFetch() is used:

```
RETCODE rc = SQL_SUCCESS;

SQLExecDirect( hstmt, "select models from cars", SQL_NTS );

while ( TRUE )
{
    rc = SQLFetch( hstmt );
    if (rc == SQL_NO_DATA_FOUND)
    {
        printf("End of data.\n" );
        break;
    }
    if ( !SQL_SUCCEEDED( rc ) )
    {
        printf("Error! status is %d\n", rc );
        break;
    }
}
```

SQLGetData() and SQLBindCol()—Bind Fetched Data

The previous SQLFetch() example fetches rows from the database, but does not make the data available to the application. The functions SQLGetData() and SQLBindCol() bind the fetched data to variables in the application. SQLGetData() binds the variables after the fetch; SQLBindCol() binds the variables before the fetch. SQLGetData() and SQLBindCol() can be used separately or together, although it is somewhat redundant to do both.

The following example expands the SQLFetch() example to show the use of SQLGetData() and SQLBindCol():

```
RETCODE rc = SQL_SUCCESS;
SQLCHAR model[21] = "\0";
SQLINTEGER orind = SQL_NTS;
SQLINTEGER orind1 = SQL_NTS;

/*
** Execute the select query.
*/
SQLExecDirect( hstmt, "select model from cars", SQL_NTS );

/*
** Bind the column to be fetched.
*/
SQLBindCol( hstmt,      /* Statement handle */
            1,          /* Column Number */
            SQL_C_CHAR, /* C type of variable */
            model,       /* The fetched data */
            20,         /* Maximum length */
            &orind );    /* Status or length indicator */

/*
** Fetch the data in a loop.
*/
while ( TRUE )
{
    rc = SQLFetch( hstmt );

    /*
    ** Break out of the loop if end-of-data is reached.
    */
    if (rc == SQL_NO_DATA_FOUND)
    {
        printf("End of data.\n" );
        break;
    }
    /*
    ** Break out of the loop if an error is found.
    */
    if ( !SQL_SUCCEEDED( rc ) )
    {
        printf("Error! status is %d\n", rc );
        break;
    }
    /*
    ** Re-bind the data to be fetched (redundant in this
```

```
    ** case).
    */
    SQLGetData( hstmt, /* Statement handle */
                1,      /* Column number */
                SQL_C_CHAR, /* C type of variable */
                model,    /* The fetched data */
                20,      /* Maximum length */
                &orind1 ); /* Status or length indicator */

    printf("model of car: %s\n", model );
}
```

The above example works, but would work just as well if only `SQLBindCol()` or `SQLGetData()` were called. Both are included in the example to show how they are used. Note that `SQLBindCol()` is called only once and serves to bind data for all successive fetches. `SQLGetData()` is called after every fetch. `SQLGetData()` is called in this way to fetch variable-length data, such as large objects.

SQLRecordScroll()—Fetch Record Sets

SQLFetchScroll() returns record sets, or blocks of data. By itself, SQLFetchScroll() does not provide enough information to describe the characteristics of the record set; instead, a series of calls to SQLSetStmtAttr (set query attributes) define the record set characteristics.

The following example shows how the cars table is fetched into a record set containing five rows:

```
#define ROWS 5
#define MODEL_LEN 21

SQLCHAR      model[ROWS][MODEL_LEN]; /* Record set */
SQLINTEGER   orind[ROWS];             /* Len or status ind */
SQLUSMALLINT rowStatus[ROWS];         /* Status of each row */
RETCODE      rc=SQL_SUCCESS;          /* Status return code */
int i;      /* Loop counter */
SQLHSTMT     hstmt;                   /* Statement handle */
SQLUINTEGER  numRowsFetched;          /* Number of rows fetched */

/*
** Declare that the record set is organized according to columns.
*/
SQLSetStmtAttr( hstmt, SQL_ATTR_ROW_BIND_TYPE,
                SQL_BIND_BY_COLUMN, 0 );
/*
** Declare that the record set has five rows.
*/
SQLSetStmtAttr( hstmt, SQL_ATTR_ROW_ARRAY_SIZE,
                (SQLPOINTER)ROWS, 0 );
/*
** Bind an array of status pointers to report on the status of
** each row fetched.
*/
SQLSetStmtAttr( hstmt, SQL_ATTR_ROW_STATUS_PTR,
                (SQLPOINTER)rowStatus, 0 );
/*
** Bind an integer that reports the number of rows fetched.
*/
SQLSetStmtAttr( hstmt, SQL_ATTR_ROWS_FETCHED_PTR,
                (SQLPOINTER)&numRowsFetched, 0 );
/*
** Bind the array describing the column fetched.
*/
SQLBindCol( hstmt, /* Statement handle */
            1,      /* Column number */
            SQL_C_CHAR, /* Bind to a C string */
            model,   /* The data to be fetched */
            MODEL_LEN, /* Maximum length of the data */
            orind ); /* Status or length indicator */
/*
** Execute the select query.
*/
SQLExecDirect(hstmt, "SELECT model from cars", SQL_NTS);
/*
** Fetch the data in a loop.
```



```
*/
while ( TRUE )
{
    rc = SQLFetchScroll( hstmt, SQL_FETCH_NEXT, 0 );
    /*
    ** Break out of the loop at end of data.
    */
    if (rc == SQL_NO_DATA_FOUND)
    {
        printf("End of record set\n" );
        break;
    }
    /*
    ** Break out of the loop if an error is found.
    */
    if ( !SQL_SUCCEEDED( rc ) )
    {
        printf( "Error on SQLFetchScroll(), status is %d\n", rc );
        break;
    }
    /*
    ** Display the result set.
    */
    for (i = 0; i < numrowsfetched; i++)
    {
        printf("Model: %s\n", model[i]);
    }
} /* end while */
```

Column-wise versus Row-wise Binding

The previous `SQLRecordScroll()` example depicts *column-wise* binding, which is the default. In column-wise binding, the variable arrays describe the columns of data to be fetched.

It is also possible to set up structures in your program that describe the rows to be fetched, rather than the columns. This is called *row-wise* binding.

The following program excerpt fetches exactly the same data as the `SQLRecordScroll()` example, but uses row-wise binding instead of column-wise binding. The structure typedef `MODEL_ROW` can be considered a snapshot of information about each row. Each column in the row structure consists of the data to be fetched and a row status indicator.

```
#define ROWS 5
#define MODEL_LEN 21

/*
** Describe a row in the result set.
*/
typedef struct
{
    SQLCHAR      model[MODEL_LEN]; /* The data to be fetched */
    SQLINTEGER   orind;             /* Len or status indicator */
} MODEL_ROW;

MODEL_ROW      model_row[ROWS]; /* The record set */
SQLUSMALLINT   rowStatus[ROWS]; /* Status of each row */
SQLHSTMT       hstmt;           /* Statement handle */
RETCODE        rc=SQL_SUCCESS;  /* Return status */
int            i;                /* Loop counter */
SQLINTEGER      numRowsFetched; /* Number of rows fetched */

/*
** Declare that the record set is organized according to rows.
*/
SQLSetStmtAttr( hstmt, SQL_ATTR_ROW_BIND_TYPE,
                (SQLPOINTER)sizeof( MODEL_ROW ), 0 );
/*
** Declare the number of rows in the result set.
*/
SQLSetStmtAttr( hstmt, SQL_ATTR_ROW_ARRAY_SIZE,
                (SQLPOINTER)ROWS, 0 );
/*
** Bind to a status array reporting on each row fetched.
*/
SQLSetStmtAttr( hstmt, SQL_ATTR_ROW_STATUS_PTR,
                (SQLPOINTER)rowStatus, 0 );
/*
** Bind to an integer reporting on the number of rows fetched.
*/
SQLSetStmtAttr( hstmt, SQL_ATTR_ROWS_FETCHED_PTR,
                (SQLPOINTER)&numRowsFetched, 0 );
/*
** Execute the select statement.
```

```

*/
SQLExecDirect( hstmt, "SELECT model from cars", SQL_NTS );

/*
** Bind each column in the record set structure.
*/
SQLBindCol( hstmt,                /* Statement handle */
            1,                    /* Column number */
            SQL_C_CHAR,           /* Bind to C string */
            &model_row[0].model,  /* Column to fetch */
            sizeof( model_row[0].model ), /* Length of data */
            &model_row[0].orind ); /* Len or status indicator */
/*
** Fetch the data in a loop.
*/
while ( TRUE )
{
    rc = SQLFetchScroll( hstmt, SQL_FETCH_NEXT, 0 ) )
    /*
    ** Break out of the loop at end-of-data.
    */
    if ( rc == SQL_NO_DATA_FOUND )
        break;

    /*
    ** Break out of the loop if an error is found.
    */
    if ( !SQL_SUCCEEDED( rc ) )
    {
        printf( "Error on SQLFetchScroll(), status is %d\n", rc );
        break;
    }
    /*
    ** Display the result set.
    */
    for ( i = 0; i < numRowsFetched; i++)
    {
        printf("Model: %s\n", model_row[i].model);
    }
} /* end while */

```

SQLSetCursorName()—Declare Cursor

The term *cursor* is an acronym for CURrent Set Of Records. A database cursor is similar to the cursor on your computer screen. However, instead of pointing at something on your screen, a database cursor points to a data row set.

Some database vendors make a distinction between client and server-side cursors. However, a cursor declared in an Ingres ODBC program is always a server-side cursor. This means that the properties of the cursor are applied only on the DBMS server of the target database.

An ODBC application can name a cursor directly via a call to `SQLSetCursorName()`:

```
SQLSetCursorName( hstmt, /* Statement handle */
                  "C1",    /* Cursor Name */
                  SQL_NTS ); /* This is a null-terminated string */
```

The above code creates a cursor named C1, which is also visible to the DBMS as C1.

Updatable Cursors

In order for a cursor to be made updatable, the Ingres ODBC driver imposes a set of syntax rules:

- The cursor must be explicitly named via `SQLSetCursorName()`.
- `SQLSetStmtAttr()` must be invoked with `SQL_ATTR_CONCURRENCY` specified as `SQL_CONCUR_VALUES`.
- The update statement must include the "where current of" clause and refer to the cursor name declared in `SQLSetCursorName()`.

The following code highlights the minimum code required to declare an updatable cursor:

```
SQLSetCursorName( hstmtS, /* Select statement handle */
                  "C1",    /* Cursor Name */
                  SQL_NTS ); /* This is a null-terminated string */

SQLSetStmtAttr( hstmtS, SQL_ATTR_CONCURRENCY,
                (SQLPOINTER)SQL_CONCUR_VALUES, 0 );

SQLExecDirect ( hstmtS,
                "select model from cars where model = 'Hummer '",
                SQL_NTS );

SQLExecDirect( hstmtU,
                "UPDATE cars SET model = 'HumV ' WHERE CURRENT OF C1",
                SQL_NTS );
```

Cursors versus Select Loops

A *loop* is an iterative set of fetches. Thus, a *cursor loop* is a set of fetches using cursors. *Select loops* are a set of fetches without a cursor defined. The ODBC uses select loops by default. This is true whether or not an ODBC DSN definition is specified.

Declaration of a cursor name is the same as a cursor loop in this discussion.

Select loops fetch multiple sets of rows from the DBMS. This is sometimes referred to as block fetching. A single fetch may appear to return only one row, but often the ODBC driver has already fetched many more rows that are cached in the driver.

Cursor loops must be specified if the cursor is scrollable or updatable.

Note: Cursors loops may need to be specified for Windows applications such as Microsoft Access or Microsoft ADO. If you are see an error message such as "API function cannot be called in the current state", and are satisfied that your application is coded correctly, try using cursor loops.

Cursor loops may offer better performance for Windows applications, because the ODBC driver returns information that it supports unlimited active statements. This signifies, for example, that ADO applications can re-use existing connections for internal procedures.

Outside of Windows applications, the performance of cursor loops is often comparable to select loops, because the ODBC driver pre-fetches rows in blocks of 100 when cursors are used. The term *pre-fetch* means that multiple rows are fetched and cached in the ODBC driver before they are presented to the application.

If a cursor is declared as updatable, pre-fetching does not occur in order to preserve the current position for the update. Thus, updatable cursors may be slower than read-only cursors or select loops.

Only one select loop can be active at a time. As a result, select loops may not be nested. For example, in ADO, multiple recordset objects may not be retrieved within [Connection].BeginTrans and [Connection].CommitTrans methods. In direct ODBC code, SQLFreeStmt() must be called with the argument SQL_CLOSE before executing another select loop. By contrast, cursors place no limits on the number of active result sets. Cursor loops may be nested.

See the *SQL Reference Guide* for more information on cursors versus select loops.

SQLFreeStmt()—Close Fetch Loop

After the ODBC application has finished fetching, the cursor associated with the statement handle must be closed. If a cursor was not declared, the ODBC application still must tell the DBMS it has completed fetching.

A cursor is closed, or a select loop completes processing, via the SQLFreeStmt() function, as shown in this example:

```
/* Stop fetching data */  
SQLFreeStmt( hstmt, SQL_CLOSE );
```

SQLFreeStmt() may be called at any time during a fetch operation. When the SQL_CLOSE argument is specified, all resources associated with the fetch are released. The statement handle can be re-used for other types of queries without having to call SQLAllocStmt() or SQLAllocHandle().

Scrollable Cursors

The ODBC driver supports scrollable cursors through SQLFetchScroll() and SQLSetPos().

The driver supports static (read-only) and keyset-driven (updatable) cursor types. These cursor types allow the cursor to be positioned in any direction within a result set.

Static and keyset-driven cursors support the position directives described in SQLFetchScroll (see page 192). In contrast, forward-only cursors support only SQL_FETCH_NEXT.

Note: Static and keyset-driven cursors can be used only if the target database is Ingres 9.2 and later. For Ingres databases prior to 9.2, the Cursor Library can be used to simulate these types of cursors.

Static Scrollable Cursors

Static cursors fetch rows as they are materialized from the current transaction isolation level. The result set is not updated if other sessions change data that applies to the result set. Static cursors are read-only.

Keyset-driven Scrollable Cursors

Keyset-driven cursors allow updates to selected records in the result set. Keyset-driven cursors require the target tables to include unique primary keys. If an attempt is made to update or delete records in a result set, and the corresponding records in the target table have been deleted, an error may be returned, depending on the transaction isolation level.

Keyset-driven cursors can be used within a read-only context, but do not perform as well as static cursors.

Scrollable Cursor Programming Considerations

The ODBC DSN definition or connection string must specify cursor loops, or name a cursor using the `SQLSetCursorName()` function. Select loops cannot be used with static or keyset-driven cursors.

Keyset-driven cursors require the cursor to be named. The cursor name must be included in the `WHERE CURRENT OF` clause in the update or delete query.

The `SQLFetchScroll()` function fetches records in the result set according to the directive specified in the `FetchOrientation` argument.

Cursor types can be specified in the `SQLSetStmtAttr()` function and queried by the `SQLGetStmtAttr()` function.

Use the `SQLSetConnectvAttr()` function to specify that the Ingres ODBC driver is to be used for scrollable cursor functions.

SQLFetchScroll()—Fetch from a Scrollable Cursor

SQLFetchScroll() positions the cursor according to the specified fetch orientation and then retrieves data.

SQLFetchScroll has the following syntax:

```
SQLFetchScroll( StatementHandle, FetchOrientation, FetchOffset )
```

where:

FetchOrientation

Specifies the fetch orientation as one of the following:

SQL_FETCH_NEXT

Fetch the next record in the result set

SQL_FETCH_FIRST

Fetch the first record in the result set

SQL_FETCH_LAST

Fetch the last record in the result set

SQL_FETCH_PRIOR

Fetch the previous record in the result set

SQL_FETCH_ABSOLUTE

Fetch a record based on the position in the result set

SQL_FETCH_RELATIVE

Fetch relative to n rows from the current position in the result set

SQLSetPos()—Scroll Cursor to Absolute Position

SQLSetPos() allows the ODBC application to scroll the cursor to an absolute position within the result set and perform updates or deletes on the selected record.

Note: The SQLSetPos() function works for keyset-driven (updatable) cursors only.

SQLSetPos() has the following syntax:

```
SQLSetPos( StatementHandle, RowNumber, Operation, LockType )
```

where:

StatementHandle

Specifies the statement handle

RowNumber

Specifies the position of the record in the result set

Operation

Specifies the operation to perform: SQL_POSITION, SQL_UPDATE, or SQL_DELETE.

SQL_REFRESH is not supported.

LockType

(Not supported) Specifies the type of table lock. SQLSetPos() ignores all settings for the LockType argument.

Static Scrollable Cursor Example

The following code demonstrates the use of static scrollable cursors:

```
/*
** Specify that the Ingres ODBC driver is used.
*/
SQLSetConnectAttr(hdbc, SQL_ATTR_ODBC_CURSORS,
    SQL_CUR_USE_DRIVER, SQL_IS_INTEGER );

/* Set the cursor name. */
SQLSetCursorName(hstmt, "C1", SQL_NTS);

/* Set the number of rows in the rowset */
SQLSetStmtAttr( hstmt,
    SQL_ATTR_ROW_ARRAY_SIZE,
    (SQLPOINTER) ROWSET_SIZE,
    0);

/* Set the cursor type */
SQLSetStmtAttr( hstmt,
    SQL_ATTR_CURSOR_TYPE,
    (SQLPOINTER) SQL_CURSOR_STATIC,
    0);

/* Set the pointer to the variable numrowsfetched: */
SQLSetStmtAttr( hstmt,
    SQL_ATTR_ROWS_FETCHED_PTR,
    &numrowsfetched,
    0);

/* Set pointer to the row status array */
SQLSetStmtAttr( hstmt,
    SQL_ATTR_ROW_STATUS_PTR,
    (SQLPOINTER) rowStatus,
    0);

/* Execute the select query. */
strcpy((char *)sqlstmt, "SELECT y,x FROM myTable");
SQLExecDirect(hstmt, sqlstmt, SQL_NTS);

/* Fetch last full result set. */
SQLFetchScroll(hstmt, SQL_FETCH_LAST, 0);

/* Fetch first result set. */
SQLFetchScroll(hstmt, SQL_FETCH_FIRST, 0);

/* Fetch next row. */
SQLFetchScroll(hstmt, SQL_FETCH_NEXT, 0);

/* Fetch the result set starting from the third row. */
SQLFetchScroll(hstmt, SQL_FETCH_ABSOLUTE, 3);

/* Fetch the result set starting after moving up one row */
SQLFetchScroll(hstmt, SQL_FETCH_RELATIVE, -1);
```

Keyset-driven Scrollable Cursor Example

The following code demonstrates keyset-driven cursors:

```
/* Set the cursor name. */
SQLSetCursorName(hstmt, "CUPD1", SQL_NTS);

/* Set the cursor type */
SQLSetStmtAttr( hstmt,
    SQL_ATTR_CURSOR_TYPE,
    (SQLPOINTER) SQL_CURSOR_KEYSET,
    0);
/* Execute select query */
SQLExecDirect(hstmtS, "SELECT x, y FROM keyset_cursor", SQL_NTS);

/* Fetch scrollable cursor */
SQLFetchScroll(hstmtS, SQL_FETCH_NEXT, 0)) != SQL_ERROR)

/* Move cursor to record 4 */
SQLSetPos(hstmtS, 4, SQL_POSITION, SQL_LOCK_NO_CHANGE);

/* Bind a string parameter */
rc = SQLBindParameter(hstmtU, 1, SQL_PARAM_INPUT,
    SQL_C_CHAR, SQL_CHAR,
    TXT_LEN, 0, x[irow-1], 0, NULL);

/* Update the record */
SQLExecDirect(hstmtU,
    "UPDATE keyset_cursor SET x=? WHERE CURRENT OF CUPD", SQL_NTS);
```

Large Objects (Blobs) Support

The `SQLGetData()` and `SQLPutData()` functions allow fetching or insertion of data in segments. Any non-atomic data type, such as char or byte varying, can be sent or fetched in segments, but generally segments are used with large objects, sometimes known as "blobs".

The Ingres ODBC driver supports the following large objects:

- Long varchar (`SQL_LONGVARCHAR`)
- Long byte (`SQL_LONGVARBYTE`)
- Long nvarchar (`SQL_WLONGVARCHAR`)

Although large object locators are supported in Ingres 9.1 and later, the Ingres ODBC driver does not support large object locators, since there is no corresponding support in the ODBC specification.

SQLPutData()—Send Data in Segments

SQLPutData() sends blob data in segments. The length of the data must be known in advance, and applied to the SQL_LEN_DATA_AT_EXEC macro prior to execution of an insert or update query.

The following example sends a blob 5,000 characters in length to a table containing a long varchar. Each segment is 1,000 characters long.

Example: SQLPutData() Function

```
SQLCHAR    b[5000], *b = &b[0]; /* The blob to be sent */
int         i;                    /* Loop counter */
SQLINTEGER  len;                  /* Segment length */
RETCODE     rc = SQL_SUCCESS;    /* Return code */
SQLPOINTER  pToken = NULL;       /* Column indicator */
SQLHSTMT    hstmt;               /* Statement handle */

/*
** Fill the blob buffer with test data.
*/
for ( i = 0; i < 5000; i++ )
    b[i] = 'x ';

/*
** Bind the blob, indicating that the length will be
** provided at runtime.
*/
SQLBindParameter(hstmt, /* Statement handle */
    1,                  /* Column number */
    SQL_PARAM_INPUT,   /* This is an input parameter */
    SQL_C_CHAR,        /* Parameter is a string */
    SQL_LONGVARBINARY, /* Destination type is long varchar */
    0,                  /* No length required */
    0,                  /* No precision required */
    (PTR)1,            /* Long parameter number 1 */
    0,                  /* No max length required */
    &len );             /* Variable blob length */

/*
```

```

** This macro definition tells the ODBC when to expect the end of
** data for the blob.
*/
len = SQL_LEN_DATA_AT_EXEC( 5000 );

/*
** Execute the insert query.
*/
rc = SQLExecDirect( hstmt,
    "insert into longv values ( ? )", SQL_NTS );
/*
** Loop, sending the data in segments of 1000.
*/
while ( rc == SQL_NEED_DATA )
{
    /*
    ** Check for EOD marker.
    */
    rc = SQLParamData( hstmt, pToken );
    if ( rc == SQL_NEED_DATA )
    {
        /*
        ** If more data to send, send it.
        */
        if ( pToken == 1 )
        {
            SQLPutData( hstmt, blob, 1000 );
            blob += 1000;
        }
    }
}
}
```

Note that `SQLBindParameter()` never directly references the data buffer "blob". Instead, the number 1 is used to indicate that parameter 1 is a variable length parameter. The overall length of the blob is calculated as a negative number into the `len` argument via the `SQL_LEN_DATA_AT_EXEC()` macro. The ODBC driver uses these clues to detect that data is to be sent in segments.

The `SQLParamData()` function tracks the progress of `SQLPutData()`. If the status return of `SQLParamData()` is `SQL_NEED_DATA`, the ODBC driver has not yet finished sending data segments. The `pToken` argument returns the parameter number of the data to be sent; thus, more than one blob or variable-length datum may be sent in segments.

The above example could have been coded in the traditional way, as shown in this example:

```
/*
** Set the maximum length of the blob.
*/
len = 5000;
/*
** Bind in the traditional way.
*/
SQLBindParameter(hstmt,      /* Statement handle */
    1,                      /* Column number */
    SQL_PARAM_INPUT,        /* This is an input parameter */
    SQL_C_CHAR,             /* Parameter is a string */
    SQL_LONGVARIABLE,       /* Destination type is long varchar */
    5000,                   /* Length of the data */
    0,                      /* No precision required */
    (PTR)blob,              /* The data itself */
    0,                      /* No max length required */
    &len );                 /* Variable blob length */

/*
** Only one call to SQLExecDirect() is required.
*/
SQLExecDirect( hstmt,
    "insert into longv values ( ? )", SQL_NTS );
```

This traditional approach may be acceptable for smaller-length blobs, but would be less practical for large blobs, especially those that consume substantial memory. The ODBC application would need to pre-allocate memory for the entire blob before using it as a query parameter. By contrast, if blobs are sent in segments, only memory for the blob segment needs to be allocated.

SQLGetData()—Fetch Data in Segments

Fetching blob segments using SQLGetData() is more straightforward than SQLPutData(). If the previous SQLPutData() example were expanded to fetch the data after insertion, the following code would fetch the 5,000-character blob in 1,000-character segments:

Example: SQLGetData() Function

```
RETCODE ret = SQL_SUCCESS;

/*
** Execute the fetch query.
*/
SQLExecDirect ( hstmt, "select * from longv", SQL_NTS );

/*
** Fetch in a loop.
*/
while ( TRUE )
{
    rc = SQLFetch ( hstmt );
    if ( rc == SQL_NO_DATA )
    {
        printf( "EOD\n" );
        break;
    }
    /*
    ** Exit the loop if an error is found.
    */
    if ( !SQL_SUCCEEDED ( rc ) )
    {
        printf("Error fetching from blob table\n" );
        break;
    }
    len = 0;
    blob = &b[0];
    *blob = '\0';
    while ( TRUE )
```

```
{
    /*
    ** Get the data in segments until
    ** the status is SQL_SUCCESS.
    */
    ret = SQLGetData( hstmt,
                     1,
                     SQL_C_CHAR,
                     1000,
                     blob,
                     &len );

    /*
    ** A status value of SQL_SUCCESS means we're done.
    ** Exit the loop.
    */
    if ( ret == SQL_SUCCESS )
        break;

    if ( ret == SQL_ERROR )
    {
        printf ("Error fetching blob segments!\n" );
        break;
    }
    /*
    ** Increment the pointer to the blob for each successful
    ** segment fetch.
    */
    blob += 1000;
}
}
```

Often, ODBC functions can share status code variables, but in this case, a second status code, "ret" must be declared in addition to "rc". This is because segment fetching uses two fetch loops: one for the fetch itself, and one for retrieving the segments. Each loop needs to track its own status.

There is no counterpart to SQLParamData() when fetching in segments. Instead, SQLGetData() returns a status of SQL_SUCCESS_WITH_INFO when there are more segments to be fetched. If the status of SQLGetData() were further analyzed, the SQLSTATE would have the value 01004 (data truncated). See Error Reporting in the next section for more information on SQLGetData().

The last argument to SQLGetData(), represented by the len variable, indicates the length of the data available in the ODBC driver cache. For the Ingres ODBC driver, it is normal for this argument to contain a larger value than the segment length until the last segment is fetched.

When all of the blob segments are fetched, SQLGetData() returns a status of SQL_SUCCESS.

As with `SQLPutData()`, the Ingres ODBC driver supports the traditional use of `SQLGetData()`. The ODBC application could make a single call to `SQLGetData()` specifying the entire length of the blob:

```
/*
** Fetch in a loop.
*/
while ( TRUE )
{
    rc = SQLFetch ( hstmt );
    /*
    ** Exit the loop at EOD.
    */
    if ( rc == SQL_NO_DATA )
    {
        printf( "EOD\n" );
        break;
    }
    /*
    ** Exit the loop if an error is found.
    */
    if ( !SQL_SUCCEEDED ( rc ) )
    {
        printf("Error fetching from blob table\n" );
        break;
    }
    len = 5000;
    blob = &b[0];
    *blob = '\0';
    /*
    ** Just one call to SQLGetData() is all that is required.
    */
    ret = SQLGetData( hstmt,          /* Statement handle */
                     1,               /* Column number */
                     SQL_C_CHAR,      /* It's a string */
                     5000,            /* Max length */
                     blob,             /* Buffer to fetch into */
                     &len );          /* Length indicator */
    /*
    ** Exit the loop if the data cannot be converted to the blob
    ** buffer.
    */
    if ( ret != SQL_SUCCESS )
    {
        printf ("Error entire fetching blob data!\n" );
        break;
    }
}
```

The ODBC application would need to know that the blob was at least 5,000 characters or less in order for the above example to work; otherwise, the application would truncate the data. Furthermore, the data buffer must be pre-allocated to 5,000 characters. Therefore, this approach may be inefficient for fetching very large blobs.

Transactions Handling

This section explores how the ODBC Driver and Ingres DBMS handle transactions, and describes ODBC support for data types.

SQLSetConnectAttr()—Enable Autocommit

The Ingres DBMS supports *standard* transaction sessions. Standard transaction sessions begin a transaction when the first query is issued, and end when a commit or rollback command is executed. By contrast, autocommit sessions commit each insert, delete or update query in the DBMS. Standard transaction sessions delete prepared statements and cursor declarations after a commit or rollback; autocommit sessions retain prepared statements and cursor declarations.

The SQLSetConnectAttr() function enables or disables autocommit. The ODBC driver default is to enable autocommit. The following example disables autocommit and manages the transaction manually.

Example: SQLSetConnectAttr() Function

```
SQLHDBC hdbc;           /* Connection handle */

/*
** Turn off autocommit.
*/
SQLSetConnectAttr( hdbc, /* Connection Handle */
                  SQL_ATTR_AUTOCOMMIT, /* Autocommit attribute */
                  SQL_AUTOCOMMIT_OFF, /* Autocommit disabled */
                  0 ); /* String length (n/a) */
```

The function SQLEndTran() commits or rolls back the transaction:

```
/*
** Roll back the current transaction.
*/
SQLEndTran(SQL_HANDLE_DBC, /* Handle type */
           hdbc,           /* Connection handle */
           SQL_ROLLBACK); /* Roll back the transaction */
```

Simulated Autocommit for Cursors

The Ingres DBMS places a restriction on cursor declarations during autocommit in that multiple cursors may not be declared. However, multiple cursors may be declared for standard transaction sessions.

The Ingres restriction has ramifications on the ODBC. The ODBC specification allows multiple cursor declarations regardless of whether autocommit is enabled or not.

In order for the Ingres ODBC driver to support multiple cursors during autocommit, the driver internally reverts to a state named *simulated* autocommit. During simulated autocommit, when the ODBC driver detects that a cursor is opened, and an update, insert or delete query is to be executed, the ODBC driver internally disables autocommit. Commits are issued internally from the ODBC driver when:

- The statement or connection handle is freed.
- A cursor is closed, and no other cursors are open.

When all cursors are closed, the ODBC driver changes back to autocommit mode.

Simulated autocommit is not used if no cursors are open or if the only DBMS queries are fetch queries.

SQLSetStmtAttr()—Set Transaction Isolation Level

The Ingres ODBC driver supports all transaction isolation levels available in the ODBC specification, including:

- Serializable
- Read committed
- Read uncommitted
- Repeatable read

The above isolation levels are specified by the ODBC attributes `SQL_ATTR_TXN_SERIALIZABLE`, `SQL_ATTR_TXN_READ_COMMITTED`, `SQL_TXN_READ_UNCOMMITTED`, and `SQL_TXN_REPEATABLE_READ`, respectively. Transaction isolation is specified by `SQLSetStmtAttr()` function via the `SQL_ATTR_TXN_ISOLATION` connection attribute. The `SQLGetStmtAttr()` function returns the current isolation level when the `SQL_ATTR_TXN_ISOLATION` option is specified.

Other types of transaction isolation supported by Ingres, such as system, are not available in `SQLSetStmtAttr()`. Use `SQLExecDirect()` to execute the SET command directly in such cases. This is also the case when locking is specified with the SET LOCKMODE command, such as for row-level locking.

Distributed (XA) Transactions

The ODBC driver supports distributed (XA) transactions in the Windows environment using the Microsoft Distributed Transaction Coordinator. See Ingres ODBC and Distributed Transactions (Windows) in this guide for more information.

Supported Data Types

The Ingres ODBC driver supports all ODBC data types except:

- SQL_GUID and SQL_C_GUID
- SQL_BOOKMARK and SQL_C_BOOKMARK
- SQL_VARBOOKMARK and SQL_C_VARBOOKMARK

The ODBC function SQLBindParameter() allows coercion from "C" data types to SQL data types. Outside of the above exceptions, the ODBC supports all ODBC data type coercions as described in the "Converting Data from SQL to C Data Types" and "Converting Data from C to SQL Data Types" tables in the Microsoft *ODBC Programmer's Reference*. The following table summarizes the coercions available:

Data Type	SQL Type
String	All types
Binary	All types
Numeric	All numeric
Timestamp, date and time	Timestamp, date and time
Interval	Interval

Date/Time Columns and Values

Prior to Ingres 9.1, the Ingres ODBC driver supported SQL_C_TYPE_DATE, SQL_C_TYPE_TIME, and SQL_C_TYPE_TIMESTAMP for the Ingres "date" data type. In Ingres 9.1 and later, support was added for ISO date/time data types, including:

- Time with local time zone
- Time with time zone
- Time without time zone
- Timestamp with local time zone
- Timestamp with time zone
- Timestamp without time zone
- Ansidate (also known as "ISO" date)
- Ingresdate (formerly known as "date")
- Year to month interval
- Day to second interval

The Ingres ODBC driver supports all ISO data types in addition to the legacy "ingresdate" type. The ODBC driver is sensitive to the connection level of the target database, and thus can work seamlessly against pre-Ingres 9.1 installations.

Support for SQL_C_INTERVAL_YEAR_TO_MONTH
SQL_INTERVAL_YEAR_TO_MONTH, SQL_C_INTERVAL_DAY_TO_SECOND,
SQL_INTERVAL_DAY_TO_SECOND is in Ingres 9.1 and later.

The "precision" argument for SQLBindParameter() is supported for SQL_C_TYPE_TIMESTAMP in Ingres 9.1 and later, since ISO timestamps can be declared with a precision for fractions of a second.

If a date/time column is bound to a string type such as SQL_C_CHAR or SQL_C_WCHAR, Ingres rules regarding II_DATE_FORMAT apply, just as if the dates were handled from the Terminal Monitor or other Ingres utility.

Ingres rules on II_DATE_FORMAT do not apply if:

- An ODBC date/time escape sequence is used such as:
 - { t 'hh:mm:ss' }
 - { d 'yyyy:mm:dd' }
 - { ts 'yyyy:mm:dd hh:mm:ss.ffffff' }
 - { interval 'yy-mm' year to month' }
 - { interval 'dd hh-mm-ss' day to second' }
- ODBC date/time structures are used, such as:
 - SQL_TIME_STRUCT
 - SQL_DATE_STRUCT
 - SQL_TIMESTAMP_STRUCT
 - SQL_INTERVAL_STRUCT

ODBC Support for ANSI Syntax

The Ingres ODBC driver supports ISO 8601 syntax in drivers released with Ingres 9.1 and later. ISO syntax is commonly referred to as ANSI syntax.

ANSI syntax is enforced as follows:

- The ODBC timestamp escape sequence { ts 'YYYY-MM-DD HH:MM:SS.[FFFFFFFF]' } is converted to the ANSI string "TIMESTAMP 'YYYY-MM-DD HH:MM:SS.[FFFFFFFF]'".
- The ODBC date escape sequence { d 'YYYY-MM-DD' } is converted to the ANSI string "DATE 'YYYY-MM-DD'".
- The ODBC time escape sequence { t 'HH:MM:SS' } is converted to the ANSI string "TIME 'HH:MM:SS'".
- The ODBC escape sequence { interval 'YY-MM' } is converted to the ANSI string "INTERVAL 'YY-MM' year to month", where YY represents the number of years.
- The ODBC escape sequence { interval 'DD HH:MM:SS.[FFFFFFFF]' } is converted to the ANSI string "INTERVAL 'DD HH:MM:SS.[FFFFFFFF]' day to second", where DD represents the number of days.

Support for Ingres Date Syntax

The Ingres syntax for date/time data types supports many of the format rules in ISO 8601, but extends or departs from the standard in several ways. The INGRESDATE data type is overloaded to represent dates, timestamps, times, and intervals in one type.

The INGRESDATE syntax is enforced as follows:

- The ODBC timestamp escape sequence { ts 'YYYY-MM-DD HH:MM:SS' } is converted to the INGRESDATE string `TIMESTAMP 'YYYY_MM_DD HH:MM:SS'`. Fractions of a second are ignored.
- The ODBC date escape sequence { d 'YYYY-MM-DD' } is converted to the INGRESDATE string `'YYYY_MM_DD 00:00:00'`.
- The ODBC time escape sequence { t 'HH:MM:SS' } is converted to the INGRESDATE string `'YYYY_MM_DD HH:MM:SS'`, where `YYYY_MM_DD` is filled with the current date when inserted into a database, and represents the insertion date when fetched from the database.
- The Ingres ODBC driver supports no Ingres equivalent of interval escape sequences.

The special Ingres syntax for dates is: `YYYY_MM_DD`.

Special Date Values Meaning "TBD"

Null Dates

Databases need to recognize a date value that corresponds to "TBD" or "Unknown". Null dates are suitable for this purpose. Ingres supports null dates for both ANSI date/time types and INGRESDATE types. The treatment in ODBC is the same as for other null data components.

Empty Dates

Ingres supports a specific type of date value that is called an "empty" date. An empty date is similar to an empty string: the contents of an empty date are non-null, but have no data. In practice, empty dates can perform the same function as null dates. Both indicate the absence of a valid date.

Magic Dates

A magic date is a specific date that represents a different value than the date itself. The ODBC uses the magic date 9999-12-31 to represent empty dates, and 9999-12-31 23:59:59 to represent empty timestamps. If Ingres syntax is in effect, any date or timestamp parameter containing the date 9999-12-31 is converted to an empty date. Similarly, if fetching an empty date into SQL_C_TIMESTAMP or SQL_C_DATE, the result is converted to the magic date.

Default Treatment of Date/Time Syntax

If the ODBC driver is connected to a pre-Ingres 9.1 database, including gateways and EDBC, Ingres syntax is applied for date and time values.

If the ODBC driver is connected to an Ingres 9.1 or later database, ANSI syntax is applied for date and time values.

"False" Magic Dates

ODBC does not know whether a given column is an ANSI date/time type or an Ingresdate type. This ambiguity can cause problems.

For example, 9999-12-31 is a valid ANSI date. Therefore, if the ODBC driver and the Ingres database support ANSI syntax, applications that insert dates of 9999-12-31 into ingresdate fields will retrieve 9999-12-31, but the result is not an empty date. Instead, this is the actual date 9999-12-31. This is a "false" empty date.

Note: If your application uses INGRESDATEs, the default ANSI syntax can cause corrupt data, or cause search queries to fail. Specify the Ingres ODBC configuration attribute "Send Date/Time as Ingres Date" or connection string attribute "SendDateTimeAsIngresDate=y" to force the Ingres ODBC driver to use INGRESDATE syntax.

Boolean Columns

The Ingres ODBC driver supports SQL_C_BIT and SQL_BIT for BOOLEAN data types. Use unsigned char (UCHAR) to define Boolean fields. Also acceptable are char, CHAR, or SCHAR.

National Character Set (Unicode) Columns

The Ingres ODBC driver supports SQL_C_WCHAR, SQL_WCHAR, SQL_WVARCHAR and SQL_WLONGVARCHAR data types for the following Ingres data types:

- nchar
- nvarchar
- long nvarchar

For databases that do not support Unicode, the ODBC treats Unicode characters as multi-byte (also known as double-byte) for:

- char
- varchar
- long varchar
- long byte

When the target columns are not Unicode columns, SQL_WCHAR, SQL_WVARCHAR and SQL_WLONGVARCHAR must be bound from SQL_C_CHAR.

Metadata (Catalog) Queries

Sometimes an ODBC application needs to know information about items in the database, such as tables, permissions, primary keys, etc. The Ingres ODBC driver supports all ODBC functions pertaining to this information. The following table summarizes the functions available:

Function	Description
SQLColumns	Column names in tables
SQLColumnPrivileges	Privileges of columns in tables
SQLForeignKeys	Foreign keys for a table
SQLPrimaryKeys	Column names of primary keys in a table
SQLTables	Table names in a database
SQLTablePrivileges	Privileges of tables in a database
SQLProcedures	Procedure names in a database
SQLProcedureColumns	Input and output names of a database procedure
SQLSpecialColumns	Columns that uniquely identify a row
SQLStatistics	Statistics and indexes associates with a table

Error Reporting

All ODBC functions return an error code. Both the Ingres ODBC driver and the Driver Manager cache a list of any errors encountered. The list of errors is deleted when the next ODBC function is called.

A return of SQL_SUCCESS means that the function completed successfully. A return of SQL_ERROR means that an error occurred. A return of SQL_SUCCESS_WITH_INFO can be considered a warning or informational status code. SQL_INVALID_HANDLE means that the handle passed to the function was invalid.

More information on a status of SQL_ERROR or SQL_SUCCESS_WITH_INFO can be retrieved from the SQLGetDiagRec() function. The following example code snippet returns error information on a call to SQLConnect().

Example: SQLGetDiagRec() Function

```
RETCODE      rc = SQL_SUCCESS;
SQLHDBC      hdbc;
SQLCHAR      buffer[SQL_MAX_MESSAGE_LENGTH + 1 ];
SQLCHAR      sqlstate[SQL_SQLSTATE_SIZE + 1 ];
SQLINTEGER   sqlcode;
SQLSMALLINT  length;
SQLSMALLINT  i;

rc = SQLConnect(hdbc,
               "myDSN",
               SQL_NTS,
               "ingres",
               SQL_NTS,
               "ingPWD",
               SQL_NTS );

if (rc == SQL_ERROR)
{
    i = 1;
    while ( SQLGetDiagRec( htype,
                          hndl,
                          i,
                          sqlstate,
                          &sqlcode,
                          buffer,
                          SQL_MAX_MESSAGE_LENGTH + 1,
                          &length ) == SQL_SUCCESS )
    {
        printf( "SQLSTATE: %s\n", sqlstate ) ;
        printf( "Native Error Code: %ld\n", sqlcode ) ;
        printf( "buffer: %s \n", buffer ) ;
        i++ ;
    }
}
```

If ingPWD was an invalid password, the following errors would be displayed from the above code:

```
SQLSTATE: 08004
Native Error Code: 786443
buffer: [Ingres][Ingres 2006 ODBC Driver][Ingres 2006]Login failure: invalid
username/password.
SQLSTATE: 08S01
Native Error Code: 13172737
buffer: [Ingres][Ingres 2006 ODBC Driver][Ingres 2006]The connection to the
server has been aborted.
```

The SQLSTATE reference is specific to ODBC and does not correlate to SQLSTATE in Ingres. The Native Error Code is an Ingres error code, viewable from the errhelp utility:

```
> %II_SYSTEM%\ingres\sig\errhelp\errhelp 786443
(786443)
<12>000b Login failure: invalid username/password.
```

Termination and Clean-up

An ODBC application can simply terminate after executing its queries, but this is considered poor programming practice. The DBMS server cannot distinguish between a program that exited without cleaning up and a program that aborted due to a serious error. As a result, spurious errors are reported in the error log.

To disconnect gracefully from the database, call `SQLDisconnect()`:

```
rc = SQLDisconnect( hdbc ); /* Disconnect */
```

The ODBC function `SQLFreeStmt()` is used to close a cursor or free all resources associated with a statement handle:

```
rc = SQLFreeStmt( hstmt, SQL_CLOSE ); /* Close a cursor or query */
rc = SQLFreeStmt( hstmt, SQL_DROP ); /* Free all resources */
```

A call to `SQLFreeStmt()` with an argument of `SQL_DROP` implicitly closes the cursor before freeing resources.

The generic `SQLFreeHandle()` function can be used with all types of handles:

```
rc = SQLFreeHandle ( SQL_HANDLE_STMT, hstmt );
rc = SQLFreeHandle ( SQL_HANDLE_DBC, hdbc );
rc = SQLFreeEnv ( SQL_HANDLE_ENV, henv );
```

A call to `SQLFreeHandle()` with a handle type argument of `SQL_HANDLE_STMT` is the equivalent of calling `SQLFreeStmt()` with an argument of `SQL_DROP`.

Once a handle has been freed, the corresponding allocate function must be re-invoked to initialize resources associated with the handle.

ODBC CLI Connection Pooling

ODBC connection pooling is a method of sharing active database connections with similar or identical connection characteristics. When a connection is released through a call to `SQLDisconnect()`, the connection is left open and added to a pool of active connections.

When an ODBC application opens a new connection, the application searches the pool for a connection with matching connection characteristics. If a match is found, the connection from the pool is used "under the covers" instead of creating a new connection. If a match is not found, a new connection is opened.

ODBC connection pooling can improve performance significantly because an ODBC application can take a long time to connect relative to the time it takes to process data.

ODBC pooled connections can be shared with single or multi-threaded ODBC applications, but are not shared between separate ODBC applications.

Note: In Windows environments, connection pooling is provided by the Windows Driver Manager rather than the Ingres ODBC CLI. 🚫

ODBC Connection Pools: Per Driver and Per Environment

ODBC connection pooling is activated by invoking `SQLSetEnvAttr()` with the attribute `SQL_ATTR_CONNECTION_POOLING`, and with the directives `SQL_CP_ONE_PER_DRIVER` or `SQL_CP_ONE_PER_HENV`.

Example: `SQL_ATTR_CONNECTION_POOLING` Attribute

```
rc = SQLSetEnvAttr( NULL, SQL_ATTR_CONNECTION_POOLING, "  
(SQLPOINTER)SQL_CP_ONE_PER_DRIVER, SQL_IS_INTEGER);
```

`SQL_CP_ONE_PER_DRIVER` means that there is only one connection pool for the entire ODBC application, regardless of the number of connections. If only one environment handle is allocated, `SQL_CP_ONE_PER_DRIVER` is essentially the same as `SQL_CP_ONE_PER_HENV`.

If multiple environment handles are allocated, it may make more sense to specify `SQL_CP_ONE_PER_HENV`, especially if the connections associated with each environment have similar characteristics. This directive will create multiple pools, each with a smaller number of connections to search through.

ODBC Connection Pool Match Criteria: Strict and Relaxed

The `SQLSetEnvAttr()` function allows the ODBC application to specify match criteria for objects in the connection pool.

Connection objects with "strict" criteria must have identical connection specifiers for connection attributes specified in the connection string from `SQLDriverConnect()`. "Strict" is the only option supported, and other specifications are ignored.

Example: Strict Match Criterion

```
rc = SQLSetEnvAttr(henv, SQL_ATTR_CP_MATCH, (SQLPOINTER)
SQL_CP_STRICT_MATCH, SQL_IS_INTEGER);
```

"Relaxed" match criteria do not apply to `SQLDriverConnect()` in the Ingres ODBC CLI. If a relaxed match criterion is specified, only key connection attributes must match. For the ODBC CLI, the following minimum attributes must match, if specified:

- DATABASE
- SERVER_TYPE
- SERVER
- DRIVER
- GROUP
- ROLENAM
- ROLEPWD
- DBMS_PWD

The following connection string attributes are ignored for relaxed match criteria:

- BLANKDATE
- DATE1582
- CATCONNECT
- SELECTLOOPS
- NUMERIC_OVERFLOW
- CATSCHEMANULL

ODBC Connection Pool Timeout

ODBC pooled connections can be configured to time out to prevent an unmanageable number of connections in the pool.

By default, the Ingres ODBC CLI allows connections to remain in the pool as long as the ODBC application is active. You can override the default connection pool timeout value using the Ingres ODBC Administrator (iiodbcadmin) on UNIX, Linux, and VMS.

To change the connection pool timeout value on UNIX, Linux, and VMS

1. Start the Ingres ODBC Administrator by issuing the **iiodbcadmin** command.

2. At the utility menu, choose Drivers, Configuration Options.

The Configuration screen appears.

3. Enter a value in the edit box from 1 to 2,147,483,647, which represents the number of seconds that unused connections remain in the pool.

The default value is -1, which means that pooled connection objects never time out. When a connection object times out, it is disconnected from the database and deleted from the pool.

Note: A thread in the ODBC CLI manages connection pool timeouts. If the time out value is left at -1, the thread is never started. For performance reasons, it is better to leave the timeout value at -1 rather than specifying a large value.

Ingres ODBC and Distributed Transactions (Windows)

The Ingres ODBC 3.5 Driver fully supports enlistment of the ODBC connection in a distributed transaction on Windows. The ODBC driver works with the Microsoft Distributed Transaction Coordinator (MSDTC), Ingres DBMS Server, and the Ingres XA Distributed Transaction Processing (DTP) subsystem to allow the Ingres connection to participate in the distributed transaction on Windows with other Ingres or non-Ingres participants.

To enlist in a transaction represented by a ITransaction interface, the Ingres ODBC driver supports the SQLSetConnectionAttr(hDBC, SQL_ATTR_ENLIST_IN_DTC, pITransaction) statement. Applications such as ODBC .NET Data Provider, Microsoft Transaction Server (MTS), or ordinary applications acquire a ITransaction pointer to a MSDTC transaction, and call the Ingres ODBC driver with the request to enlist the ODBC connection in the transaction. When the commit or rollback is required for the MSDTC transaction, the Ingres ODBC driver works as a Resource Manager (RM) with MSDTC to execute the commit or rollback for the XA transaction on the Ingres DBMS Server.

How You Enable the Use of Distributed Transactions through the Ingres ODBC Driver

To use distributed transactions through the Ingres ODBC driver and to maintain security within Windows XP, the Ingres DBA must register the Ingres ODBC driver to Windows, and the Windows MSDTC account must be enabled and registered to Ingres.

To enable the use of distributed transactions

1. Enable XA Transactions.
Select Component Services, My Computer Properties, MSDTC, Security Configuration, Enable XA Transactions.
2. Add to the Windows registry the DLL name of the Ingres ODBC Driver (caiod35.dll). (Windows XP SP2 and later requires that the DLL name of a MSDTC XA Resource Manager be defined in the Windows registry to enhance the security of the system.)

Note: This step is not necessary if distributed transactions (SQL_ATTR_ENLIST_IN_DTC feature of the driver) will not be used.

To register the Ingres ODBC driver, add the following entry to HKLM\Microsoft\MSDTC\XADLL:

```
caiod35.dll REG_SZ C:\Program  
Files\Ingres\IngresII\ingres\bin\caiod35.dll
```

3. Ensure that the Network Service account on Windows is authorized to access Ingres. (During recovery operations, the MSDTC proxy calls the ODBC driver under the NetworkService account on Windows XP SP2.)

Register the Network Service account with the Ingres Name Service. Using Ingres Visual DBA or accessdb utilities, add "networkservice" as an Ingres user. Only the name for the Ingres user account is required; no privileges are required.

Vnode Definitions When Using Distributed Transactions through ODBC

For remote Ingres servers, vnodes must be of type Global, not Private, for "Login/password data" and "Connection data" records in the vnode definition. Global definitions are required because portions of MSDTC run under the Windows System account and need the global login and connection data to connect to Ingres for transaction recovery.

If needed, the vnode definition login records must contain the username/password information, not the application connection string, because the username/password information is needed by the MS Transaction Manager and Ingres to connect to the server when XA recovery is needed. Only the vnode information, not the application connection string, is available at that time.

Troubleshooting Distributed Transactions through ODBC

[Microsoft][ODBC Driver Manager] Failed to enlist on calling object's transaction

Possible actions include:

- Check that the MSDTC service is running.
Select Start, Settings, Control Panel, Administrative Tools, Services, MSDTC.
- Check the Windows Event Viewer's application and system logs for any MS DTC messages.
- Check for messages in the client machine's Ingres log on \$II_SYSTEM\ingres\files\errlog.log.
- Check for messages in the server machine's Ingres log.
- If using MTS, check the MTS Transaction Timeout value:
 1. Start the MS Transaction Server Explorer.
 2. Select Computers.
 3. Right-click the computer (often My Computer) where the transaction was initiated.
 4. Click the Options property tab.

It is unlikely that the component's transaction aborted due to transaction timeout (default is 60 seconds) before the database enlistment is completed. However, if your transaction takes an unusually long time to enlist, you might consider increasing the Transaction Timeout value.

DTC transaction recovery is not working as expected

If DTC transaction recovery is not working as expected, check the Windows event viewer log/application screen for MS DTC messages. Also check the Ingres error log file %II_SYSTEM%\ingres\files\errlog.log on both the client and server machines.

To turn on the display of error messages and tracing of Ingres XA transaction processing

Issue this command:

```
ingsetenv II_XA_TRACE_FILE "C:\mydir\xatrace.log"
```

Ingres XA tracing should normally be turned off (ingunset II_XA_TRACE_FILE) because the trace output can accumulate to a significant volume over time.

For information on how to respond to relevant error messages, see Microsoft Knowledge Base Document 415863.

ODBC Trace Diagnostics

If you have built your ODBC directly from code such as ADO or C, you can debug your application directly using the appropriate debugger.

ODBC trace logs are a common method for debugging ODBC applications and can be used with any application that uses the Ingres ODBC. ODBC trace logs are the most meaningful if you understand ODBC programming or Ingres internals.

Standard ODBC Tracing

Standard ODBC tracing is written in a format that summarizes the ODBC functions called and provides information about the arguments passed to the ODBC functions. Here is an excerpt of a standard ODBC trace:

```
python iter      ae4-21c ENTER SQLAllocHandle
                  SQLSMALLINT                      1 <SQL_HANDLE_ENV>
                  SQLHANDLE                        00000000
                  SQLHANDLE *                      0021FCA4

python iter      ae4-21c EXIT  SQLAllocHandle with return code 0 (SQL_SUCCESS)
                  SQLSMALLINT                      1 <SQL_HANDLE_ENV>
                  SQLHANDLE                        00000000
                  SQLHANDLE *                      0x0021FCA4 ( 0x00961788)
```

Ingres Support can often generate test cases that reproduce the problem entirely from the information in the ODBC trace log.

Windows Environments

ODBC trace logs may be created for any Ingres application that uses ODBC. In addition to straight ODBC, applications include, but are not limited to:

- ADO
- OLE DB Provider for ODBC
- .NET Data Provider for ODBC
- Ingres Python DBI Driver
- Ingres PHP ODBC Driver
- Windows Access
- Windows Excel
- OpenOffice Base
- Microsoft ASP pages

Note: ASP pages, or any ODBC-based application executed from IIS (Internet Information Service) do not support ODBC tracing. See the Microsoft Knowledge Base document 836072 for more information.

The Windows ODBC Administrator is used to enable tracing. The ODBC Administrator may be visible from the Control Panel, or may be executed from the command prompt as `\WINDOWS\SYSTEM32\odbcad32.exe` (32-bit environments) or `\WINDOWS\SysWOW64\odbcad32.exe` (64-bit environments).

To enable tracing, click the Tracing tab in the ODBC Administrator's startup page, and then click the Start tracing now button. The Tracing page includes an entry box that specifies the path and file name of the ODBC trace log file. You may edit this box if you wish to redirect ODBC tracing output to another file.

UNIX, Linux and VMS Environments

This section discusses several methods for supporting ODBC tracing in UNIX, Linux and VMS environments.

Ingres ODBC CLI

For Ingres ODBC CLI applications, a standard ODBC trace is possible. Ingres does not currently release the code for ODBC trace libraries. However, you can obtain a binary of the ODBC trace library from Ingres Support.

To enable tracing for ODBC CLI applications:

1. Install the ODBC trace library
2. Select Drivers ⇒ Tracing from the Ingres ODBC Administrator, `iiodbadmin`.
3. To enable tracing for Linux, Unix and VMS, select the Set Tracing menu option to turn tracing on, and then select Save.

UnixODBC Driver Managers

UnixODBC environments support tracing. The Ingres ODBC Administrator may be used to trace in unixODBC environments, as previously documented for ODBC CLI applications. In fact, ODBC DSN definitions can be created in the Ingres ODBC Administrator and can be used in unixODBC applications.

To enable unixODBC tracing by manually editing the `odbcinst.ini` file, add this code section:

```
[ODBC]
Trace = yes
TraceFile = /tmp/odbctrace.log
```

ODBC Tracing on All Platforms—Internal ODBC Tracing

Internal ODBC tracing is available on all platforms, regardless of the ODBC application used. Internal ODBC tracing follows the conventions of GCA and API tracing and relies on specification of Ingres environment variables (logical definitions on VMS) to generate the trace.

The environment variable `II_ODBC_LOG` specifies the file name of the ODBC trace. It is recommended to specify the full path of the trace log file; otherwise it may be difficult to find the log after the application is executed.

`II_ODBC_TRACE` specifies the tracing level, which ranges from 0 to 5. (The higher the value, the more detailed the output.)

The `ingsetenv` utility may be used to specify ODBC tracing. In IIS environments, this utility is the only option.

Note: Use `ingsetenv` carefully, since all ODBC applications will "see" these variables whenever they are executed.

Example: ODBC Tracing Using ingsetenv

```
ingsetenv II_ODBC_LOG C:\temp\odbc.log  
ingsetenv II_ODBC_TRACE 5
```

If possible, it is preferable to use platform-specific commands to set the trace variables, especially if your environment executes many ODBC applications simultaneously.

On Windows, ODBC tracing is specified as follows:

```
> set II_ODBC_TRACE=5  
> set II_ODBC_LOG="C:\My Path\odbctrace.log"
```

On Unix and Linux, ODBC tracing is specified as follows for Bourne, Korn and Bash shells:

```
# II_ODBC_TRACE=5  
# export II_ODBC_TRACE  
# II_ODBC_LOG=/tmp/odbctrace.log  
# export II_ODBC_LOG
```

On Unix and Linux, ODBC tracing is specified as follows for the C shell:

```
$ setenv II_ODBC_TRACE 5  
$ setenv II_ODBC_LOG /tmp/odbctrace.log
```

On VMS, ODBC tracing is specified as follows:

```
$ DEF/JOB II_ODBC_LOG SYS$LOGIN:ODBC.LOG  
$ DEF/JOB II_ODBC_TRACE 5
```

You may wish to add GCA or API tracing. The environment variables `II_GCA_TRACE` and `II_API_TRACE` may be used in addition to `II_ODBC_LOG`. See `GCA_Tracing` for more information on these options.

Here is a sample excerpt from an ODBC trace log that specifies both ODBC and GCA tracing:

```
!ODBC Trace: SQLGetInfo (ca7cd0, 77, 21f0a4, 100, 21f144)
!ODBC Trace: ResetDbc
!ODBC Trace: SQLDriverConnect ()
!ODBC Trace: ResetDbc
!ODBC Trace: ConDriverName
!ODBC Trace: ConDriverInfo
!ODBC Trace: SQLConnect (ca7cd0, 0, 0, 0, 0, 0, 0)
! 0 GCA_REQUEST timeout=120000 async
! 0 GCA_SA_INIT status 00000000 (0)
! 0 GCA_SA_JUMP status 00000000 (1)
! 0 GCA_RQ_INIT status 00000000 (222)
! 0 GCA_REQUEST target='var/INGRES' prot=66
! 0 GCA_RQ_ALLOC status 00000000 (223)
! 0 GCA_SA_BUFFERS status 00000000 (224)
! 0 GCA_RQ_IS_NOXLATE status 00000000 (225)
! 0 GCA_RQ_IS_RSLVD status 00000000 (226)
! 0 GCA_RQ_ADDR_NSID status 00000000 (227)
! 0 GCA_RQ_IS_NMSVR status 00000000 (228)
! 0 GCA_RQ_USE_NMSVR status 00000000 (229)
```

Note: Do not define `II_API_LOG` or `II_GCA_LOG` if you wish to include ODBC trace information. `II_ODBC_LOG` must be defined.

Internal versus Standard ODBC Trace Logs

Internal ODBC trace logs are most useful for ODBC problems that are not related to the ODBC, such as:

- Problems connecting to the database
- Errors reported from the DBMS server, such as SQL syntax errors, table permission errors, or failure to return data from a query
- Errors related to commit or rollback

On Windows, internal ODBC trace logs is the only option for ODBC applications executed from IIS.

Standard ODBC trace logs are best for general use. Sometimes the best results are obtained by generating both types of trace logs.

Disable Tracing

Do not forget to disable ODBC tracing after you have generated your test case! ODBC tracing, like all tracing, can have a significant impact on performance. It is even possible that the disk may become full if tracing is enabled for a significant period of time.

How You Disable Standard Tracing

To disable standard tracing on Windows

1. Click the Tracing tab in the ODBC Administrator.
2. Click the Stop tracing now button.

To disable standard tracing for ODBC CLI applications

1. Execute the iiodbcadmin utility and select the Drivers ⇒ Tracing menu options.
2. Select the Set Tracing menu option to turn tracing off, and then select Save.

To disable standard tracing in unixODBC environments by editing the `odbcinst.ini` file

1. Open the `odbcinst.ini` file.
2. Unset tracing with the following edits:

```
[ODBC]
Trace = no
TraceFile =
```

How You Disable Internal Tracing

The `ingunset` utility turns off variables set via `ingsetenv`.

In Unix and Linux environments, internal tracing is disabled using either `unset` or `unsetenv`, depending on the execution shell.

On VMS, Ingres logical names are unset using `DEASSIGN/JOB`.

On Windows, ODBC trace variables are unset as follows:

```
> SET II_ODBC_LOG=
> SET II_ODBC_TRACE=
```

Chapter 11: Understanding JDBC Connectivity

This section contains the following topics:

[JDBC Components](#) (see page 223)

[Unsupported JDBC Features](#) (see page 227)

[JDBC Driver Interface](#) (see page 228)

[JDBC Implementation Considerations](#) (see page 239)

[Data Type Compatibility](#) (see page 256)

[JDBC Tracing](#) (see page 259)

This chapter explains the JDBC components that enable JDBC connectivity to Ingres data sources. It provides a description of each component, a list of supported API features, driver and server configuration information, and guidelines for implementing Java applications in the Ingres environment.

JDBC Components

Ingres JDBC consists of the following components:

- The JDBC driver
- The JDBC information utility

JDBC Driver

The Ingres JDBC Driver is a pure Java implementation of the JDBC 4.0 API released with the Sun Java 2 SDK, version 6.0. The driver supports application, applet, and servlet access to Ingres data sources through the Data Access Server.

The Ingres JDBC Driver supports the following JDBC features.

JDBC 4.0 features:

- Auto-loading of the driver

Applications do not need to explicitly load the driver by calling `Class.forName()`. Existing applications that currently load the Ingres JDBC driver using `Class.forName()` will continue to work without modification.

- National Character Set Support

The Ingres JDBC driver provides support for the new setter, getter, and updater methods for NCHAR, NVARCHAR, LONGNVARCHAR, and NCLOB types.

- Support for createBlob(), createClob(), and createNClob Methods

Applications can use the above methods for creating BLOB, CLOB, NCLOB objects.

- Support for java.sql.Wrapper Interface

Applications can use the Wrapper interface to access extensions to the JDBC API that are specific to the Ingres JDBC driver but currently the driver does not implement any such extensions that the applications can benefit from.

JDBC 3.0 features:

- Updatable ResultSets
- Scrollable ResultSets
- Transaction savepoints
- Named procedure parameters
- Auto-generated keys
- Parameter metadata
- Blob and clob data objects

The Ingres JDBC Driver is delivered as a single Java archive file, named `ijjdbc.jar`, located in the library directory (`lib`) of the Ingres instance. Depending on the Java environment used, access to the driver requires adding the Java archive to the `CLASSPATH` environment setting or as a resource in the appropriate utility. For browser/applet access, the Java archive must be copied to the Web Server directories.

JDBC Information Utility—Load the JDBC Driver

The JDBC information utility, `JdbcInfo`, loads the Ingres JDBC driver and displays its internal release information.

The class files for the `JdbcInfo` utility are located in the library directory (`lib`) of the Ingres instance. You must set your `CLASSPATH` environment (see page 226) before using the `JdbcInfo` utility.

You can invoke the `JdbcInfo` utility from the command line with the following parameters:

`java JdbcInfo`

Displays the internal driver release of the Ingres JDBC Driver.

`java JdbcInfo url`

Attempts to establish a JDBC connection to the target database using the specified URL. For information on formatting the URL, see `DriverManager.getConnection()` Method (see page 238). If successful, it displays the Ingres JDBC Driver name and release that serviced the URL connection.

`java JdbcInfo host port`

Attempts to establish a low-level connection to the Data Access Server associated with host port. If successful, it displays the internal driver release of the Ingres JDBC Driver.

Set CLASSPATH Environment

Before using the JdbcInfo utility to load the JDBC driver, you must set the CLASSPATH Java environment variable to point to the class files for the driver.

To set your CLASSPATH environment

Issue the following command:

Windows:

```
set CLASSPATH = %II_SYSTEM%\ingres\lib\ijdbc.jar;.;%II_SYSTEM%\ingres\lib
```

UNIX (C shell):

```
setenv CLASSPATH $II_SYSTEM\ingres\lib\ijdbc.jar:.$II_SYSTEM\ingres\lib
```

VMS:

```
$ DEFINE CLASSPATH "/II_SYSTEM/ingres/lib/ijbc.jar:/II_SYSTEM/ingres/lib"
```

or

```
DEFINE JAVA$CLASSPATH II_SYSTEM:[ingres.lib]ijdbc.jar,II_SYSTEM:[ingres.lib]
```

Unsupported JDBC Features

The Ingres JDBC Driver is compliant with the JDBC 4.0 API specification. JDBC 4.0 API interfaces are fully supported with the following exceptions:

PooledConnection interface

The Ingres JDBC driver does not support Statement pooling and does not implement the JDBC 4.0 methods `addStatementEventListener()` and `RemoveStatementEventListener()`.

Calling stored procedures

Ingres JDBC driver does not support escape syntax for calling stored procedures in `Statement` and `PreparedStatement`.

Auto-generated keys

The Ingres DBMS returns only a single table key or a single object key per `INSERT` statement. Ingres does not return table and object keys for `INSERT AS SELECT` statements. Depending on the keys that are produced by the statement executed, auto-generated key parameters in `execute()`, `executeUpdate()`, and `prepareStatement()` methods are ignored and `getGeneratedKeys()` returns a result-set containing no rows, a single row with one column, or a single row with two columns. The Ingres JDBC Driver returns table and object keys as `BINARY` values.

Result sets

Result sets generated by `executeQuery()` requests are always `CLOSE_CURSORS_AT_COMMIT` (non-holdable).

The methods `isBeforeFirst()` and `isLast()` may return invalid information when used on a `FORWARD-ONLY ResultSet`. The method `isBeforeFirst()` cannot always detect that a `ResultSet` is empty when no call to `next()` has been made and may return `true` instead of returning `false`. The method `isLast()` cannot always detect when the `ResultSet` is positioned on the last row and may return `false` instead of returning `true`.

Data types

The JDBC data types `DATALINK`, `ARRAY`, `REF`, `DISTINCT`, `STRUCT`, and `JAVA_OBJECT` are not supported. The storage or mapping of Java objects (`SQLInput`, `SQLOutput`, and `SQLData`) is also not supported. Methods associated with these data types throw exceptions when called.

Calendars

Ingres stores date/time values in GMT (same as Java). With an Ingres DBMS, the Ingres JDBC Driver handles all date/time values in GMT and calendars provided in `setXXX()` and `getXXX()` methods are ignored. EDBC servers and Enterprise Access gateways do not reference date/time values to a particular time zone. The Ingres JDBC Driver uses the local time zone when accessing a non-Ingres DBMS Server, and utilizes calendars if provided. Calendars are also used for `TIME WITHOUT TIME ZONE` and `TIMESTAMP WITHOUT TIME ZONE` values.

Batch updates

If the DBMS does not support batch processing, batched execution for `Statements`, `PreparedStatement`s, and `CallableStatement`s is supported by individual execution of each batched request.

JDBC Driver Interface

This section details the Ingres JDBC Driver interface class files and their associated properties. It also includes instructions for loading and accessing the driver.

JDBC Driver and Data Source Classes

The Ingres JDBC driver and data source classes are located in the Java package, `com.ingres.jdbc`.

These packages are contained in the Java archive `ijjdbc.jar`, which includes the following class files:

Class	Implemented JDBC Interface
<code>com.ingres.jdbc.IngresDriver</code>	The Ingres implementation of the JDBC Driver interface (<code>java.sql.Driver</code>).
<code>com.ingres.jdbc.IngresDataSource</code>	The Ingres implementation of the JDBC <code>DataSource</code> interface (<code>javax.sql.DataSource</code>).
<code>com.ingres.jdbc.IngresCPDataSource</code>	The Ingres implementation of the JDBC <code>ConnectionPoolDataSource</code> interface (<code>javax.sql.ConnectionPoolDataSource</code>).
<code>com.ingres.jdbc.IngresXADataSource</code>	The Ingres implementation of the JDBC <code>XADataSource</code> interface (<code>javax.sql.XADataSource</code>).

Note: The original Ingres JDBC Driver and DataSources classes contained in the Java archive `ijjdbc.jar` under the Java package path of `"ca.ingres.jdbc"` are moved to the package path of `"com.ingres.jdbc"`. The `ijjdbc.jar` archive included with Ingres 9.0 also contains the original classes for backward compatibility. Starting with Ingres 10.0, the `"ca.ingres.jdbc"` package is no longer included in the `ijjdbc.jar` archive. Existing references to `"ca.ingres.jdbc"` classes must be changed to `"com.ingres.jdbc"`.

JDBC Driver Properties

Driver properties allow applications to establish connection parameters that are driver-dependent. Ingres JDBC Driver properties can be specified as connection URL attributes as a Java Properties parameter to a `DriverManager.getConnection()` method, as Java system properties, or in a properties file. Attribute and property names are given below.

When specified as system properties or in a property file, the property key must be of the form

`ingres.jdbc.property.property_name`

During Ingres installation, the JDBC Driver Properties Generator (see page 233) reads the Ingres configuration and generates the corresponding JDBC driver properties file, named `ijjdbc.properties`.

A default properties file (typically `ijjdbc.properties`) is loaded automatically by the Ingres JDBC Driver when the driver class is loaded. The file must reside in a location accessible by the class loader used to load the driver. In general, this requires that the properties file directory be included in the Java environment variable `CLASSPATH`.

An alternate properties file can also be specified using the system property `ingres.jdbc.property_file`. The directory path of the property file can be specified or the property file can be placed in a directory accessible as described above for the default properties file. Properties are searched in the following order: URL attributes, `getConnection()` property set, system properties, alternate properties file, and default properties file.

The Ingres JDBC Driver supports the following properties:

Property	Attribute	Description
user	UID	<p>The user ID on the target DBMS Server machine. See the description of the <code>vnode_usage</code> property in this table.</p> <p>This property can be used if the user has no DBMS user ID and password assigned (see <code>dbms_user</code> and <code>dbms_password</code> in this table).</p>

Property	Attribute	Description
password	PWD	The user's operating system password.
role	ROLE	The desired role identifier. If a role password is required, include it with the role name as follows: <i>name/password</i> .
group	GRP	The user's group identifier.
dbms_user	DBUSR	The user name associated with the DBMS session (Ingres -u flag, can require admin privileges).
dbms_password	DBPWD	The user's DBMS password (Ingres -P flag).
connect_pool	POOL	<p>Server connection pool control. Valid values are:</p> <p>off—requests a non-pooled connection when server pooling is enabled</p> <p>on--requests a pooled connection when server pooling is optional.</p> <p>The default is to allow the DAS configuration to determine pooling.</p>
select_loop	LOOP	<p>Select loop vs. cursor queries. Valid values are:</p> <p>on—uses select loops to retrieve query results</p> <p>off—(Default) uses cursors.</p> <p>For further details, see Cursors and Select Loops (see page 244).</p>
autocommit_mode	AUTO	<p>Autocommit cursor handling mode. Valid values are:</p> <p>dbms—(Default) autocommit processing is done by the DBMS Server</p> <p>single—DAS enforces single cursor operation during autocommit</p> <p>multi—DAS simulates autocommit operations when more than one cursor is open.</p> <p>For further details, see How Transactions Are Autocommitted (see page 240).</p>
cursor_mode	CURSOR	Default cursor concurrency mode, which determines the concurrency of cursors that have no concurrency explicitly assigned. Valid values are:

Property	Attribute	Description
		<p>dbms--concurrency is determined by the DBMS Server</p> <p>update--provides updateable cursors</p> <p>readonly--(Default) provides non-updateable cursors</p> <p>Further details are provided in Cursors and Result Set Characteristics (see page 242).</p>
vnode_usage	VNODE	<p>Allows the JDBC application to control the portions of the vnode information that are used to establish the connection to the remote DBMS server. Valid values are:</p> <p>connect--(Default) Only the vnode connection information is used to establish the connection.</p> <p>login--Both the vnode connection and login information are used to establish the connection.</p> <p>For further details, see JDBC User ID Options (see page 239).</p>
char_encode	ENCODE	<p>Specifies the Java character encoding used for conversions between Unicode and character data types. Generally, the character encoding is determined automatically by the driver from the DAS installation character set. This property allows an alternate character encoding to be specified (if desired) or a valid character encoding to be used when the driver is unable to map the server's character set.</p>
timezone	TZ	<p>Specifies the Ingres time zone associated with the client's location. Corresponds to the Ingres environment variable II_TIMEZONE_NAME and is assigned the same values. This property is not used directly by the driver but is sent to the DBMS and affects the processing of dates.</p>
decimal_char	DECIMAL	<p>Specifies the character to be used as the decimal point in numeric literals. Corresponds to the Ingres environment variable II_DECIMAL and is assigned the same values. This property is not used directly by the driver but is sent to the DBMS and affects the processing of query text.</p>
date_alias	DATE	<p>Specifies the data type of columns created using the alias keyword "date". Valid values</p>

Property	Attribute	Description
		are: "ingresdate" (the default) or "ansidate". This property is not used directly by the driver but is sent to the DBMS and affects the processing of query text.
date_format	DATE_FMT	Specifies the Ingres format for date literals. Corresponds to the Ingres environment variable II_DATE_FORMAT and is assigned the same values. This property is not used directly by the driver, but is sent to the DBMS and affects the processing of query text.
money_format	MNY_FMT	Specifies the Ingres format for money literals. Corresponds to the Ingres environment variable II_MONEY_FORMAT and is assigned the same values. This property is not used directly by the driver but is sent to the DBMS and affects the processing of query text.
money_precision	MNY_PREC	Specifies the precision of money data values. Corresponds to the Ingres environment variable II_MONEY_PREC and is assigned the same values. This property is not used directly by the driver but is sent to the DBMS and affects the processing of money values.
send_ingres_dates	SEND_INGDATE	Specifies whether date/time/timestamp values should be sent as INGRESDATE data type. Valid values are: false—(Default) Values sent as ANSI TIMESTAMP WITH TIMEZONE type true—Values sent as INGRESDATE type Unlike date_format and date_alias, this property is internal to the driver, not sent to the DBMS, and affects the processing of query text.
send_integer_booleans	SEND_INTBOOL	Specifies if Boolean parameters are converted to tinyint values when sent to the DBMS. Valid values are: true—Boolean parameters are converted to tinyint values false—(Default) Boolean parameters are sent as Boolean values

Attributes can also be specified using the property name as the attribute name. Thus "UID=user1" and "user=user1" are semantically the same.

JDBC Driver Properties Generator (ijjdbcprop)

The ijjdbcprop utility automatically generates all supported JDBC properties. The utility runs automatically during installation but can be run at any time.

This utility provides the following benefits:

- Automatically generates all JDBC properties that match related Ingres environment variables, such as II_TIMEZONE_NAME, II_DECIMAL, II_DATE_FORMAT, II_MONEY_FORMAT and II_MONEY_PREC. These properties can keep the JDBC driver behavior synchronized with an Ingres installation, since the JDBC driver does not have access to the Ingres environment variables and is often deployed on a machine separate from Ingres.
- Reduces typing errors when entering JDBC properties by hand. All available JDBC properties are generated, but properties not in the Ingres configuration files are commented out.

The ijjdbcprop command writes the ijjdbc.properties file into the directory:

UNIX: \$II_SYSTEM/ingres/files

Windows: %II_SYSTEM%\ingres\files

VMS: II_SYSTEM:[INGRES.FILES]

During loading of the JDBC driver by the class loader, the directory containing the ijjdbc.properties file must be specified in the Java environment variable CLASSPATH.

If an Ingres JDBC application is connecting to a remote Data Access Server (that is, Ingres is not installed on the machine running ijjdbc.jar), the ijjdbc.properties file from the remote Ingres installation can be copied to the machine running ijjdbc.jar. The directory of the ijjdbc.properties file must be included in the CLASSPATH (see page 226).

Example Entries—For example, to turn on tracing in the driver, uncomment the desired trace entries in the ijjdbc.properties file by removing the leading # sign, and then provide appropriate values for them as shown here.

On UNIX, the following entries in ijjdbc.properties file will produce a JDBC driver trace file under the /tmp directory called jdbc_driver.log, provided the \$II_SYSTEM/ingres/files directory is in the CLASSPATH.

```
ingres.jdbc.trace.log=/tmp/jdbc_driver.log
ingres.jdbc.trace.driv=5
ingres.jdbc.trace.msg=3
```

For details on the ijjdbcprop command, see the *Command Reference Guide*.

Data Source Properties

A data source configuration is a collection of information that identifies the target database to which the driver connects. The Data Source classes support the following data source properties and associated getter/setter methods.

DS Property	Description
description	Description of the data source
serverName	<p>Server host name or network address (required).</p> <p>Multiple hosts and associated ports can be specified using the following syntax:</p> <p>host:port{,port}{;host:port{,port}}</p> <p>TCP/IPv6 addresses (colon-hexadecimal format) must be enclosed in square brackets, for example: [::1]. If a single host name or address is provided with no port, the associated port must be provided using the portName or portNumber properties. If a port (or ports) is provided in this property, then the portName and portNumber properties should not be set.</p>
portName	Symbolic port ID. Multiple ports can be provided, separated by commas. A port ID must be provided in the serverName, portName, or portNumber properties.
portNumber	Numeric port ID. A port ID must be provided either in the serverName, portName, or portNumber properties.
databaseName	Database name (required).
user	User's ID. (A user ID is required when the DAS is not on the same machine as the JDBC client; otherwise this property is optional.)
password	User's password. (A password is required when the DAS is not on the same machine as the JDBC client; otherwise this property is optional.)
roleName	DBMS role identifier
groupName	DBMS group identifier
dbmsUser	User ID for the DBMS session (-u flag)
dbmsPassword	User's DBMS password
connectionPool	Use pooled connection: 'off' or 'on'
autocommitMode	Autocommit cursor handling: 'dbms', 'single', 'multi'
selectLoop	Select loop processing: 'off', or 'on'
cursorMode	Default cursor concurrency: 'dbms', 'update', 'readonly'
vnodeUsage	Vnode usage for DBMS Server access: 'login', 'connect'

DS Property	Description
charEncode	Java character encoding
timeZone	Ingres timezone
decimalChar	Ingres decimal character
dateAlias	Ingres date alias
dateFormat	Ingres date format
moneyFormat	Ingres money format
moneyPrecision	Ingres money precision
sendIngresDates	Send date/time/timestamp values as ingresdate data type: 'true' or 'false'
sendIntegerBooleans	Send Boolean parameters as tinyint values: 'true' or 'false'

The data source properties marked as “required” correspond to parameters contained in a connection URL. For a description of these parameters, see Establish JDBC Driver Connection (see page 238). The remaining Data Source properties correspond to the driver properties defined in JDBC Driver Properties (see page 229).

Additional Data Source Properties

In addition to the DataSource class properties, the ConnectionPoolDataSource and XADataSource classes support the following properties and associated getter/setter methods:

DS Property	Description
initialPoolSize	Initial connection pool size
minPoolSize	Minimum connection pool size
maxPoolSize	Maximum connection pool size
maxIdleTime	Maximum time in connection pool
propertyCycle	Wait time for checking the connection pool

System Properties

The following system properties can be used to configure the driver:

ingres.jdbc.property_file

Path and filename containing configuration system properties. Default is `ijjdbc.properties` (must reside in CLASSPATH directory).

ingres.jdbc.property.<property name>

Driver connection properties. Defaults are property dependent.

ingres.jdbc.batch.enabled

Specifies whether batch statement processing is enabled. If set to true, batch statements will be executed together in a batch. If set to false (or if batch processing is not supported by the DBMS), batch statements will be executed individually. Default is true.

ingres.jdbc.scroll.enabled

Enable or disable scrollable result sets (true/false). If disabled, all result sets will be forward-only and an SQLWARNING will be generated if a scrollable result set is requested. Default is true.

ingres.jdbc.lob.cache.enabled

Enable/disable caching Large Objects (true/false). Default is false.

ingres.jdbc.lob.cache.segment_size

Number of bytes/characters per segment in LOB cache. Default is 8192.

ingres.jdbc.lob.locators.enabled

Enable/disable Locators for Large Objects if supported by DBMS (true/false). Default is true.

ingres.jdbc.lob.locators.autocommit.enabled

Enable/disable Locators for Large Objects during autocommit (true/false). LOB Locators must also be generally enabled. Default is false.

ingres.jdbc.lob.locators.select_loop.enabled

Enable/disable Locators for Large Objects during select loops (true/false). LOB Locators must also be generally enabled. Default is false.

ingres.jdbc.date.empty

Replacement value, in standard JDBC date/time format YYYY-MM-DD hh:mm:ss, for Ingres empty dates. Can also be set to null to have empty dates returned as null values. For default behavior, set to default or empty, as described in Date/Time Columns and Values (see page 253).

ingres.jdbc.dbms.trace.log

Path and filename of the DBMS trace log. Default is no trace log.

ingres.jdbc.trace.log

Path and filename of the driver trace log. Default is no trace log.

ingres.jdbc.trace.driv

Driver trace level. Default is 0 (no tracing).

ingres.jdbc.trace.ds

DataSource trace level. Default is 0 (no tracing).

ingres.jdbc.trace.msg

Messaging system trace level. Default is 0 (no tracing).

ingres.jdbc.trace.msg.tl

Transport layer trace level. Default is 0 (no tracing).

ingres.jdbc.trace.msg.nl

Network layer trace level. Default is 0 (no tracing).

ingres.jdbc.trace.timestamp

Include timestamps in traces (true/false). Default is false.

How the Driver Is Loaded

The Ingres JDBC Driver can be loaded by an application or applet by using one of these methods:

- Adding the driver class, `com.ingres.jdbc.IngresDriver`, to the JDBC DriverManager system property `"jdbc.drivers"`
- Adding the following Java statement to the application/applet prior to attempting to establish a connection using the Ingres JDBC Driver:

```
Class.forName( "com.ingres.jdbc.IngresDriver" ).newInstance();
```

Depending on the Java environment, calling the `forName()` method can be sufficient to load and initialize the Ingres JDBC Driver classes. Some environments, most notably older releases of Microsoft Internet Explorer, require the instantiation of an Ingres JDBC Driver object to fully initialize the driver.

While only one method is needed, both methods can be used without conflict.

DriverManager.getConnection() Method—Establish JDBC Driver Connection

An Ingres JDBC Driver connection can be established using a `DriverManager.getConnection()` method with a URL in the following format:

```
jdbc:ingres://host:port{,port}{;host:port{,port}}/db{;attr=value}
```

where:

host

Specifies the network name or address of the host on which the target Data Access Server (DAS) is running. TCP/IPv6 addresses (colon-hexadecimal format) must be enclosed in square brackets, for example: `[::1]`. Multiple hosts with associated ports must be separated by semi-colons.

port

Specifies the network port used by the DAS. This can be a numeric port number or an Ingres symbolic port address such as `II7`, the default, which is the Ingres installation ID followed by the number 7.

Multiple ports can be specified, one for each configured DAS. For example, if DAS is configured with a startup count of 4 and a listen port of `II7+`:

```
II7,II8,II9,II10
```

The driver attempts to connect to each host/port combination until a successful connection is made. The connection request between the JDBC application and the DAS fails only if all port attempts fail. If a successful connection to DAS is made, but the subsequent connection between DAS and the DBMS fails, then the connection request fails immediately and subsequent ports are not tried.

db

Specifies the target database. Any valid Ingres database designation can be used including vnode and server class (that is, `vnode::dbname/server_class`).

attr=value

(Optional) Specifies the attribute name and value pair. Multiple attribute pairs are separated by a semi-colon.

Attributes represent driver properties that are implementation-specific and can be used to configure the new connection. For details, see *JDBC Driver Properties* (see page 229).

Note: A user ID and password are required when making remote connections. They can be included as parameters to the `getConnection()` method as driver properties or as URL attributes.

Access to a Remote Instance

Before connecting to a remote Ingres DBMS, Enterprise Access, or EDBC instance, a vnode connection must be defined.

More information

Requirements for Accessing Remote Instances (see page 50)

Access Tools for Defining Vnodes (see page 51)

JDBC Implementation Considerations

To implement Java applications in the Ingres environment, you should be aware of the programming considerations and guidelines.

JDBC User Authentication

The Ingres JDBC Driver does not require a user ID and password to establish a connection when the Ingres Data Access Server (DAS) is running on the same machine as the Java client. When a userID/password is not provided, the Java client process user ID is used to establish the DBMS connection. If the target database specification includes a VNODE (see page 51), the VNODE login information is used to access the DBMS machine. Optionally, a userID/password can be provided and is handled as described below.

When the Java client and DAS are on different machines, a user ID and password are required to establish a connection to the DBMS. If the DAS and DBMS server are running in the same Ingres instance (no VNODE in target database specification), the userID/password is used to validate access to the DAS/DBMS machine.

When the DAS and DBMS servers are on different machines, a VNODE is required in the target database specification. The VNODE provides the connection and (optionally) login information needed to establish the DBMS connection.

The driver property `vnode_usage` determines how the VNODE is used to access the DBMS. The `vnode_usage` property also determines the context (DAS or DBMS) in which the application userID/password is used. VNODE usage without a userID/password is described above. If the target database specification does not contain a VNODE, the `vnode_usage` property is ignored.

When `vnode_usage` is set to 'connect', only global VNODE connection information is used to establish the DBMS connection. The application-provided user ID and password are used in the DBMS context to access the DBMS machine.

When `vnode_usage` is set to 'login', both connection and login VNODE information is used to access the DBMS machine. The application-provided user ID and password are used in the DAS context, allowing access to private and global VNODEs.

Note: Ingres may use the `ingvalidpw` program (see page 45) to validate a user password, depending on the platform requirements where the password is validated.

How Transactions Are Autocommitted

Application developers must be aware that the DBMS Server imposes severe limits on the operations that can be performed when autocommit is enabled (the JDBC default transaction mode) and a cursor is opened. In general, only one cursor at a time can be open during autocommit, and only cursor-related operations (cursor delete, cursor update) can be performed. Violating this restriction results in an exception being thrown with the message text:

```
No MST is currently in progress, cannot declare another cursor
```

Cursors are opened by the `Statement` and `PreparedStatement` `executeQuery()` methods and remain open until the associated `ResultSet` is closed. The driver closes a cursor automatically when the end of the result set is reached, but applications must not rely on this behavior. JDBC applications can avoid many problems by calling the `close()` method of each JDBC object when the object is no longer needed.

`autocommit_mode` Connection Property—Set Autocommit Processing Mode

The Ingres JDBC Driver provides alternative autocommit processing modes that help overcome the restriction of autocommitting transactions or handle problems that applications have with closing result sets.

The autocommit processing modes can be selected by setting the connection property `autocommit_mode` to one of the following values. For additional information, see JDBC Driver Properties (see page 229).

Value	Mode	Description
dbms	DBMS (default)	Autocommit processing is done by the DBMS Server and is subject to the restrictions mentioned above.
single	Single-cursor	The DAS allows only a single cursor to be open during autocommit. If a query or non-cursor operation is requested while a cursor is open, the server closes the open cursor. Any future attempts to access the cursor fails with an unknown cursor exception. This mode is useful for applications that fail to close result sets, but does not perform other queries or non-cursor related operations while the result set is being used.
multi	Multi-cursor	<p>Autocommit processing is done by the DBMS Server when no cursors are open. The DAS disables autocommit and begins a standard transaction when a cursor is opened. Because autocommit processing is disabled, multiple cursors can be open at the same time and non-cursor operations are permitted.</p> <p>When a cursor is closed, and no other cursor is open, the DAS commits the standard transaction and re-enables autocommit in the DBMS. This mode overcomes the restrictions imposed by the DBMS during autocommit, but requires the application to be very careful in closing result sets. Because the DAS does not commit the transaction until all cursors are closed, a cursor left open inadvertently eventually runs into log-file full problems and transaction aborts.</p>

Cursors and Result Set Characteristics

Ingres cursors and JDBC result sets both have an associated type specifying that the object is scrollable or forward-only. The Ingres JDBC driver by default opens cursors as forward-only.

A scrollable cursor can be opened by specifying a JDBC result set type of `ResultSet.TYPE_SCROLL_INSENSITIVE` or `ResultSet.TYPE_SCROLL_SENSITIVE` when creating the associated statement. Ingres read-only (static) scrollable cursors are insensitive to changes, while updatable (keyset) scrollable cursors are sensitive to changes. Either type can be specified for both read-only and updatable cursors. If the incorrect type is used, the JDBC driver will produce a result set with the correct type and generate a JDBC warning indicating that the result set type was changed.

Ingres cursors and JDBC result sets both have an associated concurrency characteristic specifying that the object is readonly or updatable. The Ingres JDBC Driver automatically provides an updatable `ResultSet` when the associated cursor is updatable. The JDBC readonly/update mode characteristics are used by the Ingres Driver to control the mode of the resulting cursor.

For an updatable cursor, row updates and deletes can be performed using the updatable `ResultSet` interface or by using a separate JDBC Statement to issue positioned update and delete statements on the cursor. The cursor name needed to issue a positioned update or delete statement can be assigned using the Statement method `setCursorName()` or obtained by using the `ResultSet` method `getCursorName()`.

Cursor concurrency can be specified using the `FOR READONLY` or `FOR UPDATE` clause in the `SELECT` statement. The Ingres JDBC Driver supports the JDBC syntax `SELECT FOR UPDATE` (and also `SELECT FOR READONLY`) and translates this to the correct Ingres syntax.

A cursor is opened as readonly if one of the following is true (listed in descending precedence):

- The `SELECT` statement contains the `FOR READONLY` clause.
- The associated statement was created using a Connection method that specified the concurrency as `ResultSet.CONCUR_READ_ONLY`.
- The connection is readonly (`Connection.setReadOnly(true)`).
- The connection property `cursor_mode` is set to 'readonly' (the default setting).
- The connection property `cursor_mode` is set to 'dbms' and the DBMS Server determines that the cursor cannot be updated.

A cursor is opened as updatable if one of the following is true (listed in descending precedence):

- The SELECT statement contains the FOR UPDATE clause.
- The associated statement was created using a Connection method that specified the concurrency as `ResultSet.CONCUR_UPDATABLE` and the DBMS Server determines that the cursor can be updated.
- No other readonly condition is true and the DBMS Server determines that the cursor can be updated.

Note: The Ingres JDBC Driver does not attempt to force the cursor to be updatable even when the application requests a concurrency of `ResultSet.CONCUR_UPDATABLE` when creating the associated statement or the connection property `cursor_mode` is set to 'update'. In these cases, the cursor will be updatable if the DBMS Server determines that an updatable cursor is possible, otherwise the cursor will be readonly. The JDBC specification requires "graceful degradation" with a warning rather than throwing an exception when a requested concurrency cannot be provided.

Turn Off Bi-directional Updatable Scrollable Cursors

You can use a configuration setting to turn off bi-directional updatable scrollable cursors. All result-sets will be forward-only.

To turn off bi-directional updatable scrollable cursors

Use this command:

```
java -Dingres.jdbc.scroll.enabled=false App
```

Or use this alternative method:

1. Put the following line in a property file:

```
ingres.jdbc.scroll.enabled=false
```

2. Specify the property file on the command line:

```
java -Dingres.jdbc.property_file=file App
```

Cursors and Select Loops

By default, the Ingres JDBC Driver uses a cursor to issue SQL select queries. Cursors permit other SQL operations, such as deletes or updates, to be performed while the cursor is open. (Operations can be restricted during autocommit, as described in *How Transactions Are Autocommitted* (see page 240).)

Cursors also permit multiple queries to be active at the same time. These capabilities are possible because only a limited number of result rows (frequently only a single row) are returned by the DBMS Server for each cursor fetch request. The low ratio of driver requests to returned rows results in lower performance compared to other access methods.

The Ingres JDBC Driver uses cursor pre-fetch capabilities whenever possible. Updatable cursors only return a single row for each fetch request. READONLY cursors return a fixed number of rows on each fetch request. For details, see *Cursors and Result Set Characteristics* (see page 242). By default, the Ingres JDBC Driver obtains as many rows as fit in one communications block on each fetch request.

Depending on row size, this can greatly increase data access efficiency. The application can also specify the number of rows to be retrieved for READONLY cursors by using the `setFetchSize()` method.

The Ingres JDBC Driver also permits the JDBC application to use a data access method called a select loop. In a select loop request, the DBMS Server returns all the result rows in a single data stream to the driver. Because select loops use the connection while the result set is open, no other operation or query can be performed until the result set is closed.

The statement `cancel()` method can be used to interrupt a select loop data stream when a result set needs to be closed before the last row is processed. Because the DBMS Server does not wait for fetch requests from the driver, this access method is the most efficient available.

Select loops are enabled in the Ingres JDBC Driver by setting the driver connection property `select_loop` to a value of 'on.' For more information, see *JDBC Driver Properties* (see page 229).

With select loops enabled, the driver avoids using cursors for SELECT queries unless explicitly indicated by the application. An application can request a cursor be used for a query by assigning a cursor name to the statement (`setCursorName()` method) or by using the JDBC syntax 'SELECT FOR UPDATE ...' to request an updatable cursor.

Batch Statement Execution

To take advantage of the batch query execution capability of the Ingres DBMS, you can use the `addBatch` and `executeBatch` methods supported by the Ingres JDBC Driver.

Batched statements that use repeated dynamic INSERT statements (for example, through Java-based Extract Transfer and Load tools) are specially optimized to improve performance.

If you want to force individual statement execution, use the `ingres.jdbc.batch.enabled` system property (see page 236) to disable batch query execution. If the DBMS does not support batch processing, the Ingres JDBC driver detects it and automatically executes the statements individually.

We recommend not to use `autocommit` with batch execution. If batch execution is used with `autocommit`, and you cancel the batch execution, it is impossible to tell which statements were committed.

You can improve performance of batch statement processing by following these guidelines:

- If you have large batches of inserts, we recommend that you use prepared statements. Using prepared INSERT statements with large batches (of more than 100) can significantly improve performance.
- If you are using prepared statements for batch, we recommend that you make your batches as large as possible. Larger batch sizes can make a significant difference with insert performance—even 2 or 3 times faster. In fact, when using prepared INSERT statements, the larger the batch, the better the performance. Batch sizes up to 100,000 have been noted to significantly improve performance.
- If you must use small batches (of less than 100), then you should avoid using prepared statements. The DBMS optimization works well only for large batches; for small batches, you can achieve better performance by batching non-prepared inserts.

Faster inserts also can be achieved if the following conditions are met:

- Inserts must be into a base table (not a view or index).
- The table must not have any rules or integrities defined on it.
- The table must be a regular Ingres (not gateway) table.
- The inserts must be batched.
- The batched statements must be an execution of a prepared dynamic insert where the dynamic parameters exactly match the values being inserted.

Batch Statement Execution Example

Here is a Java code segment that shows batched inserts using prepared statements:

```
PreparedStatement prep = conn.prepareStatement( "insert into emp( id, name )  
values( ?, ? )" );
```

```
prep.setInt( 1, 1001 );
```

```
prep.setString( 2, "Adam Anderson" );
```

```
prep.addBatch();
```

```
...
```

```
prep.setInt( 1, 1099 );
```

```
prep.setInt( 2, "Barry Bradley" );
```

```
prep.addBatch();
```

```
int results[] = prep.executeBatch()
```

Database Procedures

Database procedures are supported through the JDBC CallableStatement interface. The Ingres JDBC Driver supports the following database procedure syntax.

Note: Items enclosed in brackets are optional.

Database Procedure	Syntax
JDBC/ODBC CALL escape	{[? =] CALL [schema.]name[(parameters)]}
Ingres EXECUTE PROCEDURE	EXECUTE PROCEDURE [schema.]name[(parameters)] [INTO ?]
Ingres CALLPROC	CALLPROC [schema.]name[(parameters)] [INTO ?]

For all of these statements, the Ingres JDBC Driver supports a combined parameter syntax supporting features of the ODBC positional parameter syntax and the Ingres named parameter syntax:

```
parameters := param | param, parameters
```

```
param := [name =] [value]
```

```
value := ? | literal | SESSION.table_name
```

```
literal := numeric_literal | string_literal | hex_string
```

Named and Unnamed Parameters

Parameters can be named or unnamed, but mixing of named and unnamed parameters is not allowed. Dynamic parameters can also be named using CallableStatement methods introduced with JDBC 3.0. Literals can only be named using the syntax provided above. All Ingres database procedure parameters are named.

When connecting to an Ingres 9.x or earlier DBMS Server, the use of named procedure parameters is encouraged. (Using named parameters improves performance by eliminating a query of the database catalog to assign names to the parameters based on the declared order of the procedure parameters.) When connecting to an Ingres 10 or later DBMS Server, named procedure parameters are not required.

The Ingres JDBC Driver provides support for parameter default values by allowing parameter values to be omitted. This support is intended primarily for ODBC positional parameters. For Ingres named parameters, default values can be used simply by omitting the parameter entirely.

Additional Parameter Considerations

Ingres supports the parameter attributes IN, OUT, and INOUT when creating database procedures. When invoking a database procedure, the Ingres JDBC Driver marks a parameter as IN when an input value is set using a `CallableStatement.setXXX()` method. Registering a parameter for output using a `CallableStatement.registerOutParameter()` method will mark the parameter as OUT. Setting a value and registering for output will mark a parameter as INOUT. All dynamic parameters must have an input value assigned and/or be registered for output prior to executing the procedure.

Ingres database procedure parameters can also be passed by value or reference when not explicitly marked with IN, OUT, or INOUT attributes. The Ingres JDBC Driver treats parameters passed by value as IN parameters, and parameters passed by reference (BYREF) as INOUT parameters. If an input value is not provided for a parameter registered for output, the driver sends a NULL value of the output type registered for that parameter.

Ingres Global Temporary Table procedure parameters are specified by providing a parameter value in the form `session.table_name`. In this parameter, `table_name` is the name of the Global Temporary Table, and `'session.'` identifies the parameter as a Global Temporary Table parameter.

Executing Procedures

The `CallableStatement` methods `executeQuery()` and `execute()` can be used to execute a row-producing procedure. The methods `executeUpdate()` and `execute()` can be used for non-row-producing procedures. Ingres does not permit output parameters with procedures that return rows.

Procedure return values, output parameter values and rows returned by row-producing procedures are accessed by standard JDBC methods and interfaces. The `CallableStatement` `getXXX()` methods are used to retrieve procedure return and output parameter values. Rows returned by a procedure are accessed using the `ResultSet` returned by the `CallableStatement` `getResultSet()` method.

Ingres database procedures permit the use of the transaction statements COMMIT and ROLLBACK, however, the use of these statements is highly discouraged!

Using these statements in a procedure executed by the Ingres JDBC Driver can result in the unintentional commitment or rollback of work done prior to procedure execution. It is also possible that a change in transaction state during procedure execution can be interpreted as a transaction abort. For these reasons, applications must make sure that no transaction is active prior to executing a database procedure that contains COMMIT or ROLLBACK statements.

BLOB Column Handling

Large Data Objects

Long, variable length data can be stored in columns of type LONG BYTE, LONG VARCHAR, and LONG NVARCHAR. Columns of these types are collectively referred to as Large Objects (LOB) and are further distinguished as binary (BLOB), character (CLOB), and National Character Set (NCS or Unicode – NLOB). Handling values of these types is somewhat different than smaller, fixed length types such as integers and strings and, depending on the representation, can place restrictions on how the data is retrieved and used, and impact the performance of an application.

The Ingres JDBC driver can represent LOB values in three different ways:

- As a data stream within the application
- As a reference (called a LOCATOR) to the value stored in the database
- As a cached value

When first introduced, Ingres LOB values were only represented as streams of bytes or characters. Ingres 9.2 introduces the capability of retrieving a LOCATOR reference to a LOB value and accessing the value as it resides in the database through the reference. In addition, the Ingres JDBC driver provides the capability of loading a LOB data stream or LOCATOR reference and caching the LOB data in the driver.

These three representations, how they are manifested in JDBC, and the impact they have on an application are discussed here.

LOB Data Streams

A LOB data stream value accompanies the other values in a set of parameters or columns. LOB data streams are serialized in order with the other values they accompany and must be processed entirely before accessing the values that follow. LOB data streams can be accessed only once per value. The driver declares LOB values, when represented as data streams, to be of type LONGVARBINARY or LONGVARCHAR.

A LOB data stream must be accessed and read completely prior to accessing any value that follows the LOB in a result set. When a value is accessed that follows an unaccessed or partially accessed LOB data stream, the driver must read and discard the remaining LOB data so that the requested value can be accessed. If an attempt is subsequently made to access the discarded LOB value, an SQLException is generated indicating that the LOB data is no longer accessible.

LOB data streams must also be read fully before making any further request on the associated connection. Because data from the DBMS Server is serialized on the connection, the results from additional requests on the connection are queued behind any unread LOB data. The Ingres JDBC Driver avoids conflicts resulting from multiple simultaneous requests on a connection by locking the connection for the duration of each request.

When a LOB data stream value is present in a result set, the connection is not unlocked until all the data in the row, including the LOB data, has been read. An attempt to make an additional request on a connection when a LOB column has not been read completely generates an `SQLException` indicating that a request was made before the prior request had completed.

LOB data streams can be accessed only once. Because LOB data streams are not cached, only one call (to `getString()`, `getCharacterStream()`, and so on) can be made for each LOB value in each row of the result set. Additional requests to access a LOB data stream value generate an `SQLException` indicating that the LOB data is no longer available.

In general, the following recommendation from the Sun JDBC documentation must be followed: "For maximum portability, columns within a row must be read in left-to-right order, and each column must only be read once. This reflects implementation limitations in some underlying database protocols."

A result set containing a LOB data stream value is not able to perform `READONLY` cursor pre-fetching. Only one row of a result set is retrieved with each DBMS Server access when a LOB data stream is present. While this does not directly affect the JDBC application, row fetch performance is reduced when a result set contains a LOB data stream value.

The restrictions associated with LOB data streams can be avoided by configuring the driver to cache LOB data streams when received from the DBMS. When enabled, the driver reads LOB data streams as they are received and stores them in memory. All row column values are fully loaded when control is returned to the application.

An uncached LOB data stream can also be cached by accessing it using the `getBlob()` or `getClob()` method. Calling one of these methods satisfies the restrictions associated with LOB data streams and allows extended access to the LOB data using the Blob/Clob interface.

For further details on caching LOB data streams, see [Cached LOB Values](#) below.

LOB Locators

A LOB Locator is a reference to a LOB value stored in a database. Locators reduce the overhead of retrieving the entire LOB data value during row processing. Applications can use a Locator reference to retrieve the LOB data when and if it is determined that the data is needed. A Locator reference can also be used to perform certain operations on the LOB data while it resides in the database. The driver declares LOB Locator values to be of type BLOB or CLOB and wraps Locator values in objects which implement the JDBC Blob and Clob interfaces.

By default, the driver utilizes LOB Locators when supported by the DBMS. The driver can be configured to use LOB data streams instead of Locators by setting the following system property:

```
ingres.jdbc.lob.locators.enabled=false
```

When select loops are enabled, the DBMS streams all result rows back to the driver. Row returning database procedures also stream result rows. While a result stream is active, no other DBMS request is permitted. The driver does not utilize LOB Locators by default when result row streams are active since they cannot be used to access LOB data until after the result set is closed. The driver can be configured to use LOB Locators with result row streams by setting the following system property (LOB Locators must be enabled in general for this property to take effect):

```
ingres.jdbc.lob.locators.select_loop.enabled=true
```

Locators are valid during the transaction in which they are produced. Autocommit imposes a number of restrictions, depending on the autocommit mode, on the use of LOB Locators.

DBMS Mode

Locators remain valid during autocommit. Since the DBMS only supports a single active cursor during autocommit, Locators cannot be used to access a LOB value while a result set is active.

Single-Cursor Mode

Using a Locator to access a LOB value will cause any active result set to be closed.

Multi-Cursor Mode

Locators can be used to access a LOB value while the associated result set is active. Since autocommit is being simulated with standard transactions, all associated LOB Locators become invalid and unusable when their associated result set is closed.

Due to these restrictions, the driver does not utilize LOB Locators by default when autocommit is enabled. The driver can be configured to utilize LOB Locators during autocommit by setting the following system property (LOB Locators must be enabled in general for this property to take effect):

```
ingres.jdbc.lob.locators.autocommit.enabled=true
```

LOB values can be accessed through a Locator using the JDBC Blob/Clob objects returned by the ResultSet methods `getBlob()` and `getClob()`. The Ingres DBMS supports using LOB Locators to determine the length of the LOB data, search the LOB data, and read portions of the LOB data or the entire LOB value.

LOB values can also be accessed by using the other `getXXX()` methods supported for LOB data streams. The LOB value is retrieved from the database and converted to the form appropriate for the particular access method.

The Ingres DBMS does not support modifying LOB values using Locators. The Ingres JDBC driver supports the Blob/Clob modification methods by reading and caching the LOB data and performing the modification on the cached value.

Cached LOB Values

The Ingres JDBC driver will cache a LOB value in three circumstances:

- Caching of LOB data streams has been enabled with the system property `ingres.jdbc.lob.cache.enabled`.
- The application calls `getBlob()` or `getClob()` for a `LONGVARBINARY` or `LONGVARCHAR` column.
- The application calls a Blob/Clob modification method on an object representing a LOB Locator.

The driver can be configured to automatically cache LOB data streams by setting the following system property:

```
ingres.jdbc.lob.cache.enabled=true
```

Automatically caching LOB values requires sufficient memory to hold all active LOB values. Memory resources may be severely impacted when caching is enabled. To reduce the impact of extremely large LOB values, the LOB cache stores values as a series of blocks or segments. The default size of a segment is 8192 bytes or characters. The segment size is a trade off between the number of segments needed to store a value, the size of memory blocks needed to hold a segment, and the amount of unused space in the last segment. The segment size can be configured using the following system property:

```
ingres.jdbc.lob.cache.segment_size=<size>
```

The driver provides compatibility between the LOB data stream and Locator representations by allowing the same `getXXX()` method calls for both types. The driver supports `getBlob()` and `getClob()` methods for LOB data streams by caching the LOB data in the Blob/Clob object.

The driver supports Blob/Clob methods that write or truncate LOB values by reading the LOB data from the database (if necessary) and caching the data in the Blob/Clob object. Modify operations therefore modify a copy of the LOB data stored in the driver. The modified data is not automatically propagated to the database. An application can write modified LOB data back to the database by updating the row holding the LOB and providing the Blob/Clob object holding the modified data as a parameter using the `setBlob()`, `setClob()`, `updateBlob()`, or `updateClob()` methods.

Date/Time Columns and Values

The Ingres DBMS uses the timezone and date format of the client to perform various types of processing of data values. By default, the Ingres JDBC Driver uses the Java/JDBC conventions for dates by setting the client timezone to GMT and the date format to match that specified by JDBC. When using these settings, the Ingres JDBC Driver manipulates date/time values to match the requirements of both the DBMS and JDBC.

Because the DBMS does not have the actual client timezone, the following restrictions exist:

- Ingres date literal formats are not supported. JDBC specifies the format for date, time, and timestamp literals using the following escape clause syntax:

Literal	Syntax
date	{d 'yyyy-mm-dd'}
time	{t 'hh:mm:ss'}
timestamp	{ts 'yyyy-mm-dd hh:mm:ss.f...'} f... is optional

- These escape clauses must be used to include date, time, and timestamp literals in SQL text. Applications can use other date/time formats by using the classes `java.sql.Date`, `java.sql.Time`, `java.sql.Timestamp`, and `java.util.Date` with an appropriately configured date formatter (`java.text.DateFormat`).
- Ingres specific date processing, such as intervals and date functions, causes problems associated with the difference between GMT and the actual client timezone and must be avoided.

The Ingres JDBC Driver allows the Ingres timezone and date format to be passed to the DBMS. For more information, see JDBC Driver Properties (see page 229). When these property values are provided, all Ingres date processing is supported in addition to the JDBC functionality listed above.

Note: The Ingres timezone provided must correspond to the Java client default timezone. Using an arbitrary timezone results in time values that differ by the relative timezone offsets.

The Ingres JDBC Driver supports Ingres empty dates (") by returning the JDBC date/time epoch values ('1970-01-01','00:00:00') for methods `getDate()`, `getTime()` and `getTimestamp()` and a zero-length string for `getString()`. In addition, a `DataTruncation` warning is created by the driver when an empty date is returned by any of these methods. An application checks for the warning by calling the `getWarnings()` method after calling one of the previously mentioned methods. An Ingres empty date is different than a `NULL` value, and cannot be detected using the `wasNull()` method.

A `DataTruncation` warning is also created for Ingres date-only values (no time component) for the same conditions described for empty dates. While an Ingres date-only value is comparable to a JDBC `DATE` value, Ingres date columns are described as being JDBC `TIMESTAMP` types and date-only values are technically a truncation of that type.

The driver can also be configured to return an alternate value for Ingres empty dates. The system property `ingres.jdbc.date.empty` can be set to a standard JDBC format date/time value to be returned in place of empty date values. This property can also be set to null to have empty dates treated as null values. A setting of default or empty results in the behavior described above.

Ingres interval values are not supported by the methods `getDate()`, `getTime()`, and `getTimestamp()`. An exception is thrown if an Ingres date column containing an interval value is accessed using these methods. Ingres interval values can be retrieved using the `getString()` method. Because the output of `getString()` for an interval value is not in a standard JDBC date/time format, the Ingres JDBC Driver creates a warning that can be checked by calling the `getWarnings()` method following the call to `getString()`.

National Character Set Columns

The Ingres JDBC Driver supports the Ingres data types of nchar, nvarchar, and long nvarchar. Retrieval of National Character Set values is done transparently through the existing getXXX() ResultSet methods.

When using character parameters for a PreparedStatement, the data type sent by the driver is determined by the JDBC methods used to assign the parameter value, and the data types supported by the target database.

The JDBC parameter methods and resulting Ingres parameter data type for both standard and National Character Set databases are as follows:

Method	Standard Data Type	NCS Database Data Type
setString()	varchar	nvarchar
setAsciiStream()	long varchar	long nvarchar
setUnicodeStream()	long varchar	long nvarchar
setCharacterStream()	long varchar	long nvarchar
setObject(char[])	char	nchar
setObject(String)	varchar	nvarchar
setObject(Reader)	long varchar	long nvarchar
setObject(obj,CHAR)	char	nchar
setObject(obj,VARCHAR)	varchar	nvarchar
setObject(obj,LONGVARCHAR)	long varchar	long nvarchar
setObject(char[],OTHER)	char	char
setObject(String,OTHER)	varchar	varchar
setObject(Reader,OTHER)	long varchar	long varchar

Note: The driver's use of National Character Set parameters can be overridden using the JDBC SQL type of OTHER in the setObject() method.

Data Type Compatibility

With the exception of the data types listed in Unsupported JDBC Features (see page 227), the Ingres JDBC Driver supports conversion of Ingres data values into Java/JDBC values as required by the JDBC specification.

Because Ingres does not support all the JDBC data types, the following conventions are used when sending Java/JDBC parameters to the DBMS:

NULL

Generally, NULL values sent to the DBMS are associated with the data type provided in the `setNULL()` or `setObject()` method call or the data type implied by the `setXXX()` method call. A generic or typeless NULL value can be sent to the DBMS using one of the following method calls:

```
setNull( idx, Types.NULL )  
setObject( idx, null )  
setObject( idx, null, Types.NULL )
```

BOOLEAN

Boolean values are sent to the DBMS as single byte integers with the value 0 or 1.

BIGINT

Long values are sent to the DBMS as DECIMAL (if supported by the DBMS) or DOUBLE values when BIGINT is not supported by the DBMS.

DECIMAL

BigDecimal values are sent as DOUBLE values when DECIMAL is not supported by the DBMS. Avoid using the BigDecimal constructor that takes a parameter of type double. This constructor can produce decimal values that exceed the scale/precision supported by Ingres.

DATE

For earlier versions of Ingres and Enterprise Access gateways in which ANSI date/time data types are not supported, Ingres supports a single date data type, which is used for DATE, TIME, and TIMESTAMP values. Ingres dates do support date without time values and this form is used for JDBC DATE values.

TIME

For earlier versions of Ingres and Enterprise Access gateways in which ANSI date/time data types are not supported, Ingres supports a single date data type that is used for DATE, TIME, and TIMESTAMP values. Ingres dates do not support date without time values. The Ingres JDBC Driver adds the JDBC date epoch 1970-01-01 to JDBC TIME values. The Ingres DBMS adds the current date to time-only values.

CHAR

Zero length CHAR values are sent as VARCHAR values. For conventions associated with NCS enabled databases, see National Character Set Columns (see page 255). For information on automatic conversion to LONGVARCHAR, see the end of this section.

VARCHAR

For conventions associated with NCS enabled databases, see National Character Set Columns (see page 255). For information on automatic conversion to LONGVARCHAR, see the end of this section.

LONGVARCHAR

The LONGVARCHAR type is used by the driver to represent Character and NCS Large Object values passed to the driver as data streams. For conventions associated with NCS enabled databases, see National Character Set Columns (see page 255).

BINARY

Zero length BINARY values are sent as VARBINARY values.

LONGVARBINARY

The LONGVARBINARY type is used by the driver to represent Binary Large Object values passed to the driver as data streams.

BLOB

The BLOB type is used by the driver to represent Locators to Binary Large Object values residing in the database. Blob objects can be used to access the blob value. Blob values in the database are read-only, therefore Blob objects load and cache the referenced blob value locally when modification requests are made.

Driver handling of Blob parameters is dependent on the value represented by the Blob object. If a Blob object represents a Locator associated with the connection, the driver sends the Locator as the parameter value. If a Blob object represents a Locator associated with a different connection, a cached blob value, or is an object which did not originate from the driver, the driver sends the blob parameter value as a LONGVARBINARY data stream.

CLOB

The CLOB type is used by the driver to represent Locators to Character and NCS Large Object values residing in the database. Clob objects can be used to access the clob value. Clob values in the database are read-only, therefore Clob objects cache the referenced clob value when modification requests are made.

Driver handling of Clob parameters is dependent on the value represented by the Clob object. If a Clob object represents a Locator associated with the connection, the driver sends the Locator as the parameter value. If a Clob object represents a Locator associated with a different connection, a cached clob value, or is an object which did not originate from the driver, the driver sends the clob parameter value as a LONGVARCHAR data stream.

In addition to the JDBC types listed above, the following conventions are used when certain Java data values are provided to the setObject() method:

byte[]

Byte arrays are sent by default as VARBINARY values.

char[]

While not required by JDBC, character arrays are supported by the Ingres JDBC Driver and are sent by default as CHAR values. For conventions associated with NCS enabled databases, see National Character Set Columns (see page 255). For information on automatic conversion to LONGVARCHAR, see the end of this section.

String

Strings are sent by default as VARCHAR values. For conventions associated with NCS enabled databases, see National Character Set Columns (see page 255). For information on automatic conversion to LONGVARCHAR, see the end of this section.

InputStream

While not required by JDBC, InputStream objects are supported by the Ingres JDBC Driver and are sent by default as LONGVARBINARY values.

Reader

While not required by JDBC, Reader objects are supported by the Ingres JDBC Driver and are sent by default as LONGVARCHAR values. For conventions associated with NCS enabled databases, see National Character Set Columns (see page 255).

JDBC requires BINARY, VARBINARY, CHAR, and VARCHAR parameter values to be converted to LONGVARBINARY/LONGVARCHAR when their length exceeds some DBMS dependent maximum.

The default maximum used by the Ingres JDBC driver is 2000 bytes. This default maximum value can be incorrect for an Ingres database that has been configured with non-default page sizes and for EDBC or Enterprise Access gateways.

The Ingres driver uses the following entries in the iidbcapabilities system catalog to determine at runtime the appropriate size limits:

```
SQL_MAX_BYTE_COLUMN_LEN
SQL_MAX_VBYT_COLUMN_LEN
SQL_MAX_CHAR_COLUMN_LEN
SQL_MAX_VCHR_COLUMN_LEN
```

Not all releases of the Ingres DBMS, EDBC, and Enterprise Access gateways have these entries in their iidbcapabilities system catalogs. These entries can be entered manually to provide accurate size information for the Ingres driver. Depending on the DBMS involved, special permissions are required to update the system catalog.

JDBC Tracing

The Ingres JDBC Driver supports both DriverManager and DataSource tracing as documented in the JDBC 3.0 API specification. Trace information consists of JDBC API method entry and exit points with corresponding parameter and return values.

Enable internal Ingres JDBC Driver tracing by defining system properties on the java command line (-D flag) or by including the properties in the driver properties file.

DBMS trace messages are written to the internal trace log and can be directed to a separate trace log specified by a driver property.

The following properties are supported:

Property	Value	Description
ingres.jdbc.trace.log	<i>log</i>	Path and file name of the Ingres JDBC Driver trace log
ingres.jdbc.trace.driv	<i>0 - 5</i>	Tracing level for the Ingres JDBC Driver
ingres.jdbc.trace.ds	<i>0 - 5</i>	Tracing level for the Ingres JDBC DataSources
ingres.jdbc.trace.msg	<i>0 - 5</i>	Tracing level for Messaging I/O
ingres.jdbc.trace.msg.tl	<i>0 - 5</i>	Tracing level for Transport Layer I/O
ingres.jdbc.trace.msg.nl	<i>0 - 5</i>	Tracing level for Network Layer I/O
ingres.jdbc.trace.timestamp	<i>true</i>	Include timestamp in trace log

Property	Value	Description
ingres.jdbc.dbms.trace.log	<i>log</i>	Path and file name of the DBMS trace log
edbc.trace.log	<i>log</i>	Path and file name of the EDBC JDBC Driver trace log
edbc.trace.id	<i>level</i>	Tracing level for the EDBC JDBC Driver
edbc.trace.timestamp	<i>true</i>	Include timestamp in EDBC trace log

Internal tracing is also enabled by the application using the following Ingres JDBC Driver methods:

Method	Parameters	Description
setTraceLog(String)	<i>log</i>	Log file path and name
setTraceLevel(int)	<i>level</i>	Tracing level for ID 'drv'
setTraceLevel(String,int)	<i>id, level</i>	Trace ID and numeric tracing level

Internal driver tracing permits separate tracing level settings for the following trace IDs (*id*):

Trace ID	Description
drv	General driver tracing
ds	Data source tracing
msg	General messaging IO tracing
msg.tl	IO tracing: transport layer
msg.nl	IO tracing: network layer

Tracing Levels

The tracing level determines the type of information that is logged. The following levels are currently defined:

- 1 – Errors and exceptions
- 2 – High level method invocation
- 3 – High level method details
- 4 – Low level method invocation
- 5 – Low level method details

Chapter 12: Understanding .NET Data Provider Connectivity

This section contains the following topics:

- [.NET Data Provider](#) (see page 263)
- [.NET Data Provider Architecture](#) (see page 264)
- [Code Access Security](#) (see page 268)
- [.NET Data Provider Classes](#) (see page 268)
- [Data Types Mapping](#) (see page 334)
- [IngresDataReader Object—Retrieve Data from the Database](#) (see page 337)
- [ExecuteNonQuery Method—Modify and Update Database](#) (see page 340)
- [How Database Procedures Are Called](#) (see page 342)
- [Integration with Visual Studio](#) (see page 344)
- [Application Configuration File—Troubleshoot Applications](#) (see page 354)
- [Obtain File Version of the Data Provider](#) (see page 355)

This chapter describes the Ingres .NET Data Provider. It also explains how components and wizards in the provider objects help integrate the Ingres .NET Data Provider with MS Visual Studio to aid in the development of .NET applications that access Ingres data.

.NET Data Provider

The Ingres .NET Data Provider is a Microsoft .NET component that provides native .NET connectivity to Ingres databases to deliver Ingres data to the Microsoft .NET Framework. It uses the Data Access Server to access Ingres data sources.

The Ingres .NET Data Provider also supports .NET access to Enterprise Access and ODBC data sources.

.NET Data Provider Architecture

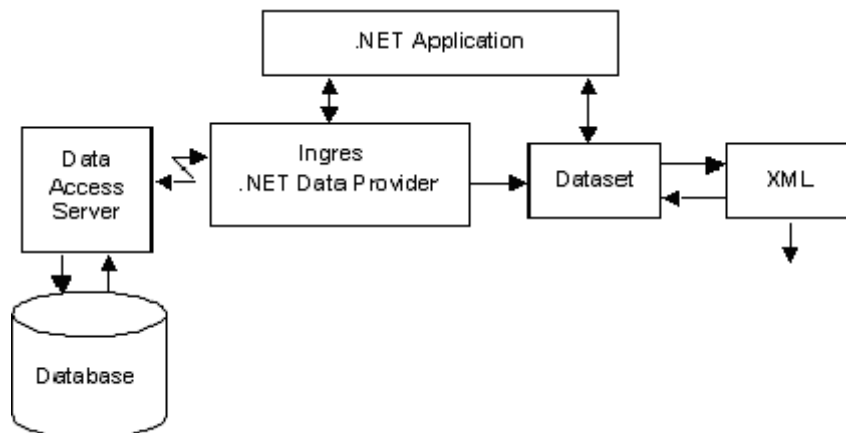
The Ingres .NET Data Provider offers a series of .NET types to describe the user's data, .NET provider classes to manipulate the data, and connection pooling to efficiently manage data connections.

The design and naming conventions of the Ingres .NET Data Provider's data types, classes, properties, and methods follow the same pattern as the Microsoft .NET Data Providers. Consequently, developers who are familiar with the Microsoft providers can easily develop or convert existing code from Microsoft databases to Ingres databases.

All Ingres .NET Data Provider modules are written in C#, a managed .NET language with full access to every .NET Framework capability. Even though the data provider is written in C#, any managed language such as VB.NET or J# can use the data provider because of .NET's language interoperability feature.

Data Provider Data Flow

A data provider in the Microsoft .NET Framework enables a connection to a data source to retrieve and modify data from that data source. Data coming out of a .NET data provider can be used directly by an application or it can be redirected into an ADO.NET DataSet where it can be processed by other application methods such as XML processing. The following figure shows the flow of data in and out of the data provider.



As shown in this figure, the Ingres .NET Data Provider uses an intermediate server called the Data Access Server to access Ingres databases.

For additional information on this server, see the chapter "Configuring the Data Access Server."

Note: The Ingres .NET Data Provider does not require Ingres Net for database connectivity. For best performance, the data provider directly communicates with the wire.

Data Provider Assembly

The Ingres .NET Data Provider includes the Ingres.Client.dll, which contains the Ingres.Client assembly. The Ingres.Client assembly contains the base runtime support.

This assembly is installed as part of a standard Ingres client installation, which automatically registers it in the Global Assembly Cache (GAC). The Ingres.Client.dll that contains the Ingres.Client assembly is also installed, by default, into the directory C:\Program Files\Ingres\Ingres .NET Data Provider\v2.1.

Data Provider Namespace

When developing .NET applications, programmers use the data types, components, and other classes in the data provider by referencing each name as defined in the namespace for the classes. The namespace for the Ingres .NET Data Provider is:

Ingres.Client

Data Retrieval Strategies

The Ingres .NET Data Provider provides ADO.NET programmers with two access strategies for retrieving data:

- **DataReader**—This program retrieves the data for read-only, forward-only access. The program opens the connection, executes the command, processes the rows in from the reader, closes the reader, and closes the connection. Resources on the database server are held until the connection is closed. For additional information, see *IngresDataReader Class* (see page 303).
- **DataAdapter**—This program opens the connection, fills a *DataSet*, closes the connection, processes the *DataSet*, opens the connection, updates the database, and closes the connection. Resources on the database server are not held during the processing of the *DataSet*. Using connection pooling, usually only one physical connection is used. For additional information, see *IngresDataAdapter Class* (see page 310).

In addition to the low-level *DataReader* and *DataAdapter* access strategies, the data provider works with Visual Studio to support *TableAdapter*, *SqlConnection*, and *DataSource* components. The data provider uses standard base classes, interfaces, and metadata methods to support higher level data-bound .NET Framework controls. .NET Framework and Visual Studio work with the Ingres .NET Data Provider to offer more rapid application development and high quality code.

Connection Pooling

Connection pooling significantly enhances the performance and scalability of some applications. Physical connections are kept in a pool after they are no longer needed by one application and are dispensed later to the same or another application (as needed) to avoid the cost of connections. All connections in a pool have identical connection strings. A new pool is created for each connection that has a different connection string from all other pools.

When an application attempts to connect to a database, the `Open` method of the `IngresConnection` object uses the connection pool to search for a physical connection that specifies the same `ConnectionString` parameters. If no match is found, a new physical connection is made to the database. If a match *is* found, a connection is returned to the application from the pool.

After the application has completed its work, committed its changes, freed its database locks and resources, and closed the connection, the physical connection is retained by the .NET data provider and placed back in the connection pool instead of being physically disconnected.

This connection is available to the same application later in the application's life span or to other applications in the same process with the same connection parameters. This avoids the overhead and delay of opening a new physical connection. An application can choose to disable connection pooling by specifying "Pooling=no" in its connection string.

The application is unaware of the connection reuse. If the connection is not reused within three minutes, the connection is physically closed to release system resources.

However, if the number of connections in the pool falls below the minimum number of connections specified by the application in the "Min Pool Size=n" value in the connection string, the connection is not physically closed and is retained in the pool for later use. For additional information on connection pooling, see `IngresConnection` Class (see page 277).

Code Access Security

The Ingres .NET Data Provider assembly requires the FullTrust permission to load, access the network, read and write certain files, and use other system resources.

The Ingres .NET Data Provider is a strongly named assembly, making it eligible for installation into the Global Assembly Cache and resistant to code tampering.

The Ingres.Client assembly contains the attribute AllowPartiallyTrustedCallersAttribute (APTCA) to allow the Ingres .NET Data Provider to be called by a partially trusted assembly. APTCA has no effect on Ingres database security. Ingres database security checks are performed as usual.

.NET Data Provider Classes

The Ingres .NET Data Provider is the runtime component that provides the interface between the .NET application and Ingres.

The Ingres .NET Data Provider namespace (Ingres.Client) and its contents follow the same pattern as the Microsoft data providers.

All public static members are safe for multithreaded operations. To reduce unnecessary overhead, instance members are not guaranteed to be thread-safe. If a thread-safe operation on the instance is needed, wrap the operation in one of .NET's System.Threading synchronization methods to protect the state of the critical section of code.

The base class and interface definition for each class is provided in C# and VB.NET syntax as shown below. However, .NET's language interoperability feature allows any managed language to use the Ingres .NET Data Provider.

C#: Public sealed class IngresParameter :
System.Data.Common.DbParameter, IDataParameter, IDbDataParameter,
ICloneable

VB.NET: NotInheritable public class IngresParameter

Inherits System.Data.Common.DbParameter
Implements IDataParameter, IDbDataParameter, ICloneable

For more information on data provider classes, including information on other .NET language syntax and inherited methods and properties, see the *Microsoft .NET Framework Developer's Guide* and Microsoft .NET Framework Class Library documentation.

IngresCommand Class

The IngresCommand class represents an SQL command or a database procedure that executes against an Ingres or Enterprise Access database.

Parameter placeholders in the SQL command text are represented by a question mark (?).

Database procedures can be invoked by either setting `CommandText="myproc"` and `CommandType=CommandType.StoredProcedure`, or by using the escape sequence format and setting `CommandText="{ call myproc }"` and `CommandType=CommandType.Text`.

Ingres .NET Data Provider does not currently support the following features:

- Multiple active results-sets
- Batched commands consisting of multiple Ingres SQL commands in one IngresCommand object
- Support for Ingres SQL command COPY TABLE
- Support for Ingres SQL command SAVEPOINT
- `IngresCommand.ExecuteReader(CommandBehavior.SchemaOnly)` is supported for SELECT commands only

IngresCommand Class Declaration

The IngresCommand class declarations are:

C#: public sealed class IngresCommand :
System.Data.Common.DbCommand, IDbCommand, IDisposable, ICloneable

VB.NET: NotInheritable Public Class IngresCommand
Inherits System.Data.Common.DbCommand
Implements IDbCommand, IDisposable, ICloneable

IngresCommand Class Example

```
IngresCommand cmd = new IngresCommand(  
"SELECT id, name FROM employee WHERE id = ?");
```

IngresCommand Class Properties

The IngresCommand class properties are:

Property	Accessor	Description
CommandText	get set	SQL statement string to execute or procedure name to call.
CommandTimeout	get set	The time, in seconds, for an attempted query to time-out if the query has not yet completed. Default is 30 seconds.
CommandType	get set	An enumeration describing how to interpret the CommandText property. Valid values are Text, TableDirect, or StoredProcedure.
Connection	get set	The IngresConnection object that is used to identify the connection to execute a command. For more information, see IngresConnection Class (see page 277).
Parameters	get	The IngresParameterCollection for the parameters associated with the SQL query or database procedure. For more information, see IngresParameterCollection Class (see page 327).
Transaction	get set	The IngresTransaction object in which the IngresCommand executes. This transaction object must be compatible with the transaction object that is associated with the Connection, (that is, the IngresTransaction must be the object (or a copy) returned by IngresConnection.BeginTransaction).
UpdateRowSource	get set	Defines how results are applied to a rowset by the DbDataAdapter.Update method. (Inherited from DbDataAdapter.)

IngresCommand Class Public Methods

The public methods for the IngresCommand class are:

Method	Description
Cancel	Cancels the execution of the SQL command or database procedure.
CreateParameter	Creates a new instance of IngresParameter. For more information, see IngresParameter Class (see page 321).
Dispose	Releases allocated resources of the IngresCommand and base Component.
ExecuteNonQuery	Executes a command that does not return results. Returns the number of rows affected by the update, delete, or insert SQL command.
ExecuteReader	Executes a command and builds an IngresDataReader. For more information, see IngresDataReader Class (see page 303).
ExecuteScalar	Executes a command and returns the first column of the first row of the result set.
Prepare	Prepares the SQL statement to be executed later.
ResetCommandTimeout	Resets the CommandTimeout property to its default value of 30 seconds.

IngresCommand Class Constructors

The constructors for the IngresCommand class are:

Constructor Overloads	Description
IngresCommand()	Instantiates a new instance of the IngresCommand class using default property values
IngresCommand(string)	Instantiates a new instance of the IngresCommand class using the defined SQL command or database procedure
IngresCommand(string, IngresConnection)	Instantiates a new instance of the IngresCommand class using the defined SQL command or database procedure and the connection to the Ingres or Enterprise Access database
IngresCommand(string, IngresConnection, IngresTransaction)	Instantiates a new instance of the IngresCommand class using the defined SQL command or database procedure, the connection to the Ingres or Enterprise Access database, and the IngresTransaction object

Sample Program Constructed with .NET Data Provider

To construct an application using the Ingres .NET Data Provider, the developer creates a series of objects from the data provider's classes. The following is a simple C# program employing four data provider classes.

.NET 2.0 Programming Model

```
using System;
using System.Configuration;
using System.Data;
using System.Data.Common;
using System.IO;
using Ingres.Client;

class App
{
    static public void Main()
    {
        ConnectionStringSettingsCollection connectionSettings =
            ConfigurationManager.ConnectionStrings;
        if (connectionSettings.Count == 0)
            throw new InvalidOperationException(
                "No connection information specified in application configuration
                file.");
        ConnectionStringSettings connectionSetting = connectionSettings[0];

        string invariantName = connectionSetting.ProviderName;
        string myConnectionString = connectionSetting.ConnectionString;

        DbProviderFactory factory = DbProviderFactories.GetFactory(invariantName);

        DbConnection conn =
            factory.CreateConnection();
        conn.ConnectionString = myConnectionString;

        conn.Open(); // open the Ingres connection

        string cmdtext =
            "select table_owner, table_name, " +
            " create_date from iitables " +
            " where table_type in ('T','V') and " +
            " table_name not like 'ii%' and " +
            " table_name not like 'II%'";
        DbCommand cmd = conn.CreateCommand();
        cmd.CommandText = cmdtext;

        // read the data using the DataReader method
        DbDataReader datareader = cmd.ExecuteReader();

        // write header labels
        Console.WriteLine(datareader.GetName(0).PadRight(18) +
            datareader.GetName(1).PadRight(34) +
            datareader.GetName(2).PadRight(34));
        int i = 0;
        while (i++ < 10 && datareader.Read())
            // read and write out a few data rows
        {
            // write out the three columns to the console
```

```
        Console.WriteLine(
            datareader.GetString(0).Substring(0,16).PadRight(18) +
            datareader.GetString(1).PadRight(34) +
            datareader.GetString(2));
    }
    datareader.Close();

    DataSet ds = new DataSet("my_list_of_tables");
    //      read the data using the DataAdapter method
    DbDataAdapter adapter = factory.CreateDataAdapter();
    DbCommand adapterCmd = conn.CreateCommand();
    adapterCmd.CommandText = cmdtext;
    adapter.SelectCommand = adapterCmd;
    adapter.Fill(ds); // fill the dataset

    //      write the dataset to an XML file
    ds.WriteXml("c:/temp/temp.xml");

    conn.Close(); // close the connection
} // end Main()
} // end class App
```

.NET 1.1 Programming Model

```
using System;
using System.IO;
using System.Data;
using Ingres.Client;

class App
{
    static public void Main()
    {
        string myConnectionString =
            "Host=myserver.mycompany.com;" +
            "User Id=myname;PWD=mypass;" +
            "Database=mydatabase";
        IngresConnection conn = new IngresConnection(
            myConnectionString );
        conn.Open(); // open the Ingres connection

        string cmdtext = "select table_owner, table_name, " +
            "create_date from iitables " +
            " where table_type in ('T','V') and " +
            " table_name not like 'ii%' and" +
            " table_name not like 'II%'";
        IngresCommand cmd = new IngresCommand(cmdtext, conn);

        //      read the data using the DataReader method
        IngresDataReader datareader = cmd.ExecuteReader();

        //      write header labels
        Console.WriteLine(datareader.GetName(0).PadRight(18) +
            datareader.GetName(1).PadRight(34) +
            datareader.GetName(2).PadRight(34));
        int i = 0;
        while (i++ < 10 && datareader.Read())
        // read and write out a few data rows
        {
            // write out the three columns to the console
```

```

        Console.WriteLine(
            datareader.GetString(0).Substring(0,16).PadRight(18) +
            datareader.GetString(1).PadRight(34) +
            datareader.GetString(2));
    }
    datareader.Close();
    DataSet ds = new DataSet("my_list_of_tables");
    //      read the data using the DataAdapter method
    IngresDataAdapter adapter = new IngresDataAdapter();
    adapter.SelectCommand = new IngresCommand(cmdtext, conn);
    adapter.Fill(ds); // fill the dataset

    //      write the dataset to an XML file
    ds.WriteXml("c:/temp/temp.xml");

    conn.Close(); // close the connection
} // end Main()
} // end class App

```

IngresCommandBuilder Class

The IngresCommandBuilder class automatically generates INSERT, DELETE, and UPDATE commands into an IngresDataAdapter object for a simple single-table SELECT query. These commands can be used to reconcile DataSet changes through the IngresDataAdapter associated with the Ingres database.

IngresCommandBuilder Class Declaration

The IngresCommandBuilder class can be declared as follows:

C#: public sealed class IngresCommandBuilder : DbCommandBuilder

VB.NET: NotInheritable Public Class IngresCommandBuilder
Inherits DbCommandBuilder

IngresCommandBuilder Class Properties

The IngresCommandBuilder class properties are:

Property	Accessor	Description
CatalogLocation	get set	Position of the catalog name in a qualified table name.
CatalogSeparator	get set	The string of characters that defines the separation between a catalog name and the table name.
ConflictOption	get set	Controls how to compare for update conflicts.

Property	Accessor	Description
DataAdapter	get set	The IngresDataAdapter object that is associated with the CommandBuilder. The IngresDataAdapter contains the InsertCommand, DeleteCommand, and UpdateCommand objects that are automatically derived from the SelectCommand.
QuotePrefix	get set	The string of characters that are used as the starting delimiter of a quoted table or column name in an SQL statement.
QuoteSuffix	get set	The string of characters that are used as the ending delimiter of a quoted table or column name in an SQL statement.
SchemaSeparator	get set	The string of characters that defines the separation between a table name and column name. Always a period (.)

IngresCommandBuilder Class Methods

The public methods available to the IngresCommandBuilder class are:

Method	Description
Derive Parameters	Retrieves the parameter metadata of the database procedure specified in the IngresCommand object and populates the IngresCommand.Parameters collection.
GetDeleteCommand	Gets the generated IngresCommand to perform DELETE operations on the table.
GetInsertCommand	Gets the generated IngresCommand to perform INSERT operations on the table.
GetUpdateCommand	Gets the generated IngresCommand to perform UPDATE operations on the table.
QuoteIdentifier	Wrap quotes around an identifier.
RefreshSchema	Refreshes the IngresCommandBuilder's copy of the metadata of a possibly changed SELECT statement in the IngresDataAdapter.SelectCommand object.

Method	Description
UnquoteIdentifier	Removes quotes from an identifier.

IngresCommandBuilder Class Constructors

The IngresCommandBuilder class has the following constructors:

Constructor Overloads	Description
IngresCommandBuilder ()	Instantiates a new instance of the IngresCommandBuilder class using default property values
IngresCommandBuilder (IngresDataAdapter)	Instantiates a new instance of the IngresCommandBuilder class using the specified IngresDataAdapter

IngresConnection Class

The IngresConnection class represents an open connection to an Ingres database. This class requires a connection string to connect to a target server and database.

Important! *An application must Close() or Dispose() on the Connection object to return it to the connection pool for reuse by other applications.*

IngresConnection Class Declaration

The IngresConnection class declaration method signature is:

C#: public sealed class IngresConnection :
System.Data.Common.DbConnection, IDbConnection, IDisposable

VB.NET: NotInheritable Public Class IngresConnection
Inherits System.Data.Common.DbConnection
Implements IDbConnection, IDisposable

IngresConnection Class Example

```
IngresConnection conn = new IngresConnection(

"Host=myserver.mycompany.com;Database=mydatabase;" +
"User ID=myuid;Password=mypassword;");

conn.Open( );
```

IngresConnection Class Properties

The IngresConnection class has the following properties:

Property	Accessor	Description
ConnectionString	get set	<p>String that specifies the target server machine and database to connect to, the credentials of the user who is connecting, and the parameters that define connection pooling and security.</p> <p>Default is "".</p> <p>Consists of keyword=value pairs, separated by semicolons. Leading and trailing blanks around the keyword or value are ignored. Case and embedded blanks in the keyword are ignored. Case and embedded blanks in the value are retained. Can only be set if connection is closed. Resetting the connection string resets the ConnectionTimeout and Database properties.</p> <p>For a list of valid keywords and their descriptions, see Connection String Keywords (see page 281).</p>
ConnectionTimeout	get	<p>The time, in seconds, for an attempted connection to abort if the connection cannot be established.</p> <p>Default is 15 seconds.</p>
Database	get	<p>The database name specified in the ConnectionString's Database value.</p> <p>Default is "".</p>
DataSource	get	<p>The name of the target server.</p>
ServerVersion	get	<p>The server version number. May include additional descriptive information about the server. This property uses an IngresDataReader. For this reason, no other IngresDataReader can be active at the time that this property is first invoked.</p>

Property	Accessor	Description
State	get	The current state of the connection: ConnectionState.Closed or ConnectionState.Open.

IngresConnection Class Public Methods

The public methods for the IngresConnection class are:

Method	Description
BeginTransaction	Begins a local transaction. The connection must be open before this method can be called. Nested or parallel transactions are not supported. Mutually exclusive with the EnlistDistributedTransaction method.
ChangeDatabase	Changes the database to be used for the connection. The connection must be closed before this method can be called.
Close	Closes the connection (rollback pending transaction) and returns the connection to the connection pool.
CreateCommand	Creates an IngresCommand object.
Dispose	Closes the connection and releases allocated resources.
EnlistDistributedTransaction	Enlists in an existing distributed transaction (ITransaction). Mutually exclusive with the BeginTransaction method.
EnlistTransaction	Enlists in an existing distributed transaction (System.Transactions.Transaction). Mutually exclusive with the BeginTransaction method.
GetSchema	Returns schema metadata from the Ingres catalog for the specified collection name. Valid collection names include: <ul style="list-style-type: none"> ■ MetadataCollections ■ DataSourceInformation ■ DataTypes ■ Restrictions

Method	Description
	<ul style="list-style-type: none">■ ReservedWords■ Tables■ Views■ Columns■ Indexes■ Procedures■ ProcedureParameters
Open	Opens a database connection or uses one from the connection pool.

IngresConnection Class Events

The events generated by the IngresConnection are:

Event	Description
InfoMessage	Generated when the database returns a warning or informational message.
StateChange	Generated when the State property changes from Closed to Open or from Open to Close. For a definition of State, see IngresConnection Class Properties (see page 278).

IngresConnection Class Constructors

The constructors for the IngresConnection class are:

Constructor Overloads	Description
IngresConnection()	Instantiates a new instance of the IngresConnection class using default property values
IngresConnection(string)	Instantiates a new instance of the IngresConnection class using the defined connection string

Connection String Keywords

Keywords for the `ConnectionString` property of the `IngresConnection` class are as follows.

Connection string keywords are case-insensitive. Certain keywords have synonyms. For example, keywords `Server` and `Address` are synonyms of `Host`. Spaces in values are retained. Values can be delimited by double quotes.

BlankDate

Specifies how an Ingres blank (empty) date result value is to be returned to the application.

`BlankDate=null` means it is returned as a null value.

The default is to return it as a `DateTime` value of "9999-12-31 23:59:59".

Character Encoding

Specifies the .NET character encoding name (for example, `windows-1252`) used for conversions between Unicode and character data types. This keyword allows an alternate .NET character encoding to be specified as an override, or a valid .NET character encoding to be used if the data provider is unable to map the Data Access Server's installation character set. A code page name can also be specified in "cp" format (for example, "cp1252").

Connect Timeout or Connection Timeout

Specifies the time, in seconds, to wait for an attempted connection to time out if the connection has not completed. Default is 15.

Cursor_Mode

Specifies the default cursor concurrency mode, which determines the concurrency of cursors that are not explicitly assigned in the command text (for example, `FOR UPDATE` or `FOR READONLY`). Available options are:

- `readonly` – (Default) Provides non-updatable cursors for best performance
- `update` – Provides updatable cursors
- `dbms` – Concurrency is assigned by the DBMS Server

Database or DB

Specifies the name of the database being connected to. If a server is required, use the syntax `dbname/server_class`.

Date_format or Date_fmt

Specifies the Ingres format for date literals. It corresponds to the Ingres environment variable `II_DATE_FORMAT` and is assigned the same values. This option is not used directly by the data provider, but is sent to the DBMS and affects the parsing of date literals in query text.

Dbms_user

Specifies the user name to be associated with the DBMS session. This name is equivalent to the Ingres `-u` flag, which can require administrator privileges.

Dbms_password

Specifies the DBMS password of the user, which is equivalent to the Ingres `-P` flag.

Decimal_char

Specifies the character that the DBMS Server will use to separate fractional and non-fractional parts of a number. It corresponds to the Ingres environment variable `II_DECIMAL` and is assigned the same values. This option is not used directly by the data provider, but is sent to the DBMS and affects the parsing and construction of numeric literals.

`Decimal_char=','` specifies the comma (,) character.

The default value is the period (.) as in 12.34.

Enlist

Specifies whether the `IngresConnection` in the transaction context is automatically enlisted if the creation thread is within a transaction context as established by `System.EnterpriseServices.ServicedComponent`. Default is true.

Group ID

Specifies the group identifier that has permissions for a group of users.

Host or Server or Address

Specifies the name of the target host server machine with the Data Access Server (DAS).

Multiple host names can be specified as a semicolon-separated list enclosed in parentheses with optional port lists:

```
Server=(hostname1[:port[,port]];hostname2[:port[,port]])
```

Example connection string:

```
Server=(myserver1;myserver2);db=mydatabase;
```

If a port ID is attached to a host name, any additional port IDs outside the list or specified in the `PORT` keyword are ignored. If a host name does not have an explicit port ID in the specification, then port IDs in the list that follows the `PORT` keyword are distributed to the hostname. For example:

```
Host=(myservera:II6;myserverb);Port=II8,II9;Database=mydb;
```

is equivalent to:

```
Host=(myservera:II6;myserverb):II8,II9;Database=mydb;
```

is equivalent to:

```
Host=(myservera:II6;myserverb:II8;myserverb:II9);Database=mydb;
```

To maintain performance, the connection string should specify a hostname and port that the DAS servers typically listen to.

Example:

Assume that four DAS servers are started and listening on symbolic ports II7, II8, II9, and II10. The following IngresConnection.ConnectionString will connect a .NET application to the Ingres database using one of the four ports:

```
Host=myserver;Port=II7,II8,II9,II10;UserID=myuserid;Pwd=mypwd;Database=mydb;
```

or

```
Host=myserver:II7,II8,II9,II10;UserID=myuserid;Pwd=mypwd;Database=mydb;
```

Max Pool Size

Specifies the maximum number of connections that can be in the pool. Default is 100.

Min Pool Size

Specifies the minimum number of connections that can be in the pool. Default is 0.

Money_format or Money_fmt

Specifies the Ingres format for money literals. It corresponds to the Ingres environment variable II_MONEY_FORMAT and is assigned the same values. This option is not used directly by the data provider, but is sent to the DBMS and affects the processing of query text.

Money_precision or Money_prec

Specifies the precision of money data values. It corresponds to the Ingres environment variable II_MONEY_PREC and is assigned the same values. This option is not used directly by the data provider, but is sent to the DBMS and affects the processing of money values.

Password or PWD

Specifies the password to the database. This value may be case-sensitive depending on the target server.

Persist Security Info

Specifies whether password information from the connection string is returned in a get of the ConnectionString property:

false—(Default) Password information is not returned.

true—Password information is returned.

Pooling

Enables or disables connection pooling:

true—(Default) Connection pooling is enabled.

false—Connection pooling is disabled.

Port

Specifies the port number on the target host server machine that the Data Access Server is listening to.

Multiple port numbers, separated by commas, can be specified, one for each configured DAS. For example, "Port=II7,II8,II9,II10;" can be specified if four DAS servers have been configured to listen on each respective port. The data provider attempts to connect to each port, in random order, until a successful connection is made. Default is II7.

Role ID

Specifies the role identifier that has associated privileges for the role.

Role Password or Role PWD

Specifies the role password associated with the Role ID.

SendIngresDate

Specifies whether .NET DateTime parameter data is sent as ingresdate data type rather than as ANSI date/time data type.

false—(Default) Sends as ANSI date/time type.

true—Sends as ingresdate type.

For more information, see SendIngresDate Connection Keyword (see page 286).

Timezone or TZ

Specifies the Ingres time zone associated with the client's location. Corresponds to the Ingres environment variable II_TIMEZONE_NAME and is assigned the same values. This information is not used directly by the data provider, but is sent to the DBMS and affects the processing of dates.

User ID or UID

Specifies the name of the authorized user connecting to the DBMS Server. This value may be case-sensitive depending on the target server.

Vnode_usage

Allows the .NET application to control the portions of the vnode information that are used to establish the connection to the remote DBMS server through the Data Access Server. Options are:

connect

(Default) Only the vnode connection information is used to establish the connection.

login

Both the vnode connection and login information are used to establish the connection.

For further details, see Data Provider User ID Options (see page 285).

User ID Options for the Data Provider

The Ingres .NET Data provider does not require a user ID and password to establish a connection when the Ingres Data Access Server (DAS) is running on the same machine as the .NET client application. When a user ID and password is not provided, the .NET client process user ID is used to establish the DBMS connection.

If the target database name specification includes a VNODE name specification, the VNODE login information is used to access the DBMS machine. Optionally, a user ID and password can be provided and is handled as described below.

When the DAS and DBMS servers are on different machines, a VNODE name is required in the target database specification of the form *vnodename::dbname*. The VNODE provides the connection and (optionally) login information needed to establish the DBMS connection.

The connection string keyword *Vnode_usage* determines how the VNODE is used to access the DBMS. *Vnode_usage* also determines the context (DAS or DBMS) in which the application user ID/password is used. If the target database specification does not contain a VNODE name, the *Vnode_usage* specification is ignored.

When *Vnode_usage* is set to *CONNECT*, only global VNODE connection information is used to establish the DBMS connection. The application-provided user ID and password are used in the DBMS context to access the DBMS machine.

When *Vnode_usage* is set to *LOGIN*, both connection and login VNODE information is used to access the DBMS machine. The application-provided User ID and Password are used in the DAS context, allowing access to private and global VNODEs on the DAS server.

The Ingres .NET Data Provider supports IPv6 addressing. IPv6 addresses should be enclosed in brackets [] because of the different address format—for example: [fe80::127:dff:fe7c:fecc].

If a hostname is associated with multiple IP addresses, the data provider sequentially tries to connect to each IP address in the *AddressList* returned by *System.Net.Dns.GetHostEntry* until it achieves a successful socket connection or until it reaches the end of the list. If the connection to the first address is down, the driver attempts a connection to the next entry in the *AddressList*. Although performance will suffer as each *Exception* from a failed connection is caught, this re-attempt allows a secondary IP (backup) for a connection to a server.

SendIngresDate Connection Keyword

The Ingres .NET Data Provider sends .NET DateTime parameter data to the Data Access Server (DAS) and the DBMS. The format of this DateTime parameter data sent by the provider depends on the level of support of the ANSI Date/Time data types.

In releases prior to Ingres 9.1 (also known as 2006 Release 2), the data is sent as a data type of INGRESDATE with a GMT timezone. Beginning with Ingres 9.1, the date/time data is sent with a data type of ANSI TIMESTAMP_WITH_TIMEZONE (TS_W_TZ) with a local datetime, local timezone, and microsecond data. This flexibility and adaptability to the capabilities of the DBMS is useful, but can have a side-effect when upgrading to new releases of Ingres.

When upgrading from a release of Ingres prior to 9.1, the behavior of the program can change under certain circumstances when datetime formerly sent as an INGRESDATE type is now sent as a TS_W_TZ type. Typically, no problem occurs because the TS_W_TZ data is converted as needed and any components of the data such as microseconds are discarded, depending on the target data type. In some cases, however, the change in data type can change the semantics of processing enough to become an issue.

For example, if datetime parameter data is sent to a query containing the the IFNULL(? , ' ') function, the function may succeed or fail depending on the data type sent by the provider. If the provider sends the data as INGRESDATE type, then IFNULL(<ingresdate>, ' ') will execute correctly (returning the date or the empty string date if the parameter is null). Upon upgrade to Ingres 9.1 and later, the datetime parameter data is sent by the drivers as TIMESTAMP_WITH_TIMEZONE data type, and the IFNULL(<ts_w_tz>, ' ') will fail because the empty string date is not permitted in combination with ANSI TIMESTAMP_WITH_TIMEZONE.

The work-around for this problem is to wrap the INGRESDATE function around the parameter: IFNULL(INGRESDATE(?) , ' '). Changing every source reference in an application, however, may not be feasible.

Using the SendIngresDate=TRUE connection keyword tells the Ingres .NET Data Provider to send the date/time parameter data as INGRESDATE type and value (as if the server was pre-Ingres 9.1) to force the old behavior.

Caution! SendIngresDate=TRUE is intended to allow older applications to run with newer versions of Ingres. It should be used only in applications that access INGRESDATE columns only. Applications accessing ANSI Date/Time columns may experience other side effects of this feature due to loss of fractional second information and timezone format differences when the datetime parameter data is sent as INGRESDATE to an ANSI date/time column. Rather than using the SendIngresDate option, which is general in scope, we recommend using the IngresType.IngresDate parameter data type when sending .NET DateTime data that requires an INGRESDATE semantic context for the parameter.

IngresConnection.GetSchema Method

The IngresConnection.GetSchema method is used to return information on the schema of a database. The collection of metadata returned describes the tables, columns, and so on, that are defined in the database. The schema information is returned in the form of a .NET Framework DataTable.

GetSchema takes zero or more parameters. The first parameter is the collection name (for example: "Tables" or "Columns"). Optional restriction parameters in a String array ("restrictionValues") allow the returned metadata to be filtered to only those rows for a specified value, for example, only those columns for a specific table name. If GetSchema is invoked with no parameters or with a collection name of "MetaDataCollections" then a DataTable is returned that lists the collection names. Each row in the DataTable lists the collection name and number of restrictions supported for that collection.

The following collections are supported:

MetaDataCollection

Column Name	Data Type	Description
CollectionName	String	Collection name support by the data provider
NumberOfRestrictions	INT32	Number of restrictions that can be specified for the collection by GetSchema
NumberOfIdentifierParts	INT32	Number of parts in the database object name.

DataSourceInformation

Column Name	Data Type	Description
CompositeIdentifierSeparatorPattern	String	Regular expression for matching composite name separator character

Column Name	Data Type	Description
DataSourceProductName	String	Product name accessed by the data provider
DataSourceProductVersion	String	Product version accessed by the data provider
DataSourceProductVersionNormalized	String	Product version accessed by the data provider in a format that will be consistent for all versions of the data provider
GroupByBehavior	GroupByBehavior	The relationship between GROUP BY columns and non-aggregated columns of the SELECT statement
IdentifierPattern	String	Regular expression for matching an identifier
IdentifierCase	IdentifierCase	Indicates if nonquoted identifiers are case-sensitive
OrderByColumnsInSelect	Boolean	Specifies whether ORDER BY columns must be specified in the SELECT list.
ParameterMarkerFormat	String	Format string for matching a parameter marker
ParameterMarkerPattern	String	Regular expression for matching a parameter marker
ParameterNameMaxLength	INT32	Maximum character length of a named parameter
ParameterNamePattern	String	Regular expression for matching a named parameter
QuotedIdentifierPattern	String	Regular expression for matching a quoted identifier
QuotedIdentifierCase	String	Indicates if quoted identifiers are case sensitive
StatementSeparatorPattern	String	Regular expression for matching a statement separator
StringLiteralPattern	String	Regular expression for matching a quoted string literal in an SQL statement
SupportedJoinOperators	SupportedJoinOperators	Specifies what SQL JOIN operators are supported

DataTypes

ColumnName	DataType	Description
TypeName	String	Ingres data type name
ProviderDbType	INT32	IngresType value to be used for a parameter data type
ColumnSize	INT64	Length of the data type, if non-numeric
CreateFormat	String	Format string for creation of a column's data type in a CREATE TABLE
CreateParameters	String	Length, precision, and/or scale parameters associated with the data type's CreateFormat
DataType	String	Name of the .NET Framework data type associated with this data type
IsAutoIncrementable	Boolean	Indicates whether this data type may be auto-incrementing
IsBestMatch	Boolean	Indicates whether this data type is the best match for the .NET Framework specified by the DataType column
IsCaseSensitive	Boolean	Indicates whether the data type is a character data type and is case-sensitive
IsFixedLength	Boolean	Indicates whether the data type is of fixed length
IsFixedPrecisionScale	Boolean	Indicates whether the data type has a fixed precision and scale
IsLong	Boolean	Indicates whether the data type is a CLOB or BLOB
IsNullable	Boolean	Indicates whether the data type is nullable
IsSearchable	Boolean	Indicates whether the data type can be used in a WHERE predicate with any operator other than LIKE
IsSearchableWithLike	Boolean	Indicates whether the data type can be used in a WHERE predicate with the LIKE operator
IsUnsigned	Boolean	Indicates whether the data type is unsigned

ColumnName	DataType	Description
MaximumScale	INT16	If the data type is numeric, the maximum number of digits to the right of the decimal point
MinimumScale	INT16	If the data type is numeric, the minimum number of digits to the right of the decimal point
IsConcurrencyType	Boolean	Indicates whether the data type is updated when the row is changed and the column value is different. DBNull.Value if this capability is not supported.
IsLiteralSupported	Boolean	Indicates whether the data type can be expressed in a literal
LiteralPrefix	String	The prefix for a literal of this type
LiteralSuffix	String	The suffix for a literal of this type

Restrictions

ColumnName	DataType	Description
CollectionName	String	Collection name
RestrictionName	String	Restriction name
RestrictionDefault	String	Ignored
RestrictionNumber	INT32	The location (numbered from 1) within the restrictions collection where this restriction is associated with

ReservedWords

ColumnName	DataType	Description
ReservedWord	String	Ingres specific reserved words

Tables

ColumnName	DataType	Description
TABLE_CATALOG ¹	String	Always DBNull.Value
TABLE_SCHEMA ²	String	Schema name (table owner name)
TABLE_NAME ³	String	Table name of user table
TABLE_TYPE	String	Always "TABLE"

Views

ColumnName	DataType	Description
TABLE_CATALOG ¹	String	Always DBNull.Value
TABLE_SCHEMA ²	String	Schema name (view owner name)
TABLE_NAME ³	String	View name of user view
TABLE_TYPE	String	Always "VIEW"

Columns

ColumnName	DataType	Description
TABLE_CATALOG ¹	String	Always DBNull.Value
TABLE_SCHEMA ²	String	Schema name (table/view owner name)
TABLE_NAME ³	String	Table/view name of user table
COLUMN_NAME ⁴	String	Column name
ORDINAL_POSITION	Int16	Position of the column within the set of the table's columns, numbered from 1
COLUMN_DEFAULT	String	Column's default value
IS_NULLABLE	Boolean	Indicates whether the column is nullable
DATA_TYPE	String	Ingres data type name

ColumnName	DataType	Description
CHARACTER_MAXIMUM_LENGTH	INT32	Maximum length in characters, if character or binary data type
CHARACTER_OCTET_LENGTH	INT32	Maximum length in bytes, if character or binary data type
NUMERIC_PRECISION	Byte	Precision length if an integer, float, real, decimal, datetime, or interval data type
NUMERIC_PRECISION_RADIX	INT16	Radix of the Precision
NUMERIC_SCALE	INT32	Scale length
DATETIME_PRECISION	INT16	Precision of ingresdate and ANSI Timestamp data types

Indexes

ColumnName	DataType	Description
TABLE_CATALOG ¹	String	Always DBNull.Value
TABLE_SCHEMA ²	String	Base table schema name (table owner name)
TABLE_NAME ³	String	Base table name
NON_UNIQUE	INT16	1 if index is unique, else 0
INDEX_QUALIFIER	String	Index owner name
INDEX_NAME ⁴	String	Index name
TYPE	INT16	Index type (always ODBC index type SQL_INDEX_OTHER)
ORDINAL_POSITION	INT16	Position of the column within the set of the index's columns, numbered from 1
COLUMN_NAME	String	Column name
ASC_OR_DSC	String	Collation. "A" for ascending, "D" for descending

Procedures

ColumnName	DataType	Description
PROCEDURE_CATALOG ¹	String	Always DBNull.Value
PROCEDURE_SCHEMA ²	String	Procedure schema (procedure owner name)
PROCEDURE_NAME ³	String	Procedure name

ProcedureParameters

ColumnName	DataType	Description
PROCEDURE_CATALOG ¹	String	Always DBNull.Value
PROCEDURE_SCHEMA ²	String	Schema name (procedure owner name)
PROCEDURE_NAME ³	String	Procedure name
COLUMN_NAME ⁴	String	Procedure parameter name
ORDINAL_POSITION	INT16	Position of the column within the set of the procedure's parameters, numbered from 1
COLUMN_DEFAULT	String	Parameter's default value
IS_NULLABLE	Boolean	Indicates whether the parameter is nullable
DATA_TYPE	String	Ingres data type name
CHARACTER_MAXIMUM_LENGTH	INT32	Maximum length in characters, if character or binary data type
CHARACTER_OCTET_LENGTH	INT32	Maximum length in bytes, if character or binary data type
NUMERIC_PRECISION	Byte	Precision length if an integer, float, real, decimal, datetime, or interval data type
NUMERIC_PRECISION_RADIX	INT16	Radix of the precision

ColumnName	DataType	Description
NUMERIC_SCALE	INT32	Scale length
DATETIME_PRECISION	INT16	Precision of ingresdate and ANSI Timestamp data types
INGRESTYPE	IngresType	Ingres .NET Data Provider data type

1. Can be used as a restriction in the first entry of restrictionValues array.
2. Can be used as a restriction in the second entry of restrictionValues array.
3. Can be used as a restriction in the third entry of restrictionValues array.
4. Can be used as a restriction in the fourth entry of restrictionValues array.

Enlistment in Distributed Transactions

The Ingres .NET Data Provider supports enlistment in distributed transactions through the MS Distributed Transaction Coordinator (MSDTC) and the XA two-phase commit protocol.

Developers should be aware of MSDTC performance with distributed transactions and the lag time in communicating with all voters of the two-phase commit protocol. For performance reasons, distributed transactions should be used carefully. While the enlistment in a distributed transaction is not slow, it is not as fast as an enlistment in a local transaction.

Enable XA Support in Windows

To use the EnlistTransaction and EnlistDistributedTransaction methods in the Ingres .NET Data Provider, the administrator of the Windows machine must enable XA transactions through Component Services.

To enable XA support in Windows

Start, Control Panel, Administrative Tools, Component Services, Computers, My Computer, Properties, MSDTC, Security Configuration, Enable XA Transactions.

System.Transactions Programming Models

The .NET Framework System.Transactions namespace offers two programming models to the .NET application programmer to create a transaction. The Ingres .NET Data Provider supports both models:

- The explicit programming model allows the programmer to create, enlist into, and control the transaction manually.
- The implicit programming model allows .NET to automatically perform these operations. The implicit programming model is recommended as a best practice since there are fewer chances for programming errors and it frees the programmer from the details of managing enlistment in the System.Transactions.Transaction.

Implicit Automatic Enlistment using TransactionScope

The `System.Transactions.TransactionScope` class allows a .NET application to establish a transaction context. When a `TransactionScope` is instantiated, a current transaction context is established by .NET. Resource managers such as Ingres by default enlist in this ambient transaction. If an `IngresConnection.Open()` is issued by the application within the scope of this current transaction, and if the `ConnectionString` contains `Enlist=yes` (the default), then the `IngresConnection` automatically enlists the `IngresConnection` into the transaction.

Within the scope of the `TransactionScope`, the application calls the `TransactionScope.Complete()` method to indicate that the database unit of work should be committed. If the method is not called, the work is rolled back. When the `TransactionScope` is `Disposed`, updates to the Ingres database are committed or rolled back as directed by the .NET Transaction Manager. The Transaction Manager examines whether `TransactionScope.Complete()` method was called and issues the appropriate commit or rollback statements to Ingres.

Note: When an Ingres connection is enlisted in the .NET transaction, the commit or rollback to Ingres to commit/rollback the database changes occur when the `TransactionScope` is disposed, not when the `IngresConnection` is closed or disposed. Even though a `IngresConnection.Close()` method has been called and the `IngresConnection` instance has been disposed, the Ingres session remains active until the .NET `TransactionScope` is disposed and the .NET Transaction Manager issues the commit/rollback to the Ingres session.

The .NET application programmer can prevent automatic enlistment of the `IngresConnection` in the transaction context if the `IngresConnection.ConnectionString` includes the `Enlist=No` keyword/value pair.

The advantage to coding a "using" statement and `TransactionScope` is that if any of the database operations throws an exception, flow of control jumps out of the "using (`TransactionScope`)" block and a rollback of the transaction automatically occurs. The ease of programming and reliability of rollback or commit within the `TransactionScope` makes this model of enlistment a better programming practice.

TransactionScope Example

This example shows the enlistment of an Ingres database session in a distributed transaction managed by a using TransactionScope block. When the IngresConnection.Open() method is issued, the Ingres connection will enlist in the distributed transaction represented by the Transaction object within the TransactionScope. After the update, the TransactionScope is marked Complete(), and the updates to the Ingres database will be committed when the TransactionScope object is disposed.

```
static void TestEnlistTransactionImplicitSample(
    string connstring)
{
    using (TransactionScope scope = new TransactionScope())
    {
        using (IngresConnection conn1 =
            new IngresConnection(connstring))
        {
            conn1.Open();

            IngresCommand cmd = conn1.CreateCommand();
            cmd.CommandText =
                "update authors set au_id = '409-56-7008' " +
                "where au_id = '409-56-7008'";
            cmd.ExecuteNonQuery();

            scope.Complete();
        } // end using (IngresConnection)
    } // end using (TransactionScope)
}
```

Explicit Enlistment by EnlistTransaction() Method

A .NET application can disable automatic transaction enlistment and manually enlist the Ingres connection in the transaction if desired. The application programmer can prevent automatic enlistment of the IngresConnection in the current transaction context if the IngresConnection.ConnectionString includes the Enlist=no or Enlist=false keyword/value pair. Later, the application can manually enlist the Ingres connection in the transaction by calling the IngresConnection.EnlistTransaction method. The .NET Transaction Manager will issue a commit to the Ingres transaction if the .NET System.Transactions.Transaction is marked complete before the Transaction object is disposed, else the Ingres transaction will be rolled back.

Ingres Enlistment in a .NET Transaction

Whether the enlistment is automatic or manual, when an Ingres connection is enlisted in a .NET transaction, the Ingres .NET Data Provider participates as a resource manager within the transaction. The data provider works with the Microsoft Distributed Transaction Coordinator (MSDTC), Ingres DBMS Server, and the Ingres XA Distributed Transaction Processing (DTP) subsystem to allow the Ingres connection to participate in the distributed transaction on Windows with other Ingres or non-Ingres participants. The participants are polled in a vote to commit in a two-phase commit protocol (2PC). If the vote to commit is unanimous, all participants are directed to commit; otherwise, they are directed to roll back the database updates as an atomic unit of work.

IngresConnectionStringBuilder Class

The IngresConnectionStringBuilder class provides a series of properties and methods to create syntactically correct connection string and to parse and rebuild existing connection strings.

IngresConnectionStringBuilder Class Declaration

The IngresConnectionStringBuilder class can be declared as follows:

C#: public sealed class IngresConnectionStringBuilder :
DbConnectionStringBuilder

VB.NET: NotInheritable Public Class IngresConnectionStringBuilder
Inherits DbConnectionStringBuilder

IngresConnectionStringBuilder Class Properties

The IngresConnectionStringBuilder class has the following properties:

Property	Accessor	Description
BrowsableConnectionString	get set	Indicates whether the ConnectionString Property is visible in Visual Studio designers.
BlankDate	get set	BlankDate=null specifies that an Ingres blank (empty) date result value is to be returned to the application as a null value. The default is to return an Ingres blank date as a DateTime value of "9999-12-31 23:59:59".

Property	Accessor	Description
CharacterEncoding	get set	Specifies the .NET character encoding (for example, ISO-8859-1) used for conversions between Unicode and character data types.
ConnectTimeout	get set	The time, in seconds, to wait for an attempted connection to time out if the connection has not completed. Default is 15.
Count	get	The number of keys contained within the ConnectionString property.
CursorMode	get set	Specifies the default cursor concurrency mode, which determines the concurrency of cursors that have no explicitly assigned option in the command text. For example, FOR UPDATE or FOR READONLY.
Database	get set	Name of the Ingres database being connected to.
DataSource	get set	The name of the target server.
DateFormat	get set	Specifies the Ingres date format to be used by the Ingres server for date literals. Corresponds to the Ingres environment variable II_DATE_FORMAT and is assigned the same values.
DbmsUser	get set	The user name associated with the DBMS session.
DbmsPassword	get set	The DBMS password of the user.
DecimalChar	get set	Specifies the character that the Ingres DBMS Server is to use to separate fractional and non-fractional parts of a number—the comma (',') or the period ('.'). Default is the period.

Property	Accessor	Description
Enlist	get set	If set to true and if the creation thread is within a transaction context as established by System.EnterpriseServices.ServicedComponent, the IngresConnection is automatically enlisted into the transaction context. Default is true.
GroupID	get set	Group identifier that has permissions for a group of users.
Item	get set	The value associated with the key. This property is the C# indexer for the IngresConnectionStringBuilder class.
Keys	get	An ICollection of keys of type String in the IngresConnectionStringBuilder.
MaxPoolSize	get set	Maximum number of connections that can be in the connection pool. Default is 100.
MinPoolSize	get set	Minimum number of connections that can be in the connection pool. Default is 0.
MoneyFormat	get set	Specifies the Ingres money format to be used by the Ingres server for money literals. Corresponds to the Ingres environment variable II_MONEY_FORMAT and is assigned the same values.
MoneyPrecision	get set	Specifies the money precision to be used by the Ingres server for money literals. Corresponds to the Ingres environment variable II_MONEY_PREC and is assigned the same values.
Password	get set	The password to the Ingres database.
PersistSecurityInfo	get set	Indicates whether password information is returned in a get of the ConnectionString.

Property	Accessor	Description
Pooling	get set	Enables or disables connection pooling. By default, connection pooling is enabled (true).
Port	get set	Port number on the target server machine that the Data Access Server is listening to. Default is 117.
RoleID	get set	Role identifier that has associated privileges for the role.
RolePassword	get set	Role password associated with the Role ID.
Server	get set	The Ingres host server to connect to.
Timezone	get set	Specifies the Ingres time zone associated with the user's location. Used by the Ingres server only. Corresponds to the Ingres environment variable II_TIMEZONE_NAME and is assigned the same values.
UserID	get set	The name of the authorized user connecting to the DBMS Server. This value may be case-sensitive depending on the target server.
Values	get	An ICollection of values of type Object in the IngresConnectionStringBuilder.
VnodeUsage	get set	<p>Allows the .NET application to control the portions of the vnode information that are used to establish the connection to the remote DBMS server through the Ingres DAS server. Valid options are:</p> <ul style="list-style-type: none"> ■ connect – Only the vnode connection information is used (default). ■ login – Both the vnode connection and login information is used.

IngresConnectionStringBuilder Class Methods

The public methods available to the IngresConnectionStringBuilder class are:

Method	Description
Add	Adds a key and value to the collection within IngresConnectionStringBuilder.
Clear	Clears all keys and values from IngresConnectionStringBuilder. Sets ConnectionString property to "".
ContainsKey	Returns true if IngresConnectionStringBuilder contains the specified key.
EquivalentTo	Returns true if keys and values are comparable to the specified IngresConnectionStringBuilder object.
Remove	Removes the entry with the specified key from IngresConnectionStringBuilder.
ToString	Returns the ConnectionString associated in the IngresConnectionStringBuilder.
TryGetValue	Returns a value corresponding to the specified key from the IngresConnectionStringBuilder. Returns false if the key was not found.

IngresConnectionStringBuilder Class Constructors

The IngresConnectionStringBuilder class has the following constructors:

Constructor Overloads	Description
IngresConnectionStringBuilder ()	Instantiates a new instance of the IngresConnectionStringBuilder class using default property values
IngresConnectionStringBuilder (string)	Instantiates a new instance of the IngresConnectionStringBuilder class using the specified connection string.

IngresDataReader Class

IngresDataReader provides a means of reading a forward-only stream of rows from a result-set created by a SELECT query or a row-producing database procedure.

When an IngresDataReader is open, the IngresConnection is busy and no other operations are allowed on the IngresConnection (other than IngresConnection.Close) until IngresDataReader.Close is issued. Created by the IngresCommand.ExecuteReader methods.

IngresDataReader Class Declaration

The IngresDataReader can be declared as follows:

C#: public sealed class IngresDataReader :
System.Data.Common.DbDataReader

VB.NET: NotInheritable Public Class IngresDataReader
Inherits System.Data.Common.DbDataReader

IngresDataReader Class Example

The following is an example implementation of the IngresDataReader class:

```
static void ReaderDemo(string connstring)
{
    IngresConnection conn = new IngresConnection(connstring);

    string strNumber;
    string strName;
    string strSSN;

    conn.Open();

    IngresCommand cmd = new IngresCommand(
        "select number, name, ssn  from personnel", conn);

    IngresDataReader reader = cmd.ExecuteReader();

    Console.Write(reader.GetName(0) + "\t");
    Console.Write(reader.GetName(1) + "\t");
    Console.Write(reader.GetName(2));
    Console.WriteLine();

    while (reader.Read())
    {
        strNumber= reader.IsDBNull(0)?
            "<none>":reader.GetInt32(0).ToString();
        strName  = reader.IsDBNull(1)?
            "<none>":reader.GetString(1);
        strSSN   = reader.IsDBNull(2)?
            "<none>":reader.GetString(2);

        Console.WriteLine(
            strNumber + "\t" + strName + "\t" + strSSN);
    }

    reader.Close();
    conn.Close();
}
```

IngresDataReader Class Properties

The IngresDataReader class contains the following properties:

Property	Accessor	Description
Depth	get	The depth of nesting for the current row. This data provider always returns a depth of zero to indicate no nesting of tables.
FieldCount	get	The number of columns in the current row.
HasRows	get	Returns true if the data reader contains one or more rows. Returns false if the data

Property	Accessor	Description
		reader contains zero rows.
IsClosed	get	A true/false indicator as to whether the data reader is closed.
Item	get	Gets the column value in its native format for a given column name or column ordinal. This property is the C# indexer for the IngresDataReader class.
RecordsAffected	get	The number of rows updated, inserted, or deleted by execution of the SQL statement. -1 is returned for SELECT statements.

IngresDataReader Class Public Methods

The public methods available to the IngresDataReader class are:

Method	Description
Close	Closes the IngresDataReader.
GetBoolean	Gets the column value as a Boolean.
GetByte	Gets the column value as an unsigned 8-bit Byte.
GetBytes	Gets the column value as a byte stream into a Byte array.
GetChar	Gets the column value as a Char.
GetChars	Gets the column value as a character stream into a Char array.
GetDataTypeName	Gets the column's data type name as known in Ingres.
GetDateTime	Gets the column value as a DateTime.
GetDecimal	Gets the column value as a Decimal.
GetDouble	Gets the column value as a double.
GetFieldType	Gets the column's .NET Type.
GetFloat	Gets the column value as a Float.
GetGuid	Gets the column value as a Guid.
GetInt16	Gets the column value as a signed 16-bit integer.
GetInt32	Gets the column value as a signed 32-bit integer.
GetInt64	Gets the column value as a signed 64-bit integer.

Method	Description
GetName	Gets the column's name using a specified ordinal.
GetOrdinal	Gets the column's ordinal using a specified name.
GetSchemaTable	Returns a DataTable that describes the resultset column metadata. If ExecuteReader(CommandBehavior.KeyInfo) was called, additional information about primary key columns, unique columns, and base names is retrieved from the database catalog and included in the returned DataTable. For column information returned, see GetSchemaTable Columns Returned (see page 306).
GetString	Gets the column value as a string.
GetTimeSpan	Gets the column value as a TimeSpan.
GetValue	Gets the column value in its native format.
GetValues	Gets all of the column values into an Object array.
IsDBNull	Returns true/false indicating whether the column value is null.
NextResult	Advances the data reader to the next result set if present.
Read	Advances the data reader to the next row in the result set.

Important! *There are no conversions performed by the GetXXX methods. If the data is not of the correct type, an `InvalidCastException` is thrown.*

Always call `IsDBNull` on a column if there is any chance of it being null before attempting to call one of the `GetXXX` accessor to retrieve the data.

GetSchemaTable Columns Returned

The `GetSchemaTable` describes the column metadata of the `IngresDataReader`.

Note: The column information is not necessarily returned in the order shown.

Column Information	Data Type	Description
ColumnName	String	The name of the column, which reflects the renaming of the column in the command text (that is, the alias).
ColumnOrdinal	Int32	The number of the column, beginning with 1.

Column Information	Data Type	Description
ColumnSize	Int32	Maximum possible length of a value in the column.
NumericPrecision	Int16	This is the maximum precision of the column if the column is a numeric data type; otherwise the value is null.
NumericScale	Int16	This is the number of decimal places in the column if the column is a numeric data type; otherwise the value is null.
DataType	Type	The .NET Framework data type of the column.
ProviderType	IngresType	The indicator of the column's data type
IsLong	Boolean	Set to true if the column contains a long varchar, long varbinary, or long nvarchar object; otherwise false.
AllowDBNull	Boolean	Set to true if the application can set the column to a null value or if the data provider cannot determine if the application can set the column to a null value. Set to false if it is known that the application is not permitted to set the column to a null. Note that a column value may be null even if the application is not permitted to set the null value.
IsReadOnly	Boolean	Set to true if it is known that the column cannot be modified; otherwise false.
IsRowVersion	Boolean	Set to true if column has a persistent row identifier that cannot be written to and serves only to identify the row. The Ingres .NET Data Provider always returns false.
IsUnique	Boolean	Set to true if no two rows in the table can have the same value in this column. Set to false if not unique or if uniqueness cannot be determined. Only set if ExecuteReader(CommandBehavior.KeyInfo) was called.

Column Information	Data Type	Description
IsKey	Boolean	Set to true if this column is in the set of columns that, taken together, uniquely identify the row. Only set if ExecuteReader(CommandBehavior.KeyInfo) was called.
IsAutoIncrement	Boolean	Set to true if the column assigns values to new rows in fixed increments. The Ingres .NET Data Provider always returns false.
BaseCatalogName	String	The name of the database catalog that contains the column. This value is null if the catalog name cannot be determined. The Ingres .NET Data Provider always returns a null value.
BaseSchemaName	String	The name of the database schema that contains the column. This value is null if the schema name cannot be determined. Only set if ExecuteReader(CommandBehavior.KeyInfo) was called.
BaseTableName	String	The name of the database table or view that contains the column. This value is null if the table name cannot be determined. Only set if ExecuteReader(CommandBehavior.KeyInfo) was called.
BaseColumnName	String	The name of the column in the database. This value is null if the column name cannot be determined. Only set if ExecuteReader(CommandBehavior.KeyInfo) was called.

Mapping of Ingres Native Types to .NET Types

The following table maps the native Ingres database types supported by the Ingres .NET Data Provider to their corresponding .NET type. It also maps the typed accessor that a .NET application uses for an Ingres native database type to be obtained as a .NET type.

IngresType	Ingres Data Type	.NET Data Type	Accessor
Binary	byte	Byte[]	GetBytes()
Char	char	String	GetString()
DateTime	date	DateTime	GetDateTime()
Decimal	decimal	Decimal	GetDecimal()
Double	double precision (float8)	Double	GetDouble()
SmallInt	smallint	Int16	GetInt16()
TinyInt	integer1	Byte	GetByte()
Int	integer	Int32	GetInt32()
BigInt	bigint	Int64	GetInt64()
LongVarBinary	long byte	Byte[]	GetBytes()
LongVarChar	long varchar	String	GetString()
LongNVarChar	long nvarchar	String	GetString()
Nchar	nchar	String	GetString()
NVarChar	nvarchar	String	GetString()
Real	real (float4)	Single	GetString()
VarBinary	byte varying	Byte[]	GetBytes()
VarChar	varchar	String	GetString()
IntervalYearToMonth	interval year to month	String	GetString()
IntervalDayToSecond	interval day to second	Timespan	GetTimeSpan()

IngresDataAdapter Class

The IngresDataAdapter class represents a set of SQL statements and a database connection that are used to fill a DataSet and, optionally, update the Ingres database. The IngresDataAdapter object acts as a bridge between a .NET DataSet and the Ingres database for retrieving and updating data.

IngresDataAdapter Class Declaration

The declarations for the IngresDataAdapter class are:

C#: public sealed class IngresDataAdapter : DbDataAdapter, IDbDataAdapter, ICloneable

VB.NET: NotInheritable Public Class DataAdapter
Inherits DbDataAdapter
Implements IDbDataAdapter, ICloneable

IngresDataAdapter Class Example

```
public DataSet CreateDataSet(  
    string dsName, string connectionString, string commandText)  
{  
    IngresConnection connection =  
        new IngresConnection(connectionString);  
    IngresCommand command =  
        new IngresCommand(commandText, connection);  
    IngresDataAdapter adapter = new IngresDataAdapter(command);  
    DataSet ds = new DataSet();  
    adapter.Fill(ds, dsName);  
    return ds;  
}
```

IngresDataAdapter Class Properties

The IngresDataAdapter class has the following properties:

Property	Accessor	Description
AcceptChangesDuringFill	get set	A true/false value indicating whether the DataRow.AcceptChanges method is called after the DataRow is added to the DataTable. Inherited from DataAdapter. Default is true.
ContinueUpdateOnError	get set	A true/false value indicating whether to generate an exception or to update the RowError property

Property	Accessor	Description
		when an error occurs during an update to the row. Inherited from DataAdapter. Default is false.
DeleteCommand	get set	Command to be used (SQL statement or database procedure) to DELETE records from the database.
InsertCommand	get set	Command to be used (SQL statement or database procedure) to INSERT records into the database.
MissingMappingAction	get set	Action to be taken if incoming data does not have a matching table or column. Default is Passthrough. Inherited from DataAdapter.
MissingSchemaAction	get set	Action to be taken if an existing DataSet schema does not match incoming data. Default is Add. Inherited from DataAdapter.
SelectCommand	get set	Command to be used (SQL statement or database procedure) to SELECT records from the database.
TableMappings	get	The collection that provides the mapping between the returned records and the DataSet. Default is an empty collection. Inherited from DataAdapter.
UpdateCommand	get set	Command to be used (SQL statement or database procedure) to UPDATE records in the database.

IngresDataAdapter Class Public Methods

The public methods available to the IngresDataAdapter Class are:

Method	Description
Dispose	Releases allocated resources.
Fill	Adds or refreshes rows in the DataSet to match the

Method	Description
	values in the database. Inherited from DBDataAdapter.
FillSchema	Adds a DataTable to a DataSet and configures the schema to match that in the database. FillSchema does not add rows to a DataTable. Inherited from DBDataAdapter.
GetFillParameters	Gets an array of IDataParameter objects that contain the parameters set by the user when executing a SELECT statement. Inherited from DBDataAdapter.
Update	Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the DataSet. Inherited from DBDataAdapter.

IngresDataAdapter Class Events

The events generated by the IngresDataAdapter class are:

Event	Description
FillError	Raised when an error occurs during a Fill operation. Inherited from DBDataAdapter.
RowUpdating	Raised as an UPDATE, INSERT, or DELETE operation on a row (by a call to one of the Update methods) is about to start.
RowUpdated	Raised after an UPDATE, INSERT, or DELETE operation on a row (by a call to one of the Update methods) is complete.

IngresDataAdapter Class Constructors

The IngresDataAdapter class contains the following constructors:

Constructor Overloads	Description
IngresDataAdapter()	Instantiates a new instance of the IngresDataAdapter class using default property values
IngresDataAdapter (IngresCommand)	Instantiates a new instance of the IngresDataAdapter class using the defined IngresCommand as the SelectCommand.
IngresDataAdapter (string, IngresConnection)	Instantiates a new instance of the IngresDataAdapter class using the defined command text for a new instance of

Constructor Overloads	Description
	IngresCommand for the SelectCommand, and the IngresConnection object
IngresDataAdapter (string, string)	Instantiates a new instance of the IngresDataAdapter class using the defined command text for the SelectCommand and a connection string

IngresError Class

The IngresError class represents error or warning information returned by the Ingres database.

IngresError Class Declaration

The IngresError class can be declared as follows:

C#: [Serializable] public sealed class IngresError

VB.NET: NotInheritable Public Class IngresError

IngresError Class Example

The following is an implementation of the IngresError class:

```
static void PrintErrorCollection(IngresErrorCollection errcol)
{
    foreach(IngresError err in errcol)
    {
        PrintError(err);
    }
    Console.WriteLine("");
}

static void PrintError(IngresError err)
{
    Console.Write(err.GetType().ToString() + ":\n");
    Console.Write("\t" + "Message" = " +
        (err.Message != null?
            err.Message.ToString() : "<null>") + "\n");
    Console.Write("\t" + "Source = " +
        (err.Source != null?err.Source.ToString() : "<null>") + "\n");
    Console.Write("\t" + "ToString: " + err.ToString() + "\n");
    Console.Write("\t" + "Number" = " +
        (err.Number.ToString()) + "\n");
    Console.Write("\t" + "SQLState" = " +
        (err.SQLState != null?
            err.SQLState.ToString() : "<null>") + "\n");
    Console.WriteLine("");
}
```

IngresError Class Properties

The IngresError class has the following properties:

Property	Accessor	Description
Message	get	A description of the error.
Number	get	The database-specific error integer information returned by the Ingres database.
Source	get	Name of the data provider that generated the error. Always "Ingres."
SQLState	get	The standard five-character SQLSTATE code.

IngresError Class Public Methods

The public methods available to the IngresError class are:

Method	Description
ToString	A description of the error in the form of "IngresError: <i>error-message-text</i> ".

IngresErrorCollection Class

The IngresErrorCollection class represents a collection of the IngresError objects returned by the Ingres database. Created by IngresException, an IngresErrorCollection collection always contains at least one instance of IngresError.

IngresErrorCollection Class Declaration

The declarations for the IngresErrorCollection class are:

C#: [Serializable]
public sealed class IngresErrorCollection : ICollection, IEnumerable

VB.NET: <Serializable>
NotInheritable Public Class IngresError
Inherits ICollection Implements IEnumerable

IngresErrorCollection Class Example

The following is an example implementation of the IngresErrorCollection class:

```
static void PrintErrorCollection(IngresErrorCollection errcol)
{
    foreach(IngresError err in errcol)
    {
        PrintError(err);
    }
    Console.WriteLine("");
}

static void PrintError(IngresError err)
{
    Console.Write(err.GetType().ToString() + ":\n");

    Console.Write("\t" + "Message          = " +
        (err.Message != null?
            err.Message.ToString() : "<null>") + "\n");
    Console.Write("\t" + "Source = " +
        (err.Source != null?err.Source.ToString() : "<null>") + "\n");
    Console.Write("\t" + "ToString: " + err.ToString() + "\n");
    Console.Write("\t" + "Number          = " +
        (err.Number.ToString()) + "\n");
    Console.Write("\t" + "SQLState          = " +
        (err.SQLState != null?
            err.SQLState.ToString() : "<null>") + "\n");
    Console.WriteLine("");
}
```

IngresErrorCollection Class Properties

The IngresErrorCollection class has the following properties:

Property	Accessor	Description
Count	get	The number of errors in the collection.
Item	get	Gets the IngresError for a given ordinal. This property is the C# indexer for IngresErrorCollection class.

IngresErrorCollection Class Public Methods

The public methods available to the IngresErrorCollection class are:

Method	Description
CopyTo	Copies the elements of IngresErrorCollection to an Array.

IngresException Class

The IngresException class represents the exception that is thrown when error information is returned by the Ingres database.

IngresException Class Declaration

The IngresException is declared as follows:

C#: [Serializable]
public sealed class IngresException : SystemException

VB.NET: <Serializable>
NotInheritable Public Class IngresException
Inherits SystemException

IngresException Class Example

The following is an example implementation of the IngresException class:

```
static void PrintException(IngresException ex)
{
    Console.Write(ex.GetType().ToString() + ":\n");
    Console.Write("\t" + "Errors          = " +
        (ex.Errors != null?ex.Errors.ToString() :
        "<null>") + "\n");
    Console.Write("\t" + "HelpLink          = " +
        (ex.HelpLink != null?ex.HelpLink.ToString() :
        "<null>") + "\n");
    Console.Write("\t" + "InnerException = " +
        (ex.InnerException != null?ex.InnerException.ToString():
        "<null>") + "\n");
    Console.Write("\t" + "Source          = " +
        (ex.Source != null?ex.Source.ToString() :
        "<null>") + "\n");
    Console.Write("\t" + "TargetSite = " +
        (ex.TargetSite != null?ex.TargetSite.ToString(): "<null>") + "\n");
    Console.WriteLine("");
}
```

IngresException Class Properties

The IngresException class contains the following properties:

Property	Accessor	Description
Errors	get	An ErrorCollection of one or more Error objects that give more detailed information on the exception generated by the provider.
InnerException	get	The nested Exception instance that caused

Property	Accessor	Description
		the current exception. Inherited from Exception.
Message	get	A concatenation of all the messages in the Errors collection.
Source	get	Name of the data provider that generated the error. Always "Ingres."
StackTrace	get	A string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite	get	The method that threw the current exception. Inherited from Exception.

IngresException Class Public Methods

The public methods available to the IngresException class are:

Method	Description
ToString	The description of the exception as a string.

IngresFactory Class

The IngresFactory class helps generate many of the other Ingres classes in an inter-operable data provider model. For each Ingres class that the factory wants to construct, simply call the Ingres constructor for that class and return the object instance.

IngresFactory Class Declaration

The IngresFactory class can be declared as follows:

C#: public sealed class IngresFactory : DbProviderFactory

VB.NET: NotInheritable Public Class IngresFactory
Inherits DbProviderFactory

IngresFactory Class Public Fields

The IngresFactory class has the following public fields:

Field	Description
Instance	The static field containing the single instance of IngresFactory.

IngresFactory Class Public Methods

The public methods available to the IngresFactory class are:

Method	Description
CreateCommand	Creates an instance of IngresCommand using the factory.
CreateCommandBuilder	Creates an instance of IngresCommandBuilder using the factory.
CreateConnection	Creates an instance of IngresConnection using the factory.
CreateConnectionStringBuilder	Creates an instance of IngresConnectionStringBuilder using the factory.
CreateDataAdapter	Creates an instance of IngresDataAdapter using the factory.
CreateDataSourceEnumerator	Always returns null.
CreateParameter	Creates an instance of IngresParameter using the factory.
CreatePermission	Creates an instance of IngresPermission using the factory.

IngresInfoMessageEventArgs Class

The IngresInfoMessageEventArgs class provides the information on warnings from the database to the delegate method that handles the InfoMessage event.

IngresInfoMessageEventArgs Class Declaration

The IngresInfoMessageEventArgs class is declared as follows:

C#: public sealed class IngresInfoMessageEventArgs : EventArgs

VB.NET: NotInheritable Public Class IngresInfoMessageEventArgs
Inherits EventArgs

IngresInfoMessageEventArgs Class Example

The following is an implementation of the IngresInfoMessageEventArgs class:

```
static void OnInfoMessage(
    Object sender, IngresInfoMessageEventArgs e)
{
    Console.WriteLine("OnInfoMessage event args: (" +
        "ErrorCode=" + e.Number +
        ", Errors=" + e.Errors +
        ", Message=\"\" + e.Message + "\"\" +
        ", Source=" + e.Source + ")");
}
```

IngresInfoMessageEventArgs Class Properties

The following are the properties of the IngresInfoMessageEventArgs class:

Property	Accessor	Description
Errors	get	An ErrorCollection of one or more Error objects that give more detailed information on the warnings generated by the provider.
Message	get	A concatenation of all the messages in the Errors collection.
Number	get	The database-specific warning integer information returned by the Ingres database.
Source	get	Name of the data provider that generated the error. Always "Ingres."

IngresInfoMessageEventHandler Class

The IngresInfoMessageEventHandler delegate represents the delegate method that handles the InfoMessage event.

IngresInfoMessageEventHandler Class Declaration

The IngresInfoMessageEventHandler class has the following message declaration:

C#: [Serializable]
public delegate void IngresInfoMessageEventHandler
(object sender, IngresInfoMessageEventArgs e)

VB.NET: <Serializable>
Public Delegate Sub IngresInfoMessageEventHandler _
(ByVal sender As Object, ByVal e As
IngresInfoMessageEventArgs)

IngresInfoMessageEventHandler Class Example

The following is an implementation of the IngresInfoMessageEventHandler class:

```
static void DoWork(string connstring)
{
    IngresConnection conn = new IngresConnection(connstring);
    conn.InfoMessage += new
        IngresInfoMessageEventHandler(OnInfoMessage);
    <do additional work>
}

static void OnInfoMessage(
    Object sender, IngresInfoMessageEventArgs e)
{
    Console.WriteLine("OnInfoMessage event args: (" +
        "ErrorCode=" + e.Number +

        ", Errors=" + e.Errors +
        ", Message=\"\" + e.Message + "\"\" +
        ", Source=" + e.Source + ")");
}
```


IngresMetaDataCollectionNames Class

The IngresMetaDataCollectionNames class presents information on metadata, such as tables, views, and columns. Each member of the class is a constant string suitable for use as collectionName argument for the IngresConnection.GetSchema method.

The collectionNames supported are:

- Columns
- Indexes
- ProcedureParameters
- Procedures
- Tables
- Views

IngresMetaDataCollectionNames Class Declaration

The IngresMetaDataCollectionNames Class can be declared as follows:

C#: public static class IngresMetaDataCollectionNames

VB.NET: Public Shared Class IngresMetaDataCollectionNames

IngresParameter Class

The IngresParameter class represents a parameter for an IngresCommand for each question mark (?) placeholder in the command and, optionally, its mapping to a DataSet column.

The IngresParameter constructor determines the data type of the parameter in the following ways:

- The constructor specifies an IngresType enumeration
- The constructor specifies a System.DbType enumeration
- Through the .NET Framework System.Type of the Value object if no specific data type is in the constructor

Positional Parameters

The data provider supports the positional (unnamed) parameter marker placeholder syntax using the question mark character (?) similar to ODBC and JDBC. Parameter data is accessed from the IngresParameterCollection in the same relative order as the parameter markers in the SQL statement.

Named Parameters

The data provider supports named parameters. A named parameter marker is constructed by an initial character of question mark (?), at-sign (@), or colon (:) followed by a name. The name is an alphanumeric identifier or an integer. Databases vary in the use of the initial character for a named parameter. While the Ingres recommended initial parameter marker character is the question mark, the Ingres .NET Data Provider supports all three characters.

If one parameter is named then all of the parameters must be named. If one parameter is unnamed (positional) then all of the parameters must be unnamed. For the sake of code readability, a mix of named parameters and unnamed parameters is not permitted.

For the sake of code readability, a mix of named parameters, each with a different initial parameter marker character in a single SQL statement is also not permitted.

Multiple occurrences of the same named parameter marker in the SQL statement is permitted.

Excess named parameters that are not referenced in the SQL statement are permitted in the IngresParameterCollection and are ignored.

The name comparison between the named parameter marker in the SQL statement and the named parameter in the IngresParameterCollection is not case sensitive.

Named Parameters Example

The following is an example of using named parameters:

```
cmd = conn.CreateCommand();
cmd.CommandText =
    "insert into mytable values ( ?key, ?col1 ?col2 ?col3 )";
cmd.Parameters.AddWithValue("?key", mykey);
cmd.Parameters.AddWithValue("?col3", mystring3);
cmd.Parameters.AddWithValue("?col1", mystring1);
cmd.Parameters.AddWithValue("?col2", mystring2);
cmd.ExecuteNonQuery();

cmd = conn.CreateCommand
cmd.CommandText =
    "insert into mytable values ( @key, @col1 @col2 @col3 )";
cmd.Parameters.AddWithValue("@key", mykey);
cmd.Parameters.AddWithValue("@col3", mystring3);
cmd.Parameters.AddWithValue("@col1", mystring1);
cmd.Parameters.AddWithValue("@col2", mystring2);
cmd.ExecuteNonQuery();

cmd = conn.CreateCommand();
cmd.CommandText =
    "insert into mytable values ( :key, :col1 :col2 :col3 )";
cmd.Parameters.AddWithValue(":key", mykey);
cmd.Parameters.AddWithValue(":col3", mystring3);
cmd.Parameters.AddWithValue(":col1", mystring1);
cmd.Parameters.AddWithValue(":col2", mystring2);
cmd.ExecuteNonQuery();
```

IngresParameter Class Example

The following is an implementation of the IngresParameter class:

```
static string DemoParameterSSN(
    string connstring, int intNumber, string strSSN)
{
    IngresConnection conn = new IngresConnection(connstring);

    string strName = null;

    conn.Open();

    IngresCommand cmd = new IngresCommand(
        "select name from demo_personnel where number = ? and ssn = ?",
        conn);

    // add two parameters to the command's IngresParameterCollection
    cmd.Parameters.Add(new IngresParameter("Number", intNumber));
    IngresParameter parm =
        new IngresParameter("SSN", IngresType.VarChar);
    parm.Value = strSSN;
    cmd.Parameters.Add(parm);

    IngresDataReader reader = cmd.ExecuteReader();

    while (reader.Read())
    {
        if (reader.IsDBNull(0))
            break;
        strName = reader.GetString(0);
    }

    reader.Close();
    conn.Close();

    return strName;
}
```

IngresParameter Class Declaration

The class declaration for IngresParameter class is:

C#: public sealed class IngresParameter :
System.Data.Common.DbParameter, IDataParameter, IDbDataParameter,
ICloneable

VB.NET: NotInheritable Public Class IngresParameter
Inherits System.Data.Common.DbParameter
Implements IDataParameter, IDbDataParameter, ICloneable

IngresParameter Class Properties

The properties for the IngresParameter are:

Property	Accessor	Description
DbType	get set	The type that the parameter must be converted to before being passed to the database server. Setting this parameter induces a setting of IngresType property. Default is DbType.String.
Direction	get set	Indicators whether the parameter has an input, input/output, output, or procedure return value.
IngresType	get set	The type that the parameter must be converted to before being passed to the database server. Setting this parameter induces a setting of DbType property. Default is IngresType.NVarChar if the database supports Unicode UTF-16; otherwise the default is IngresType.VarChar.
IsNullable	get set	Indicates whether the parameter accepts null values. True = accepts null values. False = does not accept null values.
ParameterName	get set	The name of the parameter. Default is "".
Precision	get set	Maximum number of digits for decimal parameters. Default is 0.
Scale	get set	Number of decimal places for decimal parameters. Default is 0.
Size	get set	Maximum size of binary and string data to be sent to the server. Default is inferred from the parameter value.
SourceColumn	get set	The name of the source column mapped to a DataSet. Default is "".

Property	Accessor	Description
SourceVersion	get set	The DataRowVersion used by an UpdateCommand during an Update operation for setting the parameter. Default is DataRowVersion.Current.
Value	get set	The value of the parameter. Default is null.

Important! .NET strings are Unicode based. If the application is sending a Unicode string as a parameter to a database field that is ASCII char or varchar, the application can direct the data provider to coerce the Unicode string to an ASCII string on the client side by setting the parameter DbType property to DbType.AnsiString, or the IngresType property to IngresType.VarChar.

IngresParameter Class Public Methods

The public methods available for the IngresParameter class are:

Method	Description
ToString	The name of the parameter.

IngresParameter Class Constructors

The constructors available to the IngresParameter class are:

Constructor Overloads	Description
IngresParameter()	Instantiates a new instance of the IngresParameter class using default property values
IngresParameter(string, DbType)	Instantiates a new instance of the IngresParameter class using the defined parameter name and DbType.
IngresParameter(string, IngresType)	Instantiates a new instance of the IngresParameter class using the defined parameter name and IngresType.
IngresParameter(string, object)	Instantiates a new instance of the IngresParameter class using the defined parameter name and System.Object. The DbType and IngresType are inferred from the .NET Type of the object.

Constructor Overloads	Description
<code>IngresParameter (string, DbType, string)</code>	Instantiates a new instance of the <code>IngresParameter</code> class using the defined parameter name, <code>DbType</code> , and source column name.
<code>IngresParameter (string, IngresType, string)</code>	Instantiates a new instance of the <code>IngresParameter</code> class using the defined parameter name, <code>IngresType</code> , and source column name.
<code>IngresParameter (string, DbType, int)</code>	Instantiates a new instance of the <code>IngresParameter</code> class using the defined parameter name, <code>DbType</code> , and column size.
<code>IngresParameter (string, IngresType, int)</code>	Instantiates a new instance of the <code>IngresParameter</code> class using the defined parameter name, <code>IngresType</code> , and column size.
<code>IngresParameter (string, DbType, int, string)</code>	Instantiates a new instance of the <code>IngresParameter</code> class using the defined parameter name, <code>DbType</code> , column size, and source column name.
<code>IngresParameter (string, IngresType, int, string)</code>	Instantiates a new instance of the <code>IngresParameter</code> class using the defined parameter name, <code>IngresType</code> , column size, and source column name.
<code>IngresParameter (string, DbType, int, ParameterDirection, bool, byte, byte, string, DataRowVersion, object)</code>	Instantiates a new instance of the <code>IngresParameter</code> class using the defined parameter name, <code>DbType</code> , column size, parameter direction, boolean indication of whether or not the field can be null, precision, scale, source column name, <code>DataRowVersion</code> describing the version of a <code>System.Data.DataRow</code> , and the value of the object.
<code>IngresParameter (string, IngresType, int, ParameterDirection, bool, byte, byte, string, DataRowVersion, object)</code>	Instantiates a new instance of the <code>IngresParameter</code> class using the defined parameter name, <code>IngresType</code> , column size, parameter direction, boolean indication of whether or not the field can be null, precision, scale, source column name, <code>DataRowVersion</code> describing the version of a <code>System.Data.DataRow</code> , and the value of the object.

IngresParameterCollection Class

The `IngresParameterCollection` class represents a collection of all parameters for an `IngresCommand` object and their mapping to a `DataSet` object.

IngresParameterCollection Class Declaration

The class declaration for IngresParameterCollection class is as follows:

C#: public sealed class IngresParameterCollection :
System.Data.Common.DbParameterCollection, IDataParameterCollection,
IList, ICollection, IEnumerable

VB.NET: NotInheritable Public Class IngresParameterCollection
Inherits System.Data.Common.DbParameterCollection
Implements IDataParameterCollection, IList, ICollection,
IEnumerable

IngresParameterCollection Class Example

For an example of adding parameters to an IngresParameterCollection, see IngresParameter Class Example (see page 324).

IngresParameterCollection Class Properties

The IngresParameterCollection has the following properties:

Property	Accessor	Description
Count	get	The number of parameters in the collection.
Item	get set	The IngresParameter object for a given ordinal or parameter name. This property is the C# indexer for IngresParameterCollection class.

IngresParameterCollection Class Public Methods

The public methods available to the IngresParameterCollection class are:

Method	Description
Add	Adds an IngresParameter to the parameter collection.
AddRange	Adds an array of values or IngresParameters to the parameter collection
AddWithValue	Adds an IngresParameter with a name and value to the parameter collection.
Clear	Removes all items from the collection.
Contains	Indicates whether IngresParameter exists in the collection. True = does exist; False = does not exist.

Method	Description
CopyTo	Copies the elements of IngresParameterCollection to an Array.
IndexOf	Returns the index of the IngresParameter in the collection.
Insert	Inserts the IngresParameter into the collection.
Remove	Removes the IngresParameter from the collection.
RemoveAt	Removes an IngresParameter with a specified index or name from the collection.

IngresPermission Class

The IngresPermission class provides additional information that a user has a security level sufficient to access the Ingres database. At present, the class is not used by the Ingres .NET Data Provider. The class is included in the Ingres data provider to provide completeness for the Factory interoperability model. Instead, standard Ingres security and access control is used.

The IngresPermission class can be declared as follows:

C#: public sealed class IngresPermission : DBPermission

VB.NET: NotInheritable Public Class IngresPermission
Inherits DBPermission

IngresRowUpdatedEventArgs Class

The IngresRowUpdatedEventArgs class provides the information for the RowUpdated event of an IngresDataAdapter.

IngresRowUpdatedEventArgs Class Declaration

The class declaration for IngresRowUpdateEventArgs is as follows:

C#: public sealed class IngresRowUpdatedEventArgs : RowUpdatedEventArgs

VB.NET: NotInheritable Public Class IngresRowUpdatedEventArgs
Inherits RowUpdatedEventArgs

IngresRowUpdatedEventArgs Class Properties

The IngresRowUpdatedEventArgs has the following properties:

Property	Accessor	Description
Command	get	The IngresCommand executed when DbDataAdapter.Update method is called.
Errors	get	The Exception containing any errors generated by the Ingres .NET Data Provider during the execution of the IngresCommand. Inherited from RowUpdatedEventArgs.
RecordsAffected	get	The number of rows updated, inserted, or deleted during the execution of the UPDATE, INSERT, or DELETE SQL statement. Inherited from RowUpdatedEventArgs.
Row	get	The System.Data.DataRow sent through the DbDataAdapter.Update. Inherited from RowUpdatedEventArgs.
StatementType	get	The type of SQL statement executed. Inherited from RowUpdatedEventArgs.
Status	get	The System.Data.UpdateStatus of the IngresCommand. Inherited from RowUpdatedEventArgs.
TableMapping	get	The DataTableMapping sent through a DbDataAdapter.Update. Inherited from RowUpdatedEventArgs.

IngresRowUpdatedEventHandler Class

The IngresRowUpdatedEventHandler delegate represents a delegate method that handles the RowUpdated event of an IngresDataAdapter.

IngresRowUpdatedEventHandler Class Declaration

The IngresRowUpdateEventHandler class has the following declaration:

C#: [Serializable]
public delegate void Ingres RowUpdated EventHandler
(object sender, IngresRowUpdated EventArgs e)

VB.NET: <Serializable>
Public Delegate Sub Ingres RowUpdatedEventHandler _
(ByVal sender As Object, ByVal e As
IngresRowUpdatedEventArgs)

IngresRowUpdatingEventArgs Class

The IngresRowUpdatingEventArgs class provides the information for the RowUpdating event of an IngresDataAdapter.

IngresRowUpdatingEventArgs Class Declaration

The IngresRowUpdatingEventArgs class has the following declaration:

C#: public sealed class IngresRowUpdatingEventArgs :
RowUpdatingEventArgs

VB.NET: NotInheritable Public Class Ingres RowUpdatingEventArgs
Inherits RowUpdatingEventArgs

IngresRowUpdatingEventArgs Class Properties

The IngresRowUpdatingEventArgs class has the following properties:

Property	Accessor	Description
Command	get set	The IngresCommand executed when DbDataAdapter.Update method is called.
Errors	get	The Exception containing any errors generated by the Ingres .NET Data Provider during the execution of the IngresCommand. Inherited from RowUpdatedEventArgs.
RecordsAffected	get	The number of rows updated, inserted, or deleted during the execution of the UPDATE, INSERT, or DELETE SQL statement. Inherited from RowUpdatedEventArgs.

Row	get	The System.Data.DataRow sent through the DbDataAdapter.Update. Inherited from RowUpdatedEventArgs.
StatementType	get	The type of SQL statement executed. Inherited from RowUpdatedEventArgs.
Status	get	The System.Data.UpdateStatus of the IngresCommand. Inherited from RowUpdatedEventArgs.
TableMapping	get	The DataTableMapping sent through a DbDataAdapter.Update. Inherited from RowUpdatedEventArgs.

IngresRowUpdatingEventHandler Class

The IngresRowUpdatingEventHandler delegate represents a delegate method that handles the RowUpdating event of an IngresDataAdapter.

IngresRowUpdatingEventHandler Class Declaration

The IngresRowUpdatingEventHandler class declaration is as follows:

```
C#: [Serializable]
    public delegate void IngresRowUpdatingEventHandler
        ( object sender, IngresRowUpdatingEventArgs e)

VB.NET: <Serializable>
    Public Delegate Sub Ingres RowUpdatingEventHandler _
        (ByVal sender As Object, ByVal e As
        IngresRowUpdatingEventArgs)
```

IngresTransaction Class

The IngresTransaction class represents a local, non-distributed database transaction. Each connection is associated with a transaction. This object allows manual commit or rollback control over the pending local transaction.

Created by the BeginTransaction method in the IngresConnection object. After Commit() or Rollback has been issued against the transaction, the IngresTransaction object can not be reused and a new IngresTransaction object must be obtained from the IngresConnection.BeginTransaction method if another transaction is desired.

IngresTransaction Class Declaration

The IngresTransaction class declaration is as follows:

C#: public sealed class IngresTransaction :
System.Data.Common.DbTransaction

VB.NET: NotInheritable Public Class IngresTransaction
Inherits System.Data.Common.DbTransaction

IngresTransaction Class Example

The following is an implementation of the IngresTransaction class:

```
static void DemoTxn(string connstring)

{
    IngresConnection conn = new IngresConnection(connstring);
    ProviderTransaction txn;

    conn.Open();
    txn = conn.BeginTransaction();

    IngresCommand cmd = new IngresCommand(
        "update demo_personnel set name = 'Howard Lane' "+
        " where number = 200", conn, txn);

    int numberOfRecordsAffected = cmd.ExecuteNonQuery();

    Console.WriteLine(numberOfRecordsAffected.ToString() +
        " records updated.");

    txn.Commit();

    conn.Close();
}
```

IngresTransaction Class Properties

The IngresTransaction class has the following properties:

Property	Accessor	Description
Connection	get	The IngresConnection object that is associated with the transaction. Null if the transaction or connection is no longer valid.
IsolationLevel	get	The IsolationLevel for this transaction as set in the IngresConnection.BeginTransaction method.

IngresTransaction Class Methods

The IngresTransaction class contains the following methods:

Method	Description
Commit	Commit the database changes.
Dispose	Release allocated resources.
Rollback	Rollback the database changes.

Data Types Mapping

The Ingres .NET Data Provider defines its own enumeration of supported data types in addition to the standard System.Data.DbType enumeration.

The following table shows the mapping of the Ingres .NET Data Provider's data types to its .NET data type counterparts. For information on the typed accessors that a .NET application uses for an Ingres native database type to be obtained as a .NET type, see IngresDataReader Class (see page 303).

IngresType	Ingres Data Type	Description	.NET Data Type
Binary	byte	Fixed length stream of binary data	Byte[]
Boolean	boolean	Boolean values of true and false	Boolean
Char	char	Fixed length stream of character data	String
Date	ansidate	Date data	DateTime
Time	time	Time data	DateTime
DateTime	timestamp	Date and time data	DateTime
IngresDate	ingresdate	Ingres format date	DateTime
IntervalYearToMonth	interval year to month	Interval year to month	String
IntervalDayToSecond	interval day to second	Interval day to second	TimeSpan
Decimal	decimal	Exact numeric data	Decimal
Double	double precision	Approximate numeric data	Double

IngresType	Ingres Data Type	Description	.NET Data Type
	(float8)		
SmallInt	smallint	Signed 16-bit integer data	Int16
TinyInt	integer1	Signed 8-bit integer data	SByte
Int	integer	Signed 32-bit integer data	Int32
BigInt	bigint	Signed 64-bit integer data	Int64
LongVarBinary	long byte	Binary large object	Byte[]
LongVarChar	long varchar	Character large object	String
LongNVarChar	long nvarchar	Unicode large object	String
NChar	nchar	Fixed length stream of Unicode data	String
NVarChar	nvarchar	Variable length stream of Unicode data	String
Real	real (float4)	Approximate numeric data	Single
VarBinary	byte varying	Variable length stream of binary data	Byte[]
VarChar	varchar	Variable length stream of character data	String

Notes:

- DateTime literals within SQL CommandText must be specified in the form of {d 'yyyy-mm-dd'} for dates and {ts 'yyyy-mm-dd hh-mm-ss'} for timestamps. However, it is preferable to pass .NET DateTime parameters rather than literals for these values.
- For Ingres servers, DateTime parameters are converted to UTC values before being sent to the Ingres servers. DateTime values retrieved from Ingres servers are converted from UTC values to Local values. For non-Ingres servers, DateTime values are passed between the data provider and servers unchanged.
- IngresType.DateTime parameter data is sent by the data provider depending on the level of support of the ANSI Date/Time data types. You can override the default behavior by specifying the IngresType.IngresDate data type for the parameter data rather than IngresType.DateTime. IngresType.IngresDate parameter data type forces the data provider to send the date/time parameter data as INGRES DATE type and value. For more information, see SendIngresDate Connection Keyword (see page 286).

DbType Mapping

.NET's System.Data.DbType for a parameter is mapped to the IngresType data type as follows:

DbType	IngresType
AnsiString	VarChar
AnsiStringFixedLength	Char
Binary	VarBinary
Boolean	Boolean
Byte	Binary, if supported by the database; otherwise, Char
Currency	Decimal
Date	DateTime
DateTime	DateTime
Decimal	Decimal
Double	Double
Guid	Not supported
Int16	SmallInt
Int32	Int
Int64	Decimal
Object	Not supported
SByte	TinyInt, if supported by the database; otherwise, SmallInt
Single	Real
String	NVarChar , if supported by the database; otherwise, VarChar
StringFixedLength	NChar, if supported by the database; otherwise, Char
Time	DateTime
UInt16	Int
UInt32	Decimal
UInt64	Decimal
VarNumeric	Decimal

Coercion of Unicode Strings

.NET strings are Unicode based. If the application is sending a Unicode string as a parameter to a database field that is ASCII char or varchar, the application can direct the data provider to coerce the Unicode string to an ASCII string on the client side by setting the IngresParameter's DbType property to DbType.AnsiString or the IngresType property to IngresType.VarChar.

The following is an example of coercing a Unicode string:

```
IngresCommand cmd = new IngresCommand(  
    "select name from personnel where ssn = ?",  
    conn);  
  
IngresParameter parm =  
    new IngresParameter("SSN", IngresType.VarChar);  
parm.Value = strSSN;  
cmd.Parameters.Add(parm);
```

IngresDataReader Object—Retrieve Data from the Database

Ingres .NET Data Provider provides the IngresDataReader object to retrieve a read-only, forward-only stream of data from an Ingres result-set created by a SELECT query or row-producing database procedure. Using the IngresDataReader increases application performance and reduces system overhead because only one row of data at a time is in memory.

Build the IngresDataReader

After creating an instance of the IngresCommand object, call Command.ExecuteReader to build the IngresDataReader to retrieve rows from a data source, as shown in the following example:

Visual Basic:

```
Dim rdr As IngresDataReader = cmd.ExecuteReader()
```

C#:

```
IngresDataReader rdr = cmd.ExecuteReader();
```

IngresDataReader Methods

Use the Read method of the IngresDataReader object to obtain a row from the results of the query. To access each column of the returned row, pass the name or ordinal reference of the column to the IngresDataReader.

For best performance, the IngresDataReader provides a series of methods that allow you to access column values in their native data types (GetDateTime, GetDouble, GetGuid, GetInt32, and so on). For a list of typed accessor methods, see the IngresDataReader Class (see page 303). Use the typed accessor when you know the underlying data type. When using a type accessor, use the correct accessor to avoid an InvalidCastException being thrown when no type conversions are performed.

Example: Using the IngresDataReader

The following is an implementation of IngresDataReader.

```
static void DataReaderDemo(string connstring)
{
    IngresConnection conn = new IngresConnection(connstring);

    string strNumber;
    string strName;
    string strSSN;

    conn.Open();

    IngresCommand cmd = new IngresCommand(
        "select number, name, ssn from personnel", conn);

    IngresDataReader reader = cmd.ExecuteReader();

    Console.Write(reader.GetName(0) + "\t");
    Console.Write(reader.GetName(1) + "\t");
    Console.Write(reader.GetName(2));
    Console.WriteLine();

    while (reader.Read())
    {
        strNumber= reader.IsDBNull(0)?
            "<none>":reader.GetInt32(0).ToString();
        strName = reader.IsDBNull(1)?
            "<none>":reader.GetString(1);
        strSSN = reader.IsDBNull(2)?
            "<none>":reader.GetString(2);

        Console.WriteLine(
            strNumber + "\t" + strName + "\t" + strSSN);
    }

    reader.Close();
    conn.Close();
}
```

ExecuteScalar Method—Obtain a Single Value from a Database

To return database information that is a single value rather than in the form of a table or data stream—for example, to return the result of an aggregate function such as Count(*), Sum(Price), or Avg(Quantity)—use the IngresCommand object's ExecuteScalar method. The ExecuteScalar method returns as a scalar value the value of the first column of the first row of the result set.

The following code example uses the Count aggregate function to return the number of records in a table:

Visual Basic:

```
Dim cmd As IngresCommand = New IngresCommand("SELECT Count(*) FROM Personnel",  
conn)  
Dim count As Int32 = CInt(cmd.ExecuteScalar())
```

C#:

```
IngresCommand cmd = new IngresCommand("SELECT Count(*) FROM Personnel", conn);  
Int32 count = (Int32)cmd.ExecuteScalar();
```

GetBytes Method—Obtain BLOB Values from a Database

When you access the data in the BLOB field, use the GetBytes typed accessor of the DataReader, which fills a byte array with the binary data. Specify a buffer size of data to be returned and a starting location for the first byte read from the returned data. GetBytes will return a long value that represents the number of bytes returned. If you pass a null byte array to GetBytes, the long value returned is the total number of bytes in the BLOB. Optionally, specify an index in the byte array as a start position for the data being read.

GetSchemaTable Method—Obtain Schema Information from a Database

The Ingres .NET Data Provider enables you to obtain schema information from Ingres data sources. Such schema information includes database schemas or catalogs available from the data source, database tables and views, and constraints that exist for database tables.

The Ingres .NET Data Provider exposes schema information using the GetSchemaTable method of the IngresDataReader object. This method returns a DataTable that describes the resultset column metadata. For more information on this metadata, see GetSchemaTable Columns Returned (see page 306).

If the ExecuteReader(CommandBehavior.KeyInfo) method was called in the IngresCommand object when building the IngresDataReader, additional information about primary key columns, unique columns, and base names are retrieved from the database catalog and included in the returned DataTable.

ExecuteNonQuery Method—Modify and Update Database

Using the Ingres .NET Data Provider, you can use the IngresCommand's ExecuteNonQuery method to process SQL statements that modify data but do not return rows, such as INSERT, UPDATE, DELETE, and other non-resultset commands such as CREATE TABLE.

Although rows are not returned by the ExecuteNonQuery method, input and output parameters and return values can be passed and returned using the Parameters property of the IngresCommand object.

The following code example executes an UPDATE statement to update a record in a database using ExecuteNonQuery:

```
static void DemoUpdate(string connstring)
{
    IngresConnection conn = new IngresConnection(connstring);

    conn.Open();
    IngresCommand cmd = new IngresCommand(
        "update demo_personnel set name = 'Howard Lane' "+
        " where number = 200", conn);

    int numberOfRecordsAffected = cmd.ExecuteNonQuery();

    Console.WriteLine(numberOfRecordsAffected.ToString() +
        " records updated.");

    conn.Close();
}
```

IngresDataAdapter Object—Manage Data

An IngresDataAdapter object has four properties for retrieving and updating data source records:

- SelectCommand returns selected data from the data source.

The SelectCommand property must be set before calling the Fill method of the IngresDataAdapter.

- InsertCommand inserts data into the data source.
- UpdateCommand updates data in the data source.
- DeleteCommand deletes data from the data source.

The InsertCommand, UpdateCommand, and DeleteCommand properties must be set before the Update method of the IngresDataAdapter is called, depending on what changes were made to the data in the DataSet. For example, if rows have been added, the InsertCommand must be set before calling Update.

When Update is processing an inserted, updated, or deleted row, the IngresDataAdapter uses the respective Command property to process the action. Current information about the modified row is passed to the Command object through the Parameters collection.

For example, when updating a row, the UPDATE statement uses a unique identifier to identify the row in the table being updated. The unique identifier is commonly the value of a primary key field, or unique non-null index. The UPDATE statement uses parameters that contain the unique identifier, the columns, and the values to be updated, as shown in the following SQL statement:

```
static void DemoAdapter(string connstring)
{
    IngresConnection conn = new IngresConnection (connstring);
    IngresDataAdapter adapter = new IngresDataAdapter ();

    adapter.SelectCommand = new IngresCommand (
        "select * from personnel", conn);

    adapter.UpdateCommand = new IngresCommand (
        "update personnel set name = ?, number = ? where ssn = ?",
        conn);
    adapter.UpdateCommand.Parameters.Add(
        "@name", IngresType.Char, "name");
    adapter.UpdateCommand.Parameters.Add(
        "@number", IngresType.Int, "number");
    adapter.UpdateCommand.Parameters.Add(
        "@oldssn", IngresType.Char, "ssn").SourceVersion =
        DataRowVersion.Original;

    DataSet ds = new DataSet();
    adapter.Fill(ds, "Personnel");

    ds.Tables["Personnel"].Rows[195]["number"] = 4199;
    adapter.Update(ds, "Personnel");
}
```

IngresDataAdapter Events

The IngresDataAdapter exposes the following events, which you can use to respond to changes made to data source data:

RowUpdating

An UPDATE, INSERT, or DELETE operation on a row (by a call to one of the Update methods) is about to start.

RowUpdated

An UPDATE, INSERT, or DELETE operation on a row (by a call to one of the Update methods) is complete.

FillError

An error has occurred during a Fill operation. Inherited from DBDataAdapter.

How Database Procedures Are Called

The Ingres .NET Data Provider supports calling Ingres database procedures. Input, output, and return values can be passed to and from the execution of the procedure on the DBMS server.

The .NET application can call database procedures in either of two programming styles.

The first technique simply sets the IngresCommand.CommandText property to the name of the database procedure, sets the IngresCommand.CommandType to CommandType.StoredProcedure, optionally sets IngresCommand.Parameters with a collection of parameters, and executes the command.

The second technique uses the "{call ...}" escape sequence syntax that is commonly used in the ODBC and JDBC APIs. The syntax supported is:

```
{ [ ? = ] CALL [schemaname.]procedurename [( [parameters, ...])]
```

```
parameters := param | param, parameters
param      := [parametername =] [value]
value      := ? | literal | SESSION.tablename
literal    := numeric_literal | string_literal | hex_string
```

Zero or more parameters can be passed to the procedure. In its `IngresParameter.Direction` property, a parameter can have a `ParameterDirection` of `Input`, `InputOutput`, `Output`, or `ReturnValue`.

Ingres Global Temporary Table (GTT) procedure parameters are specified by providing a parameter value in the form of `SESSION.tablename`. In this parameter, `tablename` is the name of the GTT, and the keyword `SESSION` identifies the parameter as a GTT parameter.

When using the `CALL` syntax, parameters can be named or unnamed. Named parameters specify a `parametername=` qualifier and can be specified in any order in the `CALL` syntax. Unnamed (or positional) parameters must be specified in the same order in the `CALL` syntax as the parameters defined in the `CREATE PROCEDURE` declaration. Having a mix of named and unnamed parameters in the `CALL` statement is not permitted.

When connecting to an Ingres 9.x or earlier DBMS Server, the use of named procedure parameters is encouraged. (Using named parameters improves performance by eliminating a query of the database catalog to assign names to the parameters based on the declared order of the procedure parameters.) When connecting to an Ingres 10 or later DBMS Server, named procedure parameters are not required.

Ingres database procedures permit the use of the transaction statements `COMMIT` and `ROLLBACK`. The use of these statements, however, is highly discouraged due to the potential for conflict in the transaction processing state between the .NET client and DBMS Server sides of the session. Including these statements in a procedure called by the data provider can result in the unintentional commit or rollback of work done prior to procedure execution. It is also possible that a change in transaction state during procedure execution can be misinterpreted as a transaction abort. For these reasons, applications must make sure that no transaction is active prior to executing a database procedure that contains `COMMIT` or `ROLLBACK` statements.

Row Producing Procedures

The result-set from Ingres row-producing database procedures can be read by the Ingres .NET Data Provider like any other result set.

If the database procedure was defined as:

```
create procedure myrowproc
  result row(char(32)) as
  declare tabname char(32);
begin
  for select table_name into :tabname from iitables
  do
    return row(:tabname);
  endfor;
end;
```

The application code fragment to read the result set might be:

```
IngresCommand cmd = new IngresCommand(
    "myrowproc", conn);
cmd.CommandType = CommandType.StoredProcedure;
IDataReader reader = cmd.ExecuteReader();

while (reader.Read())
{
    Console.Write(reader.GetString(0));
}

Console.WriteLine();
reader.Close();
```

Integration with Visual Studio

The Ingres .NET Data Provider is integrated with Visual Studio 2005 and Visual Studio 2008.

The .NET Framework has design-time support. .NET objects, derived from certain component objects, can exist within an application runtime environment and in a designer environment such as Microsoft Visual Studio.

Integration with Visual Studio visual tools allows a programmer to drag-and-drop the data provider design component onto a control. Integration also allows the programmer to use wizards and editors to aid application development.

The Visual Studio Toolbox contains a series of tabs (for example, Data, Components, and Window Forms) that list objects for the Visual Studio design environment. These objects can be dragged-and-dropped onto design surfaces such as the Windows Form control (WinForm). This operation can trigger wizards, designers, or simply a paint of a control on the design surface.

Install the Data Provider into the Toolbox

The Ingres .NET Data Provider must be installed into the Visual Studio Toolbox before using it for the first time.

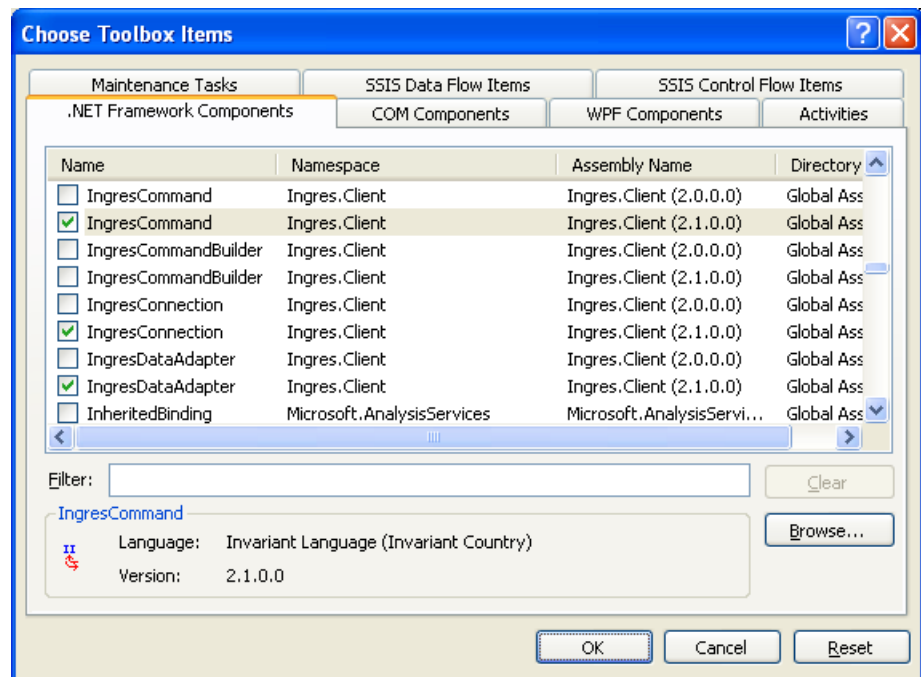
To install the data provider components into the Toolbox

1. Create an empty Winform application.
2. Right-click the Data tab of the toolbox, and select Choose Items.

The Choose Toolbox Items dialog is displayed.

3. Select the IngresCommand, IngresConnection, and IngresDataAdapter components on the .NET Framework Components tab, and then click OK.

The Ingres .NET Data Provider components are installed in the Toolbox, as shown in this example:



If the IngresCommand, IngresConnection, and IngresDataAdapter components do not appear in the Choose Toolbox Items dialog, you can add them.

To add the Ingres .NET Data Provider components to the Choose Toolbox Items dialog

1. Click Browse on the Choose Toolbox Items dialog and browse to the directory C:\Program Files\Ingres\Ingres .NET Data Provider\v2.1.
2. Open the Ingres.Client dll.

The components are added.

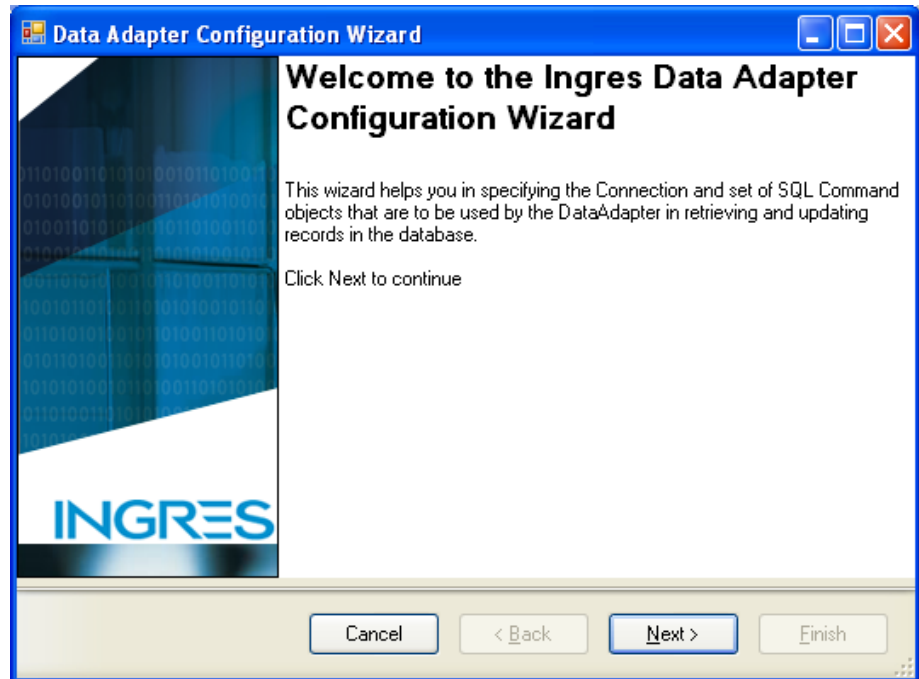
Start the Ingres Data Adapter Configuration Wizard

The Toolbox's Data tab lists the .NET data provider components that are available during the application's design.

To start the Ingres Data Adapter Configuration Wizard

1. Drag the IngresDataAdapter component from the list on the Toolbox's Data tab onto the Windows Form design surface ("Form1").

The welcome page of the Data Adapter Configuration Wizard is displayed.



An "ingesDataAdapter1" component and its icon are added to the Visual Studio designer component tray.

2. Click Cancel on the welcome page.

Only the IngresDataAdapter component is created.

Configure a Connection

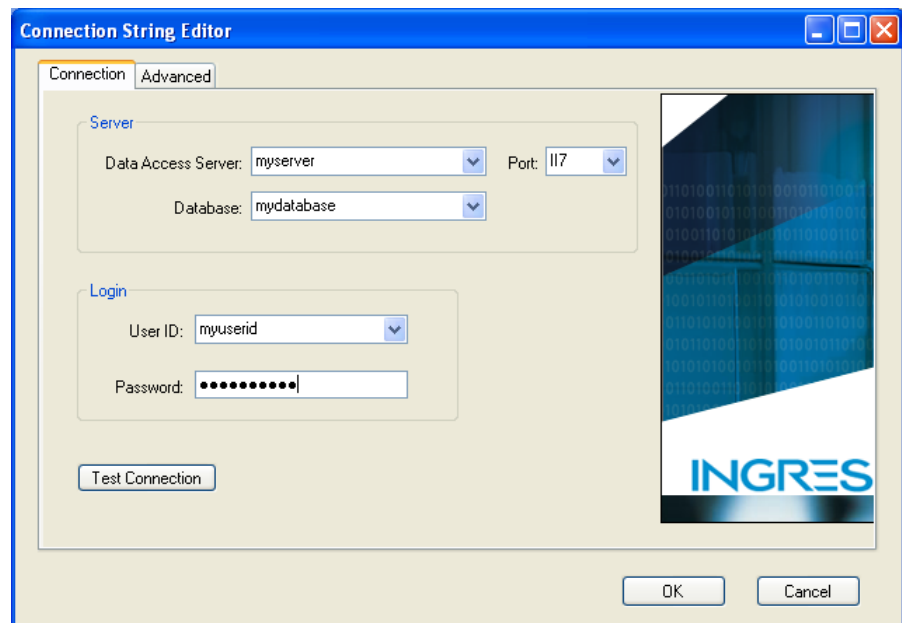
Ingres Data Adapter Configuration Wizard assists you in specifying the design properties of the `ingresDataAdapter1` component, including its connection string definition.

A connection string is a collection of information that identifies the target server machine and database to connect to, the permissions of the user who is connecting, and the parameters that define connection pooling and security. You must configure a connection string before connecting to a database using a .NET application.

To configure a connection string

1. Click Next in the Data Adapter Configuration Wizard welcome screen.

The Connection String Editor dialog is displayed.



2. Enter the required information. For details, see Connection String Editor (see page 348).

Click OK.

The Connection string is created.

Connection String Editor (Data Adapter Configuration Wizard)

The Connection String Editor of the Ingres Data Adapter Configuration Wizard has the following tabs:

Connection Tab

Data Access Server

Identifies the name of the Data Access Server that services .NET application requests for the target DBMS Server.

Port

Identifies the port number on the host server machine that the Data Access Server is listening to.

Default: 117

Database

Specifies the name of the target database that the application will connect to by default.

User ID

Specifies the name of the authorized user that is connecting to the DBMS Server.

Password

Specifies the password associated with the specified User ID for connecting to the DBMS Server.

Advanced Tab

Timeout

Defines the number of seconds after which an attempted connection will abort if it cannot be established.

Default: 15

Enable Connection Pooling

Enables or disables connection pooling.

Default: Connection pooling is enabled

Return password text in Connection.ConnectionString property

Determines whether password information from the connection string is returned in a get of the **ConnectionString** property.

Default: Password information is not returned

Role ID

Specifies the role identifier that has associated privileges for the role.

Role Password

Specifies the password associated with the specified Role ID.

DBMS User

Specifies the user name associated with the DBMS session (equivalent to the -u flag).

DBMS Password

Specifies the DBMS password of the user (equivalent to the -P flag).

Group ID

Specifies the group identifier that has associated privileges for a group of use.

Design a Query Using the Query Builder

The Ingres .NET Data Provider uses SQL statements to retrieve and update data in Ingres databases. In the Data Adapter Configuration Wizard, you can enter your SQL command or use the Query Builder tool to generate the SELECT statement.

To design a query using the Query Builder

1. Click Query Builder to develop your SELECT statement.

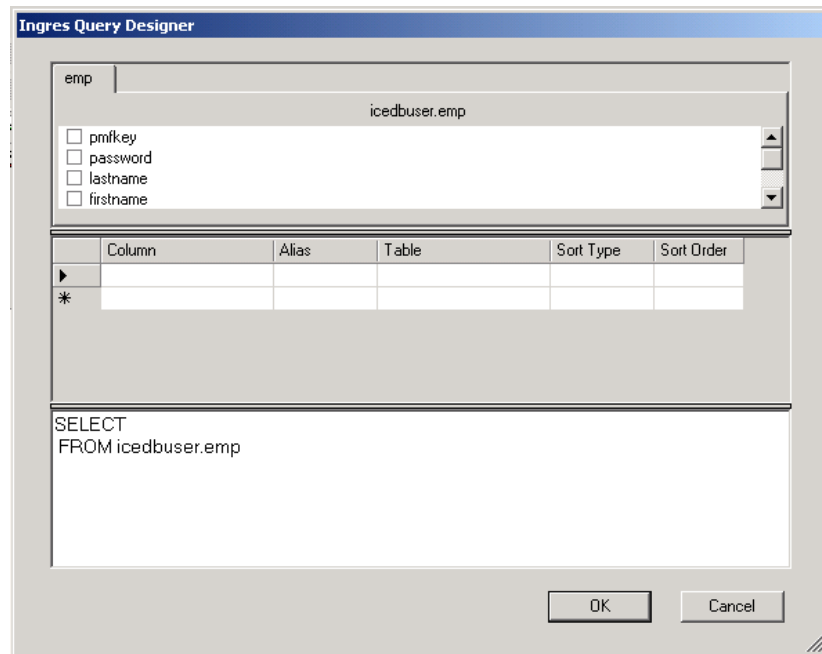
The Add Table dialog opens.

2. Click User Tables or All Tables.

A list of available tables is displayed.

3. Choose your table from the list, and then click Add, Close.

Ingres Query Designer opens.



The Ingres Query Designer has three horizontal panels:

The top panel

Consists of tab pages, one for each table reference in the FROM clause of the query. Each tab page contains a list of check boxes for each column defined in the table. The columns are listed as they are written in the table's catalog definition.

Check or uncheck each column to add or remove the column reference from the SELECT statement.

The middle panel

Is a grid that lists the column names and or expressions in the SELECT statement's column reference list. It provides a convenient tabular format for entering the column references.

The bottom panel

Displays the query text as it is being built. The query text can be directly edited and is automatically formatted for readability.

4. Enter column references you want to add to your query into any one of the three panels.

The other two panels are automatically updated.

5. Click OK.

The query builder returns to the Ingres Data Adapter Wizard and displays the generated SELECT statement.

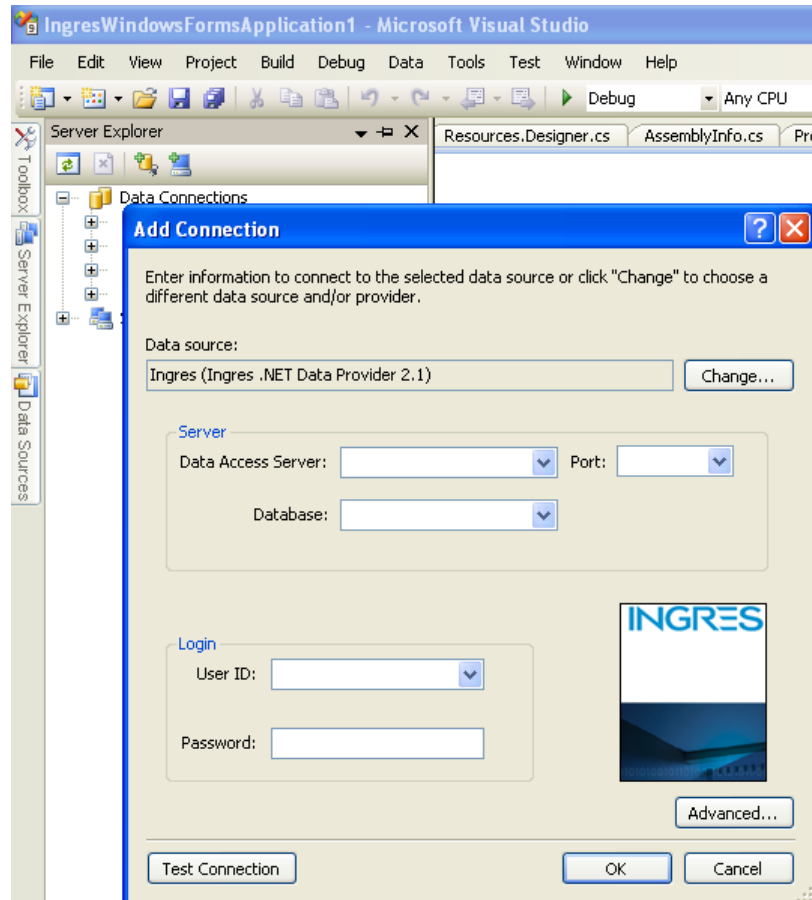
6. Click Finish.

The Ingres Data Adapter Wizard is closed.

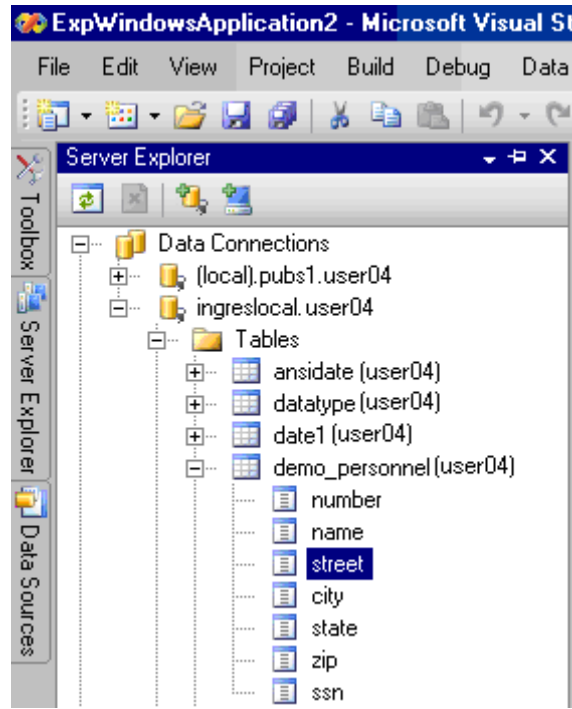
Server Explorer Integration

The Ingres .NET Data Provider is integrated with the Visual Studio Server Explorer and Data Sources tabs.

A Data Connection definition for Ingres can be defined in the Server Explorer. This Data Connection definition can subsequently be used by other wizards and designers in Visual Studio.



The Data Connection definition can also be used to examine the metadata information of tables and views in the Ingres connection.



Note: The Visual Studio integration does not currently support database procedures.

Application Configuration File—Troubleshoot Applications

The Ingres .NET Data Provider offers a basic trace facility to assist the application developer in identifying a sequence of data provider method calls that may be called incorrectly by an application program. The developer can create a .NET application configuration file that contains keys for Ingres.trace.

For example, you can create a file called myApplication.exe.config in the same directory as the myApplication.exe executable. The myApplication.exe.config text file contains the following:

```
<?xml version="1.0" encoding = "utf-8" ?>
<configuration>
  <appSettings>
    <add key="Ingres.trace.log" value="C:\temp\Ingres.trace.log" />
    <add key="Ingres.trace.timestamp" value="true" />
    <add key="Ingres.trace.driv" value="2" />
  </appSettings>
</configuration>
```

The value= key controls the level of tracing that is produced. Possible key values are as follows:

- 0 - no tracing (default)
- 1 - Basic function name detail
- 2 - Internal connection and messaging detail
- 3 - Internal state detail
- 4 - Internal status, length, and count detail

Obtain File Version of the Data Provider

Ingres Technical Support may ask what file version of the data provider is installed on the client.

The file version of the data provider's Ingres.Client.dll is incremented to mark changes to the data provider. You can obtain the change level of the data provider by displaying the file version of the DLL.

The Ingres.Client.dll is installed (by default) into the C:\Program Files\Ingres\Ingres .NET Data Provider\v2.1 directory and also into the Global Assembly Cache (GAC) that can be displayed at C:\WINDOWS\assembly.

To display the file version of the Ingres.Client.dll using the Windows file explorer

Right-click the Ingres.Client.dll in one of the directories mentioned previously and select Properties.

The Version tab of the Properties dialog displays the file version of the data provider.

Appendix A: TCP/IP Protocol

This section contains the following topics:

[Listen Address Format](#) (see page 357)

[Network Address Format](#) (see page 358)

[Connection Data Entry Information](#) (see page 359)

This appendix describes the format of a listen address when the protocol between two machines is TCP/IP. It also gives you the protocol- and platform-specific information to set up connection data entries.

Listen Address Format

A listen address is a unique identifier used for interprocess communications. A Communications Server has two kinds of listen addresses. It uses one to receive messages from local processes and the other to receive messages from remote Communications Servers.

This section describes the format of the listen address used to receive messages from remote processes when the network protocol is TCP/IP.

Note: To view or change your instance's listen addresses, use the Configuration-By-Forms (cbf) utility or Configuration Manager (vcbf).

When you install Ingres Net on a machine that is using the TCP/IP protocol, the listen address has two possible formats, as follows:

ax[n]

or

ppppp

where:

a

Is an alphabetic character (case is not significant)

x

Is an alphabetic character or a decimal digit (0-9)

n

Is a numeric digit from 0 - 7, inclusive

p

Is a numeric digit. The range depends on your operating system. For specific details, see your operating system documentation.

The format *ax[n]* is the default format, where *ax* is the installation ID (found in `II_INSTALLATION`) and *n* = 0. The digit *n* is incremented by 1 for each successive Communications Server started in an installation. For example, if the installation has three Communications Servers using the default format for their listen addresses, the addresses are *ax0*, *ax1*, and *ax2*.

Network Address Format

The network address in a TCP/IP environment has the following format:

host_name | *ip_address*

where:

host_name

Is the name of the remote node in character form

ip_address

Is the internet address of the remote node in the following format:

For IPv4: d.d.d.d (For example: 123.45.67.89) dotted decimal

For IPv6: x:x:x:x:x:x:x (For example: fe80::208:dff:fe7c:fe7c::) colon-hexadecimal

Connection Data Entry Information

Windows

Protocol: tcp_ip or wintcp

Network Address: See Network Address Format (see page 358).

Listen Address: See Listen Address Format (see page 357).

UNIX

Protocol: tcp_ip

Network Address: See Network Address Format (see page 358).

Listen Address: See Listen Address Format (see page 357).

VMS

Protocol: tcp_wol | tcp_dec

- Use tcp_wol if the protocol between the two machines is Wollongong TCP/IP or Multinet TCP/IP when running in Wollongong emulation.
- Use tcp_dec if the protocol between the two machines is TCP/IP Services for OpenVMS or Multinet TCP/IP when running in TCP/IP Services emulation.

Network Address: See Network Address Format (see page 358).

Listen Address: See Listen Address Format (see page 357).

MVS

When you install Ingres Net on an MVS machine using the TCP/IP protocol, the listen address is stored in the IGWFPSVR macro:

```
INSTALL = xx  
      TYPE = tcp_ibm | tcp_knet | tcp_sns  
      PORT = ax[n] | ppppp
```

where:

xx

Is the installation ID

tcp_ibm

Is the Ingres keyword indicating the IBM TCP/IP protocol

tcp_knet

Is the Ingres keyword indicating the KNET TCP/IP protocol

tcp_sns

Is the TCP/IP protocol for SNS TCP/IP

ax[n] ppppp

Is the listen address

Connection Data Entry Information:

Protocol: tcp_ibm | tcp_knet

Network Address: See Network Address Format (see page 358).

Listen Address: See Listen Address Format (see page 357).

Appendix B: SNA LU0 Protocol

This section contains the following topics:

[Listen Address Format](#) (see page 361)

This appendix describes the format of a listen address when the protocol between two machines is SNA LU0. It also gives you the protocol- and platform-specific information to set up connection data entries.

Listen Address Format

A listen address is a unique identifier used for interprocess communications. A Communications Server has two kinds of listen addresses. It uses one to receive messages from local processes and the other to receive messages from remote Communications Servers.

This section describes the format of the listen address used to receive messages from remote processes when the network protocol is SNA LU0.

Note: To view or change your instance's listen addresses, use the Configuration-By-Forms (cbf) utility or Configuration Manager (vcbf).

MVS

When you install Ingres Net on an MVS machine, the listen address is determined by the IGWFPSVR macro:

```
INSTALL = xx  
        TYPE=sna_lu0  
        ACB=acb_name
```

where:

xx

Is the installation ID, found in II_INSTALLATION

acb_name

Is the ACB name defined for the Ingres Net application using the VTAM APPL statement

In the APPL statement, you can specify the ACB name explicitly using the ACBNAME parameter, or implicitly by omitting ACBNAME. If you omit ACBNAME, the label on the APPL statement is used as the ACB name.

Connection Data Entry Information:

Protocol: sna_lu0

Network address: This parameter is ignored by MVS. Enter **x**.

Listen Address: *lu_name*

where:

lu_name

Is the name of the LU that corresponds to the listen address specified on the server node. This name does not necessarily have to match the name specified as the configured listen address on the server instance.

If the server instance is on an MVS system, the *lu_name* is the application minor node name for the target Ingres Net installation. The application minor node name is usually, but not always, the ACB name.

Appendix C: SNA LU62 Protocol

This section contains the following topics:

[Listen Address Format](#) (see page 363)

This appendix describes the format of a listen address when the protocol between two machines is SNA LU62. It also gives you the protocol- and platform-specific information to set up connection data entries.

Listen Address Format

A listen address is a unique identifier used for interprocess communications. A Communications Server has two kinds of listen addresses. It uses one to receive messages from local processes and the other to receive messages from remote Communications Servers.

This section describes the format of the listen address used to receive messages from remote processes when the network protocol is SNA LU62.

For the SNA LU62 protocol, the format of the listen address is platform specific.

Note: To view or change your instance's listen addresses, use the Configuration-By-Forms (cbf) utility or Configuration Manager (vcbf).

MVS

When you install Ingres Net on an MVS system, the listen address is determined by the IGWFPSVR macro:

```
INSTALL=xx  
      TYPE=sna_lu62  
      ACB=acb_name
```

where:

xx

Is the installation ID, found in II_INSTALLATION

acb_name

Is the ACB name defined for the Net LU6.2 application in the VTAM APPL statement

In the APPL statement, you can specify the ACB name by using the ACBNAME parameter or by omitting ACBNAME. If you omit ACBNAME, the label on the APPL statement is used as the ACB name.

Connection Data Entry Information:

Protocol: sna_lu62

Network address: *lu_name*[*.mode_name*]

where:

lu_name

Is the name of the LU on the remote instance that supports the Net SNA_LU62 protocol driver.

If the remote instance is on an MVS system, the *lu_name* is the minor node name for the target Net LU6.2 application. This is usually, but not always, the ACB name.

If the remote instance is on a Sun-4 system, the *lu_name* is any LU defined for the SunLink SNA Peer-to-Peer Gateway.

Default: None.

mode_name

Is the SNA logon mode name to be used for sessions with the remote instance.

Default: The mode name specified in the LOGMODE parameter of the IGWFPSVR macro.

Listen Address: *tp_name*

where:

tp_name

Is the transaction program name that is used by the Net SNA_LU62 protocol driver on the server node. If the remote instance is on an MVS system, there is no transaction program name. Enter "x". For all other systems, the *tp_name* must be the transaction program name specified in the remote instance. This is generally found in the remote instance's listen address.

Solaris

When you install Ingres Net on a Solaris system that is using the SNA LU62 protocol, the listen address has the format:

gateway_name.tp_name

where:

gateway_name

Is the name of the SunLink SNA Peer-to-Peer Gateway

This name must match a *gateway_name* contained in the */etc/appcs* file or NIS database, as described in the SunLink SNA Peer-to-Peer Administrator's Guide.

tp_name

Is the transaction program name used by the listening process. The name is an arbitrary string of up to 16 characters.

The rate at which Ingres Net polls for incoming connection requests can be controlled through the Ingres configuration variable (in *config.dat* file) *ii.<host>.gcc.*.sna_lu62.poll* (*xx* is the installation ID.) This specifies the polling rate in milliseconds. The polling rate defaults to 4000 (4 seconds); values smaller than this are not recommended. If there are no incoming connection requests, you can inhibit polling by setting the environment variable to the special value of -1.

Connection Data Entry Information:

Protocol: `sna_lu62`

Network address: *session_name*

where:

session_name

Is the unique session name defined to the SunLink SNA Peer-to-Peer Gateway for sessions between the Sun-4 client and the remote server instance. Default: None

Listen Address: *tp_name*

where:

tp_name

Is the transaction program name that is used by the Net SNA_LU62 protocol driver on the server node. If the remote instance is on an MVS system, there is no transaction program name. Enter **x**. For all other systems, the *tp_name* must match the transaction program name specified in the remote server instance. This is generally found in the remote server instance's listen address for SNA_LU62.

HP-UX

When you install Ingres Net on an HP-UX system using the HP-UX SNAplus product, the listen address has the format:

tp_name

where:

tp_name

Is the transaction program name used by the listening process.

Unless you inhibit polling for incoming connections, the name must be configured as an Invocable TP name in the SNAplusAPI configuration file. For more information, see the appendix Netu Procedures.

The rate at which Ingres Net polls for incoming connection requests can be controlled through the configuration variable (in config.dat file)
ii. `<host>.gcc.*.sna_lu62.poll` (xx is the installation ID.) This specifies the polling rate in milliseconds. The polling rate defaults to 4000 (4 seconds); values smaller than this are not recommended. If there are no incoming connection requests, you can inhibit polling by setting the environment variable to the special value of -1.

Connection Data Entry Information:

Protocol: sna_lu62

Network address: *[lu_alias].plu_alias[.mode_name]*

where:

lu_alias

Is the alias for the Local LU to be used by the connection.

The alias must match the name of a Local LU alias established during configuration. If an LU from the pool of default Local LUs is to be used, the alias is omitted.

plu_alias

Is the alias by which the Partner LU for the remote instance is known.

The alias must match the name of a Remote LU alias established during configuration. Additionally, the alias must have been configured as a Partner LU for the specified local LU.

mode_name

Is the name of a set of networking characteristics defined during configuration.

The name must match the name of a mode assigned during configuration with the pair of the specified Local LU and Partner LU. If a blank mode name has been configured, the name (and the preceding ".") is omitted.

Listen Address: *tp_name*

where:

tp_name

Is the transaction program name that is used by the Net SNA_LU62 protocol driver on the server node. If the remote instance is on an MVS system, there is no transaction program name. Enter **x**. For all other systems, the *tp_name* must be the transaction program name specified in the remote instance. This is generally found in the remote instance's listen address.

RS/6000

When you install Ingres Net on a RS/6000 that is using the SNA LU62 protocol, the listen address has the format:

[/pathname/] connection_profile.tp_profile.tp_name

where:

pathname

Is the name of the SNA device driver.

Default: dev/sna

connection_profile tp_profile

Refers to the names of configuration profiles that must be defined before running Ingres Net.

For information on how to define these profiles, see Using AIX SNA Services/6000 and AIX SNA Services/6000 Reference.

The AIX SNA Services/6000 Configuration Profiles appendix contains samples of connection and tp profiles suitable for Ingres Net.

tp_name

Is the transaction program name used by the listening process

Connection Data Entry Information:

Protocol: sna_lu62

Network address: *connection_profile*

The *connection_profile* is a profile defined by AIX SNA Services/6000 configuration utilities for sessions between the AIX client and the remote instance. There is no default for the *connection_profile*.

Listen Address: *tp_name*

where:

tp_name

Is the transaction program name that is used by the Net SNA_LU62 protocol driver on the server node. If the remote instance is on an MVS system, there is no transaction program name. Enter **x**. For all other systems, the *tp_name* must be the transaction program name specified in the remote instance. This is generally found in the remote instance's listen address.

Appendix D: SPX/IPX Protocol

This section contains the following topics:

[Listen Address Format](#) (see page 371)

This appendix describes the format of a listen address when the protocol between two machines is Novell Netware SPX/IPX. It also gives you the protocol- and platform-specific information to set up connection data entries.

Listen Address Format

A listen address is a unique identifier used for interprocess communications. A Communications Server has two kinds of listen addresses. It uses one to receive messages from local processes and the other to receive messages from remote Communications Servers.

This section describes the format of the listen address used to receive messages from remote processes when the network protocol is SPX/IPX.

Note: To view or change your instance's listen addresses, use the Configuration-By-Forms (cbf) utility or Configuration Manager (vcbf).

Windows

An SPX/IPX listen address has the following format:

xxxxxxxx

where:

xxxxxxxx

Is a hexadecimal number from 00000000 to ffffffff

The listen address is the SPX/IPX "net number," an 8-digit hexadecimal number. Both the net address and the 12-digit node number for a server are repeated in the errlog.log file when the Communications Server starts.

Connection Data Entry Information:

Protocol: nvlspix

Network address: *node*

where:

node

Is a 12-digit hexadecimal SPX node number

UNIX and VMS

SPX treats listen addresses as a 16-bit quantity, normally represented in hexadecimal as 0000 - FFFF. As a convenience, the Ingres SPX/IPX protocol driver recognizes an installation ID (followed by an optional digit) as a listen address and translates that into a 16-bit value in the range 4000 - 4FFF, which Novell has reserved for dynamically allocated listen addresses.

An SPX/IPX listen address has two possible formats:

ab[n] or *xxxx*

where:

a

An alphabetic character (case is not significant)

b

An alphabetic character or a decimal digit (0-9)

n

The 0 or 1 digit

xxxx

A hexadecimal number from 0000 to ffff

The format *ab[n]* is the default format, where *ab* is the installation ID (found in `II_INSTALLATION`) and *n* = 0. The digit *n* is incremented by 1 for each successive Communications server started in an installation. For example, if the installation has two Communications Servers using the default format for their listen addresses, the addresses are *ab0* and *ab1*. Only two Communications Servers using the default listen addresses for SPX/IPX can be started in a single Ingres instance.

To accept a connection from Ingres running on a PC, the listen address must be set to 6582. The PC currently does not support selecting alternate listen addresses.

Connection Data Entry Information:

Protocol: spx

Network address: *network.node*

where:

network

Is a hexadecimal SPX network number

node

Is a hexadecimal SPX node number

The Novell utility getlan reports the local network and node numbers for the internal network and all external network connections. Because most hosts have only a single external network connection, getlan normally reports two networks. The internal network is always the first in the list (Lan number 0), and it is the address Ingres Net requires.

For example, if getlan reports:

LAN	Network	Node	Mux ID	State	Stream
0	119D9D21	000000000001	00000000	UNBOUND	OK
1	00000900	080020101FC7	00000059	IDLE	OK

The network address is:

119D9D21.1

Leading zeroes are not important.

Listen Address: *ab[n] | xxxx*

Appendix E: DECnet Protocol

This section contains the following topics:

[Listen Address Format](#) (see page 375)

This appendix describes the format of a listen address when the protocol between two machines is DECnet. It also gives you the protocol- and platform-specific information to set up connection data entries.

Listen Address Format

A listen address is a unique identifier used for interprocess communications. A Communications Server has two kinds of listen addresses. It uses one to receive messages from local processes and the other to receive messages from remote Communications Servers.

This section describes the format of the listen address used to receive messages from remote processes when the network protocol is DECnet.

Note: To view or change your instance's listen addresses, use the Configuration-By-Forms (cbf) utility or Configuration Manager (vcbf).

VMS

A DECnet listen address is a DECnet object and has the format:

II_GCC[xx]_nnnnn

where :

xx

The installation ID, found in II_INSTALLATION.

The installation ID is only present for group level installations.

nnnnn

A five-digit number that you specify when you install Net. The default for this number is 0.

The default listen address is II_GCC[xx]_0.

In netutil, the network address prompt is in DECnet node name in character form. You can specify the node address and name in the format *area-number.node_number* (for example: 1.234) instead of the DECnet node name. If you are running DECnet-Plus, you can specify the format *namespace::directory_path.node_object* (for example, local::mynode). DECnet-Plus users can specify node names larger than six characters.

Connection Data Entry Information:

Protocol: decnet

Network address: *node_name*

The *node_name* is the DECnet node name in character form.

Listen address: II_GCC[xx]_nnnnn

Appendix F: LAN Manager Protocol

This section contains the following topics:

[LAN Manager Listen Address—Enable Communications](#) (see page 377)

This appendix describes the format of a listen address when the protocol between two machines is Microsoft LAN Manager. It also gives you the protocol- and platform-specific information to set up connection data entries.

LAN Manager Listen Address—Enable Communications

A listen address is a unique identifier used for interprocess communications. A Communications Server has two kinds of listen addresses. It uses one to receive messages from local processes and the other to receive messages from remote Communications Servers.

This section describes the format of the listen address used to receive messages from remote processes when the network protocol is LAN Manager.

A LAN Manager listen address is the installation ID (found in II_INSTALLATION).

Note: To view or change your instance's listen addresses, use the Configuration-By-Forms (cbf) utility or Configuration Manager (vcbf).

Connection Data Entry Information:

Protocol: lanman

Network address: *computername*

where:

computername

Is the Computer Name assigned during installation of Windows or through the Network applet in the Control Panel. The current Computer Name can be viewed along with the name of the current logged-in user at the top of the Program Manager window. It can also be viewed in the operating system environment variable COMPUTERNAME by typing **echo %COMPUTERNAME%**.

Listen address: *installation ID*

Appendix G: SunLink Gateway Configuration Files

This section contains the following topics:

[SunLink Gateway Configuration File](#) (see page 379)

[Solaris Independent LUs](#) (see page 380)

[Solaris Dependent LUs](#) (see page 382)

[SunOS \(or Sun-4\) Independent LUs](#) (see page 384)

[SunOS \(or Sun-4\) Dependent LUs](#) (see page 386)

Ingres Net supports communications over both dependent and independent Logical Units (LUs). This appendix contains sample configuration file excerpts that show how to configure both types of LUs. For information about setting up the configuration files, see the *SunLink SNA Peer-to-Peer System Administrator's Guide*.

SunLink Gateway Configuration File

The gateway configuration file is named `/etc/appcs` and must be present on the SunLink Gateway machine as well as on each Sun Solaris or Sun-4 machine running Net providing SNA LU62 support. It is not necessary to have Net installed on the same machine as the SunLink Gateway, but it must be connected and accessible through TCP/IP.

The following entry defines the SunLink Gateway to other Sun machines that require access to it:

```
ws406sgw0 ws406s:ws406sgw0
```

where:

ws406sgw0

Is the SunLink Gateway name.

ws406s

Is the machine name on which the SunLink Gateway is running.

Solaris Independent LUs

The following example is for Solaris independent LUs. This file is usually in the /opt/SUNWconn/snap2p/p2p_etc/config directory.

```
:DEFINE_PU:
pu_name          = S1MVS, network_name = RTIBM
contents_id      = 01234567

:DEFINE_NODE:
pu_name          = S1MVS, node_id = NODE0

:DEFINE_LOCAL_LU:
fql_lu_name      = S1115001  # An LU name in VTAM/NCP gen

lu_local_address = 1
lu_name          = S1115001  # An LU name in VTAM/NCP gen

lu_session_limit = 16

:DEFINE_PARTNER_LU:
fql plu name     = A04IS1G2  # VTAM Applid for Ingres
u plu name       = A04IS1G2  # Enterprise Access to DB2
parallel_session = yes       # Must set to this value
lu_is_dependent  = no        # Must set to this value
initiate_type    = INITIATE_ONLY
security_acceptance NONE

:DEFINE_MODE:
mode_name        = INGLU62
unique_session_name = s1      # This is the name specified
                                # as the Node Address
                                # in NETU entries

snd_pac_window   = 0          # Recommended
rcv_pac_window   = 0          # Recommended
snd_max_ru_size  = 4096       # Recommended
rcv_max_ru_size  = 4096       # Recommended
sync_level       = none
sess_reinit      = INIT_OPERATOR
auto_activate_limit = 0
session_limit     = 64        # Allows for 64 parallel sessions
min_conwinner_limit = 32
min_conloser_limit = 32
```

The following example is for a Synchronous Data Link Control (SDLC) connection through the serial port:

```
:DEFINE_DLC:
dlc_name          = XLINK000
dlc_driver_name    = /dev/sdlc
port_driver_name   = zsh0
dlc_type           = sdlc
npr_timeout        = 240
pause_timeout      = 2

idle_timeout       = 1400      # for maxdata = 1033
                                # & line speed = 9600
maxdata            = 1033      # [frm_size - 8]
retries            = 32
window_size        = 7
sdlc_addr          = 0x1
full_duplex        = yes
nrzi               = no
multipoint         = yes
switched_line      = no

block_number       = 056      # MUST be first of
                                # xid parameters
id_number          = E2E43
role               = secondary
tx_rx_capability   = simultaneous
max_rcv_iframe_size = 7
include_control_point = yes    # xid control vector
include_link_station_name = yes # xid control vector

:DEFINE_ALS:
dlc_name          = XLINK000
pu_name           = S1MVS
als_name          = XXALS000
```

Solaris Dependent LUs

The following example is for Solaris dependent LUs. This file is usually in the opt/SUNWconn/snap2p/p2_etc/config directory.

```
:DEFINE_PU:
pu_name          = S1MVS, network_name = RTIBM
contents_id      = 01234567

:DEFINE_NODE:
pu_name          = S1MVS, node_id = NODE0

:DEFINE_LOCAL_LU:
fql_lu_name      = RTIBM.S1115001 # An LU name in VTAM/NCP gen
lu_local_address = 1
lu_name          = S1115001      # An LU name in VTAM/NCP gen
lu_session_limit = 1

:DEFINE_PARTNER_LU:
fql_plu_name     = RTIBM.A04IS1G2 # VTAM Applid for
                                     # Ingres
                                     # Enterprise Access
                                     # to DB2
parallel_session = no             # Must set to this value
lu_is_dependent  = yes           # Must set to this value
initiate_type    = INITIATE_ONLY
security_acceptance = NONE

:DEFINE_MODE:
mode_name        = INGLU62
unique_session_name = s1          # This is the name specified
                                     # as the Node Address in
                                     # NETU entries
snd_pac_window   = 0              # Recommended
rcv_pac_window   = 0              # Recommended
snd_max_ru_size  = 4096           # Recommended
rcv_max_ru_size  = 4096           # Recommended
sync_level       = none
sess_reinit      = INIT_PLU_OR_SLU
auto_activate_limit = 0
session_limit     = 1             # Must set to this value
min_conwinner_limit = 1
min_conloser_limit = 0
```

The following example is for an SDLC connection through the serial port:

```
:DEFINE_DLC:
dlc_name          = XLINK000
device_driver_name = /dev/sdlc
port_driver_name  = zsh0
dlc_type          = sdlc
npr_timeout       = 240
pause_timeout     = 2
idle_timeout      = 1400      # for maxdata = 1033
                                # & line speed = 9600
maxdata           = 1033      # [frm_size - 8]
retries           = 32
window_size       = 7
sdlc_addr         = 0x1
full_duplex       = yes
nrzi              = no
multipoint        = yes
switched_line     = no
block_number      = 056      # MUST be first of xid
                                # parameters
id_number         = E2E43
role              = secondary
tx_rx_capability  = simultaneous
max_rcv_iframe_size = 7
include_control_point = yes    # xid control vector
include_link_station_name = yes # xid control vector

:DEFINE_ALS:
dlc_name          = XLINK000
pu_name           = S1MVS
als_name          = XXALS000
```

SunOS (or Sun-4) Independent LUs

The following example is for SunOS (or Sun-4) independent LUs. This file is usually in the opt/SUNWconn/snap2p/p2p_etc/config directory.

```
:DEFINE_PU:
pu_name          = S1MVS, network_name = RTIBM
contents_id      = 01234567

:DEFINE_NODE:
pu_name          = S1MVS, node_id = NODE0

:DEFINE_LOCAL_LU:
fql_lu_name      = S1115001      # An LU name in VTAM/NCP gen

lu_local_address = 1
lu_name          = S1115001      # An LU name in VTAM/NCP gen
lu_session_limit = 16

:DEFINE_PARTNER_LU:
fql_plu_name     = A04IS1G2      # VTAM Applid for
                                # Ingres
u_plu_name       = A04IS1G2      # Enterprise Access to DB2
parallel_session = 1             # Must set to this value
cnos_supported   = 1             # Must set to this value
remote_is_sscp   = 0             # Must set to this value
initiate_type    = INITIATE_ONLY
security_acceptance = NONE

:DEFINE_MODE:
mode_name        = INGLU62
unique_session_name = s1         # This is the name specified
                                # as the Node Address in
                                # NETU entries

snd_pac_window   = 0             # Recommended
rcv_pac_window   = 0             # Recommended
snd_max_ru_size  = 4096          # Recommended
rcv_max_ru_size  = 4096          # Recommended
sync_level       = 0
sess_reinit      = INIT_OPERATOR
auto_activate_limit = 0
session_limit     = 64           # Allows for 64
                                # parallel sessions

min_conwinner_limit = 32
min_conloser_limit  = 32
```


The following example is for an SDLC connection through the serial port:

```
:DEFINE_DLC:
dlc_name          = XLINK000
device_driver_name = /dev/dfd0
dlc_type          = 0
npr_timeout       = 240
pause_timeout     = 2
idle_timeout      = 400          # for maxdata = 1033
                                   # & line speed = 9600
                                   # [frm_size - 8]
frm_size          = 1033
retries           = 32
window_size       = 7
rxaddr            = 0x1
txaddr            = 0x1
full_duplex       = yes
nrzi              = no
multipoint        = yes
addr.search       = no
switched_line     = no
send_reject       = no
rcv_reject        = no
block_number      = 056          # MUST be first of xid
                                   # parameters
id_number         = E2E43
abm_support       = no
max_btu_rev       = 265
sim_rlm           = no
role              = secondary
tx_rx_capability  = simultaneous
max_btu_rcv       = 265
max_rcv_iframe_siz = 7
include_control_point = yes      # xid control vector
include_link_station_name = yes  # xid control vector
product_set_id = 161101130011f9f4f0f4c3f1f0f1f0f0f0f2f4f1f6f4

:DEFINE_ALS:
dlc_name          = XLINK000
pu_name           = S1MVS
als_name          = XXALS000
remote_addr       = 0x10
```

SunOS (or Sun-4) Dependent LUs

The following example is for SunOS (or Sun-4) dependent LUs. This file is usually in the opt/SUNWconn/snap2p/p2_etc/config directory.

```
:DEFINE_PU:
pu_name           = S1MVS, network_name = RTIBM
contents_id       = 01234567

:DEFINE_NODE:
pu_name           = S1MVS, node_id = NODE0

:DEFINE_LOCAL_LU:
fql_lu_name       = RTIBM.S1115001 # An LU name in VTAM/NCP gen
lu_local_address  = 1
lu_name           = S1115001      # An LU name in VTAM/NCP gen
lu_session_limit  = 1

:DEFINE_PARTNER_LU:
fql_plu_name      = RTIBM.A04IS1G2 # VTAM Applid for
                                     # Ingres
                                     # Enterprise Access to DB2
parallel_session  = 0              # Must set to this value
cnos_supported    = 0              # Must set to this value
remote_is_sscp    = 1              # Must set to this value
initiate_type     = INITIATE_ONLY
security_acceptance = NONE

:DEFINE_MODE:
mode_name         = INGLU62
unique_session_name = s1           # This is the name
                                     # specified
                                     # as the Node Address in
                                     # NETU entries
snd_pac_window    = 0              # Recommended
rcv_pac_window    = 0              # Recommended
snd_max_ru_size   = 4096           # Recommended
rcv_max_ru_size   = 4096           # Recommended
sync_level        = 0
sess_reinit       = INIT_PLU_OR_SLU
auto_activate_limit = 0
session_limit     = 1              # Must set to this value
min_conwinner_limit = 1
min_conloser_limit = 0
```

The following example is for an SDLC connection through the serial port:

```
:DEFINE_DLC:
dlc_name           = XLINK000
device_driver_name = /dev/ird0
dlc_type           = 0
npr_timeout        = 240
pause_timeout      = 2

idle_timeout       = 400           # for maxdata = 1033 & line
                                   # speed = 9600
                                   # [frm_size - 8]
frm_size           = 1033
retries            = 32
window_size        = 7
rxaddr             = 0x1
txaddr             = 0x1
full_duplex        = yes
nrzi               = no
multipoint         = yes
addr.search        = no
switched_line      = no
send_reject        = no
rcv_reject         = no
block_number       = 056          # MUST be first of xid
                                   # parameters
id_number          = E2E43
abm_support        = no
max_btu_rcv        = 265
sim_rlm            = no
role               = secondary
tx_rx_capability   = simultaneous
max_btu_rcv        = 265
max_rcv_iframe_size = 7
include_control_point = yes       # xid control vector
include_link_station_name = yes   # xid control vector
product_set_id = 161101130011f9f4f0f4c3f1f0f1f0f0f0f2f4f1f6f4

:DEFINE_ALS:
dlc_name           = XLINK000
pu_name            = S1MVS
als_name           = XXALS000
remote_addr        = 0x10
```


Appendix H: AIX SNA Services/6000 Configuration Profiles

This section contains the following topics:

[Sample Configuration Profiles](#) (see page 389)

AIX SNA Services/6000 configuration is done by defining a series of profiles. For detailed information about this process, see the guides *Using AIX SNA Services/6000* and *AIX SNA Services/6000 Reference*. This appendix includes information about only those aspects of configuration particular to netutil.

Sample Configuration Profiles

The following excerpts provide examples of profiles suitable for running netutil. The format shown here is roughly the same as the output from the AIX SNA Services/6000 `exportsna` command. Comments (denoted by the symbol `#`) are included only for the purpose of explanation; these do *not* appear in the actual configuration files.

The configuration below reflects an environment using both dependent and independent LUs. Note that independent and dependent LUs can share the same ATTACHMENT, TPN, and REMOTETPN profiles. However, separate CONNECTION, LOCALLU, and MODE profiles must be provided for dependent and independent LUs respectively.

The following samples include profiles for independent and dependent CONNECTION, LOCALLU, and MODE LUs.

CONNECTION Profile for Independent LUs

```
indconn_CONNECTION:
  type = CONNECTION
  profile_name = indconn                                #this is the name specified
                                                         # as the Network Address
                                                         # in netutil entries

  attachment_profile_name = rs6_attach
  local_lu_profile_name = indlu
  network_name = RTIBM                                #actual SNA network name
  remote_lu_name = GCVDEV1                            #actual target LU name
  stop_connection_on_inactivity = no                  #must set to this value
  lu_type = lu6.2                                     #must set to this value
  interface_type = extended                           #must set to this value
  remote_tpn_list_name = INET
  mode_list_name = INDMODE
  node_verification = no
  inactivity_timeout_value = 0
  notify = no
  parallel_sessions = parallel                        #independent LUs only
  negotiate_session_limits = yes                     #independent LUs only
  security_accepted = none
  conversation_security_access_list_name =
```

CONNECTION Profile for Dependent LUs

```
depconn_CONNECTION:
  type = CONNECTION
  profile_name = depconn                                #this is the name specified
                                                         # as the Network Address
                                                         # in netutil entries

  attachment_profile_name = rs6_attach
  local_lu_profile_name = deplu
  network_name = RTIBM                                #actual SNA network name
  remote_lu_name = GCVDEV1                            #actual target LU name
  stop_connection_on_inactivity = no                  #must set to this value
  lu_type = lu6.2                                     #must set to this value
  interface_type = extended                           #must set to this value
  remote_tpn_list_name = INET
  mode_list_name = DEPMODE
  node_verification = no
  inactivity_timeout_value = 0
  notify = no
  parallel_sessions = single                          #dependent LUs only
  negotiate_session_limits = no                       #dependent LUs only
  security_accepted = none
  conversation_security_access_list_name =
```

LOCALLU Profile for Independent LU

```
indlu_LOCALLU:  
  type = LOCALLU  
  profile_name = indlu  
  local_lu_name = S1114000  
  network_name = RTIBM  
  lu_type = lu6.2  
  independent_lu = yes  
  tpn_list_name = IIGCC  
  local_lu_address = 99  
  sscp_id =  
  number_of_rows = 24  
  number_of_columns = 80
```

```
#actual local LU name  
#actual SNA network name  
#must set to this value  
#indicates independent LU  
  
#ignored for independent LUs  
#ignored for independent LUs
```

LOCALLU Profile for Dependent LU

```
deplu_LOCALLU:
  type = LOCALLU
  profile_name = deplu
  local_lu_name = S111400G          #actual local LU name
  network_name = RTIBM             #actual SNA network name
  lu_type = lu6.2                  #must set to this value
  independent_lu = no              #indicates dependent LU
  tpn_list_name = IIGCC
  local_lu_address = 1             #actual local LU address
  sscp_id = 0500000000001         #actual SSCP id
  number_of_rows = 24
  number_of_columns = 80

inet_REMOTETPN:
  type = REMOTETPN
  profile_name = inet
  tpn_name = x                     #this is the name specified
                                   # as Listen Address
                                   # in netutil entries

  tpn_name_hex = A7
  pip_data = no                   #must set to this value
  conversation_type = mapped      #must set to this value
  recovery_level = no_reconnect   #must set to this value
  sync_level = none               #must set to this value
  tpn_name_in_hex = no

INET_REMOTETPNLIST:
  type = REMOTETPNLIST
  Listname = INET
  list_members = inet

iigcc_TPN:
  type = TPN
  profile_name = iigcc
  tpn_name = iigcc
  tpn_name_hex = 8989878383
  conversation_type = mapped      #must set to this value
  pip_data = no                  #must set to this value
  sync_level = none              #must set to this value
  recovery_level = no_reconnect   #must set to this value
  full_path_to_tpn_executable = /x #ignored for Net
  multiple_instances = yes
  user_id = 777
  server_synonym_name =
  restart_action = once
  communication_type = signals
  stdin = /dev/null
  stdout = /tmp/tpn_output
  stderr = /tmp/tpn_error
  subfields = 0
  communication_ipc_queue_key = 0
  tpn_name_in_hex = no
  security_required = none
  resource_security_access_list_name =

IIGCC_TPNLIST:
  type = TPNLIST
```



```
Listname = IIGCC
list_members = iigcc
```

MODE Profile for Independent LUs

```
indmode_MODE:
  type = MODE
  profile_name = indmode
  mode_name = INGLU62                #actual mode name
  maximum_number_of_sessions = 200
  minimum_contention_winners = 50
  minimum_contention_losers = 5
  receive_pacing = 3                 #default
  send_pacing = 3                    #default
  maximum_ru_size = 2816             #default
  recovery_level = no_reconnect      #must set to this value

INDMODE_MODELIST:
  type = MODELIST
  Listname = INDMODE
  list_members = indmode
```

MODE Profile for Dependent LUs

```
depmode_MODE:
  type = MODE
  profile_name = depmode
  mode_name = INGSLU62              #actual mode name
  maximum_number_of_sessions = 200
  minimum_contention_winners = 50
  minimum_contention_losers = 5
  receive_pacing = 3                 #default
  send_pacing = 3                    #default
  maximum_ru_size = 2816             #default
  recovery_level = no_reconnect      #must set to this value

DEPMODE_MODELIST:
  type = MODELIST
  Listname = DEPMODE
  list_members = depmode
```


Appendix I: HP-UX SNAplus Configuration

This section contains the following topics:

[Sample Configuration File Excerpts](#) (see page 395)

Connectivity supports communications over both dependent and independent Logical Units (LUs). This appendix contains sample configuration file excerpts (as produced by the HP-UX SNAplus configuration file print utility, `snapshotcfg`) that show how to configure both types of LUs. Additionally, an excerpt is included that illustrates the required configuration for a dynamically loadable TP, which is required if connections incoming to HP-UX are to be supported.

Sample Configuration File Excerpts

The following samples include excerpts for independent and dependent LUs and for a dynamically loadable TP.

Independent LUs

The following are sample excerpts for independent LUs:

```
*****
* APPC Local LU Record *
*****
Local LU name .....S1110000
Description .....[LU for MVS DB2 access]
Owning local node name .....NODE
Network ID .....[RTIBM]
Network name .....S1110000

Session limit .....64
Default LU? .....No
Locally usable? .....No
LU number .....0
Conversation-level security? ...No
Prevalidation ability? .....No
Number of remote LUs ..... 3
Remote LU #1:
  Remote LU name ..... GCVDEV1
  Number of modes ..... 1
  List of mode IDs ..... 000
*****
* APPC Mode Data Record *
*****
Mode name ..... INGLU62
Description ..... [INGLU62 mode for MVS]
Owning connection name ... [CONN1]
Mode ID ..... 000

High priority mode? ..... Yes
Session limit ..... 64
Auto activation limit .... 16
Min contention losers .... 32
Min contention winners ... 32
Send pacing count ..... 0
Receive pacing count ..... 0
Send RU size ..... 256 (min) to 4096 (max)
Receive RU size ..... 256 (min) to 4096 (max)
*****
* APPC Remote LU Record *
*****
LU alias .....GCVDEV1
Description .....[Remote LU for DB2 Gateway]
Network ID .....RTIBM
Remote LU name .....C5X6IC55

Prevalidation ability? .....No
Parallel sessions? .....Yes
Conversation-level security? ...No
Uninterpreted LU name .....C5X6IC55
```

Dependent LUs

The following are sample excerpts for dependent LUs:

```
*****
* APPC Local LU Record *
*****
Local LU name .....S1110008
Description .....[LU for MVS DB2 access]
Owning local node name .....NODE
Network ID .....[RTIBM]
Network name .....S1110008

Session limit .....1
Default LU? .....No
Locally usable? .....No
LU number .....1
Conversation-level security? ...No
Prevalidation ability? .....No
Number of remote LUs .....1
Remote LU #1:
  Remote LU name .....GCVSDEV1
  Number of modes .....1
  List of mode IDs .....003

*****
* APPC Mode Data Record *
*****
Mode name .....INGSLU62
Description .....[INGLU62 mode for MVS]
Owning connection name .....[CONN1]
Mode ID .....003

High priority mode? .....Yes
Session limit .....1
Auto activation limit .....0
Min contention losers .....0
Min contention winners ....1
Send pacing count .....0
Receive pacing count .....0
Send RU size .....256 (min) to 4096 (max)
Receive RU size .....256 (min) to 4096 (max)

*****
* APPC Remote LU Record *
*****
LU alias ..... GCVSDEV1
Description ..... [Remote LU for DB2 Gateway]
Network ID ..... RTIBM
Remote LU name ..... C5X6ICS5

Prevalidation ability? ..... No
Parallel sessions? ..... No
Conversation-level security? . No
Uninterpreted LU name ..... C5X6ICS5
```

Dynamically Loadable TP

The following is a sample excerpt for a dynamically loadable TP to support Connectivity connections incoming to HP-UX. Note that the executable file specified is not used by “Queued - operator started” TPs, and that the “Receive allocate timeout” must be set to 0 seconds to avoid the Communications server blocking for incoming connections.

```
*****
* Dynamically Loadable TP Record *
*****
Local TP name ..... TEST
Description ..... [test invocable TP]
TP type ..... APPC
Queueing scheme ..... Queued - operator started
Conversation security? ..... No
Accept already-verified? ..... No
Full TP name ..... test
Executable file ..... [junk]

Parameters ..... []
Environment string ..... []
Target machine name ..... []
Attach timeout ..... 3600 sec
Receive allocate timeout ..... 0 sec
```

Appendix J: Netu Procedures

This section contains the following topics:

[Netu \(Deprecated\)](#) (see page 399)

[Start Netu](#) (see page 399)

[Netu User Interface](#) (see page 400)

[Remote Node Definition Operations](#) (see page 402)

[Remote User Authorization Operations](#) (see page 410)


[Netu Options for Stopping the Communications Server](#) (see page 417)

Netu (Deprecated)

Netu (Net Management Utility) was provided in Ingres 6.4 and previous releases for establishing and maintaining remote connections. The netu utility is replaced by the forms-based netutil utility. It is still possible, however, to establish remote connections using netu.

Start Netu

To access netu, follow the instructions below for your operating system:

Windows: At the command prompt, change directory to %II_SYSTEM%/ingres/sig/netu, and then type **netu**. 

UNIX: Verify that the environment variable II_SYSTEM is set to the location of your current Ingres instance and that \$II_SYSTEM/ingres/sig/netu is in the search path of the user who owns the installation. If not, do this now.


C shell:

```
set path=($path $II_SYSTEM/ingres/sig/netu)
```

Bourne shell:

```
PATH=$PATH:$II_SYSTEM/ingres/sig/netu 
```

VMS: Enter the following at your operating system prompt:

```
netu== $ii_system:[ingres.sig.netu]netu.exe 
```

Netu User Interface

The netu user interface's primary component is a menu that appears when you start netu. The menu looks like this:

```
Select one of the following:
Q <Comm_Server_Id> - Quiesce Ingres/Net
S <Comm_Server_Id> - Stop Ingres/Net
N - Modify Node Entry
A - Modify Remote Authorization Entry
E - Exit
```

Remember these tips when using netu:

- The netu user interface is not case sensitive. When you make a selection from the main menu, use either an upper- or lowercase entry for the selection.
- Press Enter after every response you make or every menu item that you select.
- The netu utility does not check for valid entries in response to prompts. Make sure your entries are accurate.
- Abort an operation by pressing Esc and Enter.

Stop the Communications Server

Choosing Q or S stops the Communications Server. The Q selection stops the server after all sessions currently in progress terminate. The S selection stops it immediately, disconnecting any sessions that are open. For more information about these procedures, see the chapter "Maintaining Connectivity."

Modify Node Entry

Choosing N allows you to:

- Add remote node definitions
- Merge remote node definitions (add a remote node definition whose vnode name matches that of an existing node definition)
- Delete remote node definitions
- Retrieve remote node definitions for viewing

It is also possible to change an existing remote node definition, using the Add option or a combination of Add and Delete. See *How You Change Remote Node Definitions* (see page 406).

If you are a system administrator with Ingres privileges, define global or private node definitions. If you are not a system administrator with Ingres privileges, define only private remote node definitions. For a discussion of the differences between private and global definitions, see the background information in *Remote Node Definition Operations* (see page 402).

Modify Remote Authorization Entry

Choosing A lets you:

- Add remote user authorizations
- Delete remote user authorizations
- Retrieve remote user authorizations for viewing

It is also possible to change an existing remote user authorization using the Add option or a combination of the Add and Delete options. See *Change Remote User Authorizations* (see page 413).

If you are an Ingres administrator, establish global or private remote node authorizations. If you are not a system administrator with Ingres privileges, set up only private remote node authorizations.

Exit Netu

Choosing E exits the utility.

You exit the netu utility from its main menu. You can reach this menu from any netu operation by choosing exit at that operation's command line prompt.

If you are in the middle of an operation and want to quit without completing the operation, press Esc and Enter to open the netu menu. From there, exit netu or choose another operation.

Note: If your keyboard lacks an Escape key, use Control + [to quit without completing the operation.

Remote Node Definition Operations

There are four operations associated with remote node definitions:

- Adding new definitions
- Merging new definitions
- Deleting existing definitions
- Viewing existing definitions

To perform each of these operations, netu asks for the following information:

- The vnode name of the remote node
- The network protocol used by the remote node
- The remote node address
- The listen address of the Communications server at the remote node

Add or Merge Remote Node Definitions

A remote node definition identifies a particular node and a listen address for that node's Communications server and associates that node and address combination with a vnode name.

When a user uses a vnode name to connect to a database on a remote instance, the local instance must have a remote node definition that defines that vnode name for the remote instance to complete the connection. The `netu` utility offers two options for adding remote node definitions: `add` and `merge`.

`add` and `merge` differ in how they handle the addition of a node definition whose vnode name matches the vnode name of an existing node definition. If you are using the `add` option, `netu` overwrites any existing node definitions that have the matching vnode name. If you are using the `merge` option, `netu` does not overwrite the existing definitions, but simply establishes another definition. The `merge` option is very useful if you want to run more than one Communications Server at a server node.

For example, assume that you want to run three Communications Servers at the node "eugenie." Each server has its own unique listen address for interprocess communications. If you use the `add` option to establish the remote node definitions for "eugenie" from "napoleon," you must provide a unique vnode name for each listen address. For example, you have the following vnode name and listen address combinations:

From napoleon:

```
Royal addr1
Lady addr2
Second addr3
```

If you use the `merge` option, you need only one vnode name. Using `merge`, set up the three node definitions at "napoleon" using the same vnode name for each and simply changing the listen address.

For example:

From napoleon:

```
Royal addr1
Royal addr2
Royal addr3
```

When users connect using the vnode name "Royal" Ingres Net connects them to one of the three Communications Servers. Ingres Net automatically tries each server, in random order, until it finds one of the three that is available. Users do not need to remember three vnode names or make three connection attempts. Using merge allows you to keep it simple for users, regardless of how many Communications Servers are running on an installation.

Note: Ingres Net does not allow two definitions that are exactly the same at the same node.

To add or merge a remote node definition

1. Start netu by entering netu at the operating system prompt.

The netu menu appears.

2. Select N (Modify Node Entry) from the menu.

The following prompt appears:

Enter command (add, merge, del, show, exit):

3. Select add or merge. (Type a or m instead of the full word.)

The add and merge options behave differently if the definition you are adding matches an existing vnode name. Be sure to read the paragraphs preceding this procedure before making a choice.

4. Define the account as a private or global account. The default is private.

- To accept the default, press Enter.
- To define a global node entry, enter G. You must be a user with Ingres privileges to define a global node entry.

5. Enter a remote vnode name.

6. Enter the network software type.

This is the name of the protocol that the remote node is using. For a list of valid entries see Network Protocol Keywords (see page 58).

7. Enter the remote node address.

8. Enter the listen address of the remote node's Communications Server.

Netu adds one remote node definition for your local node and displays this prompt:

Enter operation (add, merge, del, show, exit):

9. Select another operation or select exit to Enter to the netu menu.

Delete Remote Node Definitions

To remove a remote node definition from a local node, the netu utility allows you to remove one or several definitions at a time. When you select the operation that deletes node definitions, netu responds with a series of prompts. When you have answered all the prompts, netu deletes all node definitions that match the answers you supplied.

To remove several definitions at once, use the asterisk (*) in response to the appropriate prompt. This is a wild card character that matches any entry. For example, if you want to remove all private node definitions for the vnode name "general," complete the deletion procedure, responding to the prompts in the following manner:

```
Enter Private or Global (P): <Enter>
Enter the vnode name of the remote node: general
Enter the node address of the remote node: *
Enter the network software type:
Enter the remote node address: *
Enter the remote Ingres/Net server listen address:
appropriate address
```

You cannot answer with an asterisk in response to the Global or Private prompt.

To remove node definitions from the local Communications Server

1. Start netu by entering netu at the operating system prompt.
The netu menu appears.
2. Select N (Modify Node Entry) from the menu.
The following prompt appears:
Enter command (add, merge, del, show, exit):
3. Enter del. (Type d in place of del.)
4. When netu asks you to specify if the definition is a private or global definition, do one of the following steps:
 - If the node definition is private, press Enter.
 - If the node definition is global, enter G. You must be a user with Ingres privileges to select G.
5. Enter the vnode name of the remote node. Enter the network software type.
This is the name of the protocol that the remote node is using. Valid entries are described in Network Protocol Keywords (see page 58).
6. Enter the node address of the remote node.

7. Enter the listen address of the remote node's Communications Server.
Netu removes the node definition(s) from the local Communications Server and displays the following prompt:
Enter command (add, merge, del, show, exit):
8. Select another operation, or Enter to the netu menu by entering exit.

How You Change Remote Node Definitions

The procedure you use for changing an existing remote node definition depends on whether the vnode name for the definition is unique among the node definitions for the installation.

If the vnode name is unique (that is, associated with only one node definition), overwrite the existing definition. For instructions, see Overwrite an Existing Definition (see page 406).

If the vnode name is not unique (that is, associated with more than one node definition), you must delete the incorrect definition and set up the new definition. For instructions, see Delete Old and Add New Definition (see page 407).

Overwrite an Existing Definition

Use this procedure to change a node definition only if the vnode name associated with that definition *is not* associated with any other node definitions within the local set of remote node definitions.

To overwrite an existing definition

1. Start netu by entering netu at the operating system prompt.
The main menu appears.
2. Select N (Modify Node Entry) from the menu.
The following prompt appears:
Enter the operation (add, merge, del, show, exit):
3. Enter add. (Type a instead of add.)
Prompts appear.
4. Answer the prompts using the values of the new, correct node definition.
The utility overwrites the existing node definition with the values that you have just supplied and displays the following prompt:
Enter operation (add, merge, del, show, exit):
5. Continue with other node definition tasks or Enter to the netu menu by choosing exit.

Delete Old and Add New Definition

Use this procedure when there are two or more remote node definitions (at the same node) that are associated with the vnode name belonging to the definition that you want to change.

When the merge option has been used to set up several node definitions that use the same vnode name, you must be very careful when changing one of these definitions. You *cannot* overwrite the incorrect definition using the add option as in Overwrite an Existing Definition (see page 406). If you try to do this, netu overwrites all of the definitions that have the specified vnode name, effectively deleting them all and leaving you with only the definition you have just added.

To safely change a node definition that has a vnode name in common with other node definitions, you must delete the old definition and use merge to add the new definition.

To delete an old definition and add a new one

1. Start netu by entering netu at the operating system prompt.

The main menu appears.

2. Select N (Modify Node Entry) from the menu.

The following prompt appears:

Enter operation (add, merge, del, show, exit):

3. Enter del. (Enter d instead of del.)
4. Answer the prompts that appear with the values from the definition you want to change.

When all the prompts are answered, netu deletes the node definition that has the values that you have just supplied and Enters you to the prompt:

Enter operation (add, merge, del, show, exit):

5. Enter merge. (Enter m instead of merge.)
6. Answer the prompts that appear with the values for the correct definition.

The utility adds a node definition with the values that you have just supplied and displays the following prompt:

Enter operation (add, merge, del, show, exit):

7. Continue with node definition tasks or Enter to the main netu menu by choosing exit.

Retrieve Remote Node Definition Information

To see the remote node definitions associated with a particular local node, the following procedure asks for information about the definitions and displays all the definitions that match the information you provide.

Use an asterisk in response to any prompt other than the one asking if the definition is private or global. An asterisk is the wild card character that matches any value.

To retrieve remote node definition information

1. Start netu by entering netu at the operating system prompt.
The netu menu appears.
2. Select N (Modify Node Entry) from the menu.
This prompt appears:
Enter operation (add, merge, del, show, exit):
3. Enter show. (Type s instead of show.)
4. When netu asks if the entry you want to see is a private or global entry, do one of the following:
 - If the entry is private, press Enter.
 - If the entry is global, enter G.
5. Enter the vnode name of the remote node definition.
6. Enter the network software type.
7. Enter the remote node's address.
8. Enter the listen address of the remote node's Communications Server.

The utility displays all of the definitions that match the information that you gave. If you used an asterisk as a wild card for any of the prompts, you receive more than one definition. The definitions appear in table format. You *cannot* change any node name definition while it is displayed in this format.

After the definition or list of definitions is displayed, the utility displays the following prompt:

Enter operation (add, merge, del, show, exit):

9. Continue with node definition tasks or choose exit to Enter to the netu menu.

Displayed Node Definition Examples

Here are some examples of the format and type of information that you receive when you retrieve node definition information for viewing.

Windows: The following table shows Windows Displayed Node Definition examples:

Global:	V_Node	Net Software	Node Address	Listen Address
	london	tcp_ip	uk1	II0
	rome	tcp_ip	italy2	II0
	n_york	tcp_ip	usa1	II0
	chicago	lanman	usa2	USA2_II

UNIX: The following table shows UNIX Displayed Node Definition examples:

Global:	V_Node	Net Software	Node Address	Listen Address
	london	tcp_ip	uk1	II0
	rome	tcp_ip	italy2	II0
	n_york	tcp_ip	usa1	II0

VMS: The following table shows VMS Displayed Node Definition examples:

Global:	V_Node	Net Software	Node Address	Listen Address
	n_york	sna_lu0	GW1	NYMVSPLU
	chicago	sna_lu0	GW2	CHMVSPLU
	london	decnet	uk1	II_GCC_0
	rome	decnet	rome	II_GCC_0
	s_fran	decnet	s_fran	II_GCC_0
	n_york	tcp_wol	usa1	II
	chicago	tcp_wol	usa2	KK

Remote User Authorization Operations

Remote user authorizations, along with node definitions, make it possible to use Ingres Net to access databases on remote nodes. A remote user authorization associates a specified vnode name with a specified account on the remote node. When the user requests a connection using that vnode name, Ingres Net makes the connection to the DBMS Server on the remote node through that account.

Three operations concern remote user authorizations:

- Adding new remote user authorizations
- Deleting existing authorizations
- Viewing existing authorizations

In addition, change an existing authorization by overwriting the authorization or by deleting it and adding a new authorization.

To perform any of these operations, you must have the following information about the authorization:

- The type of the authorization, private or global
- The vnode name of the remote node
- The name of the account on the remote node
- The password for the account on the remote node

Define Remote User Authorizations

A remote user authorization associates a specified vnode name with a specified account on the node represented by that vnode name.

To define a remote user authorization

Note: You can exit this procedure at any time without making an entry by pressing Esc and Enter.

1. Start netu by entering netu at the operating system prompt.
The netu menu appears.
2. Select A (Modify Remote User Authorization Entry).
The following prompt appears:
Enter operation (add, del, show, exit):
3. Enter add. (Type a instead of add.)
Netu asks if you want a private or global authorization.
4. Do one of the following:
 - To accept the default (private), press Enter.
 - To select global, enter G.
5. Enter the vnode name of the remote node.
6. Enter the name of the account at the remote node.
7. Enter the password for the remote account.
The default is an asterisk (*). Use this *only* if the remote account has no password.
8. Enter the password for the remote account again.
The utility finishes defining a remote user authorization and displays the following prompt:
Enter operation (add, del, show, exit):
9. Continue with remote user authorization tasks or choose exit to Enter to the netu menu.

Delete Remote User Authorizations

The netu utility lets you delete one or several authorizations at a time. When you select the delete operation, netu asks for the values that comprise the authorization and deletes any and all authorizations that match those values. To delete a single authorization, all the values must match. If the match is not exact, the authorization is not deleted. To delete multiple authorizations, use the wild card character (*), which matches any value.

If you are unsure of the values for the authorization that you want to delete, use the "show" operation to check the values before you delete them. For instructions on using the show operation, see Retrieve Remote User Authorizations (see page 416).

To delete more than one authorization in one operation, use an asterisk in response to the prompts asking for the remote user name. The asterisk is a wild card character that matches any value. For example, assume that you want to delete all of your private user authorizations from "napoleon" that are associated with the account having the userid "tommy" on "josephine." To do this, run netu from "napoleon," selecting A and the del operation. Respond to the prompts in this manner:

```
Enter Private or Global (P): <Enter>
Enter the remote vnode name: *
Enter the remote user name: tommy
```

When you have completed the procedure, you deleted all of the private user authorizations for the account "tommy." If you have defined other remote user authorizations to "josephine" through a different account, these remain undeleted.

Only a user with Ingres privileges can delete global authorizations.

To delete remote user authorizations

Note: You can exit the procedure at any point by pressing Esc and Enter.

1. Start netu by entering netu at the operating system prompt.
The netu menu appears.
2. Select A (Modify Remote User Authorization Entry) from the menu.
The following prompt appears:
Enter operation (add, del, show, exit):
3. Enter del. (Enter d instead of del.)
Netu asks if the authorization is a private or global authorization.

4. Do one of the following:
 - If the authorization is private, press Enter.
 - If the authorization is global, enter G.
5. Enter the remote vnode name.
6. Enter the remote user name.

The utility displays the number of authorizations that it deleted and the following prompt:

Enter operation (add, del, show, exit):

7. Select del again to delete another authorization, or choose a different operation.

Change Remote User Authorizations

If necessary, change an existing remote user authorization entry by using one of two methods:

- Overwrite the existing, incorrect entry.

Use this method if the vnode names for both the old incorrect entry and the new correct entry are the same. For instructions on performing this method, see [Overwrite an Incorrect Entry](#) (see page 414).
- Delete the existing, incorrect entry and add the new, correct entry.

Use this method if you must correct a vnode name. For instructions on performing this method, see [Delete Old and Add New Definition](#) (see page 407).

Overwrite an Incorrect Entry

Use this procedure to modify a remote user authorization when some part of the authorization information, other than the vnode name, has changed. For example, perhaps the passwords to accounts are changed on a regular basis. When this happens, the remote user authorizations must be modified to allow users continued access to remote accounts.

When you use this procedure, netu overwrites the existing authorization whose vnode name matches the vnode name that you specify. Use the values of the new, correct authorization to respond to the prompts.

To modify a remote user authorization

1. Start netu by entering netu at the operating system prompt.

The netu menu appears.

2. Select A (Modify Remote User Authorization Entry).

The following prompt appears:

Enter command (add, del, show, exit):

3. Enter add. (Type a instead of add.)

Netu asks if the authorization is private or global.

4. Do one of the following:

- If the authorization is private, press Enter.
- If the authorization is global, enter G.

5. Enter the remote vnode name.

6. Enter the remote user name.

7. Enter the password.

8. Enter the password a second time.

Netu replaces the existing authorization with the new one and displays the following prompt:

Enter operation (add, del, show, exit):

9. Continue with other user authorization operations or Enter to the netu menu by selecting exit.

Delete and Add an Entry

To change the vnode name associated with a remote user authorization

1. Start netu by entering netu at the operating system prompt.
The netu menu appears.
2. Select A (Modify Remote User Authorization Entry).
The following prompt appears:
Enter command (add, del, show, exit):
3. Enter del. (Enter d instead of del.)
4. Answer the prompts that appear using the values of the incorrect authorization.

When you have answered all the prompts, netu deletes the incorrect authorization and displays the following prompt:
Enter command (add, del, show, exit):
5. Enter add. (Enter a instead of add.)
6. Answer the prompts that appear using the values of the new, correct authorization.

The utility adds the new authorization and displays the following prompt:
Enter command (add, del, show, exit):
7. Choose another operation or Enter to the netu menu by choosing exit.

Retrieve Remote User Authorizations

When you want to see a list of authorizations for a single node or several nodes, use the procedure in this section. It produces a read-only display of authorization information. For example, use this procedure to produce a list of all your private authorizations to a single node or to all nodes. You can also use this procedure to find out if there are any global authorizations to a particular node.

For example, to see any global authorizations, you run the procedure and make the following responses to the following prompts:

```
Enter Private or Global (P): Global
Enter the remote vnode name: *
Enter the remote user name: *
```

The utility displays all of the global authorizations that have been defined from the node on which you are working. The display is in a table format. For display examples, see Displayed Remote User Authorization Examples (see page 417).

If you select Private, the utility displays the appropriate private authorizations that belong to you.

To display remote user authorizations

Note: You can exit the procedure at any point by pressing Esc and Enter.

1. Enter **netu** at the operating system prompt.

The main menu appears.

2. Select A.

The following prompt appears:

```
Enter operation (add, del, show, exit):
```

3. Enter show. (Enter s instead of show.)

4. The utility asks if you want a list of private or global authorizations.

You must choose one or the other. You cannot enter the wildcard character for this prompt.

- If you want a list of private authorizations, press Enter.
- If you want a list of global authorizations, enter G.

5. Enter the remote vnode name.

6. Enter the remote user name.

The netu utility shows you the authorizations that match the responses you provided. The display appears in table format. You cannot change any of the information while it is in this format.

After the authorizations are displayed, netu Enters you automatically to the prompt:

Enter command (add, del, show, exit):

7. Choose another authorization operation or Enter to the netu menu by choosing exit.

Displayed Remote User Authorization Examples

Here is an example of the information that you receive when you view remote user authorizations:

Private:	V_Node	User Name
^	london	janetd
^	rome	janetd
^	n_york	janetd
^	s_fran	janetd

Netu Options for Stopping the Communications Server

Netu provides two options for stopping the Communications Server:

quiesce

Stops the server after any open sessions with the server have terminated. Use the quiesce option to stop the server gracefully, waiting until open sessions terminate.

stop

Stops the server immediately, regardless of whether there are any open sessions using the server.

Both options require you to know the GCF (General Communication Facility) address of the Communications Server.

Obtain GCF Address

The GCF address is the Ingres-specific symbolic address that the Communications Server uses to communicate with local Ingres processes. This address is also called the GCA address.

You must know the Communications Server's GCF address before you can stop the server. Use the `iinamu` utility to obtain this address.

To obtain the GCF address

1. Enter the following command at the operating system prompt:

```
iinamu
```

The `iinamu` menu appears.

2. Enter this command:

```
show comsvr
```

The utility displays a list of Communications Servers running in the installation, in the format:

```
COMSVR * GCF_ADDRESS
```

The GCF address is the value in the third column.

3. Note the GCF address shown in the display, and then enter quit.
You exit the utility.

Stop Communications Server

Before you begin this procedure, you must have the GCF address of the Communications Server. To obtain this address, see Obtain GCF Address (see page 418).

To stop the Communications Server using netu

1. Enter this command at the operating system prompt:

```
netu
```

The netu menu appears.

2. Enter one of the following:

- *Q Comm_server_id*

where *Comm_server_id* is the GCF address of the Communications Server.

The server stops after all open sessions are closed. In most cases, use this option.

- *S Comm_server_id*

where *Comm_server_id* is the GCF address of the Communications Server.

The Communications Server is stopped immediately, terminating any open sessions.

The netu menu automatically reappears.

3. To exit netu, select E.

Note: If you enter the *Comm_server_id* on the command line when you start netu (for example, **\$ netu *Comm_server_id***), you do not have to enter the *Comm_server_id* when you select Q or S. By default, netu stops the Communications Server associated with the ID that you specified on the command line.

Appendix K: IPv6 Configuration

This section contains the following topics:

[IPv6 Configuration Overview](#) (see page 421)

[TCP/IP and Ingres Communications](#) (see page 422)

[Parameters for Controlling IPv6 Support](#) (see page 422)

[Options for Disabling IPv6 Support](#) (see page 425)

[IPv6 in the JDBC Driver and Ingres .NET Data Provider](#) (see page 430)

[Examples of Disabling IPv6 Support](#) (see page 431)

This appendix describes the parameters that control IPv6 support in Ingres and how to disable IPv6 support, if necessary.

IPv6 Configuration Overview

In most cases, no configuration in Ingres is required to implement IPv6 support. It is automatically enabled in the tcp_ip (tcp_dec on VMS) network protocol starting with Ingres 9.1.

The tcp_ip protocol driver also continues to support IPv4 addresses without any configuration changes needed; the driver will detect and process IPv4 addresses, IPv6 addresses, and both, if present concurrently. For example, on Windows, it is not necessary to use the older wintcp protocol on Windows systems that do not have the IPv6 protocol enabled.

Some versions of operating systems, however, provide no or limited support for IPv6. In some cases, IPv6 support is available, but must be enabled or configured in the operating system before it can be used. On systems with partial or no IPv6 functionality, Ingres will adjust automatically to the level of IPv6 support available. On some systems, however, Ingres may have difficulty, particularly when starting up or establishing connections.

For those rare situations, configuration parameters can be used to "back out" the IPv6 support, if required, or to restrict TCP support to IPv6 addresses only. These parameters (see page 422), which are not available in the configuration utilities, must be set with set, ingsetenv, or iisetres commands.

On Windows, the iicvtwintcp utility (see page 428) can be used to convert wintcp settings to tcp_ip settings, or to subsequently restore the settings to their previous values.

TCP/IP and Ingres Communications

On all platforms, Ingres Net typically uses TCP/IP to communicate between Ingres installations. A typical scenario is where applications in the client installation communicate through Ingres Net with the DBMS Server in the server installation. If using JDBC or .NET applications with Ingres, then TCP/IP is used to communicate between the Ingres JDBC or .NET driver running under the application and the Data Access Server (process iigcd).

On Linux and UNIX only, TCP/IP is typically used to communicate between Ingres processes—that is, as the local IPC. The tcp_ip protocol driver used for network communications is used for local communications also. Therefore, if you experience trouble with IPv6 across the network, local communications are likely to have trouble too. If so, basic Ingres server processes such as the Name Server (iigcn) and the DBMS Server (iidsbms) may not even start.

Parameters for Controlling IPv6 Support

The parameters for controlling IPv6 support are as follows:

- II_TCPIP_VERSION environment variable
- II_GC_PROT environment variable
- ii.hostname.gcX.*.protocol.status (and port) in config.dat, where the gcX server can be gcc, gcd, or jdbc.

VMS: The only configuration options are II_TCPIP_VERSION and ii.hostname.gcX.*.tcp_ip.version.

II_TCPIP_VERSION Environment Variable—Specify Version of TCP/IP to Use

The II_TCPIP_VERSION environment variable determines the version of IP addresses that the tcp_ip (or tcp_dec on VMS) protocol driver uses. It can be set using the ingsetenv command.

Note: The equivalent configuration parameter in config.dat is tcp_ip.version.

This variable has the following format:

II_TCPIP_VERSION = *value*

value

Controls which tcp_ip protocol driver or which version of IP addresses to use, as one of the following:

ALL

(Default) Uses both IPv4 and IPv6 addresses.

VMS: The default behavior is to use both IPv4 and IPv6 addresses, and to map the IPv4 addresses as IPv6.

6

Uses IPv6 addresses only.

VMS: The IPv6 versions of the listen, accept, and connect are used.

4

Windows: Uses only IPv4 addresses with IPv6-capable functions.

Linux, UNIX, VMS: Uses the IPv4-only version of the protocol driver.

46

Linux and UNIX: Uses only IPv4 addresses with IPv6-capable functions.

II_GC_PROT Environment Variable—Set IPC Protocol

The II_GC_PROT environment variable sets the local interprocess communications (IPC) protocol. You can set this variable using the `ingsetenv` command.

This variable has the following format:

`II_GC_PROT = protocol`

protocol

Specifies the local IPC protocol as one of the following:

TCP_IP

(Default for Linux and UNIX) Uses the current TCP_IP protocol driver.

Windows: TCP_IP is the only valid value. The default IPC protocol is named pipes. 🚫

TCP_IPV4

(Linux and UNIX only) Uses the previous version of the TCP_IP protocol driver, which supports IPv4 addresses only.

ii.hostname.gcX.*.protocol.status Resource—Set Network Communications Protocol

This resource in config.dat sets the network communications protocol for the designated server (gcX) to the appropriate Ingres network protocol driver. The gcX server can be gcc, gcd, or jdbc. The resource can be set by using the `iiisetres` command.

The format is as follows:

```
ii.hostname.gcX.*.protocol.status
```

and

```
ii.hostname.gcX.*.protocol.port
```

protocol

Specifies the Ingres network protocol driver, which can be one of the following:

tcp_ip

(All environments except VMS) (Default) Uses the current TCP_IP protocol driver.

tcp_ipv4

(Linux and UNIX only) Uses the previous version of the TCP_IP protocol driver, which supports IPv4 addresses only.

wintcp

(Windows only) Uses the previous version of the TCP_IP protocol driver, which supports IPv4 addresses only.

tcp_dec

(VMS only) Uses the current TCP_IP protocol driver.

Options for Disabling IPv6 Support

Two approaches can be used to disable IPv6 support:

- Restrict the `tcp_ip` protocol driver to only use IPv4 addresses
- Completely back out the enhanced `tcp_ip` protocol driver and use the old version of the driver.

Use IPv4 Addresses Only

The `tcp_ip` protocol driver can be restricted to listen and connect only with IPv4 style addresses. (IPv4 is the standard IP version that was used prior to IPv6).

To restrict the `tcp_ip` protocol driver to use IPv4 addresses only:

Use any one of the following options, which are functionally equivalent:

- Set the `II_TCP_IP_VERSION` operating system variable as follows:

Windows:

```
set II_TCPIP_VERSION=4
```

Linux and UNIX:

```
set II_TCPIP_VERSION=46
```

- Set the Ingres `II_TCPIP_VERSION` environment variable as follows:

Windows:

```
ingsetenv II_TCPIP_VERSION 4
```

Linux and UNIX:

```
ingsetenv II_TCPIP_VERSION 46
```

- Set the Ingres resource as follows:

If using Ingres Net:

```
iisetres ii.machine.gcc.*.tcp_ip.version 4
```

If using Data Access Server:

```
iisetres ii.machine.gcd.*.tcp_ip.version 4
```

- (VMS only) (Optional) Set the "lnm" (logical name) attribute to cause `II_TCPIP_VERSION` to be defined when the servers start up:

```
iisetres ii.machine.lnm.ii_tcpip_version 4
```

Back Out IPv6 Support

To completely back out IPv6 support, you must use the old driver, which supports IPv4 only. The old driver is renamed to `tcp_ipv4` on UNIX and Linux, and is `wintcp` on Windows. The old driver is expected to be made obsolete in a future release of Ingres.

To back out the enhanced `tcp_ip` Ingres protocol driver

1. Back out network protocol for servers with `tcp_ip.status = ON`

Linux, UNIX, Windows:

```
iisetres ii.machine.gcc.*.tcp_ip.status OFF      (All platforms)
iisetres ii.machine.gcc.*.tcp_ipv4.status ON     (Unix/Linux)
iisetres ii.machine.gcc.*.tcp_ipv4.port II      (Unix/Linux)
iisetres ii.machine.gcc.*.wintcp.status ON      (Windows)
iisetres ii.machine.gcd.*.tcp_ip.status OFF      (All platforms)
iisetres ii.machine.gcd.*.tcp_ipv4.status ON     (Unix/Linux)
iisetres ii.machine.gcd.*.tcp_ipv4.port II7     (Unix/Linux)
iisetres ii.machine.gcd.*.wintcp.status ON      (Windows)
```

Set the port value to same as that used by the same server for `tcp_ip`.

VMS:

```
define/group II_TCPIP_VERSION 4
or
define/system II_TCPIP_VERSION 4
```

2. Back out local IPC protocol (Linux and UNIX only):
3. Back out network and local IPC protocol (Linux and UNIX only):

```
ingsetenv II_GC_PROT tcp_ipv4
```

```
set II_TCPIP_VERSION=4
```

or

```
ingsetenv II_TCPIP_VERSION 4
```

Note: This step is equivalent to steps 1 and 2 above, and is the simplest way to back out to the IPv4-only driver on Linux and UNIX.

iicvtwintcp Command—Convert wintcp to tcp_ip Protocol Setting

The iicvtwintcp command converts vnode definitions and GCx server protocol settings in config.dat from the deprecated wintcp network protocol to the newer tcp_ip protocol. This utility runs automatically during an upgrade, but can also be run standalone.

A backup file is created (or appended to) that can be used in subsequent iicvtwintcp commands to restore converted protocol entries to their previous settings.

Note: Dynamic vnodes are not affected by this utility.

The iicvtwintcp command has the following format:

```
iicvtwintcp [-help] [-verbose] [-noupdate] [-force]
  [-action convert_wintcp | convert_tcp_ip | restore]
  [-bf filename] [-nf filename]
  [-scope all | vnodes | config]
```

-help

Displays command usage information.

-verbose

Displays each record affected. If not specified, only totals are displayed.

-noupdate

Reports what the impact of running the utility will be, but does not update Name Server or backup files.

If the Name Server is running, the **-force** parameter must also be used.

When used with the **-verbose** parameter, produces a report from which the vnode updates can be done manually from one of the Ingres network configuration utilities.

-force

Runs iicvtwintcp even if the Name Server or another instance of the program is currently running. A warning message is issued. This parameter is useful with the **-noupdate** parameter or when running against a copy of the Name Server node file.

Note: Use this parameter with caution.

-action

Specifies the action to be performed:

convert_wintcp

Converts wintcp to tcp_ip.

convert_tcp_ip

Converts tcp_ip to wintcp.

restore

Restores converted protocol entries back to prior values.

Note: If no action is specified, displays usage, which prevents unintentional conversion if command with no parameters is entered.

-bf

Specifies the name of the restore (back off) file. If not specified, the default is:

%II_SYSTEM%\ingres\files\name\IINODE_ *hostname*.BK

Can be specified with or without path name. If no path is specified, defaults to %II_SYSTEM%\ingres\files\name.

Note: Written to as output if -action is convert_wintcp or convert_tcp_ip. Read from as input if -action is restore, and is required for a successful restore operation.

-nf

Specifies the name of the Name Server connection (NODE) file that contains the vnode definitions that will be updated. If not specified, the default is:

%II_SYSTEM%\ingres\files\name\IINODE_ *hostname*

Can be specified with or without path name. If no path is specified, defaults to %II_SYSTEM%\ingres\files\name.

-scope

Converts or restores the following entities:

all

(Default) Vnode definitions and config.dat file

vnodes

Vnode definitions only

config

Config.dat file only

Return codes:**>=0**

Indicates the program succeeded, where return code is the number of records modified.

-1

Indicates the program failed.

iicvtwintcp Examples

In these examples of the iicvtwintcp command, the command should be run while the Name Server (iigcn) is stopped.

1. Convert wintcp to tcp_ip in current Ingres installation:

```
iicvtwintcp -action convert_wintcp
```

2. Restore records converted to tcp_ip back to wintcp:

```
iicvtwintcp -action restore
```

3. Convert tcp_ip to wintcp in current Ingres installation.

This example is similar to Example 2 except that all tcp_ip records will be set to wintcp, not just the ones that were originally converted by **-action convert_wintcp**. Such a command is likely to be used if a restore is wanted but the backup file is lost.

```
iicvtwintcp -action convert_tcp_ip
```

4. List connection (NODE) information for all vnodes:

```
iicvtwintcp -action convert_wintcp -noupdate -verbose
```

IPv6 in the JDBC Driver and Ingres .NET Data Provider

No Ingres parameters control or restrict IPv6 in the JDBC driver or the .NET Data Provider. The DAS (server side of the connection) can be configured as documented in Options for Disabling IPv6 Support (see page 425). However, there are some Java-specific networking system properties that can be set to control the IPv6 behavior in the Ingres JDBC driver (and other Java applications).

To return IPv6 addresses before IPv4 addresses:

```
java.net.preferIPv6Addresses=true
```

To restrict driver to IPv4 only:

```
java.net.preferIPv4Stack=true
```

Examples of Disabling IPv6 Support

In the following examples, assume: machine=host1, Ingres installation id=AA, startup count is 1 for Ingres Net and Data Access Server.

1. Revert to the old IPv4-only driver

Linux and UNIX:

```
ingsetenv II_GC_PROT tcp_ipv4
iisetres ii.host1.gcc.*.tcp_ip.status OFF
iisetres ii.host1.gcc.*.tcp_ipv4.status ON
iisetres ii.host1.gcc.*.tcp_ipv4.port AA
iisetres ii.host1.gcd.*.tcp_ip.status OFF
iisetres ii.host1.gcd.*.tcp_ipv4.status ON
iisetres ii.host1.gcd.*.tcp_ipv4.port AA7
```

Windows:

```
iisetres ii.host1.gcc.*.tcp_ip.status OFF
iisetres ii.host1.gcc.*.wintcp.status ON
iisetres ii.host1.gcd.*.tcp_ip.status OFF
iisetres ii.host1.gcd.*.wintcp.status ON
```

Note: wintcp port is typically already set correctly on Windows.

VMS:

```
define/group II_TCPIP_VERSION 4
```

or

```
define/sys II_TCPIP_VERSION 4
```

Optionally, set the "lnm" (logical name) attribute to cause II_TCPIP_VERSION to be defined when the servers start up:

```
iisetres ii.machine.lnm.ii_tcpip_version 4
```

2. Restrict all remote communications to IPv4 addresses only on Windows, Unix or Linux:

```
ingsetenv II_TCPIP_VERSION 4
```


Index

.

- .NET data provider
 - classes • 268
 - data types • 334
 - integration with Visual Studio • 344
 - sample program • 273
 - troubleshooting applications • 354
- .NET System.Data.DbType
 - data types • 334
 - mapping to EdbcType • 336
 - mapping to IngresType • 336

A

- accessing remote installations • 50
- AIX SNA Services/6000
 - installation requirements • 41
 - sample configuration file • 389
- architecture
 - .Net Data Provider • 264
 - Ingres Net • 28
- attributes, configuring vnode • 64, 92
- authentication_mechanism (vnode attribute) • 64, 92
- authorization
 - distributed databases • 51
 - of users • 49
 - remote user • 30, 50
- autocommit
 - alternative processing modes • 240
 - transactions • 240

B

- BLOB columns • 249
- Bridge Server • 131
 - described • 22
 - monitoring • 137
 - starting options • 135
 - starting using command line • 136
 - stopping with ingstop • 138
 - stopping with IVM • 138
- Bridge, starting • 134
- BYREF parameters • 248

C

- cbf (utility)
 - configuration parameters • 47, 119
- classes
 - IngresCommand • 269
 - IngresConnection • 277
 - IngresDataAdapter • 310
 - IngresDataReader • 303
 - IngresError • 313
 - IngresErrorCollection • 314
 - IngresException • 316
 - IngresInfoMessageEventArgs • 318
 - IngresInfoMessageEventHandler • 319
 - IngresParameter • 321
 - IngresParameterCollection • 327
 - IngresRowUpdatedEventArgs • 329
 - IngresRowUpdatedEventHandler • 330
 - IngresRowUpdatingEventArgs • 331
 - IngresRowUpdatingEventHandler • 332
 - IngresTransaction • 332
- client installation
 - NFS • 42
 - setup parameters • 42
- cluster
 - and Ingres Net • 21, 24
 - Name Server files • 120
 - running netutil • 51
- commands
 - used with Ingres Net • 102
- Communications Server
 - described • 22, 23
 - inbound/outbound session limits • 110
 - listen address • 127
 - starting • 105
 - stopping • 106, 400, 417, 419
- Configuration Manager • 47
- configuration parameters • 47
 - Data Access Server • 144
 - default values • 119
 - default_server_class • 100
 - described • 47
 - inbound_limit • 110
 - named and unnamed • 247
 - outbound_limit • 110

- remote_vnode • 114
- configuring a data source
 - Windows • 153
- connect statement • 101
- connecting to a data source • 156
- connection data • 57, 59
 - described • 29
 - global and private • 31
 - listen address • 29
 - table • 57, 59
- connection data entries
 - creating • 67, 91
 - deleting • 69, 70
 - editing • 72, 73
 - valid protocol keywords • 58
- connection errors
 - local • 127
 - remote • 128
- connection pooling
 - .NET • 264, 267, 348
 - Data Access Server • 144
 - ODBC • 148, 156, 212
- connection string keywords • 156, 281
- connection_type (vnode attribute) • 64, 92
- connectivity problems • 122
- constructors
 - IngresConnection class • 272, 280, 312, 326
- conventions
 - syntax • 19
- create user statement • 49, 103
- cursors
 - and result set characteristics • 242
 - and select loops • 244
 - scrollable • 190, 242

D

- Data Access Server (DAS)
 - configuring • 144
 - described • 22
 - parameters • 144
 - starting • 115
 - stopping • 115
 - tracing support • 146
- data provider namespaces • 265
- data provider user ID options • 285
- data retrieval
 - strategies • 266
 - using DataAdapter • 266

- using DataReader • 266
- data types
 - .NET data provider • 334
 - .NET System.Data.DbType • 334
- database
 - access syntax • 98
 - procedures • 246
- DataSource tracing • 259
- DataTable
 - GetSchemaTable • 306, 340
- date/time columns • 253
- DECnet
 - installation requirements, VMS • 40
 - listen address • 375
- default_server_class (configuration parameter)
 - 100, 119
- defined • 14
- direct connect feature • 64, 92
- Driver Manager • 150, 259

E

- encryption • 64, 92
- encryption_mechanism (vnode attribute) • 64, 92
- encryption_mode (vnode attribute) • 64, 92
- Enterprise Access products
 - with Ingres Net • 25
 - with Ingres Star • 26
- errlog.log • 111, 122
- errors
 - connection • 127
 - diagnose • 111, 122
 - local connection • 127
 - Net registration • 129
 - remote connection • 128
 - security and permission • 129
- events
 - IngresConnection class • 280
 - IngresDataAdapter class • 312

G

- GCF (General Communication Facility)
 - Bridge Server • 22
 - Communications Server • 22
 - Data Access Server • 22
 - described • 22
 - General Communications Area (GCA) • 22
 - Name Server • 22

- servers, dynamic tracing • 113, 146
- GetSchemaTable
 - DataTable • 306, 340
- global registration types • 31
- Global Temporary Table parameters (JDBC) • 248

H

- HP-UX SNAplus
 - installation requirements • 41
 - sample configuration file • 395

I

- II_GCA_LOG • 112
- iicvtwintcp command • 428, 430
- iigcb
 - described • 22
 - starting • 135
- iigcc
 - described • 22, 23
 - inbound/outbound session limits • 110
 - listen address • 127
 - starting • 105
 - stopping • 106, 400
- iigcd
 - described • 22
 - starting • 115
 - stopping • 115
- iigcn
 - database files • 120
 - described • 22
 - Name Server database • 120
 - starting • 135
- iinamu (utility) • 127
- inbound/outbound sessions
 - changing limits • 110
- inbound_limit (configuration parameter) • 110, 119
- Ingres
 - basic components • 16
 - installation, described • 17
 - server classes • 100
 - tools • 16
- Ingres Bridge
 - Bridge Server component • 131
 - diagnosing problems • 139
 - installation configurations • 132
 - installing • 134

- overview • 131
- sample configuration • 133
- sample configuration setup files • 140
- security • 131
- setting up client • 136
- tracing a connection • 139
- vs. Ingres Star • 132

- Ingres Net
 - accessing remote databases • 97
 - architecture • 28, 34
 - benefits • 27
 - commands available • 102
 - configuration parameters • 47
 - described • 21
 - diagnosing problems • 122
 - Enterprise Access products • 25
 - establishing connections • 117
 - Ingres Star and • 25
 - NET_ADMIN privilege • 34
 - permission errors • 130
 - resolving connection errors • 127
 - sample configuration • 24
 - security • 23
 - security errors • 130
 - SERVER_CONTROL privilege • 34
 - setup parameters • 41
 - user roles • 33
 - using the connect statement • 101
- Ingres Star
 - with Ingres Enterprise Access products • 26
 - with Ingres Net • 25
- IngresCommand class • 269
- IngresConnection class • 277
- IngresDataAdapter class • 310
- IngresDataReader class • 303
- IngresError class • 313
- IngresErrorCollection class • 314
- IngresException class • 316
- IngresInfoMessageEventArgs class • 318
- IngresInfoMessageEventHandler class • 319
- ingstart -iigcb • 136
- ingstart -iigcc • 105
- ingstart -iigcd • 115
- ingstart -iigcn • 135
- ingstop -iigcc • 106, 400
- ingstop -iigcd • 115
- ingvalidpw (executable) • 46, 130
- installation
 - client • 17

- server • 17
- types of • 25
- installation code
 - format • 42
- Installation Password
 - defining • 93
 - defining during installation • 42
 - defining on local installation • 74, 82
 - in contrast to login account password • 30
- installing Ingres Bridge • 134
- installing Ingres Net
 - for existing installations • 43
 - installation components • 37
 - setup parameters • 41
- IPv6 • 358
- ISO • 23

J

- JDBC
 - and Data Access Server • 143
 - connectivity components • 223
 - implementation considerations • 239
 - JDBC Information Utility • 225
 - tracing • 259
 - unsupported features • 227
- JDBC driver
 - accessing • 238
 - and cursor pre-fetch capabilities • 244
 - and updateable cursors • 242
 - BLOB columns • 249
 - class files • 228
 - cursors and select loops • 244
 - data source properties • 234
 - database procedures • 246
 - date/time columns • 253
 - driver properties and attributes • 229
 - National Character Set values • 255
 - properties generator (iijdbcprop) • 229, 233
 - support for parameter default values • 247
 - supported features • 223

L

- LAN Manager
 - listen address • 377
- listen address
 - configuration parameter • 119
- DECnet • 375
- defined • 29

- LAN Manager • 377
- SNA LU0 • 361
- SNA LU62 • 363
- SPX/IPX • 371
- TCP/IP • 357
- LOB locator • 249
- local_vnode (configuration parameter) • 119
- log_level (configuration parameter) • 119
- logging
 - directing to a file • 112
 - levels • 111
- Login/password data table
 - described • 55
 - Installation Password • 56
 - login account password • 56

M

- mapping
 - .NET System.Data.DbType to IngresType • 336
- methods
 - IngresCommand class • 271
 - IngresConnection class • 279
 - IngresDataAdapter class • 311
 - IngresDataReader class • 305
 - IngresError class • 314
 - IngresErrorCollection class • 315
 - IngresException class • 317
 - IngresParameter class • 326
 - IngresParameterCollection class • 328
 - IngresTransaction class • 334
- mkvalidpw • 46, 130
- MultiNet TCP/IP
 - installation requirements, VMS • 40

N

- Name Server
 - database files • 120
 - described • 22
 - IICOMSVR_nodename • 120
 - IIINGRES_nodename • 120
 - IILOGIN_nodename • 120
 - iiname.all • 120
 - IINODE_nodename • 120
 - IISTAR_nodename • 120
- namespace
 - data provider • 265
- National Character Set values • 255

NET_ADMIN (privilege) • 34

netu (utility)

- aborting • 402

- adding remote node definitions • 403

- adding vs. merging remote node definitions • 403

- changing remote node definitions • 406

- changing remote user authorizations • 413

- defining remote user authorizations • 411

- deleting remote node definitions • 405

- deleting remote user authorizations • 412

- described • 53

- displayed remote node definitions • 409

- displayed remote user authorizations • 417

- merging remote node definitions • 403

- retrieving remote node definitions • 408

- retrieving remote user authorizations • 416

- stopping Communications Server (iigcc) • 417

- user interface • 400

netutil (non-interactive mode)

- command line flags • 76

- creating a connection data entry • 83

- creating a remote user authorization • 78

- defining local Installation Password • 82

- deleting a connection data entry • 84

- deleting a remote user authorization • 80

- displaying connection data entries • 86

- displaying remote user authorizations • 81

- functions available • 75

- input control file • 76

- invariant fields • 77

- quiescing the Communications Server • 88

- stopping the Communications Server • 88

- wildcards • 78

netutil (utility)

- accessing • 51

- adding vnode attributes • 64

- clusters and • 51

- Connection data table • 57, 59

- creating a connection data entry • 53, 67, 136

- creating a remote user authorization • 53, 68

- defining a local Installation Password • 74

- deleting entries • 68

- editing entries • 70

- establishing and testing a connection • 62

- global and private registration types • 31

- list of user tasks • 60

- Login/password data table • 55

- non-interactive mode • 75

- startup screen • 53

- stopping the Communications Server • 108

- tables • 53

- virtual node name (vnode) table • 54

network

- installing and testing • 38

- protocol types • 58

- terms and concepts • 14

network address • 57, 59

network protocol • 58

Network Utility

- accessing • 51

- adding vnode attributes • 92

- altering vnodes • 89

- creating a private remote user authorization • 92

- creating vnodes • 89

- deleting vnodes • 89

- list of user tasks • 89

O

ODBC

- CLI implementation considerations • 159

- connection pooling • 212

- Data Source Administrator • 153

- Driver Manager • 150

ODBC driver

- configuring a data source (Windows) • 153

- described • 147

- read-only option • 149

- requirements • 149

Open Systems Interconnect • 23

OSI • 28, 34

outbound_limit (configuration parameter) • 110, 119

P

parameters

- BYREF • 248

- Global Temporary Table (JDBC) • 248

passwords

- defining a local Installation Password • 74, 82

private registration types • 31

privileges, user • 103

procedures, executing • 248

properties

 IngresCommand class • 270

 IngresConnection class • 278

 IngresDataAdapter class • 310

 IngresDataReader class • 304

 IngresError class • 314

 IngresErrorCollection class • 315

 IngresException class • 316, 319

 IngresParameter class • 325

 IngresParameterCollection class • 328, 330, 331

 IngresTransaction class • 333

protocol

 configuration parameter • 119

 keywords • 58

Q

query builder • 350

R

RDBMS (Relational Database Management System) • 16

read-only ODBC driver • 149

region and time zone, defined • 42

remote databases

 command syntax for accessing • 98

remote installation

 access, requirements • 50, 51

remote user authorization

 creating • 68

 defining during installation • 42

 deleting • 69

 described • 30

 editing • 71

 global and private • 31

 IILOGIN_nodename • 120

 Installation Password and login account password • 30

 storage of • 120

remote_vnode (configuration parameter) • 114, 119

S

security

 .NET Data Provider • 268

 Ingres Net • 23

 resolving problems • 130

select loops and cursors • 244

server class keywords • 100

server classes • 95

 default • 100

server connections, establishing • 117

server installation

 setup parameters for a • 42

SERVER_CONTROL (privilege) • 34

servers

 Bridge • 131

 Communications • 105

 Data Access • 143

 DBMS • 16

 Name • 22

 tasks related to • 95

set host command • 40

SNA LU0

 listen address • 361

SNA LU62

 listen address • 363

 SunLink Gateway configuration files • 379

SPX/IPX

 installation requirements • 38

 listen address • 371

SQL statement

 connect • 101

Star Server

 IISTAR_nodename • 120

SunLink Gateway

 sample configuration file • 379

SunLink SNA Peer-to-Peer

 installation requirements, Solaris and Sun-4 • 40

T

TCP/IP

 installation requirements, Windows • 38

 listen address • 357

time zone and region, defined • 42

toolbar, virtual nodes • 89

tools

 for managing Net • 32

 Ingres • 16

tracing

 Data Access Server • 146

 Driver Manager • 259

 enabling using Ingres JDBC Driver methods • 259

- Ingres Bridge connection • 139
- JDBC DataSource • 259
- levels • 146
- trace IDs • 259
- transaction mode
 - autocommit • 240
- troubleshooting
 - connectivity problems • 122
 - remote connection failure • 127
 - security and permission errors • 129
 - UNIX installation checklist • 124
 - VMS installation checklist • 126
 - Windows installation checklist • 123

U

- u command flag • 103
- unixODBC Driver Manager • 150

V

- vcbf (utility)
 - configuration parameters • 47
- virtual node name (vnode)
 - described • 28
 - table • 54
- virtual nodes toolbar
 - in Network Utility and VDBA • 89
- Visual DBA
 - accessing • 51
 - adding vnode attributes • 92
 - altering vnodes • 89
 - creating vnodes • 89
 - deleting vnodes • 89
 - list of user tasks • 89
- vnodes
 - adding attributes for • 64, 92
 - advanced, defined • 90
 - altering using Network Utility • 89
 - altering using VDBA • 89
 - creating using netutil • 62
 - creating using Network Utility • 89
 - creating using VDBA • 89
 - deleting using netutil • 69
 - deleting using Network Utility • 89
 - deleting using VDBA • 89
 - disconnecting from • 94
 - editing using netutil • 71
 - naming rules • 54
 - opening a utility window • 94

- private vs. global • 90
- refreshing • 93
- setting a default • 114
- simple, defined • 90
- testing a connection • 94
- tools for defining • 51