

Ingres® 2006 Release 3

Connectivity Guide

INGRES®

October 2007

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

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

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

This right to print copies is limited to the period during which the license for the product remains in full force and effect. The user consents to Ingres obtaining injunctive relief precluding any unauthorized use of the Documentation. Should the license terminate for any reason, it shall be the user's responsibility to return to Ingres the reproduced copies or to certify to Ingres that same have been destroyed.

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

The manufacturer of this Documentation is 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 © 2007 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**

| | |
|--|----|
| Connectivity Solutions Not in This Guide | 13 |
| Basic Networking Concepts | 14 |
| Ingres Components and Tools | 15 |
| Ingres Instance | 16 |
| System-specific Text in This Guide | 17 |
| Terminology Used in This Guide | 18 |
| Syntax Conventions Used in This Guide | 18 |

Chapter 2: Exploring Net **19**

| | |
|--|----|
| Ingres Net | 19 |
| General Communication Facility | 20 |
| Net Security | 21 |
| Installation Configurations That Require Net | 22 |
| Net and Other Ingres-related Products | 22 |
| Net and Enterprise Access and EDBC Products | 23 |
| Net and Ingres Star | 23 |
| Net Product Integration Summary | 24 |
| Benefits of Net | 25 |
| Net Concepts | 26 |
| Virtual Nodes | 26 |
| Connection Data | 27 |
| Remote User Authorizations | 28 |
| Global and Private Definitions | 29 |
| Net Management Tools | 30 |
| Net and Bridge Users | 31 |
| System Administrator and Ingres Net | 32 |
| Database Administrator and Ingres Net | 32 |
| End Users and Ingres Net | 32 |
| GCA Privileges | 33 |

Chapter 3: Installing and Configuring Net **35**

| | |
|--|----|
| Installation Components | 35 |
| How You Prepare for Installation | 35 |
| Network Installation and Testing | 35 |
| Setup Parameters for Net | 39 |

| | |
|--|----|
| How Net Setup Works on an Existing Installation | 41 |
| How Communications Are Enabled | 41 |
| How You Install Net..... | 42 |
| invalidpw Program | 43 |
| Create Password Validation Program (UNIX) | 44 |
| Net Configuration Parameters—Customize Ingres Net..... | 45 |

Chapter 4: Establishing Communications 47

| | |
|---|----|
| How User Access Is Established | 47 |
| Requirements for Accessing Remote Instances | 48 |
| Requirements for Accessing Distributed Databases | 49 |
| Access Tools for Defining Vnodes..... | 50 |
| Netutil (Net Management Utility) | 51 |
| Netutil Startup Screen | 51 |
| Virtual Node Name Table in Netutil..... | 52 |
| Login and Password Data Table in Netutil | 53 |
| Connection Data Table in Netutil | 55 |
| Other Attribute Data Table in Netutil..... | 57 |
| Netutil Operations..... | 58 |
| Prerequisites to Establish and Test a Remote Connection | 59 |
| Establish and Test a Remote Connection Using Netutil..... | 60 |
| Delete an Entry | 66 |
| Change an Entry..... | 68 |
| Define an Installation Password for the Local Instance | 72 |
| Netutil Non-Interactive Mode | 73 |
| Command Line Flags in Netutil Non-interactive Mode | 74 |
| Create Function—Create a Remote User Authorization..... | 76 |
| Destroy Function—Destroy a Remote User Authorization | 78 |
| Show Function—Display Remote User Authorizations | 79 |
| Create Function—Define an Installation Password for the Local Instance | 80 |
| Create Function—Create a Connection Data Entry..... | 81 |
| Destroy Function—Destroy a Connection Data Entry | 82 |
| Show Function—Display Connection Data Entries..... | 84 |
| Stop and Quiesce Commands—Stop or Quiesce One or More Communications Servers..... | 86 |
| Network Utility and Visual DBA..... | 87 |
| Virtual Nodes Toolbar | 87 |
| Simple and Advanced Vnodes | 88 |
| Advanced Vnode Parameters | 88 |
| Installation Password Definitions for the Local Instance | 91 |
| Changing Installation Passwords | 91 |
| Additional Vnode-Related Tasks | 91 |
| Server-related Tasks | 93 |

| | |
|---|----------------|
| Chapter 5: Using Net | 95 |
| Connecting to Remote Databases | 95 |
| Database Access Syntax—Connect to Remote Database | 96 |
| Using the SQL Connect Statement with Net | 99 |
| Commands and Net | 100 |
| User Identity on Remote Instance | 101 |
| -u Command Flag—Impersonate User | 101 |
| Verify Your Identity | 102 |
| Chapter 6: Maintaining Connectivity | 103 |
| Start Communications Server | 103 |
| Stop Communications Server | 104 |
| Network Server Control Screen in Netutil | 104 |
| Stop or Quiesce a Communications Server Using Netutil | 106 |
| Inbound and Outbound Session Limits | 108 |
| How You Set Inbound and Outbound Session Limits | 108 |
| Logging Levels | 109 |
| How You Change the Logging Level | 109 |
| How You Direct Logging Output to a File | 110 |
| Default Remote Nodes | 111 |
| How You Set Default Remote Nodes | 111 |
| Start Data Access Server (DAS) | 112 |
| Stop Data Access Server (DAS) | 112 |
| Chapter 7: Troubleshooting Connectivity | 113 |
| How Connection Between the Application and DBMS Server Is Established | 113 |
| Where Ingres Net Information Is Stored | 114 |
| config.dat—Store Net Configuration Values | 115 |
| Name Server Database—Store Remote Access Information | 116 |
| Causes of Connectivity Problems | 118 |
| How You Diagnose Connectivity Problems | 118 |
| General Net Installation Check | 119 |
| Connection Errors | 123 |
| How You Resolve Net Registration Problems | 126 |
| Security and Permission Errors | 126 |
| Chapter 8: Exploring Bridge | 129 |
| Ingres Bridge | 129 |
| How the Bridge Server Works | 129 |
| Tools for Configuring Bridge | 130 |

| | |
|---|-----|
| Installation Configurations That Require Bridge | 130 |
| How Bridge Is Installed | 132 |
| How Bridge Is Started..... | 132 |
| config.dat File—Store Bridge Configuration | 133 |
| ingstart Command—Start the Bridge Server | 133 |
| iigcb Command—Start the Bridge Server..... | 134 |
| How the Client Is Set Up | 134 |
| vnode Definition—Enable Client Access to Remote Servers Through Bridge..... | 135 |
| Bridge Server Monitoring | 135 |
| Stop the Bridge Server | 136 |
| How a Connection Is Established Through Bridge | 136 |
| Bridge Troubleshooting | 137 |
| Sample Bridge Server Configuration | 138 |

Chapter 9: Configuring the Data Access Server 141

| | |
|---|-----|
| Data Access Server | 141 |
| Data Access Server Parameters—Configure DAS | 142 |
| How You Enable Data Access Server Tracing | 143 |
| Tracing Levels | 144 |

Chapter 10: Understanding ODBC Connectivity 145

| | |
|--|-----|
| ODBC Driver | 145 |
| ODBC Call-level Interface | 146 |
| Unsupported ODBC Features..... | 147 |
| Read-Only Driver Option | 147 |
| ODBC Driver Requirements..... | 147 |
| ODBC Driver Manager Programs | 148 |
| Protocols Supported by ODBC Driver | 148 |
| Support for Previously Released ODBC Drivers | 149 |
| Backward Compatibility Issues for ODBC DSN Definitions | 149 |
| Configure a Data Source (Windows)..... | 150 |
| Configure a Data Source (UNIX and VMS)..... | 151 |
| iiodbcadmin Utility | 152 |
| Connection String Keywords | 152 |
| ODBC CLI Implementation Considerations | 154 |
| Configuration on UNIX, Linux, and VMS | 154 |
| Optional Data Source Definitions..... | 155 |
| Ingres ODBC and Distributed Transactions (Windows) | 155 |
| How You Enable the Use of Distributed Transactions through the Ingres ODBC Driver..... | 156 |
| Vnode Definitions When Using Distributed Transactions through ODBC | 156 |
| Troubleshooting Distributed Transactions Through ODBC..... | 157 |

| | |
|---|-----|
| Ingres ODBC and .NET Framework | 158 |
| Sample Program Using MS ODBC .NET Data Provider Access..... | 159 |
| Sample .NET Configuration for the Sample Program..... | 161 |

Chapter 11: Understanding JDBC Connectivity 163

| | |
|---|-----|
| JDBC Components..... | 163 |
| JDBC Driver | 163 |
| JDBC Information Utility—Load the JDBC Driver | 164 |
| Unsupported JDBC Features..... | 165 |
| JDBC Driver Interface | 166 |
| JDBC Driver and Data Source Classes | 166 |
| JDBC Implementation Considerations | 174 |
| JDBC User Authentication | 175 |
| How Transactions Are Autocommitted..... | 176 |
| Cursors and Result Set Characteristics | 178 |
| Cursors and Select Loops..... | 180 |
| Database Procedures..... | 181 |
| Named and Unnamed Parameters | 181 |
| Additional Parameter Considerations..... | 182 |
| Executing Procedures | 182 |
| BLOB Column Handling..... | 183 |
| Date/Time Columns and Values | 187 |
| National Character Set Columns..... | 189 |
| Data Type Compatibility | 190 |
| JDBC Tracing..... | 193 |
| Tracing Levels | 195 |

Chapter 12: Understanding .NET Data Provider Connectivity 197

| | |
|--|-----|
| .NET Data Provider..... | 197 |
| .NET Data Provider Architecture | 197 |
| Data Provider Data Flow | 198 |
| Data Provider Assembly..... | 198 |
| Data Provider Namespace | 199 |
| Data Retrieval Strategies | 199 |
| Connection Pooling | 200 |
| Code Access Security..... | 201 |
| .NET Data Provider Classes..... | 201 |
| IngresCommand Class..... | 202 |
| Sample Program Constructed with .NET Data Provider | 205 |
| IngresCommandBuilder Class | 207 |
| IngresConnection Class | 210 |

| | |
|--|-----|
| IngresConnectionStringBuilder Class | 218 |
| IngresDataReader Class | 223 |
| IngresDataAdapter Class | 231 |
| IngresError Class..... | 234 |
| IngresErrorCollection Class | 236 |
| IngresException Class | 238 |
| IngresFactory Class | 240 |
| IngresInfoMessageEventArgs Class | 241 |
| IngresInfoMessageEventHandler Class..... | 242 |
| IngresMetaDataCollectionNames Class..... | 243 |
| IngresParameter Class | 244 |
| IngresParameterCollection Class..... | 249 |
| IngresPermission Class..... | 251 |
| IngresRowUpdatedEventArgs Class | 251 |
| IngresRowUpdatedEventHandler Class..... | 252 |
| IngresRowUpdatingEventArgs Class..... | 252 |
| IngresRowUpdatingEventHandler Class | 253 |
| IngresTransaction Class..... | 254 |
| Data Types Mapping | 256 |
| DbType Mapping..... | 257 |
| Coercion of Unicode Strings | 258 |
| IngresDataReader Object—Retrieve Data from the Database | 258 |
| Build the IngresDataReader | 259 |
| IngresDataReader Methods | 259 |
| Example: Using the IngresDataReader..... | 260 |
| ExecuteScalar Method—Obtain a Single Value from a Database | 261 |
| GetBytes Method—Obtain BLOB Values from a Database | 261 |
| GetSchemaTable Method—Obtain Schema Information from a Database..... | 262 |
| ExecuteNonQuery Method—Modify and Update Database | 262 |
| IngresDataAdapter Object—Manage Data..... | 263 |
| Integration with Visual Studio 2005 | 264 |
| Install the Data Provider into the Toolbox..... | 265 |
| Start the Ingres Data Adapter Configuration Wizard..... | 266 |
| Design a Query Using the Query Builder..... | 270 |
| Server Explorer Integration..... | 272 |
| Application Configuration File—Troubleshoot Applications..... | 274 |

Chapter 13: Configuring Ingres to Use Kerberos 275

| | |
|--|-----|
| Kerberos..... | 275 |
| Kerberos Configuration in the Enterprise..... | 276 |
| Kerberos Configuration Files—Configure Kerberos for Ingres | 278 |
| The Ingres Service Principal—Authorize Client Connections | 279 |

| | |
|---|-----|
| How You Configure Ingres to Use Kerberos | 281 |
| iisukerberos Command—Perform Basic Kerberos Configuration | 282 |
| Ingres Configuration Options for Kerberos | 283 |
| Basic Configuration for Kerberos | 283 |
| remote_mechanism Parameter—Configure Client in a Homogeneous Kerberos Environment | 284 |
| vnode Connection Attributes—Configure Client in a Heterogeneous Kerberos Environment..... | 285 |
| Encryption Parameters—Enable Kerberos Encryption..... | 286 |
| Use Kerberos for Local Authentication..... | 287 |
| How Name Server Delegation Works | 287 |

Appendix A: TCP/IP Protocol 289

| | |
|---|-----|
| Listen Address Format | 289 |
| Network Address Format | 290 |
| Connection Data Entry Information | 290 |
| Windows | 290 |
| UNIX | 290 |
| VMS | 291 |
| MVS | 291 |

Appendix B: SNA LU0 Protocol 293

| | |
|-----------------------------|-----|
| Listen Address Format | 293 |
| MVS | 293 |

Appendix C: SNA LU62 Protocol 295

| | |
|-----------------------------|-----|
| Listen Address Format | 295 |
| MVS | 295 |
| Solaris | 297 |
| HP-UX..... | 298 |
| RS/6000..... | 300 |

Appendix D: DECnet Protocol 301

| | |
|-----------------------------|-----|
| Listen Address Format | 301 |
| VMS | 301 |

Appendix E: SPX/IPX Protocol 303

| | |
|-----------------------------|-----|
| Listen Address Format | 303 |
| Windows | 303 |
| UNIX and VMS | 304 |

Appendix F: LAN Manager Protocol **307**

| | |
|--|-----|
| LAN Manager Listen Address—Enable Communications | 307 |
|--|-----|

Appendix G: SunLink Gateway Configuration Files **309**

| | |
|--|-----|
| SunLink Gateway Configuration File | 309 |
| Solaris Independent LUs | 310 |
| Solaris Dependent LUs | 312 |
| SunOS (or Sun-4) Independent LUs | 314 |
| SunOS (or Sun-4) Dependent LUs | 316 |

Appendix H: AIX SNA Services/6000 Configuration Profiles **319**

| | |
|--|-----|
| Sample Configuration Profiles | 319 |
| CONNECTION Profile for Independent LUs | 319 |
| CONNECTION Profile for Dependent LUs | 320 |
| LOCALLU Profile for Independent LU | 320 |
| LOCALLU Profile for Dependent LU | 321 |
| MODE Profile for Independent LUs | 322 |
| MODE Profile for Dependent LUs | 322 |

Appendix I: HP-UX SNAplus Configuration **323**

| | |
|--|-----|
| Sample Configuration File Excerpts | 323 |
| Independent LUs | 324 |
| Dependent LUs | 325 |
| Dynamically Loadable TP | 326 |

Appendix J: Netu Procedures **327**

| | |
|---|-----|
| Start Netu | 327 |
| Netu User Interface | 328 |
| Stop the Communications Server | 328 |
| Modify Node Entry | 329 |
| Modify Remote Authorization Entry | 329 |
| Exit Netu | 330 |
| Remote Node Definition Operations | 330 |
| Add or Merge Remote Node Definitions | 331 |
| Delete Remote Node Definitions | 333 |
| How You Change Remote Node Definitions | 334 |
| Retrieve Remote Node Definition Information | 336 |
| Remote User Authorization Operations | 338 |
| Define Remote User Authorizations | 339 |

| | |
|---|-----|
| Delete Remote User Authorizations | 340 |
| Change Remote User Authorizations | 341 |
| Retrieve Remote User Authorizations | 344 |
| Netu Options for Stopping the Communications Server | 345 |
| Obtain GCF Address | 346 |
| Stop Communications Server | 347 |

| | |
|--------------|------------|
| Index | 349 |
|--------------|------------|

Chapter 1: Introducing Ingres Connectivity

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 EDBC 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

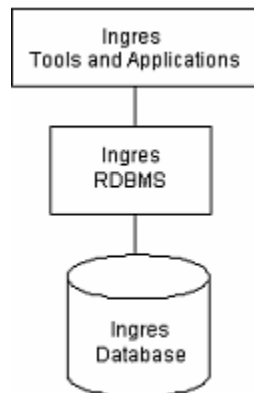
To use this guide effectively, you should be familiar with the basic components of Ingres, client/server concepts, and the Ingres tools required to configure, maintain, and view data.

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*.

The following figure illustrates the relationships among Ingres components and tools:



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 (vdca)
- 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 chapter describes Ingres Net and its many benefits in establishing and maintaining Ingres communications. It also explains the role that various connectivity components play in establishing communications with a remote DBMS server. This chapter also introduces Ingres connectivity concepts used in this guide. Lastly, it describes the types of Ingres users and the connectivity-related tasks that each performs.

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

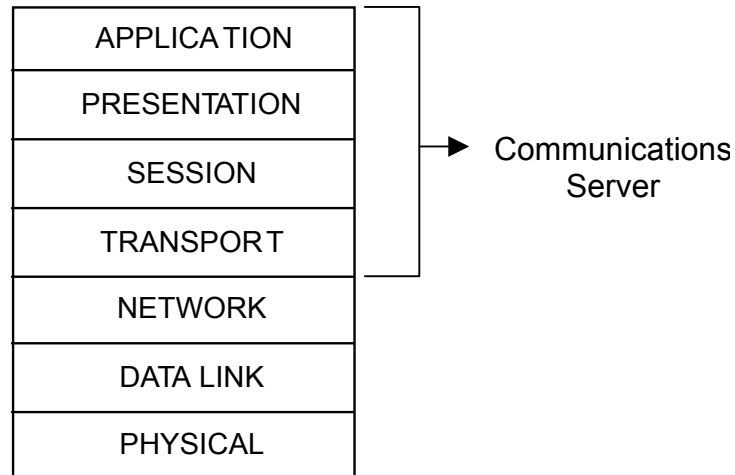
Ingres Net works with the basic Ingres components to enable connectivity between client and server instances. It also uses the General Communication Facility to manage communication among various components of Ingres.

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. For additional information on the security system, see the "Ensuring Access Security" chapter in the *Database Administrator Guide*.

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*.

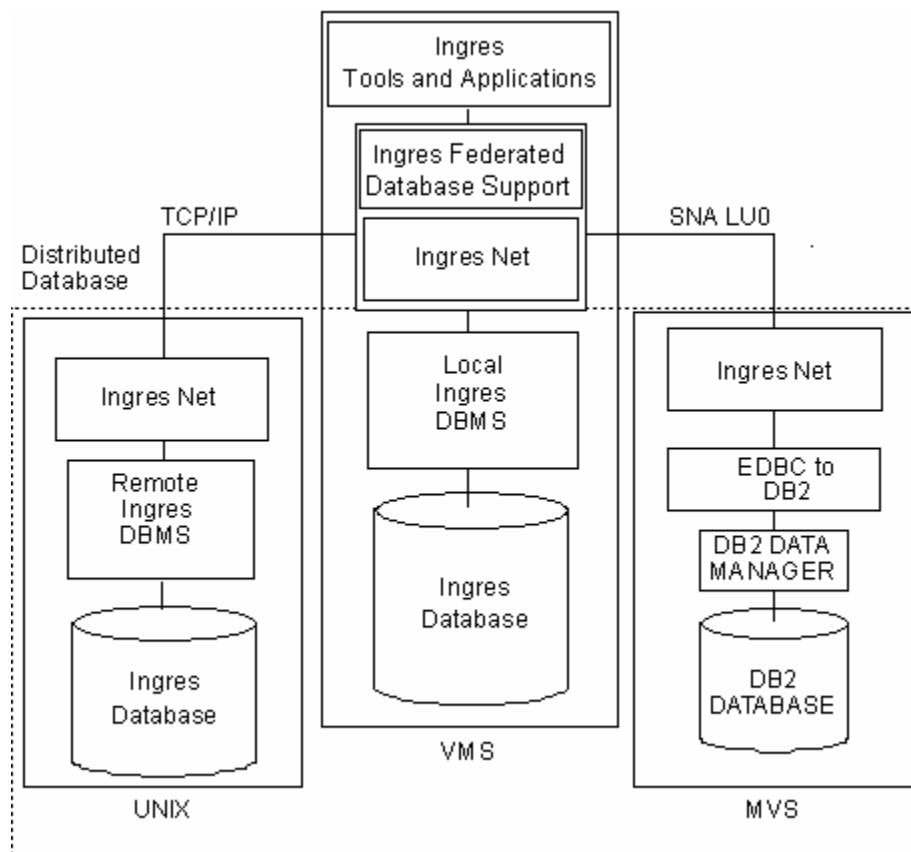
For a full explanation of how to use Ingres Star, see the *Ingres Star User Guide*.

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."

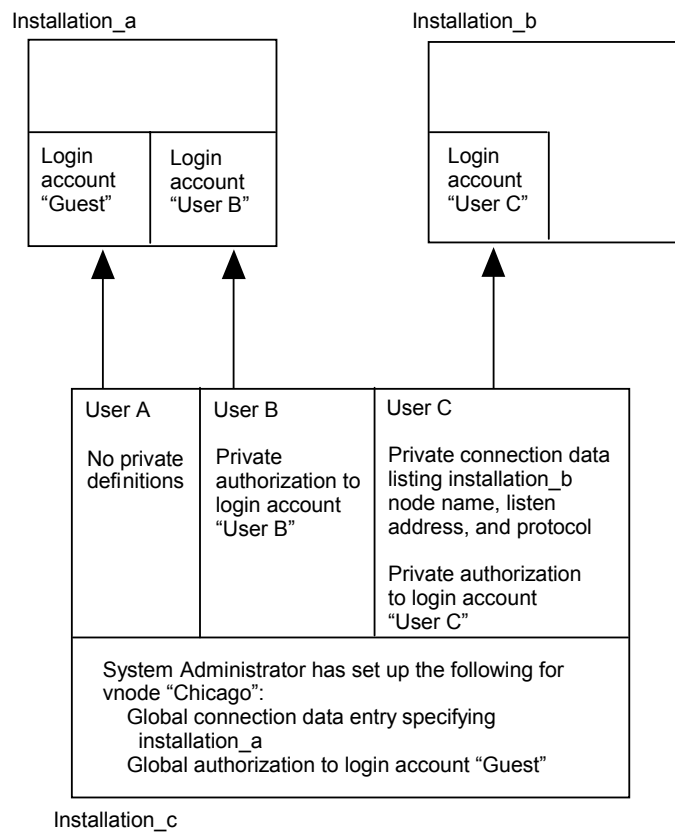
Global and Private Definitions

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 chapter explains how to install and configure Ingres Net as part of a new or existing installation.

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 .NET Data Provider
- 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.
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 35) 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 systems, make sure the ingvalidpw program is installed. For details, see Create Password Validation Program (UNIX) (see page 44).

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 chapter contains the following topics:

- An overview of the process of creating connection data entries and authorizing user access to Ingres using virtual nodes (vnodes)
- An overview of the tables and menu of the forms-based Net Management Utility (netutil) startup screen
- Procedures for establishing and maintaining connections to remote Ingres instances using the following tools:
 - Netutil
 - The non-interactive mode of netutil
 - Network Utility and Visual DBA

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 28).

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 74).

- 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 54).

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 29).

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 | * | Installation Password of remote instance |
| Local Installation Password | | Global | * | 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 56).

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:

wintcp

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

Note: This keyword is obsolete. It is provided for backward compatibility and will be removed in the future. Use `tcp_ip` instead.

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

tcp_ip

TCP/IP Internet protocol for UNIX and Windows (using WinSock 2.2 API)

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 62).

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 56).
- 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:

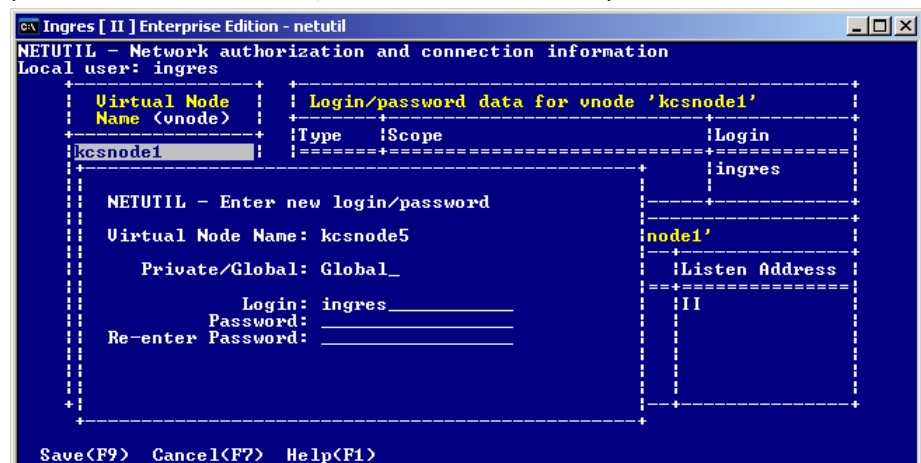
Choose type of login to be created

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.

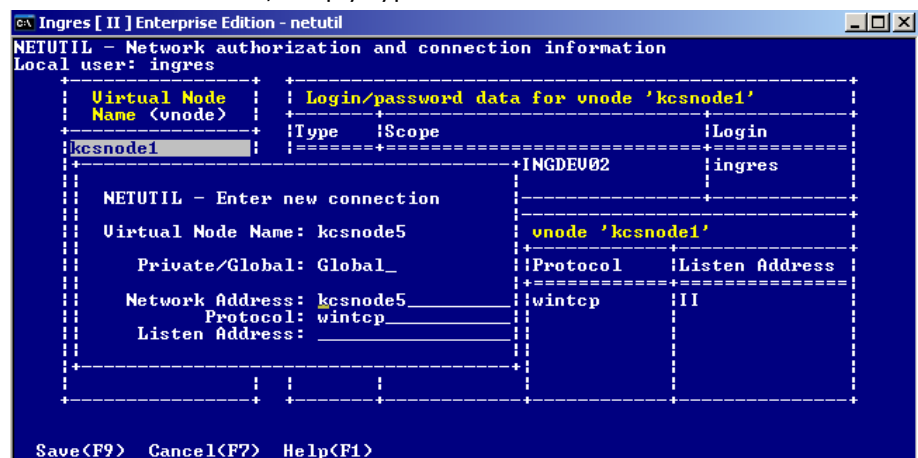
The Enter new login/password pop-up window appears. It displays prompts for the login of the account that is used on the remote node, the password of that account, and verification of the password.



5. 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.



6. 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.

7. 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.

8. 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.

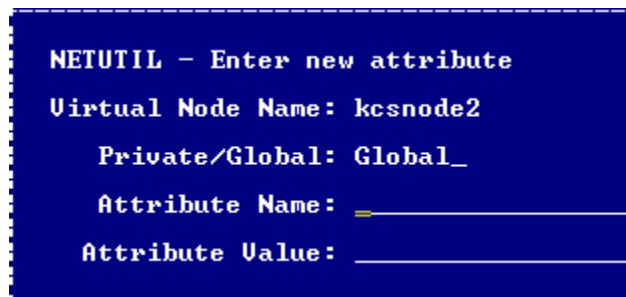
To configure one or more attributes for a vnode

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, 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 Name Server processing.

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) and the environment variable `II_CHARSET` must be identical on both the client and server machines.
- On Windows, the client and server machines must be logged into the same Windows network domain.
- 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 `ENABLE` to allow direct connections. For important information on network security and `II_GC_REMOTE`, see the *System Administrator Guide*.

encryption_mode

Determines 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

Determines 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**.

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.

Create an Additional Connection Data Entry

If a remote instance has more than one Communications Server or can be accessed by more than one network protocol, include that information in your vnode definition by adding extra entries to the Connection Data table. 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.

Know the following information before beginning this procedure:

- 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 that is used to make the connection. For more information, see Network Protocol Keywords (see page 56).

To define an additional connection data entry

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.

Create an Additional Remote User Authorization

End users can create a private remote user authorization for an existing vnode by adding an entry to the Login/password data table.

For the user who sets it up, a private authorization overrides a global authorization defined to the same vnode. In other words, Ingres Net uses the private authorization whenever the user who created it uses the vnode.

Know the following information before beginning this procedure:

- 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

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. (Notice that 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.

```

Ingres [ II ] Enterprise Edition - netutil
NETUTIL - Network authorization and connection information
Local user: ingres

+-----+-----+
| Virtual Node | Login/password data for vnode 'kcsnode2' |
| Name (vnode) | |
+-----+-----+
| kcsnode1    | Type | Scope | Login |
| kcsnode2    | +-----+-----+-----+
| kcsnode4    | Global | Any user on INGDEU02 | ingres |
| kcsnode5    | +-----+-----+-----+
| kcsnode1    | |
+-----+-----+

+-----+
| NETUTIL - Edit login and password |
+-----+
| Virtual Node Name: kcsnode2       |
| Private/Global: Global_           |
| Login: ingres                     |
| Password:                         |
| Re-enter Password:                |
+-----+

Save(F9)  Cancel(F7)  Help(F1)

```

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.

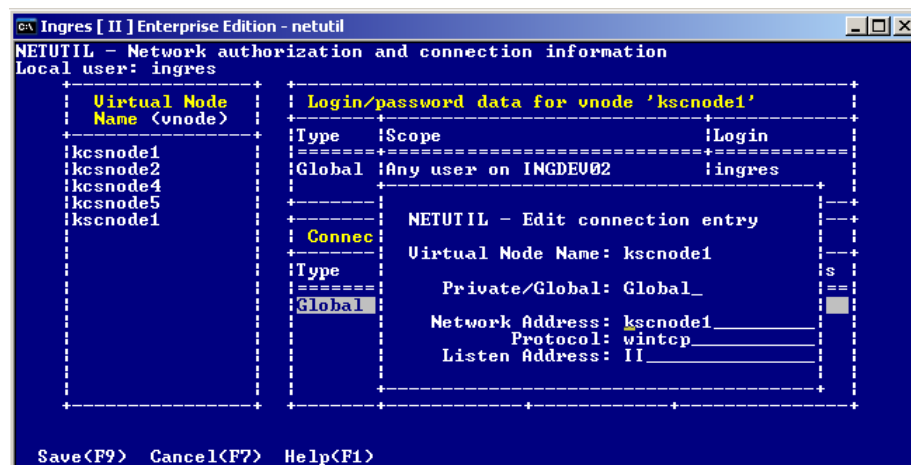
Netutil returns to the startup screen. The edited remote user authorization is displayed in the Login/password data table.

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

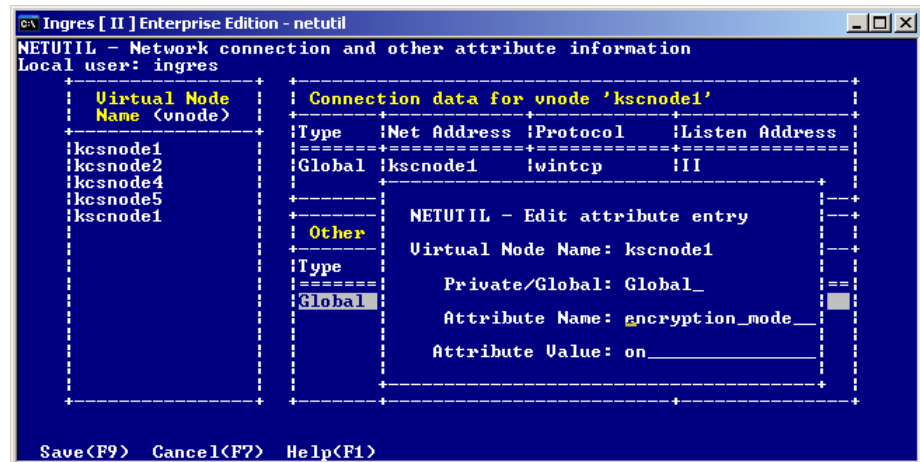
To edit attribute data for a particular vnode

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 62). 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 56).

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 56). 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 56). 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 performing 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

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.

Additional Connection Data Entry Creation

If a remote instance has more than one Communications Server or can be accessed by more than one network protocol, its information can be added in the vnode definition. 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, each server is tried in random order until an available one is found. Similarly, when a connection fails over one network protocol, an attempt to make a connection with any other protocol that has been defined is tried automatically.

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 for the same vnode. In other words, Ingres Net uses the private connection data entry whenever the user who created the entry uses the vnode.

Note: Have the following information on hand before beginning the procedures mentioned below: the network address of the node on which the remote instance resides, the listen address of the remote instance's Communications Server, and the keyword for the network protocol that is used to make the connection.

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

- Maintaining Vnode Connection Information
- Adding a Connection Data Definition
- Altering a Connection Data Definition
- Dropping a Connection Data Definition

Additional Remote User Authorization Creation

End users can create (as well as alter and drop) a private remote user authorization for an existing vnode. For the user who sets it up, a private authorization overrides a global authorization defined to the same vnode. In other words, Ingres Net uses the private authorization whenever the user who created it uses the vnode.

Know the following information before beginning these procedures:

- 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.

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

- 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 performing 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 the following vnode-related tasks.

Refreshing Vnodes

Refreshing is used to update loaded vnode data. You can selectively choose to refresh particular vnodes by either selecting the Force Refresh button on the toolbar or choosing Node, Force Refresh. You can configure your background refresh settings by choosing File, Preferences.

Testing Vnodes

Network Utility and Visual DBA provide a way to test if a connection to a specific node can be established. From the Virtual Nodes toolbar, choose Node, Test Node to initiate a connection test. If the connection fails, an error message is returned.

Disconnecting 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 performing this procedure can be found in the Procedures section of online help for Network Utility and Visual DBA. See the following 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 performing 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

Servers represent server classes associated with a particular vnode. In Network Utility and Visual DBA, 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 performing 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 chapter contains general information for working in the Ingres Net environment. It explains how to connect to remote databases and which SQL commands are valid with Ingres Net. This chapter also includes a general discussion of how the system perceives you, the user, when you are working in a remote database.

Connecting 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 111).

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 97).

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 98).

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, 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 56).

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 62).

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 does the same as the previous example and uses an attribute to set up a direct connection:

```
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 EDBC products are developed. Check the Readme file for the most up-to-date set of products.

Using 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 EDBC 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 | upgrafe |
| 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 | iimonitor |
| 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 "Authorizing User Access" chapter in the *Database Administrator 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*

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 chapter contains connectivity maintenance procedures, including:

- Starting and stopping a Communications Server
- Changing the Communications Server's connected session limits
- Changing the level of informational and error logging
- Directing logging information to an additional file
- Establishing default remote nodes
- Starting and stopping a Data Access Server

Start Communications Server

The Communications Server starts up automatically when you start up 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:

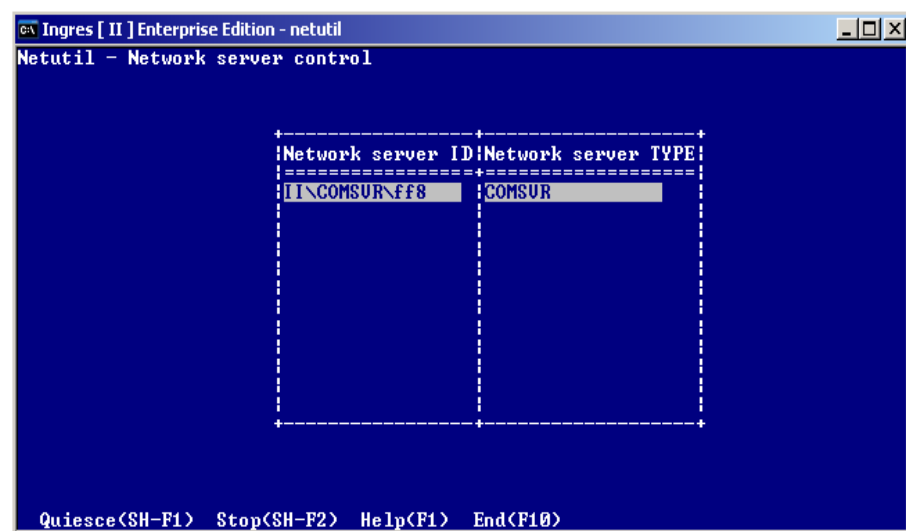
```
ingstop -iigcc
```

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. 

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 chapter provides information and procedures for diagnosing and solving network connectivity problems. It begins with some useful background information:

- A description of the internal steps necessary to complete a connection between a client and a server Using Net
- A description of Ingres Net configuration parameters and files

It then discusses common network connectivity problems and describes a procedure for diagnosing them. It concludes with several procedures that check for and resolve more specific problems.

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
IIIUSVR_nodename1
IIDB2UDB_nodename1
IIORACLE_nodename1
IIRDB_nodename1
IIRMS_nodename1
IILTICKET_nodename1
IIRTICKET_nodename1
IIINGRES_nodename2
IICOMSVR_nodename2
IISTAR_nodename2
IINODE_nodename2
IILOGIN_nodename2
IIIUSVR_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_GCNxx_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_GCNxx_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 (dmfrcp)
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 113) 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 ingvalidpw executable exists in \$II_SYSTEM/ingres/bin by typing:

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

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

Note: Not all Ingres UNIX releases use ingvalidpw to enforce C2 security. If the ingvalidpw executable is required for your release, it will be documented in the Readme file for your platform.

5. If the ingvalidpw executable exists:
 - a. Verify that it is owned by root. If not, log in as root and issue the command:
 - b. Verify that it has the "set uid" bit set. If not, issue the command:
 - c. Verify that the Ingres variable II_SHADOW_PWD is set to the full path to the ingvalidpw executable. Type:

```
$ ingprenv | grep II_SHADOW_PWD
```

The ingprenv utility displays the II_SHADOW_PWD variable.

6. If the ingvalidpw executable is not installed, create it using the mkvalidpw script. For details, see Create Password Validation Program (UNIX) (see page 44).

Chapter 8: Exploring Bridge

This chapter introduces the Ingres Bridge component and describes how it interacts with Ingres. It also describes how Ingres Bridge can be used in an Ingres Net and Ingres Enterprise Access configuration. This chapter concludes with information about configuring and using Ingres Bridge.

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

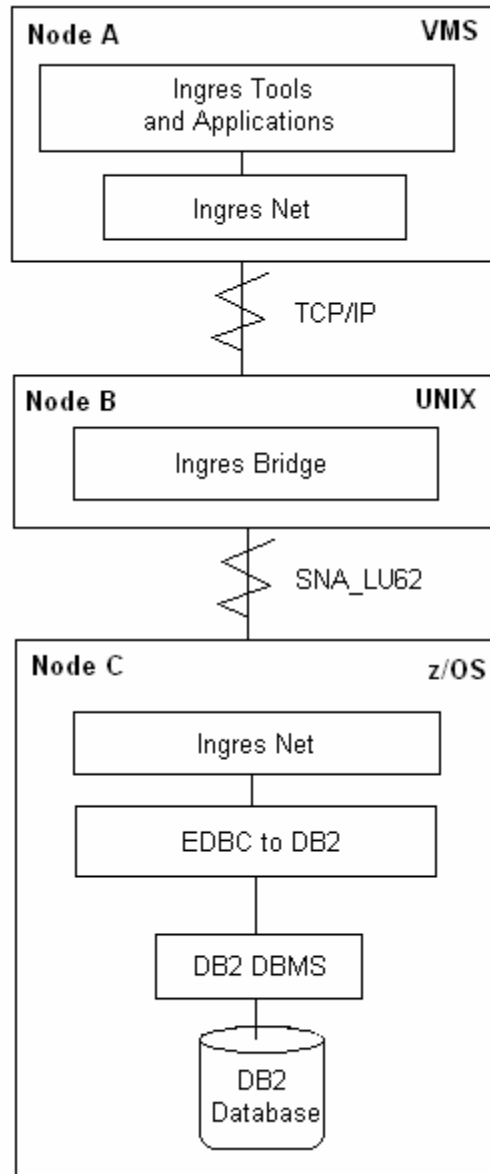
Ingres Bridge is required in any installation configuration where the client and server processes do not reside on the same machine *and* the client machine is on one type of local area network and the server machine is on another type of network.

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 56).

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 iinamu 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          = wintcp
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 chapter introduces the Data Access Server and explains how it can be configured and traced.

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 symbol port identifier such as II7. This port must not be used by any other network server on the platform.

<protocol>.status

Specifies the status of the network protocol. Options are:

on

Indicates that the DAS must listen/accept connection requests on the protocol.

off

Disables the protocol.

How You Enable Data Access Server Tracing

Because the DAS (iigcd) is a GCA-based server, it 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 either 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.

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 | <i>log</i> | Path and file name of the trace log |
| gcd_trace_level | II_GCD_TRACE | <i>0 – 5</i> | 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 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. (In Windows environments, connection pooling is provided by the Windows Driver Manager.)

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 can use ODBC data sources configured with the Microsoft ODBC Administrator on Windows or the iiodbcadm utility on non-Windows platforms. For more information, see [Configure a Data Source \(Windows\)](#) (see page 150) and [Configure a Data Source \(UNIX and VMS\)](#).

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()`
- `SQLSetPos()`

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 SET 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 150) and [Configure a Data Source \(UNIX and VMS\)](#).

ODBC Driver Requirements


The following sections list the ODBC driver software, platform, and protocol requirements. For additional information relating to the ODBC driver, see the Ingres Corporation web site. The latest release of the ODBC Driver is also available for free download on the Ingres Corporation web site.

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). The ODBC 3.0 SDK can be downloaded from the Microsoft Universal Data Access web site at <http://www.microsoft.com/data>. 

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 available as freeware and can be downloaded from <http://unixODBC.org>. The download includes a Readme file with instructions for UNIX, Linux and VMS. On Unix and Linux, the Ingres ODBC driver can also be used with the CAI/PT Driver Manager, which is available from Computer Associates. 

Note: The Ingres ODBC driver does not support the Merant ODBC Driver Manager.

Protocols Supported by ODBC Driver

The Ingres ODBC driver supports the following protocols:

- TCP/IP
- NetBIOS
- DECNet on VMS

Support for Previously Released ODBC Drivers

Each release of Ingres requires a compatible ODBC driver. On Windows, if your machine contains an Ingres 2.8 driver, the driver was registered with a driver name of "Ingres" and possibly "Ingres 2.8" if ODBC patches were installed. The Ingres installer registers the Ingres 2006 ODBC driver as "Ingres 3.0." Previous installations of Ingres r3, which were also registered as "Ingres 3.0," are overwritten by the Ingres installer.

The ODBC 2.8 driver is ODBC 2.0 API compliant, while the ODBC 3.5 driver is ODBC 3.0 API compliant.

UNIX, Linux, and VMS use an odbinst.ini configuration file instead of a registry. A succession of ODBC driver names are maintained that correspond to the Ingres release. No ODBC 2.8 drivers are supported on UNIX, Linux, or VMS.

Backward Compatibility Issues for ODBC DSN Definitions

On Linux, if you want to run the Ingres 2006 Release 2 ODBC driver with earlier versions of Ingres on the same machine, existing DSN definitions that point at "Ingres" will now reference the Ingres 2006 Release 2 version of the Ingres ODBC.

On Windows, existing DSN definitions will still point at the prior driver settings because the driver path is hard-coded in the registry and takes precedence over the driver name. We recommend that you delete your existing ODBC DSN definitions and create new ones after installing the Ingres ODBC driver.

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.

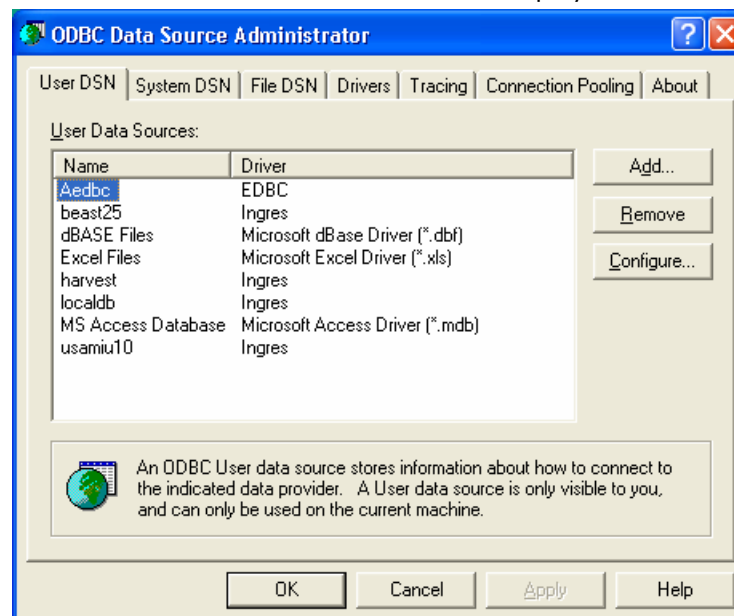
ODBC data sources are a convenient way of connecting to a database. You can, however, connect to a database without them by using only a connection string. For details, see Connection String Keywords (see page 152).

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:



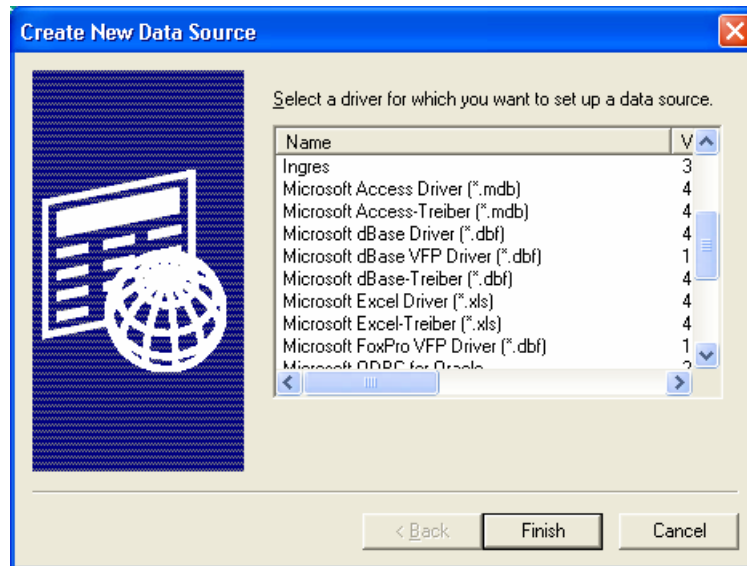
You can define one or more data sources for each installed driver. The data source name must provide a unique description of the data; for example, Payroll or Accounts Payable.

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 Data Source Administrator utility, **iiodbcadmin**.

iiodbcadmin Utility

The 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.

DSN-less connection strings have the form:

```
CONNECTSTR=SERVER=server_name; SERVERTYPE=server_type;  
DATABASE=database;[attribute=value]...
```

The following table provides the keyword for each connection string attribute.

| Keyword | Attribute Value Description |
|---------|---|
| DSN | Data source name. |
| DRIVER | Driver description as returned by <code>SQLDrivers()</code> |
| UID | User ID to override vnode definition. If specified, PWD must also be specified. |

| Keyword | Attribute Value Description |
|------------------|---|
| 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. |
| SERVER | Vnode name |
| 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. |
| CATSCHEMANULL | =Y Returns NULL for schema names form ODBC catalog functions |

ODBC CLI Implementation Considerations

The ODBC CLI includes two include files for compiling applications:

- `sql.h`
- `sqltext.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 `sqltext.h`.

Windows: On Windows, the library is named `INGODBC.DLL` and resides in `%II_SYSTEM%\ingres\lib`. ■

UNIX: On 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`. ■

VMS: On VMS, the library is named `IIODBC.EXE` and resides in `I_SYSTEM:[INGRES.LIBRARY]`. ■

ODBC CLI applications link against the appropriate shared library.

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 Linux:

```
cc -c myOdbcApp.c /I$II_SYSTEM/ingres/files
ld -o myOdbcApp myOdbcApp.o -L$II_SYSTEM/ingres/lib -liiodbc.1
```

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 \(Windows\)](#) (see page 150) and [Configure a Data Source \(UNIX and VMS\)](#).

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 use `SQLConnect()` by using 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 `iiodbcadm` utility (UNIX, Linux, and VMS) or Microsoft ODBC Administrator (Windows) is optional.

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.

Follow these steps:

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 Knowledge Document 415863.

Ingres ODBC and .NET Framework

.NET applications can access Ingres in two ways:

- Through the Microsoft ODBC .NET Data Provider
- Through the native Ingres .NET Data Provider

The Ingres ODBC Driver supports the ODBC .NET Data Provider. For more information on the native Ingres .NET Data Provider, see the chapter "Understanding .NET Data Provider Connectivity."

The .NET classes in the .NET System.Data.Odbc namespace (for example, OdbcConnection, OdbcCommand, and so on) can be used to access the Ingres ODBC DSN and ODBC driver. Like all other Ingres ODBC applications, .NET applications using the Microsoft ODBC .NET Data Provider to access the Ingres ODBC Driver require the Ingres Net Client to be installed on each client application machine in order to pick up the ODBC driver, Ingres API, Communications Server, and other Ingres components. More detail on the Microsoft ODBC .NET Data Provider can be found in the MSDN documentation at <http://msdn2.microsoft.com/en-us/library/system.data.odbc.aspx> (<http://msdn2.microsoft.com/en-us/library/system.data.odbc.aspx>).

If the application requires use of the .NET System.Transactions.TransactionScope, you must register the Ingres ODBC Driver to Windows. For details, see Ingres ODBC and Distributed Transactions (Windows) (see page 155).

Sample Program Using MS ODBC .NET Data Provider Access

The following is a sample .NET program that accesses Ingres through the Microsoft ODBC .NET Data Provider.

```
using System;
using System.Configuration;
using System.Data;
using System.IO;

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["myODBCConnectionString"];

        // get the System.Data.Odbc provider name
        string invariantName = connectionSetting.ProviderName;

        // get the ODBC connection string such as
        // "DSN=mydsn;UID=myuserid;PWD=mypass"
        string myConnectionString = connectionSetting.ConnectionString;

        DbProviderFactory factory = 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
```


Sample .NET Configuration for the Sample Program

A .NET configuration XML file can be used to define connection parameters rather than hard-coding them into the application source. Different configuration files can be deployed with the same application executable to direct the application to a different data provider, database host, or with different user credentials.

The following is an example for the above sample program:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionStrings>
    <clear />
    <!-- clear any inherited references in memory -->
    <add name="myODBCConnectionString"
          providerName="System.Data.Odbc"
          connectionString="DSN= mydsn;" />
  </connectionStrings>
</configuration>
```


Chapter 11: Understanding JDBC Connectivity

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 3.0 API released with the Sun Java 2 SDK, version 5.0. The driver supports application, applet, and servlet access to Ingres data sources through the Data Access Server.

Note: The JDBC driver provided in the Ingres 2.6 release continues to be supported in this current release. For migration instructions related to the JDBC driver, see the *Migration Guide*.

The Ingres JDBC Driver supports the following JDBC 3.0 features:

- Updateable 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 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. 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.

Unsupported JDBC Features

The Ingres JDBC Driver is compliant with the JDBC 3.0 API specification. JDBC 3.0 API interfaces are fully supported with the following exceptions:

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 `isLast()` method 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

Batched execution for `Statements`, `PreparedStatement`s, and `CallableStatement`s is supported by individual execution of each batched request. The driver implementation for batch updates is only as efficient as an application making individual update requests.

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 DataSource interface (<code>javax.sql.DataSource</code>). |
| <code>com.ingres.jdbc.IngresCPDataSource</code> | The Ingres implementation of the JDBC ConnectionPoolDataSource interface (<code>javax.sql.ConnectionPoolDataSource</code>). |
| <code>com.ingres.jdbc.IngresXADataSource</code> | The Ingres implementation of the JDBC XADataSource 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 2006 also contains the original classes for backward compatibility. The original `"ca.ingres.jdbc"` package path and classes will be removed from `ijjdbc.jar` in the next major release. Existing references to `"ca.ingres.jdbc"` classes will continue to work, but should be changed when convenient. New references should use the package path of `"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>`. A default properties file is loaded automatically by the Ingres JDBC Driver when the driver class is loaded. The default properties file is named `ijjdbc.properties` and must reside in a location accessible by the class loader used to load the driver. In general, this requires the properties file directory to 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 | The user ID on the target DBMS Server machine. See the description of the <code>vnode_usage</code> property in this table. 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). |
| 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). |

| Property | Attribute | Description |
|-----------------|-----------|--|
| connect_pool | POOL | <p>Server connection pool control. Available options 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. Available options are:</p> <p>on--uses select loops to retrieve query results</p> <p>off--uses cursors (default).</p> <p>For further details, see Cursors and Select Loops (see page 180).</p> |
| autocommit_mode | AUTO | <p>Autocommit cursor handling mode. Available options are:</p> <p>dbms--autocommit processing is done by the DBMS Server (default)</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 176).</p> |
| cursor_mode | CURSOR | <p>Default cursor concurrency mode, which determines the concurrency of cursors that have no concurrency explicitly assigned. Available options are:</p> <p>dbms--concurrency is determined by the DBMS Server</p> <p>update--provides updateable cursors</p> <p>readonly--provides non-updateable cursors (default)</p> <p>Further details are provided in Cursors and Result Set Characteristics (see page 178).</p> |

| Property | Attribute | Description |
|--------------|-----------|---|
| 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. Available options are:</p> <p>connect—Only the vnode connection information is used to establish the connection. (default)</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 175).</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 <code>II_TIMEZONE_NAME</code> 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 <code>II_DECIMAL</code> 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> |

| Property | Attribute | Description |
|-----------------|-----------|--|
| date_alias | DATE | Specifies the data type of columns created using the alias keyword "date". Should be set to either "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. |

Attributes can also be specified using the property name as the attribute name. Thus "UID=user1" and "user=user1" are semantically the same.

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 | Server host name or network address (required). |

| DS Property | Description |
|----------------|--|
| portName | Symbolic port ID (required). A port ID must be provided either numerically or symbolically. |
| portNumber | Numeric port ID (required). A port ID must be provided either numerically or symbolically. |
| 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'. |
| 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 |

Note that 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 174). The remaining Data Source properties correspond to the driver properties defined in JDBC Driver Properties (see page 167).

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 ingres.properties (must reside in CLASSPATH directory).

ingres.jdbc.property.<property name>

Driver connection properties. Defaults are property dependent.

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 187).

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.

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/db;attr=value
```

where:

host

Is 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].

port

Is the network port used by the DAS. This can be a numeric port number or an Ingres symbolic port address such as II7.

db

Is the target database specification. Any valid Ingres database designation can be used including vnode and server class (that is, vnode::dbname/server_class).

attr=value

(Optional) Is 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 167).

Note: A user ID and password are required when making remote connections and can be included as parameters to the getConnection() method as driver properties or as URL attributes.

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, 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 `invalidpw` program (see page 43) 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 167).

| 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. |

| Value | Mode | Description |
|--------------|--------------|--|
| 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 updateable (keyset) scrollable cursors are sensitive to changes. Either type can be specified for both read-only and updateable 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 updateable. The Ingres JDBC Driver automatically provides an updateable `ResultSet` when the associated cursor is updateable. The JDBC readonly/update mode characteristics are used by the Ingres Driver to control the mode of the resulting cursor.

For an updateable cursor, row updates and deletes can be performed using the updateable `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 updateable 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 updateable 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 updateable if the DBMS Server determines that an updateable 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.

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. For more information, see *How Transactions Are Autocommitted* (see page 176).

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. Updateable 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 178). 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 167).

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 updateable cursor.

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.

If parameter names are not provided to the Ingres JDBC Driver, the driver must query the database and assign names to the parameters based on the declared order of the procedure parameters. Because querying the database reduces the performance of database procedure execution, using named parameters in your applications is strongly encouraged.

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 2006 Release 3 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...'} |

- 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 167). 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 |
|---|---------------------------|----------------------------|
| <code>setString()</code> | <code>varchar</code> | <code>nvarchar</code> |
| <code>setAsciiStream()</code> | <code>long varchar</code> | <code>long nvarchar</code> |
| <code>setUnicodeStream()</code> | <code>long varchar</code> | <code>long nvarchar</code> |
| <code>setCharacterStream()</code> | <code>long varchar</code> | <code>long nvarchar</code> |
| <code>setObject(char[])</code> | <code>char</code> | <code>nchar</code> |
| <code>setObject(String)</code> | <code>varchar</code> | <code>nvarchar</code> |
| <code>setObject(Reader)</code> | <code>long varchar</code> | <code>long nvarchar</code> |
| <code>setObject(obj,CHAR)</code> | <code>char</code> | <code>nchar</code> |
| <code>setObject(obj,VARCHAR)</code> | <code>varchar</code> | <code>nvarchar</code> |
| <code>setObject(obj,LONGVARCHAR)</code> | <code>long varchar</code> | <code>long nvarchar</code> |
| <code>setObject(char[],OTHER)</code> | <code>char</code> | <code>char</code> |
| <code>setObject(String,OTHER)</code> | <code>varchar</code> | <code>varchar</code> |
| <code>setObject(Reader,OTHER)</code> | <code>long varchar</code> | <code>long varchar</code> |

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 165), 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

When 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

When 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 189). 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 189). 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 189).

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 189). 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 189). 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 189).

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.drv | <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 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 2005 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.

Note: The Ingres .NET Data Provider also supports .NET access to Enterprise Access data sources.

.NET Data Provider Architecture

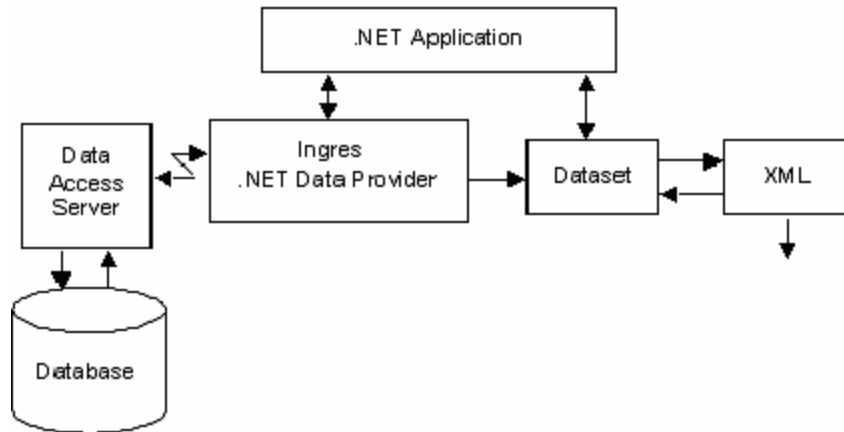
The Ingres .NET Data Provider offers a series of .NET types to describe the user's data, .NET provider classes to manipulate the data, and connection pooling to efficiently manage data connections.

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\dotnet\assembly\v2.0.

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 223).
- **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 231).

In addition to the low-level `DataReader` and `DataAdapter` access strategies, the data provider works with Visual Studio to support `TableAdapter`, `DataConnection`, 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 210).

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
- Cursor direction other than forward
- 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 210). |
| Parameters | get | The IngresParameterCollection for the parameters associated with the SQL query or database procedure. For more information, see IngresParameterCollection Class (see page 249). |
| 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. |

| Method | Description |
|---------------------|--|
| CreateParameter | Creates a new instance of IngresParameter. For more information, see IngresParameter Class (see page 244). |
| 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 223). |
| 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.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 = 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. |
| 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. |
| 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 214).</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 | The name of the target server. |
| 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 | Not supported at this time. Consider using EnlistDistributedTransaction as a work-around. |

| Method | Description |
|-----------|--|
| GetSchema | <p>Returns schema metadata from the Ingres catalog for the specified collection name. Valid collection names include:</p> <ul style="list-style-type: none"> ■ MetaDataCollections ■ DataSourceInformation ■ DataTypes ■ Restrictions ■ 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 211). |

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 |

| Constructor Overloads | Description |
|--------------------------|---|
| IngresConnection(string) | Instantiates a new instance of the IngresConnection class using the defined connection string |

Connection String Keywords

Connection string keywords are case-insensitive. Certain keywords are accepted as synonyms of each other. For example, keywords "Server" and "Address" are synonyms of "Host." Spaces in values are retained. Values may be delimited by double-quotes.

The connection string keywords for the IngresConnection class are:

| Keyword | Description |
|--|--|
| BlankDate | 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". |
| 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 Connection Timeout | The time, in seconds, to wait for an attempted connection to time out if the connection has not completed. Default is 15 seconds. |
| Cursor_Mode | 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: <ul style="list-style-type: none">■ readonly – Provides non-updateable cursors for best performance (default)■ update – Provides updateable cursors■ dbms – Concurrency is assigned by the DBMS Server |
| Database DB | Name of the database being connected to. If a server is required, use the syntax <i>dbname/server_class</i> . |

| Keyword | Description |
|---------------------------|---|
| Date_format 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 | 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 | The DBMS password of the user, which is equivalent to the Ingres -P flag. |
| Decimal_char | Decimal_char=', ' specifies that the DBMS Server is to use the comma (,) character 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. The default value is the period (.) as in 12.34. |
| Enlist | If set to true and if the creation thread is within a transaction context as established by System.EnterpriseServices.ServicedComponent, the IngresConnection in the transaction context is automatically enlisted. Default is true. |
| Group ID | Group identifier that has permissions for a group of users. |
| Host Server Address | Name of the target host server machine with the Data Access Server. |
| Max Pool Size | Maximum number of connections that can be in the pool. Default is 100. |
| Min Pool Size | Minimum number of connections that can be in the pool. Default is 0. |
| Money_format 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. |

| Keyword | Description |
|-------------------------------|---|
| Money_precision 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 PWD | The password to the database. This value may be case-sensitive depending on the target server. |
| Persist Security Info | If set to false, password information from the connection string is not returned in a get of the ConnectionString property. Default is false. If true, then password information from the connection string <i>is</i> returned in a get of the ConnectionString property. |
| Pooling | Enables or disables connection pooling. By default, connection pooling is enabled (true). If set to false, connection pooling is disabled. |
| Port | Port number on the target host server machine that the Data Access Server is listening to. Default is II7. |
| Role ID | Role identifier that has associated privileges for the role. |
| Role Password Role PWD | Role password associated with the Role ID. |
| Timezone 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 UID | The name of the authorized user connecting to the DBMS Server. This value may be case-sensitive depending on the target server. |
| Vnode_usage | <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 Data Access Server. Available options are:</p> <ul style="list-style-type: none"> ■ connect – Only the vnode connection information is used to establish the connection. This is the default. ■ login – Both the vnode connection and login information are used to establish the connection. <p>For further details, see Data Provider User ID Options (see page 217).</p> |

Data Provider User ID Options

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].

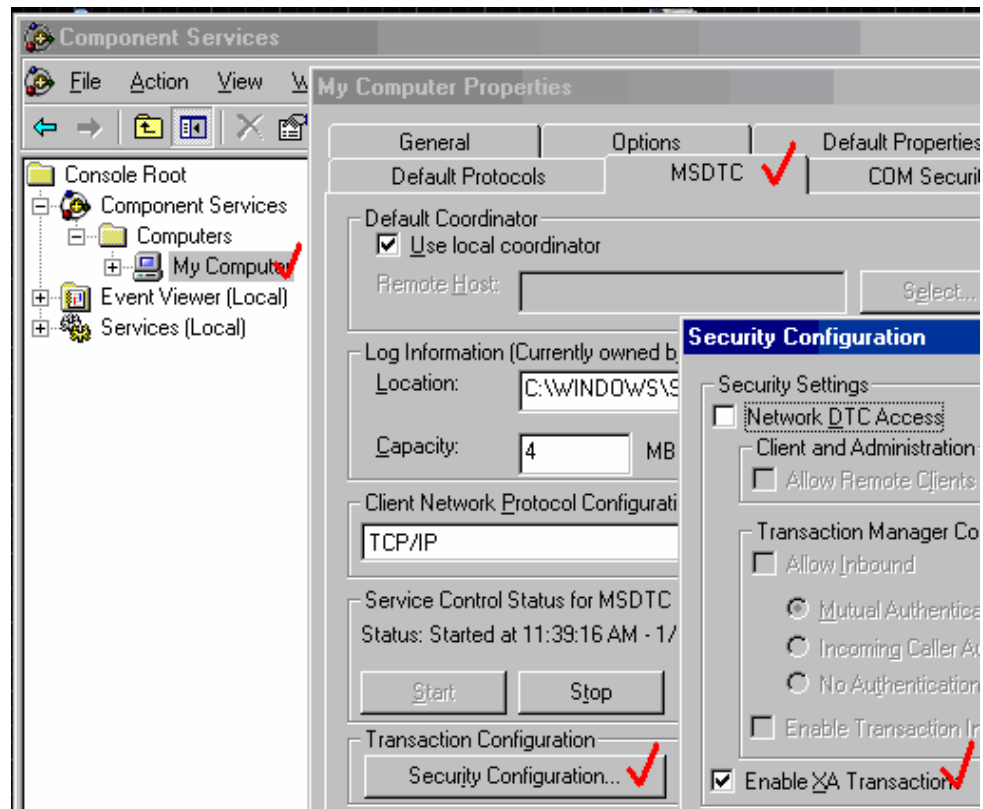
If a hostname is associated with multiple IP addresses, the data provider sequentially tries to connect to each IP address in the AddressList returned by System.Net.Dns.GetHostEntry until it achieves a successful socket connection or until it reaches the end of the list. If the connection to the first 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.

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.

To use the distributed transaction support in the data provider, the administrator of the Windows machine must enable XA transactions through Component Services.



IngresConnectionStringBuilder Class

The IngresConnectionStringBuilder class represents 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". |
| 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. |

| Property | Accessor | Description |
|--------------|----------|---|
| 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. |
| Enlist | get set | If set to true and if the creation thread is within a transaction context as established by System.EnterpriseServices.ServiceComponent, 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. |

| Property | Accessor | Description |
|---------------------|----------|--|
| 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. |
| 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. |

| Property | Accessor | Description |
|------------|----------|--|
| 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 | 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: <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. |

| Method | Description |
|-------------|--|
| 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 Example—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();
```

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 reader contains zero rows. |
| IsClosed | get | A true/false indicator as to whether the data reader is closed. |

| Property | Accessor | Description |
|-----------------|----------|---|
| 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. |
| GetName | Gets the column's name using a specified ordinal. |
| GetOrdinal | Gets the column's ordinal using a specified name. |

| Method | Description |
|----------------|--|
| 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 227). |
| 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. |
| ColumnSize | Int32 | Maximum possible length of a value in the column. |

| Column Information | Data Type | Description |
|--------------------|------------|---|
| 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 |
|--------------------|-----------|--|
| IsKeyColumn | 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. |

| 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 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 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 |

| Constructor Overloads | Description |
|---|--|
| 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 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 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 generates many of the other Ingres classes in an interoperable 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. |

| Method | Description |
|-------------------------------|---|
| 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
- ForeignKeys
- 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

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. |

| Property | Accessor | Description |
|---------------|----------|---|
| 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 "". |
| 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. |
| IngresParameter (string, DbType, string) | Instantiates a new instance of the IngresParameter class using the defined parameter name, DbType, and source column name. |
| IngresParameter (string, IngresType, string) | Instantiates a new instance of the IngresParameter class using the defined parameter name, IngresType, and source column name. |
| IngresParameter (string, DbType, int) | Instantiates a new instance of the IngresParameter class using the defined parameter name, DbType, and column size. |
| IngresParameter (string, IngresType, int) | Instantiates a new instance of the IngresParameter class using the defined parameter name, IngresType, and column size. |
| IngresParameter (string, DbType, int, string) | Instantiates a new instance of the IngresParameter class using the defined parameter name, DbType, column size, and source column name. |
| IngresParameter (string, IngresType, int, string) | Instantiates a new instance of the IngresParameter class using the defined parameter name, IngresType, column size, and source column name. |

| Constructor Overloads | Description |
|--|---|
| <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 245).

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. |
| Clear | Removes all items from the collection. |
| Contains | Indicates whether IngresParameter exists in the collection. True = does exist; False = does not exist. |
| 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. |

| Property | Accessor | Description |
|---------------|----------|--|
| 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 223).

| IngresType | Ingres Data Type | Description | .NET Data Type |
|-------------------|---------------------------|--|-----------------------|
| Binary | byte | Fixed length stream of binary data | Byte[] |
| Char | char | Fixed length stream of character data | String |
| DateTime | date | Date data | DateTime |
| Decimal | decimal | Exact numeric data | Decimal |
| Double | double precision (float8) | Approximate numeric data | Double |
| 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.

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 | TinyInt, if supported by the database; otherwise, SmallInt |
| 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 |

| DbType | IngresType |
|-------------------|---|
| 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 223). 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 227).

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.

Integration with Visual Studio 2005

The Ingres .NET Data Provider is integrated with Visual Studio 2005.

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's Visual Studio 2005.

Integration with Visual Studio 2005 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 2005 Toolbox contains a series of tabs (for example, Data, Components, and Window Forms) that list objects for the Visual Studio 2005 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 2005 Toolbox before using it for the first time.

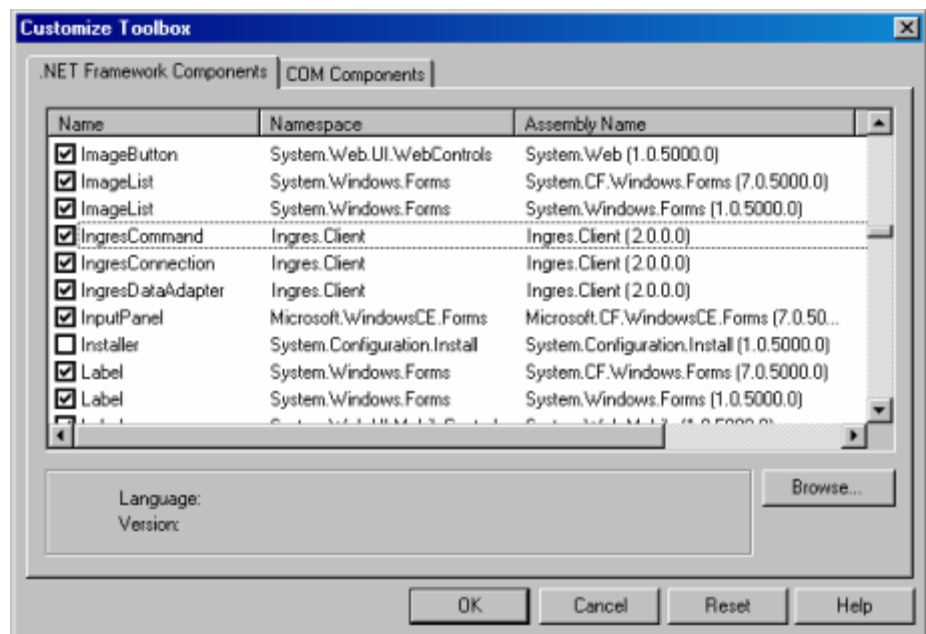
To install the data provider components into the Toolbox

1. Create an empty Winform application.
2. Right-click the Data tab of the toolbox, and select Customize Toolbox.

The Customize Toolbox 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 Customize Toolbox dialog, you can add them.

To add the Ingres .NET Data Provider components to the Customize Toolbox dialog

1. Click Browse on the Customize Toolbox dialog and browse to the directory C:\Program Files\Ingres\dotnet\assembly\v2.0.
2. Open the Ingres.Client.dll.

The components are added.

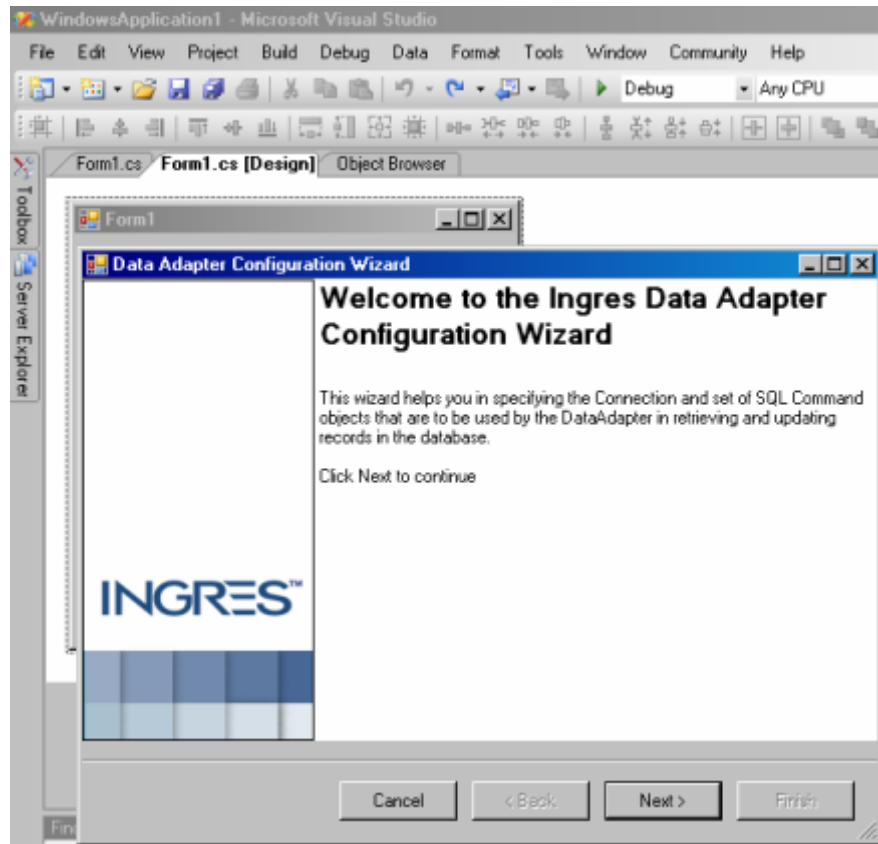
Start the Ingres Data Adapter Configuration Wizard

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 "ingresDataAdapter1" component and its icon are added to the Visual Studio 2005 designer component tray.

2. Click Cancel on the welcome page.

Only the IngresDataAdapter component is created.

Configure a Connection

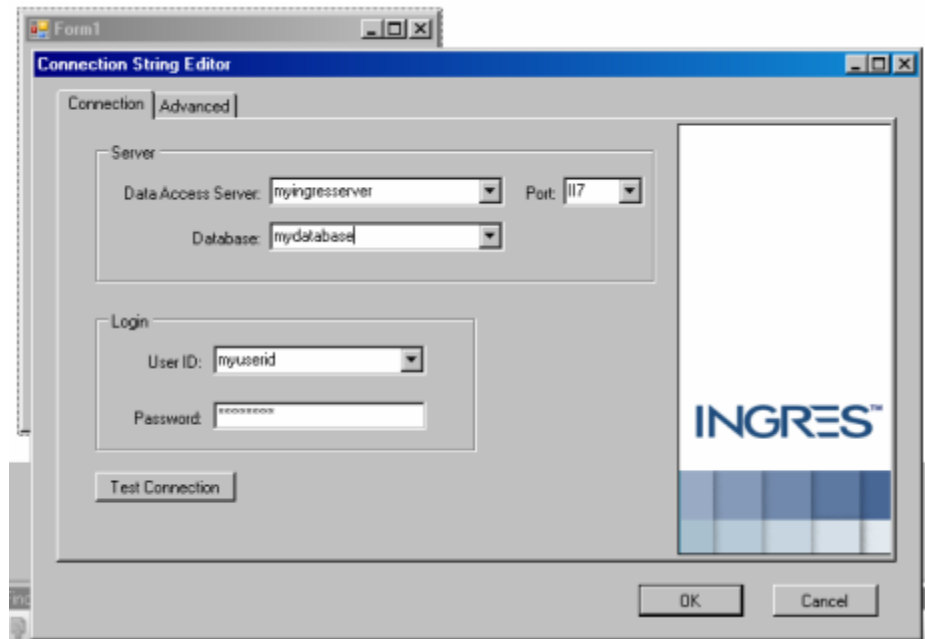
Ingres Data Adapter Configuration Wizard assists you in specifying the design properties of the `ingresDataAdapter1` component, including its connection string definition.

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 268).

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.

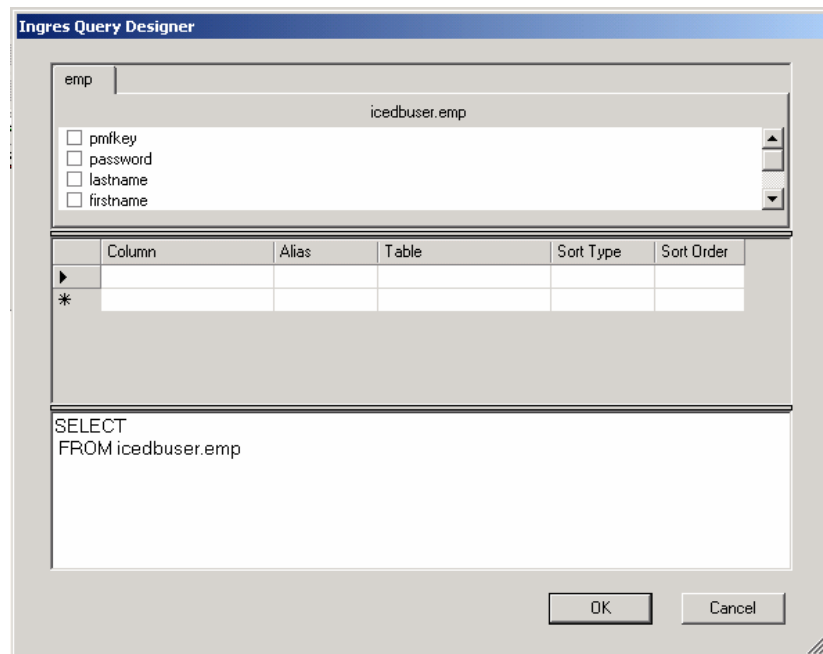
The Add Table dialog opens.

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.

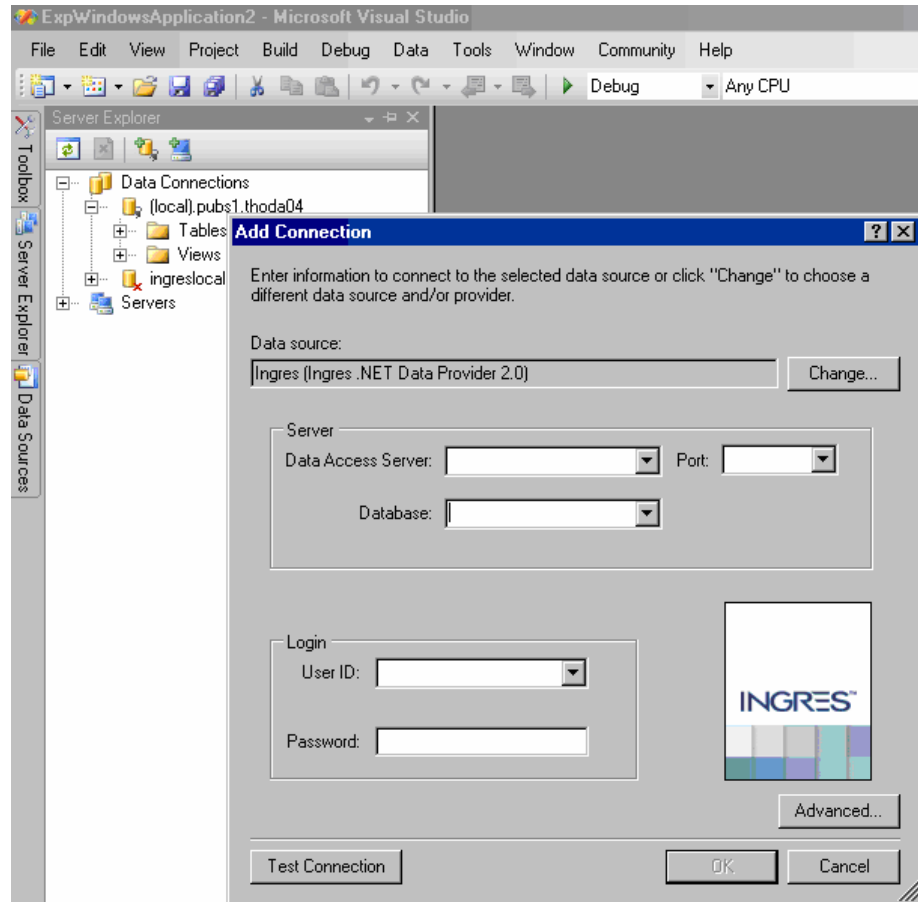
6. Click Finish.

The Ingres Data Adapter Wizard is closed.

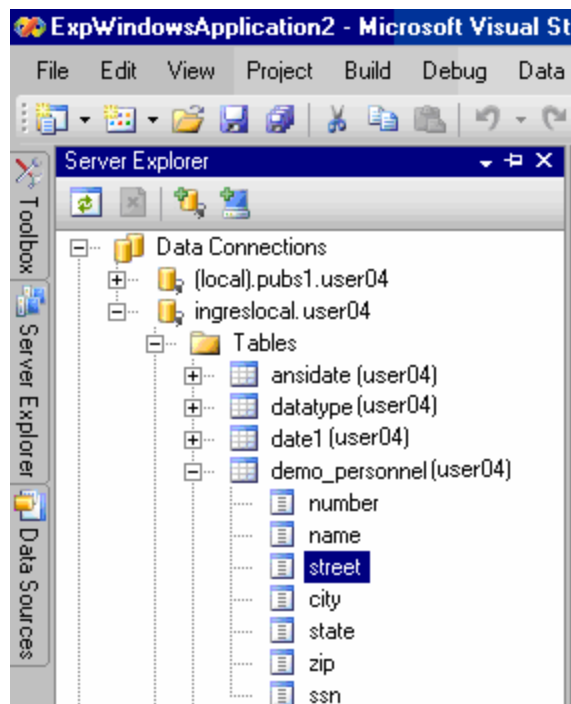
Server Explorer Integration

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

Chapter 13: Configuring Ingres to Use Kerberos

This chapter describes how to configure Ingres to use Kerberos network authentication and encryption protocol.

Kerberos

The Kerberos authentication mechanism can be used as an alternative to the Ingres or System security mechanisms. Kerberos provides a highly secure alternative to operating system-level password authentication, and optionally allows encryption of the entire data stream exchanged between the DBMS server and client.

The Ingres and System security mechanisms are called “static” mechanisms, because they are embedded in Ingres. The Kerberos security mechanism is called a “dynamic” mechanism, because it depends on third-party software that is dynamically loaded into Ingres executable images at runtime.

Kerberos is available as freeware from the Massachusetts Institute of Technology at <http://web.mit.edu/kerberos/>. Kerberos is also available commercially or may be available natively on certain operating systems, such as Linux. The MIT site contains extensive documentation on Kerberos installation and configuration.

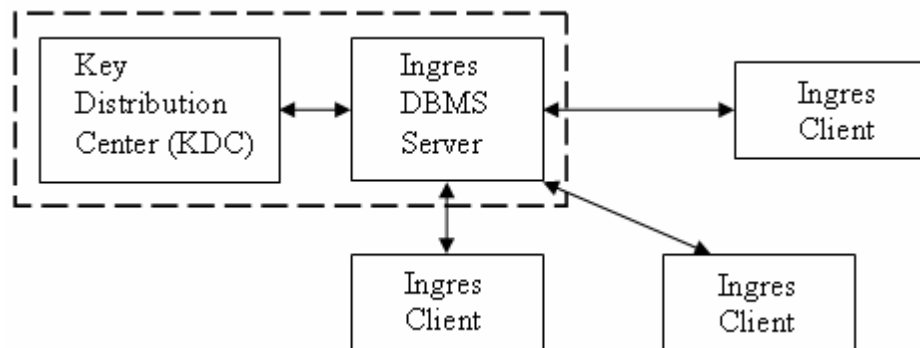
The Ingres Kerberos driver references authentication and encryption routines in the Kerberos environment, most notably, the shared library or DLL containing GSS API authentication routines.

Kerberos Configuration in the Enterprise

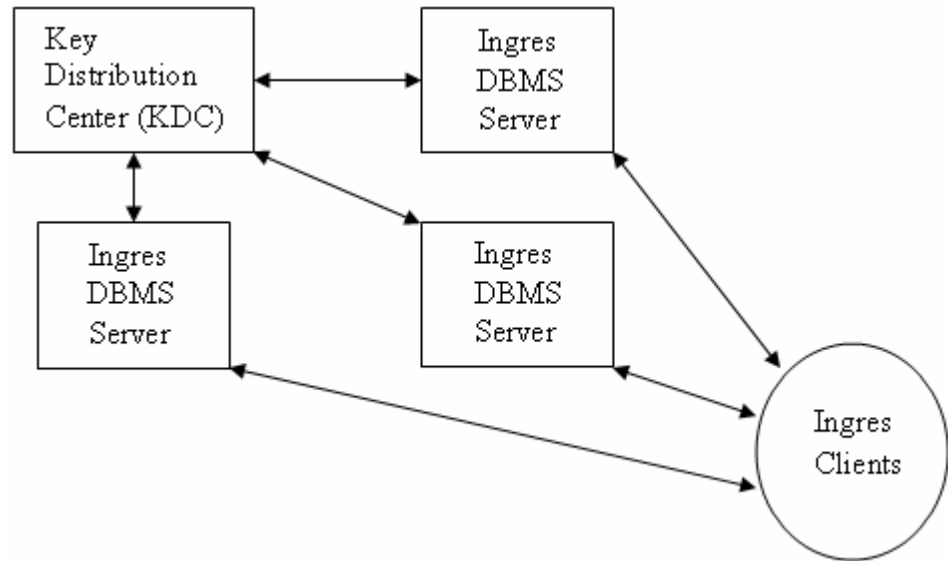
Before using Kerberos with Ingres, Kerberos should be appropriately configured in your enterprise.

A primary component of Kerberos is the Key Distribution Center (KDC). The KDC is a server process that performs the core authentication. The authentication protocol is a set of encrypted tickets that are passed from the KDC to client processes or intermediate agents known as “service principals.” For the sake of simplicity, let us assume that a single KDC will perform the Kerberos authentication.

If the enterprise contains only one Ingres DBMS Server, a possible option is to execute the KDC on the same machine as the DBMS Server:



If enough resources are available, it is desirable to install the KDC on a network node separate from the Ingres installation. In this way, security restrictions can be imposed on the Kerberos node that may not be possible if Kerberos resided on the same machine as an Ingres DBMS:



The example above demonstrates why Kerberos is sometimes referred to as “distributed authentication.” The KDC performs authentication for all Ingres nodes in the enterprise, even though the KDC itself resides on a separate network node.

Note: The above example assumes all the Ingres nodes will use Kerberos for authentication, but this is not a requirement; some nodes may continue to use Ingres or System authentication.

Kerberos Configuration Files—Configure Kerberos for Ingres

Here are examples of Kerberos configuration files. These examples assume that the KDC resides on the node foo.xyz.com and the Kerberos domain is named MYDOMAIN.XYZ.COM,

The krb5.conf file may look like this:

```
[libdefaults]
    default_realm = MYDOMAIN.XYZ.COM

[realms]
    SSF.XYZ.COM = {
        kdc = foo.xyz.com
        admin_server = foo.xyz.com
    }

[domain_realm]
    .xyz.com = MYDOMAIN.XYZ.COM
    xyz.com = MYDOMAIN.XYZ.COM

[logging]
    kdc = FILE:/var/log/krb5kdc.log
    admin_server = FILE:/var/log/kadmin.log
    default = FILE:/var/log/krb5lib.log
```

The kdc.conf file may look like this:

```
[kdcdefaults]
    kdc_ports = 88

[realms]
    MYDOMAIN.XYZ.COM = {
        kadmind_port = 749
        max_life = 12h 0m 0s
        max_renewable_life = 7d 0h 0m 0s
        master_key_type = des3-hmac-sha1
        supported_encetypes = des3-hmac-sha1:normal des-cbc-crc:normal des-cbc-
        crc:v4
    }
```

The Ingres Service Principal—Authorize Client Connections

A Kerberos principal is an entity to which credentials (validated tickets) may be assigned. Most principals of concern to Ingres are simply those that correspond to the login names of the Ingres users. For instance, for the domain MYDOMAIN.XYZ.COM, a principal representing the “ingres” user is “ingres@MYDOMAIN.XYZ.COM”.

Note: The credentials associated with the “ingres” user are valid for all “ingres” logins in the Kerberos domain, regardless of the system passwords associated with the “ingres” login name on each machine.

A KDC must define user principals for each Ingres user that exists in the enterprise, and for each Ingres service principal. An Ingres service principal does not correspond to a login user name. Instead, the Ingres service principal represents an Ingres process that performs authentication on behalf of the user.

User principals get tickets directly from the KDC through the kinit or Leash Ticket Manager or Network Identify Manager programs, but an Ingres service principal requires no such initialization. Instead, the Ingres service principal relies on the Kerberos keytab file to establish its credentials.

An Ingres service principal definition is required for each node on the Kerberos domain that has an Ingres installation. The KDC installation must define a keytab file for all Ingres service principals in order to decrypt tickets received from the KDC. A copy of the keytab file must be installed on each Ingres node in the Kerberos domain. For the best security, set ownership of the keytab file to “ingres” or the installation owner, and set read-only permissions to the keytab file.

We strongly recommend that you define the KR5_KTNAME environment variable as the full path and file name of the keytab file. On Windows, this is mandatory.

The Ingres service principal uses the standard Kerberos “primary/instance@realm” format, as follows:

```
$ingres/hostname@realm
```

hostname

Is the fully-qualified domain name of the host on which the Ingres installation is running. To find the fully-qualified host name for your machine, execute the `ifconfig` utility.

realm

Is the Kerberos administrative domain name.

In the example host name `foo.xyz.com`, the Ingres service principal would be named `"$ingres/foo.xyz.com@MYDOMAIN.COM"`.

Note: The fully-qualified host name is required when defining the Ingres service principal. Thus, the name `"$ingres.foo@MYDOMAIN.COM"` is not a valid Ingres service principal name. The `"$ingres/"` prefix is mandatory. A principal name such as `"ingres.foo@MYDOMAIN.COM"` is invalid due to the missing dollar sign (\$).

How You Configure Ingres to Use Kerberos

The process for configuring Ingres to use Kerberos is as follows:

1. Set the basic configuration for using Kerberos by doing **either** of the following:
 - Run the `iisukerberos` utility (see page 282).
 - Set parameters in Configuration-By-Forms, as described in Basic Configuration for Kerberos (see page 283).
2. Set other parameters in Configuration-By-Forms, as needed, according to your environment.
3. Obtain authorization tickets by using the `kinit` command (Windows, VMS and UNIX), the Leash Utility (Windows), or the Network Identity Manager (Windows).
4. Stop and restart Ingres.

Startup will be successful if the Kerberos GSS API library exists in your `LD_LIBRARY_PATH` definition (UNIX and Linux), if the `GSSAPI32.DLL` file resides in your system environment path (Windows), or if the file `SYS$LIBRARY:GSS$RTL32.EXE` is installed (VMS).

5. Test your server using a loopback test.

To test a loopback connection using Kerberos, the local Name Server must be configured for Kerberos authentication by using the `iisukerberos` utility or by setting the `remote_mechanism` setting in the Name Server to `kerberos` in the Configuration-By-Forms utility. In addition, your loopback vnode entry, as defined in `netutil`, must have an attribute named `authentication_mechanism` and an attribute value of `kerberos`, as described in vnode Connection Attributes (see page 285).

If you do not want to define a loopback vnode, proceed to step 7.

6. Test your connection using the Terminal Monitor, as follows:

```
sql loopback::iiddb
```

The loopback vnode should be as described in the preceding step.

7. Set up your clients. Your `netutil` definitions are almost the same as when using os-level authentication, but you should leave the login/password data blank.

iisukerberos Command—Perform Basic Kerberos Configuration

The iisukerberos utility provides a simple interface for the most commonly-used Kerberos configuration options.

This command has the following format:

UNIX, Linux, Windows:

```
iisukerberos
```

VMS:

```
@II_SYSTEM: [ingres.utility]iisukerberos.com
```

The iisukerberos utility configures Kerberos at the following levels:

- Client configuration

Client configuration enables this installation to use Kerberos to authenticate against a remote installation that has been configured to use Kerberos for authentication. This is the minimum level of Kerberos authentication.

Note: You must add the "authentication_mechanism" attribute in netutil for each remote node you wish to authenticate using Kerberos. The "authentication_mechanism" attribute must be set to "kerberos". There is no need to define users or passwords for vnodes using Kerberos authentication.

- Name Server authentication

Name Server authentication allows the local Name Server to authenticate using Kerberos.

- Server-level authentication

Server-level authentication forces all local servers, such as the DBMS Server and the Communications Server, and applications to authenticate against the Name Server using Kerberos.

You can select any combination of client, Name Server, and server-level authentications.

Ingres Configuration Options for Kerberos

To configure Ingres to use Kerberos, you must set certain parameters in Configuration-By-Forms. In addition, connection attributes may be required depending on the requirements of the enterprise.

The following system components in Configuration-By-Forms are relevant:

- Name Server
- Net Server
- Security, Configure, System
- Security, Configure, System, Mechanisms

Basic Configuration for Kerberos

When you configure Ingres to use Kerberos, you should first check the basic configuration. The basic configuration consists of the mechanisms parameter and the domain parameter in the Security component.

mechanisms Parameter—Specify Dynamic Mechanism

For Ingres to use Kerberos as a dynamic mechanism, the mechanism parameter must be set to kerberos. In Configuration-By-Forms, the mechanisms parameter is located in Security, Configure, System.

The setting should look similar to this:

| Name | Value | Units |
|------------|----------|----------------|
| mechanisms | kerberos | mechanism list |

Note: The ingres, system, or null mechanisms are invalid entries to this list, since its purpose is to specify the dynamic authentication mechanisms.

domain Parameter—Specify Domain Name

In addition to the mechanism parameter, the domain parameter is must be set to configure Ingres to use Kerberos.

In Configuration-By-Forms, the domain parameter is located in Security, Configure, System, Mechanisms, Kerberos.

The domain parameter must contain the fully qualified host name of the local installation. This name will correspond to the Ingres service principal name. For example, for machine foo.xyz.com, the value for the domain parameter should be "foo.xyz.com." If the entry reads simply "foo," edit and correct the entry.

The setting should look similar to this:

| Name | Value | Units |
|--------|-------------|----------|
| domain | foo.xyz.com | hostname |

remote_mechanism Parameter—Configure Client in a Homogeneous Kerberos Environment

The Name Server can be configured to use Kerberos for authentication for all remote targets. If so configured, connection attempts on non-Kerberos targets will fail. Use the remote_mechanism parameter for this purpose.

In Configuration-By-Forms, the remote_mechanism parameter is located in the Name Server component. Add kerberos to the mechanism list (if not already added in the Security configuration), and specify kerberos as the value on the remote_mechanism parameter.

The setting should look similar to this:

| Name | Value | Units |
|------------------|----------|-------------------------------|
| remote_mechanism | kerberos | none, default, mechansim name |

In a homogeneous Kerberos environment, it is not necessary to add login/password information for the vnode definitions in netutil. They are ignored at connect time.

vnode Connection Attributes—Configure Client in a Heterogeneous Kerberos Environment

Heterogeneous Kerberos environments are those in which both Kerberos and non-Kerberos connection targets exist in the enterprise. In such an environment, the Name Server settings in Configuration-By-Forms must remain at their default values. The local client behavior must change, depending on the connection target.

To configure the client in a heterogeneous Kerberos environment, specify connection attributes for a vnode using the netutil utility.

Here is a sample vnode configuration in netutil:

| Connection data for vnode 'newyork' | | | |
|--|--------------------------|------------|----------------|
| Type | Net Address | Protocol | Listen Address |
| Global | newyork-xp1. | wintcp | TS |
| Other attribute data for vnode 'newyork' | | | |
| Type | Attr_Name | Attr_Value | |
| Private | authentication_mechanism | kerberos | |

Note: The login/password entry for a Kerberos target should remain blank. A login/password entry is not required because the local Kerberos user principal is used for authentication, and the KDC authenticates using the ticket cache of the local user, rather than the system password on the remote connection target.

Encryption Parameters—Enable Kerberos Encryption

To specify encryption, the following options are available in Configuration-By-Forms under the Net Server (also known as Communications Server) component:

ib_encrypt_mech

Determine the encryption mechanism for inbound connections. Valid values are

kerberos

Specifies that Kerberos be used.

*

Specifies that Kerberos will be used if included as an item on the mechanism list.

ob_encrypt_mech

Determine the encryption mechanism for outbound connections. Valid values are the same as for `ib_encrypt_mech`.

ib_encrypt_mode

Determines the encryption mode for the inbound data stream. Valid values are as follows:

Off

Specifies that encryption be neither requested nor allowed.

Optional

Specifies that encryption may occur but is not requested.

On

Specifies that encryption is requested, if possible (if both ends support it).

Required

Specifies that encryption must always occur.

ob_encrypt_mode

Determines the encryption mode for the outbound data stream. Valid values are the same as for `ib_encrypt_mode`.

Outbound connection items may be configured as connection attributes in `netutil`.

The following example specifies Kerberos encryption for all inbound connections:

| Name | Value | Units |
|-----------------|----------|-----------------------------|
| ib_incrypt_mech | kerberos | *, mechanism name |
| ib_incrypt_mode | required | off, optional, on, required |

Use Kerberos for Local Authentication

Kerberos authentication can be extended to local connections.

To use Kerberos for local authentication

1. Start Configuration-By-Forms and select the Security parameters.
2. Set the user_mechanism to kerberos, as shown here:

| Name | Value | Units |
|----------------|----------|-------------------------------|
| user_mechanism | kerberos | none, default, mechanism name |

Note: The security_mechanism or server_mechanism parameters are not valid for Kerberos. Attempts to set these to kerberos will return errors.

How Name Server Delegation Works

Delegation provides an alternate method of acquiring and forwarding authentication. When delegation is configured, the Name Server generates authentication certificates as if it were the client.

This method requires Kerberos to be configured as both the local and remote authentication mechanism. The client process generates an authentication certificate for the local Name Server. The local Name Server, in turn, uses its delegation capabilities to generate an authentication certificate, and forwards the certificate on behalf of the client to the remote Name Server.

If delegation is not enabled, or Kerberos is not configured as the local authentication mechanism, then the Name Server cannot generate the remote authentication certificate. Instead, the client acquires the authentication certificate prior to making the remote connection. The client then forwards the credentials directly to the remote Name Server. Either method is valid for making secure connections through Kerberos.

Set Delegation

The process of acquiring and forwarding authentication can be delegated to the Name Server.

To set delegation

1. Start Configuration-By-Forms and select Security, Configure, Mechanisms, Kerberos.
2. Set the delegation parameter to on.

Appendix A: TCP/IP Protocol

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: wintcp or tcp_ip

Network Address: See Network Address Format (see page 290).

Listen Address: See Listen Address Format (see page 289).

UNIX

Protocol: tcp_ip

Network Address: See Network Address Format (see page 290).

Listen Address: See Listen Address Format (see page 289).

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 290).

Listen Address: See Listen Address Format (see page 289).

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 290).

Listen Address: See Listen Address Format (see page 289).

Appendix B: SNA LU0 Protocol

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 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 SNAPplus 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 SNAPplusAPI 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: DECnet Protocol

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 E: SPX/IPX Protocol

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 | 0000000000001 | 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 F: LAN Manager Protocol

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

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
                                # [frm_size - 8]
maxdata           = 1033
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/ifd0
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 = 161101130011f9f4f0f4c3f1f0f1f0f0f2f4f1f6f4

:DEFINE_ALS:
dlc_name          = XLINK000
pu_name           = S1MVS
als_name          = XXALS000
remote_addr       = 0x10
```


Appendix H: AIX SNA Services/6000 Configuration Profiles

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                             #actual local LU name
  network_name = RTIBM                                  #actual SNA network name
  lu_type = lu6.2                                       #must set to this value
  independent_lu = yes                                  #indicates independent LU
  tpn_list_name = IIGCC
  local_lu_address = 99
  sscp_id =
                                                         #ignored for independent LUs
                                                         #ignored for independent LUs
  number_of_rows = 24
  number_of_columns = 80
```


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 = 050000000001          #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        #must set to this value
  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

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 .....C5X6ICS5

Prevalidation ability? .....No
Parallel sessions? .....Yes
Conversation-level security? ...No
Uninterpreted LU name .....C5X6ICS5
```

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 appendix contains procedures for establishing and maintaining remote connections using the Net Management Utility (netu) provided in Ingres 6.4 and previous releases. The netu utility is replaced by the forms-based netutil utility; however it is still possible to establish remote connections using netu.

Start Netu

Netu is delivered in the [Ingres.SIG] directory.

To access netu, follow the instructions below for your operating system:

Windows: Run netu from the command prompt. 

UNIX: Verify that the environment variable II_SYSTEM is set to the location of your current Ingres instance and that \$II_SYSTEM/ingres/sig 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)
```

Bourne shell:

```
PATH=$PATH:$II_SYSTEM/ingres/sig 
```

VMS: Run INGSYSDEF.COM or enter the following at your operating system prompt:

```
netu:== $i_system:[ingres.sig]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 334).

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 330).

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 341).

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 56).

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 56).
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 334).

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 335).

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 334). 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 | wintcp | uk1 | II0 |
| | rome | wintcp | italy2 | II0 |
| | n_york | wintcp | 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 344).

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 342).

- 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 335).

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 345).

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 346).

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.

Index

.

- .NET data provider
 - classes • 201
 - data types • 256
 - integration with Visual Studio .NET • 264
 - sample program • 205
 - troubleshooting applications • 274
- .NET System.Data.DbType
 - data types • 256
 - mapping to EdbcType • 257
 - mapping to IngresType • 257

A

- accessing remote installations • 48
- AIX SNA Services/6000
 - installation requirements • 39
 - sample configuration file • 319
- architecture
 - .Net Data Provider • 197
 - Ingres Net • 26
- attributes, configuring vnode • 62, 90
- authentication_mechanism (vnode attribute) • 62, 90
- authorization
 - distributed databases • 49
 - of users • 47
 - remote user • 28, 48
- autocommit
 - alternative processing modes • 176
 - transactions • 176

B

- BLOB columns • 183
- Bridge Server • 129
 - described • 20
 - monitoring • 135
 - starting options • 133
 - starting using command line • 134
 - stopping with ingstop • 136
 - stopping with IVM • 136
- Bridge, starting • 132
- BYREF parameters • 182

C

- cbf (utility)
 - configuration parameters • 45, 115
- classes
 - IngresCommand • 202
 - IngresConnection • 210
 - IngresDataAdapter • 231
 - IngresDataReader • 223
 - IngresError • 234
 - IngresErrorCollection • 236
 - IngresException • 238
 - IngresInfoMessageEventArgs • 241
 - IngresInfoMessageEventHandler • 242
 - IngresParameter • 244
 - IngresParameterCollection • 249
 - IngresRowUpdatedEventArgs • 251
 - IngresRowUpdatedEventHandler • 252
 - IngresRowUpdatingEventArgs • 252
 - IngresRowUpdatingEventHandler • 253
 - IngresTransaction • 254
- client installation
 - NFS • 40
 - setup parameters • 40
- cluster
 - and Ingres Net • 19, 22
 - Name Server files • 116
 - running netutil • 50
- commands
 - used with Ingres Net • 100
- Communications Server
 - described • 20, 21
 - inbound/outbound session limits • 108
 - listen address • 124
 - starting • 103
 - stopping • 104, 328, 345, 347
- Configuration Manager • 45
- configuration parameters • 45
 - Data Access Server • 142
 - default values • 115
 - default_server_class • 98
 - described • 45
 - inbound_limit • 108
 - named and unnamed • 181
 - outbound_limit • 108
 - remote_vnode • 111

- configuring a data source
 - Windows • 150
- connect statement • 99
- connecting to a data source • 152
- connection data • 55, 57
 - described • 27
 - global and private • 29
 - listen address • 27
 - table • 55, 57
- connection data entries
 - creating • 65, 89
 - deleting • 67, 68
 - editing • 70, 71
 - valid protocol keywords • 56
- connection errors
 - local • 124
 - remote • 125
- connection pooling • 200
- connection string keywords • 152, 214
- connection_type (vnode attribute) • 62, 90
- connectivity problems • 118
- constructors
 - EdbcConnection class • 204, 213, 233, 248
 - IngresConnection class • 204, 213, 233, 248
- conventions
 - syntax • 18
- create user statement • 47, 101
- cursors
 - and result set characteristics • 178
 - and select loops • 180
 - scrollable • 178

D

- Data Access Server (DAS)
 - configuring • 142
 - described • 20
 - parameters • 142
 - starting • 112
 - stopping • 112
 - tracing support • 143
- data provider namespaces • 199
- data provider user ID options • 217
- data retrieval
 - strategies • 199
 - using DataAdapter • 199
 - using DataReader • 199
- data types
 - .NET data provider • 256

- .NET System.Data.DbType • 256
- database
 - access syntax • 96
 - procedures • 181
- DataSource tracing • 193
- DataTable
 - GetSchemaTable • 227, 262
- date/time columns • 187
- DECnet
 - installation requirements, VMS • 37
 - listen address • 301
- default_server_class (configuration parameter) • 98, 115
- defined • 14
- direct connect feature • 62, 90
- Driver Manager • 148, 193

E

- encryption • 62, 90, 286
- encryption_mechanism (vnode attribute) • 62, 90
- encryption_mode (vnode attribute) • 62, 90
- Enterprise Access products
 - with Ingres Net • 23
 - with Ingres Star • 24
- errlog.log • 109, 118
- errors
 - connection • 123
 - diagnose • 109, 118
 - local connection • 124
 - Net registration • 126
 - remote connection • 125
 - security and permission • 126
- events
 - EdbcConnection class • 213
 - EdbcDataAdapter class • 233
 - IngresConnection class • 213
 - IngresDataAdapter class • 233

G

- GCF (General Communication Facility)
 - Bridge Server • 20
 - Communications Server • 20
 - Data Access Server • 20
 - described • 20
 - General Communications Area (GCA) • 20
 - Name Server • 20
- GetSchemaTable
 - DataTable • 227, 262

global registration types • 29
Global Temporary Table parameters (JDBC) • 182

H

HP-UX SNAPplus
 installation requirements • 38
 sample configuration file • 323

I

II_GCA_LOG • 110
iigcb
 described • 20
 starting • 133
iigcc
 described • 20, 21
 inbound/outbound session limits • 108
 listen address • 124
 starting • 103
 stopping • 104, 328
iigcd
 described • 20
 starting • 112
 stopping • 112
iigcn
 database files • 116
 described • 20
 Name Server database • 116
 starting • 133
iinamu (utility) • 124
inbound/outbound sessions
 changing limits • 108
inbound_limit (configuration parameter) • 108, 115
Ingres
 basic components • 15
 installation, described • 16
 server classes • 98
 tools • 15
Ingres Bridge
 Bridge Server component • 129
 diagnosing problems • 137
 installation configurations • 130
 installing • 132
 overview • 129
 sample configuration • 131
 sample configuration setup files • 138
 security • 129
 setting up client • 134

tracing a connection • 137
vs. Ingres Star • 130
Ingres Net
 accessing remote databases • 95
 architecture • 26, 32
 benefits • 25
 commands available • 100
 configuration parameters • 45
 described • 19
 diagnosing problems • 118
 Enterprise Access products • 23
 establishing connections • 113
 Ingres Star and • 23
 NET_ADMIN privilege • 32
 permission errors • 127
 resolving connection errors • 123
 sample configuration • 22
 security • 21
 security errors • 127
 SERVER_CONTROL privilege • 32
 setup parameters • 39
 user roles • 31
 using the connect statement • 99
Ingres Star
 with Ingres Enterprise Access products • 24
 with Ingres Net • 23
IngresCommand class • 202
IngresConnection class • 210
IngresDataAdapter class • 231
IngresDataReader class • 223
IngresError class • 234
IngresErrorCollection class • 236
IngresException class • 238
IngresInfoMessageEventArgs class • 241
IngresInfoMessageEventHandler class • 242
ingstart -iigcb • 134
ingstart -iigcc • 103
ingstart -iigcd • 112
ingstart -iigcn • 133
ingstop -iigcc • 104, 328
ingstop -iigcd • 112
ingvalidpw (executable) • 44, 127
installation
 client • 16
 server • 16
 types of • 23
installation code
 format • 40
Installation Password

- defining • 91
- defining during installation • 39
- defining on local installation • 72, 80
- in contrast to login account password • 28
- installing Ingres Bridge • 132
- installing Ingres Net
 - for existing installations • 41
 - installation components • 35
 - setup parameters • 39
- IPv6 • 290
- ISO • 21

J

- JDBC
 - and Data Access Server • 141
 - connectivity components • 163
 - implementation considerations • 174
 - JDBC Information Utility • 164
 - tracing • 193
 - unsupported features • 165
- JDBC Driver
 - accessing • 174
 - and cursor pre-fetch capabilities • 180
 - and updateable cursors • 178
 - BLOB columns • 183
 - class files • 166
 - cursors and select loops • 180
 - data source properties • 170
 - database procedures • 181
 - date/time columns • 187
 - driver properties and attributes • 167
 - National Character Set values • 189
 - support for parameter default values • 181
 - supported features • 163

K

- Kerberos • 275

L

- LAN Manager
 - listen address • 307
- listen address
 - configuration parameter • 115
 - DECnet • 301
 - defined • 27
 - LAN Manager • 307
 - SNA LU0 • 293
 - SNA LU62 • 295
 - SPX/IPX • 303

- TCP/IP • 289
- LOB locator • 183
- local_vnode (configuration parameter) • 115
- log_level (configuration parameter) • 115
- logging
 - directing to a file • 110
 - levels • 109
- Login/password data table
 - described • 53
 - Installation Password • 54
 - login account password • 54

M

- mapping
 - .NET System.Data.DbType to EdbcType • 257
 - .NET System.Data.DbType to IngresType • 257
- methods
 - EdbcCommand class • 203
 - EdbcConnection class • 212
 - EdbcDataAdapter class • 233
 - EdbcDataReader class • 226
 - EdbcError class • 236
 - EdbcErrorCollection class • 238
 - EdbcException class • 240
 - EdbcParameter class • 247
 - EdbcParameterCollection class • 250
 - EdbcTransaction class • 255
 - IngresCommand class • 203
 - IngresConnection class • 212
 - IngresDataAdapter class • 233
 - IngresDataReader class • 226
 - IngresError class • 236
 - IngresErrorCollection class • 238
 - IngresException class • 240
 - IngresParameter class • 247
 - IngresParameterCollection class • 250
 - IngresTransaction class • 255
- mkvalidpw • 44, 127
- MultiNet TCP/IP
 - installation requirements, VMS • 37

N

- Name Server
 - database files • 116
 - described • 20
 - IICOMSVR_nodename • 116
 - IIINGRES_nodename • 116

- IILOGIN_nodename • 116
- iiname.all • 116
- IINODE_nodename • 116
- IISTAR_nodename • 116
- namespace
 - data provider • 199
- National Character Set values • 189
- NET_ADMIN (privilege) • 32
- netu (utility)
 - aborting • 330
 - adding remote node definitions • 331
 - adding vs. merging remote node definitions • 331
 - changing remote node definitions • 334
 - changing remote user authorizations • 341
 - defining remote user authorizations • 339
 - deleting remote node definitions • 333
 - deleting remote user authorizations • 340
 - described • 51
 - displayed remote node definitions • 337
 - displayed remote user authorizations • 345
 - merging remote node definitions • 331
 - retrieving remote node definitions • 336
 - retrieving remote user authorizations • 344
 - stopping Communications Server (iigcc) • 345
 - user interface • 328
- netutil (non-interactive mode)
 - command line flags • 74
 - creating a connection data entry • 81
 - creating a remote user authorization • 76
 - defining local Installation Password • 80
 - deleting a connection data entry • 82
 - deleting a remote user authorization • 78
 - displaying connection data entries • 84
 - displaying remote user authorizations • 79
 - functions available • 73
 - input control file • 74
 - invariant fields • 75
 - quiescing the Communications Server • 86
 - stopping the Communications Server • 86
 - wildcards • 76
- netutil (utility)
 - accessing • 50
 - adding vnode attributes • 62
 - clusters and • 50
 - Connection data table • 55, 57
 - creating a connection data entry • 65, 134
 - creating a remote user authorization • 66

- defining a local Installation Password • 72
- defining connection data entries • 51
- defining remote user authorizations • 51
- deleting entries • 66
- editing entries • 68
- establishing and testing a connection • 60
- global and private registration types • 29
- list of user tasks • 58
- Login/password data table • 53
- non-interactive mode • 73
- startup screen • 51
- stopping the Communications Server • 106
- tables • 51
- virtual node name (vnode) table • 52

- network
 - installing and testing • 35
 - protocol types • 56
 - terms and concepts • 14
- network address • 55, 57
- network protocol • 56
- Network Utility
 - accessing • 50
 - adding vnode attributes • 90
 - altering vnodes • 87
 - creating a private remote user authorization • 90
 - creating additional connection data entries • 89
 - creating vnodes • 87
 - deleting vnodes • 87
 - list of user tasks • 87

O

- ODBC Data Source Administrator • 150
- ODBC driver
 - configuring a data source (Windows) • 150
 - described • 145
 - read-only option • 147
 - requirements • 147
- ODBC Driver Manager • 148
- ODBC implementation considerations • 154
- Open Systems Interconnect • 21
- OSI • 26, 32
- outbound_limit (configuration parameter) • 108, 115

P

- parameters
 - BYREF • 182

- Global Temporary Table (JDBC) • 182
- passwords
 - defining a local Installation Password • 72, 80
- private registration types • 29
- privileges, user • 101
- procedures, executing • 182
- properties
 - EdbcCommand class • 203
 - EdbcConnection class • 211
 - EdbcDataAdapter class • 231
 - EdbcDataReader class • 225
 - EdbcError class • 235
 - EdbcErrorCollection class • 237
 - EdbcException class • 239, 242
 - EdbcParameter class • 246
 - EdbcParameterCollection class • 250, 251, 253
 - EdbcTransaction class • 255
 - IngresCommand class • 203
 - IngresConnection class • 211
 - IngresDataAdapter class • 231
 - IngresDataReader class • 225
 - IngresError class • 235
 - IngresErrorCollection class • 237
 - IngresException class • 239, 242
 - IngresParameter class • 246
 - IngresParameterCollection class • 250, 251, 253
 - IngresTransaction class • 255
- protocol
 - configuration parameter • 115
 - keywords • 56

Q

- query builder • 270

R

- RDBMS (Relational Database Management System) • 15
- read-only ODBC driver • 147
- region and time zone, defined • 40
- remote databases
 - command syntax for accessing • 96
- remote installation
 - access, requirements • 48, 49
- remote user authorization
 - creating • 66
 - creating private • 90

- defining during installation • 39
- deleting • 67
- described • 28
- editing • 69
- global and private • 29
- IILOGIN_nodename • 116
- Installation Password and login account password • 28
- storage of • 116
- remote_vnode (configuration parameter) • 111, 115

S

- security
 - .NET Data Provider • 201
 - Ingres Net • 21
 - resolving problems • 127
- select loops and cursors • 180
- server class keywords • 98
- server classes • 93
 - default • 98
- server connections, establishing • 113
- server installation
 - setup parameters for a • 40
- SERVER_CONTROL (privilege) • 32
- servers
 - Bridge • 129
 - Communications • 103
 - Data Access • 141
 - DBMS • 15
 - Name • 20
 - tasks related to • 93
- set host command • 37
- SNA LU0
 - listen address • 293
- SNA LU62
 - listen address • 295
 - SunLink Gateway configuration files • 309
- SPX/IPX
 - installation requirements • 36
 - listen address • 303
- SQL statement
 - connect • 99
- Star Server
 - IISTAR_nodename • 116
- SunLink Gateway
 - sample configuration file • 309
- SunLink SNA Peer-to-Peer

-
- installation requirements, Solaris and Sun-4
 - 38

T

TCP/IP

- installation requirements, Windows • 36
- listen address • 289
- time zone and region, defined • 40
- toolbar, virtual nodes • 87
- tools
 - for managing Net • 30
 - Ingres • 15
- tracing
 - Data Access Server • 143
 - Driver Manager • 193
 - enabling using Ingres JDBC Driver methods
 - 193
 - Ingres Bridge connection • 137
 - JDBC DataSource • 193
 - levels • 144
 - trace IDs • 193
- transaction mode
 - autocommit • 176
- troubleshooting
 - connectivity problems • 118
 - remote connection failure • 123
 - security and permission errors • 126
 - UNIX installation checklist • 120
 - VMS installation checklist • 122
 - Windows installation checklist • 119

U

- u command flag • 101
- unixODBC Driver Manager • 148

V

- vcbf (utility)
 - configuration parameters • 45
- virtual node name (vnode)
 - described • 26
 - table • 52
- virtual nodes toolbar
 - in Network Utility and VDBA • 87
- Visual DBA
 - accessing • 50
 - adding vnode attributes • 90
 - altering vnodes • 87
 - creating a private remote user authorization
 - 90

- creating additional connection data entries • 89

- creating vnodes • 87
- deleting vnodes • 87
- list of user tasks • 87

vnodes

- adding attributes for • 62, 90
- advanced, defined • 88
- altering using Network Utility • 87
- altering using VDBA • 87
- creating using netutil • 60
- creating using Network Utility • 87
- creating using VDBA • 87
- deleting using netutil • 67
- deleting using Network Utility • 87
- deleting using VDBA • 87
- disconnecting from • 92
- editing using netutil • 69
- naming rules • 52
- opening a utility window • 92
- private vs. global • 88
- refreshing • 91
- setting a default • 111
- simple, defined • 88
- testing a connection • 91
- tools for defining • 50