Third party acknowledgements — See the “Third party acknowledgements” section on page Preface–9.
# Contents

<table>
<thead>
<tr>
<th>Preface</th>
<th>Preface–1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
<td>1–1</td>
</tr>
<tr>
<td>DataServer for ODBC overview</td>
<td>1–2</td>
</tr>
<tr>
<td>DataServer components</td>
<td>1–3</td>
</tr>
<tr>
<td>DataServer for ODBC logic</td>
<td>1–5</td>
</tr>
<tr>
<td>The schema holder</td>
<td>1–6</td>
</tr>
<tr>
<td>Security</td>
<td>1–7</td>
</tr>
<tr>
<td>DataServer utilities</td>
<td>1–8</td>
</tr>
<tr>
<td>DataServer demonstration databases</td>
<td>1–9</td>
</tr>
<tr>
<td>DataServer configurations</td>
<td>1–10</td>
</tr>
<tr>
<td>The local DataServer configuration</td>
<td>1–10</td>
</tr>
<tr>
<td>The remote DataServer configuration</td>
<td>1–11</td>
</tr>
<tr>
<td>Configuring distributed DataServer applications using ProBroker</td>
<td>1–12</td>
</tr>
<tr>
<td>Configuring distributed DataServer applications using the Unified Broker Framework</td>
<td>1–13</td>
</tr>
<tr>
<td>Software requirements</td>
<td>1–15</td>
</tr>
<tr>
<td>Guidelines for using the DataServer</td>
<td>1–16</td>
</tr>
<tr>
<td>Using the DataServer for ODBC for the first time</td>
<td>1–16</td>
</tr>
<tr>
<td>Migrating an OpenEdge database to an ODBC data source</td>
<td>1–17</td>
</tr>
<tr>
<td>Upgrading to the Release 10 DataServer for ODBC</td>
<td>1–17</td>
</tr>
<tr>
<td>Documentation resources</td>
<td>1–18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2. Programming Considerations</th>
<th>2–1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database design issues</td>
<td>2–3</td>
</tr>
<tr>
<td>OpenEdge database objects and ODBC data-source objects</td>
<td>2–3</td>
</tr>
<tr>
<td>Naming conventions</td>
<td>2–4</td>
</tr>
<tr>
<td>ODBC data source limits</td>
<td>2–5</td>
</tr>
<tr>
<td>Code pages</td>
<td>2–5</td>
</tr>
<tr>
<td>Indexes and sorting</td>
<td>2–7</td>
</tr>
<tr>
<td>Case sensitivity</td>
<td>2–9</td>
</tr>
<tr>
<td>Interaction of code page, collation, and case sensitivity</td>
<td>2–10</td>
</tr>
<tr>
<td>ODBC data-source views</td>
<td>2–10</td>
</tr>
<tr>
<td>Triggers</td>
<td>2–11</td>
</tr>
<tr>
<td>OpenEdge sequences</td>
<td>2–12</td>
</tr>
<tr>
<td>Contents</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Configuring the remote DataServer</td>
<td>3-6</td>
</tr>
<tr>
<td>Configuring with Progress Explorer</td>
<td>3-6</td>
</tr>
<tr>
<td>Configuring from the command line</td>
<td>3-7</td>
</tr>
<tr>
<td>Creating a schema holder</td>
<td>3-14</td>
</tr>
<tr>
<td>Before you begin</td>
<td>3-14</td>
</tr>
<tr>
<td>Establishing permissions</td>
<td>3-14</td>
</tr>
<tr>
<td>Preparing to create the schema holder</td>
<td>3-14</td>
</tr>
<tr>
<td>Creating an empty OpenEdge database</td>
<td>3-16</td>
</tr>
<tr>
<td>Creating a schema holder</td>
<td>3-17</td>
</tr>
<tr>
<td>Maintaining a schema holder</td>
<td>3-21</td>
</tr>
<tr>
<td>Updating a deployed schema holder</td>
<td>3-21</td>
</tr>
<tr>
<td>Typical configuration for a remote client to connect to a remote DataServer</td>
<td>3-22</td>
</tr>
<tr>
<td>Configuring an ODBC data source</td>
<td>3-22</td>
</tr>
<tr>
<td>Starting a broker</td>
<td>3-22</td>
</tr>
<tr>
<td>Starting the schema holder to accept remote client connections</td>
<td>3-23</td>
</tr>
<tr>
<td>Connecting remote clients to an ODBC DataServer broker</td>
<td>3-23</td>
</tr>
</tbody>
</table>

4. Connecting the DataServer                                4-1
   Starting a local DataServer                               4-2
   Starting a remote DataServer                              4-3
      Starting and stopping a broker process from
      the Progress Explorer and connecting a client          4-3
      Starting and stopping a broker process using ODBMAN    4-5
      Starting and stopping a broker process from the command line 4-6
      Starting the Windows client process                    4-6
      Starting the UNIX client process                        4-7
   Connection guidelines                                    4-8
      Connections and security in Windows                    4-8
   Connecting a schema holder                                4-9
      Connecting a schema holder at startup                  4-9
      Unsupported connection parameters                      4-14
      Special connection issues                             4-15
      Query tuning with connection and startup parameters     4-16
      Binding with connection and startup parameters         4-18
      Analyzing performance                                  4-19
      Local schema caching                                    4-20
   Connection failures and OpenEdge responses                4-21
      Connection troubleshooting                             4-21
      Managing connections to an ODBC data source            4-23
      Accessing the DataServer log                           4-24

5. The DataServer Tutorial                                   5-1
   Demonstration databases for ODBC DataServers              5-2
   Preparing to create demonstration databases              5-3
   DataServer utilities                                      5-7
      Creating a schema holder                               5-8
      Updating a schema holder                               5-12
      Verifying a schema holder                              5-14
      Changing connection information in a schema holder      5-18
      Changing a code page in a schema holder                5-19
      Deleting a schema holder                               5-20
   Migrating an OpenEdge database to an ODBC data source     5-21
      Preparing a data source for the utility                5-21
      Running the OpenEdge DB-to-ODBC utility                 5-21
      Running the utility interactively                      5-22
   Using the Adjust Schema utility                           5-28
Contents

Modifying a schema holder .................................................. 5–29
  Modifying table-level information ................................... 5–29
  Modifying field-level information ................................. 5–30
  Defining the ROWID .......................................................... 5–31
  Modifying tables to support ROWID function for Sybase .... 5–33
Adding extended ABL support .......................................... 5–35
  Modifying tables to support arrays ............................... 5–35
  Modifying tables to support case-insensitive indexes ....... 5–36

6. Troubleshooting .............................................................. 6–1
  Tuning your environment with the -Dsrv startup parameter .... 6–2
    ODBC options ............................................................... 6–2
    DataServer options ......................................................... 6–4
    Using ODBC and DataServer options ............................... 6–6
  ODBC driver problems in Windows platforms ................. 6–11

A. Upgrading DataServer Applications ................................. A–1
  Upgrading schema holders ................................................. A–2
  Upgrading from one data source version to another ............ A–3

B. Stored Procedure Reference ........................................... B–1
  CLOSE STORED-PROCEDURE statement ............................... B–2
  PROC-HANDLE function ..................................................... B–3
  PROC-STATUS function ..................................................... B–4
  RUN STORED-PROCEDURE statement .................................. B–5

C. Server-Related Command Line Utilities and Startup Parameters .... C–1
  Progress Explorer command line utilities for the DataServer .... C–2
  NSCONFIG utility ............................................................ C–3
  NSMAN utility ............................................................... C–5
  ODBCConfig utility ........................................................ C–7
  ODBMAN utility ............................................................. C–9
  PROBKR command .......................................................... C–11
  DataServer startup parameters ....................................... C–12

D. ODBC Data Source Data Types ........................................... D–1

Index .................................................................................. Index–1
<table>
<thead>
<tr>
<th>Figures</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1–1:</td>
<td>Architecture for DataServer for ODBC</td>
<td>1–4</td>
</tr>
<tr>
<td>Figure 1–2:</td>
<td>DataServer logic</td>
<td>1–5</td>
</tr>
<tr>
<td>Figure 1–3:</td>
<td>The schema-loading process</td>
<td>1–6</td>
</tr>
<tr>
<td>Figure 1–4:</td>
<td>The local DataServer for ODBC</td>
<td>1–10</td>
</tr>
<tr>
<td>Figure 1–5:</td>
<td>The remote DataServer for ODBC</td>
<td>1–11</td>
</tr>
<tr>
<td>Figure 1–6:</td>
<td>The distributed DataServer for ODBC using ProBroker</td>
<td>1–12</td>
</tr>
<tr>
<td>Figure 2–1:</td>
<td>DataServer processes and code pages</td>
<td>2–5</td>
</tr>
<tr>
<td>Procedures</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------</td>
<td></td>
</tr>
<tr>
<td>find.p</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2–37</td>
</tr>
</tbody>
</table>
This Preface contains the following sections:

- Purpose
- Audience
- Organization
- Using this manual
- Typographical conventions
- Examples of syntax descriptions
- OpenEdge messages
- OpenEdge messages
- Third party acknowledgements
Purpose

This manual explains how to use the OpenEdge™ DataServer for ODBC. It provides startup instructions and a brief tutorial that introduces the utilities that support the DataServer. Additionally, it discusses database design and programming issues to consider when creating applications that access the OpenEdge and ODBC database management systems.

Audience

This book is intended for programmers who want to develop OpenEdge applications with OpenEdge ABL and that run with ODBC-compliant databases. It assumes a fundamental knowledge of both OpenEdge and ODBC.

Organization

Chapter 1, “Introduction”

Describes the DataServer architecture (components and configurations) and software requirements, and provides guidelines for using the DataServer.

Chapter 2, “Programming Considerations”

Discusses the differences between OpenEdge databases and supported ODBC data sources and the DataServer strategies for resolving these differences. Includes database design issues, application issues, OpenEdge ABL issues, stored procedures, and DataServer performance enhancement.

Chapter 3, “Configuring the DataServer”

Presents instructions for configuring the DataServer and for creating, maintaining, and deploying a schema holder.

Chapter 4, “Connecting the DataServer”

Presents various methods for starting and for connecting the DataServer and describes connecting a schema holder. In addition, it provides connection guidelines and lists connection failures and OpenEdge responses.

Chapter 5, “The DataServer Tutorial”

Provides an overview of the ODBC demonstration databases and the opportunity to work with the DataServer utilities for maintaining a schema holder. In addition, it describes the OpenEdge-to-ODBC migration utility.

Chapter 6, “Troubleshooting”

Describes common issues and how to work around them, including tuning your environment and resolving ODBC driver problems.
Appendix A, “Upgrading DataServer Applications”

Provides information on how to upgrade from an earlier version of either the DataServer or an ODBC data source.

Appendix B, “Stored Procedure Reference”

Describes the OpenEdge ABL statements and functions that support running ODBC data-source stored procedures.

Appendix C, “Server-Related Command Line Utilities and Startup Parameters”

Describes the utilities you use to configure, manage, start, and stop the DataServer host and client.

Appendix D, “ODBC Data Source Data Types”

Provides details on DB2 UDB, SQL-ODBC and OpenEdge data types.

Using this manual

OpenEdge provides a special purpose programming language for building business applications. In the documentation, the formal name for this language is ABL (Advanced Business Language). With few exceptions, all keywords of the language appear in all UPPERCASE, using a font that is appropriate to the context. All other alphabetic language content appears in mixed case.

For the latest documentation updates see the OpenEdge Product Documentation Overview page on PSDN: http://communities.progress.com/pcom/docs/DOC-16074.

References to ABL compiler and run-time features

ABL is both a compiled and an interpreted language that executes in a run-time engine. The documentation refers to this run-time engine as the ABL Virtual Machine (AVM). When the documentation refers to ABL source code compilation, it specifies ABL or the compiler as the actor that manages compile-time features of the language. When the documentation refers to run-time behavior in an executing ABL program, it specifies the AVM as the actor that manages the specified run-time behavior in the program.

For example, these sentences refer to the ABL compiler’s allowance for parameter passing and the AVM’s possible response to that parameter passing at run time: “ABL allows you to pass a dynamic temp-table handle as a static temp-table parameter of a method. However, if at run time the passed dynamic temp-table schema does not match the schema of the static temp-table parameter, the AVM raises an error.” The following sentence refers to run-time actions that the AVM can perform using a particular ABL feature: “The ABL socket object handle allows the AVM to connect with other ABL and non-ABL sessions using TCP/IP sockets.”
References to ABL data types

ABL provides built-in data types, built-in class data types, and user-defined class data types. References to built-in data types follow these rules:

- Like most other keywords, references to specific built-in data types appear in all UPPERCASE, using a font that is appropriate to the context. No uppercase reference ever includes or implies any data type other than itself.

- Wherever integer appears, this is a reference to the INTEGER or INT64 data type.

- Wherever character appears, this is a reference to the CHARACTER, LONGCHAR, or CLOB data type.

- Wherever decimal appears, this is a reference to the DECIMAL data type.

- Wherever numeric appears, this is a reference to the INTEGER, INT64, or DECIMAL data type.

References to built-in class data types appear in mixed case with initial caps, for example, Progress.Lang.Object. References to user-defined class data types appear in mixed case, as specified for a given application example.

Typographical conventions

This manual uses the following typographical conventions:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bold</td>
<td>Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements.</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Italic typeface indicates the title of a document, or signifies new terms.</td>
</tr>
<tr>
<td>SMALL, BOLD CAPITAL LETTERS</td>
<td>Small, bold capital letters indicate OpenEdge key functions and generic keyboard keys; for example, GET and CTRL.</td>
</tr>
<tr>
<td>KEY1+KEY2</td>
<td>A plus sign between key names indicates a simultaneous key sequence: you press and hold down the first key while pressing the second key. For example, CTRL+X.</td>
</tr>
<tr>
<td>KEY1 KEY2</td>
<td>A space between key names indicates a sequential key sequence: you press and release the first key, then press another key. For example, ESCAPE H.</td>
</tr>
<tr>
<td><strong>Syntax:</strong></td>
<td></td>
</tr>
<tr>
<td>Fixed width</td>
<td>A fixed-width font is used in syntax statements, code examples, system output, and filenames.</td>
</tr>
<tr>
<td><strong>Fixed-width italics</strong></td>
<td>Fixed-width italics indicate variables in syntax statements.</td>
</tr>
<tr>
<td><strong>Fixed-width bold</strong></td>
<td>Fixed-width bold indicates variables with special emphasis.</td>
</tr>
<tr>
<td>Convention</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>UPPERCASE fixed width</td>
<td>Uppercase words are ABL keywords. Although these are always shown in uppercase, you can type them in either uppercase or lowercase in a procedure.</td>
</tr>
<tr>
<td>![three arrows]</td>
<td>This icon (three arrows) introduces a multi-step procedure.</td>
</tr>
<tr>
<td>![one arrow]</td>
<td>This icon (one arrow) introduces a single-step procedure.</td>
</tr>
<tr>
<td>Period (.) or colon (:)</td>
<td>All statements except DO, FOR, FUNCTION, PROCEDURE, and REPEAT end with a period. DO, FOR, FUNCTION, PROCEDURE, and REPEAT statements can end with either a period or a colon.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Large brackets indicate the items within them are optional.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Small brackets are part of ABL.</td>
</tr>
<tr>
<td>{ }</td>
<td>Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams.</td>
</tr>
<tr>
<td>{ }</td>
<td>Small braces are part of ABL. For example, a called external procedure must use braces when referencing arguments passed by a calling procedure.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Ellipses indicate repetition: you can choose one or more of the preceding items.</td>
</tr>
</tbody>
</table>
Examples of syntax descriptions

In this example, ACCUM is a keyword, and aggregate and expression are variables:

Syntax

\[
\text{ACCUM aggregate expression}
\]

FOR is one of the statements that can end with either a period or a colon, as in this example:

Syntax

\[
\text{FOR EACH Customer:}
\text{DISPLAY Name.}
\text{END.}
\]

In this example, STREAM stream, UNLESS-HIDDEN, and NO-ERROR are optional:

Syntax

\[
\text{DISPLAY [ STREAM stream ] [ UNLESS-HIDDEN ] [ NO-ERROR ]}
\]

In this example, the outer (small) brackets are part of the language, and the inner (large) brackets denote an optional item:

Syntax

\[
\text{INITIAL [ constant [ , constant ] ]}
\]

A called external procedure must use braces when referencing compile-time arguments passed by a calling procedure, as shown in this example:

Syntax

\[
\{ \ &\text{argument-name} \}
\]

In this example, EACH, FIRST, and LAST are optional, but you can choose only one of them:

Syntax

\[
\text{PRESELECT [ EACH | FIRST | LAST ] record-phrase}
\]

In this example, you must include two expressions, and optionally you can include more. Multiple expressions are separated by commas:

Syntax

\[
\text{MAXIMUM ( expression , expression [ , expression ] ... )}
\]

In this example, you must specify MESSAGE and at least one expression or SKIP \([ n ]\), and any number of additional expression or SKIP \([ n ]\) is allowed:
Syntax

\[
\text{MESSAGE \{ } \text{expression} | \text{SKIP \[ (n) \]} \text{ \} \ldots}
\]

In this example, you must specify \{include-file, then optionally any number of argument or \&argument-name = “argument-value”, and then terminate with \}: 

Syntax

\[
\text{\{ } \text{include-File} \\
\quad \text{[ } \text{argument} | \&\text{argument-name} = “argument-value” \text{ ] \ldots } \text{\}}
\]

Long syntax descriptions split across lines

Some syntax descriptions are too long to fit on one line. When syntax descriptions are split across multiple lines, groups of optional and groups of required items are kept together in the required order.

In this example, WITH is followed by six optional items:

Syntax

\[
\text{WITH \[ } \text{ACCUM max-length } | \text{ expression DOWN } \\
\quad \text{[ CENTERED ] [ n COLUMNS ] [ SIDE-LABELS ]} \\
\quad \text{[ STREAM-IO ]}
\]

Complex syntax descriptions with both required and optional elements

Some syntax descriptions are too complex to distinguish required and optional elements by bracketing only the optional elements. For such syntax, the descriptions include both braces (for required elements) and brackets (for optional elements).

In this example, ASSIGN requires either one or more field entries or one record. Options available with field or record are grouped with braces and brackets:

Syntax

\[
\text{ASSIGN \{ } \text{FRAME frame \{ field [ = expression ] \}} \\
\quad \text{[ WHEN expression ] \} \ldots} \\
\quad | \text{ \{ record [ EXCEPT field \ldots ] \} }
\]
OpenEdge messages

OpenEdge displays several types of messages to inform you of routine and unusual occurrences:

- **Execution messages** inform you of errors encountered while OpenEdge is running a procedure; for example, if OpenEdge cannot find a record with a specified index field value.

- **Compile messages** inform you of errors found while OpenEdge is reading and analyzing a procedure before running it; for example, if a procedure references a table name that is not defined in the database.

- **Startup messages** inform you of unusual conditions detected while OpenEdge is getting ready to execute; for example, if you entered an invalid startup parameter.

After displaying a message, OpenEdge proceeds in one of several ways:

- Continues execution, subject to the error-processing actions that you specify or that are assumed as part of the procedure. This is the most common action taken after execution messages.

- Returns to the Procedure Editor, so you can correct an error in a procedure. This is the usual action taken after compiler messages.

- Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.

- Terminates the current session.

OpenEdge messages end with a message number in parentheses. In this example, the message number is 200:

```
** Unknown table name table. (200)
```

If you encounter an error that terminates OpenEdge, note the message number before restarting.

**Obtaining more information about OpenEdge messages**

In Windows platforms, use OpenEdge online help to obtain more information about OpenEdge messages. Many OpenEdge tools include the following Help menu options to provide information about messages:

- Choose **Help→Recent Messages** to display detailed descriptions of the most recent OpenEdge message and all other messages returned in the current session.

- Choose **Help→Messages** and then type the message number to display a description of a specific OpenEdge message.

- In the Procedure Editor, press the **HELP** key or **F1**.
Third party acknowledgements

OpenEdge includes AdventNet - Agent Toolkit licensed from AdventNet, Inc. http://www.adventnet.com. All rights to such copyright material rest with AdventNet.

OpenEdge includes ANTLR (Another Tool for Language Recognition) software Copyright © 2003-2006, Terence Parr All rights reserved. Neither the name of the author nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes software developed by the Apache Software Foundation (http://www.apache.org/). Copyright © 1999 The Apache Software Foundation. All rights reserved (Xerces C++ Parser (XML) and Xerces2 Java Parser (XML)); Copyright © 1999-2002 The Apache Software Foundation. All rights reserved (Xerces Parser (XML)); and Copyright © 2000-2003 The Apache Software Foundation. All rights reserved (Ant). The names “Apache,” “Xerces,” “ANT,” and “Apache Software Foundation” must not be used to endorse or promote products derived from this software without prior written permission. Products derived from this software may not be called “Apache”, nor may “Apache” appear in their name, without prior written permission of the Apache Software Foundation. For written permission, please contact apache@apache.org. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes Concurrent Java software Copyright 1994-2000 Sun Microsystems, Inc. All Rights Reserved. Neither the name of or trademarks of Sun may be used to endorse or promote products including or derived from the Java Software technology without specific prior written permission; and Redistributions of source or binary code must contain the above copyright notice, this notice and the following disclaimers: This software is provided "AS IS," without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC. AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN MICROSYSTEMS, INC. OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN MICROSYSTEMS, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

OpenEdge includes DataDirect software Copyright © 1991-2007 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect for JDBC Type 4 driver); Copyright © 1993-2009 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect for JDBC); Copyright © 1988-2007 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect for ODBC); and Copyright © 1988-2007 Progress Software
Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect64 for ODBC).

OpenEdge includes DataDirect Connect for ODBC and DataDirect Connect64 for ODBC software, which include ICU software 1.8 and later - Copyright © 1995-2003 International Business Machines Corporation and others All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

OpenEdge includes DataDirect Connect for ODBC and DataDirect Connect64 for ODBC software, which include software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/). Copyright © 1998-2006 The OpenSSL Project. All rights reserved. And Copyright © 1995-1998 Eric Young (eay@cryptsoft.com). All rights reserved.

OpenEdge includes DataDirect products for the Microsoft SQL Server database which contain a licensed implementation of the Microsoft TDS Protocol.

OpenEdge includes software authored by David M. Gay. Copyright © 1991, 2000, 2001 by Lucent Technologies (dtoa.c); Copyright © 1991, 1996 by Lucent Technologies (g_fmt.c); and Copyright © 1991 by Lucent Technologies (rnd_prods). Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software. THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

OpenEdge includes software authored by David M. Gay. Copyright © 1998-2001 by Lucent Technologies All Rights Reserved (decsrtod.c; strtod.g); Copyright © 1998, 2000 by Lucent Technologies All Rights Reserved (decsrtod.c; strtod.g); Copyright © 1998 by Lucent Technologies All Rights Reserved (dtnmisc.c; gdtoa.h; gethex.c; gmisc.c; sum.c); Copyright © 1998, 1999 by Lucent Technologies All Rights Reserved (gdtoa.c; misc.c; smisc.c; ulp.c); Copyright © 1998-2000 by Lucent Technologies All Rights Reserved (gdtoaimp.h); Copyright © 2000 by Lucent Technologies All Rights Reserved (hd_init.c). Full copies of these licenses can be found in the installation directory, in the c:/OpenEdge/licenses folder. Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
OpenEdge includes http package software developed by the World Wide Web Consortium. 
Copyright © 1994-2002 World Wide Web Consortium, (Massachusetts Institute of 
Technology, European Research Consortium for Informatics and Mathematics, Keio 
University). All rights reserved. This work is distributed under the W3C® Software License 
[http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231] in the hope 
that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty 
of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

OpenEdge includes ICU software 1.8 and later - Copyright © 1995-2003 International Business 
Machines Corporation and others All rights reserved. Permission is hereby granted, free of 
charge, to any person obtaining a copy of this software and associated documentation files (the 
"Software"), to deal in the Software without restriction, including without limitation the rights 
to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit 
persons to whom the Software is furnished to do so, provided that the above copyright notice(s) 
and this permission notice appear in all copies of the Software and that both the above copyright 
notice(s) and this permission notice appear in supporting documentation.

OpenEdge includes Imaging Technology copyrighted by Snowbound Software 1993-2003. 

OpenEdge includes Infragistics NetAdvantage for .NET v2009 Vol 2 Copyright © 1996-2009 Infragistics, Inc. All rights reserved.

OpenEdge includes JSTL software Copyright 1994-2006 Sun Microsystems, Inc. All Rights 
Reserved. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY 
KIND, either express or implied. See the License for the specific language governing rights and 
limitations under the License agreement that accompanies the product.

OpenEdge includes OpenSSL software developed by the OpenSSL Project for use in the 
OpenSSL Toolkit (http://www.openssl.org/). Copyright © 1998-2007 The OpenSSL 
Project. All rights reserved. This product includes cryptographic software written by Eric 
Young (eay@cryptsoft.com). This product includes software written by Tim Hudson 
tjh@cryptsoft.com). Copyright © 1995-1998 Eric Young (eay@cryptsoft.com) All rights 
reserved. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse 
or promote products derived from this software without prior written permission. For written 
permission, please contact openssl-core@openssl.org. Products derived from this software may 
not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written 
permission of the OpenSSL Project. Software distributed on an "AS IS" basis, WITHOUT 
WARRANTY OF ANY KIND, either express or implied. See the License for the specific 
language governing rights and limitations under the License agreement that accompanies the 
product.

OpenEdge includes Quartz Enterprise Job Scheduler software Copyright © 2001-2003 James 
House. All rights reserved. Software distributed on an “AS IS” basis, WITHOUT 
WARRANTY OF ANY KIND, either express or implied. See the License for the specific 
language governing rights and limitations under the License agreement that accompanies the 
product. This product uses and includes within its distribution, software developed by the 
Apache Software Foundation (http://www.apache.org/).

OpenEdge includes code licensed from RSA Security, Inc. Some portions licensed from IBM 
are available at http://oss.software.ibm.com/icu4j/.

OpenEdge includes the RSA Data Security, Inc. MD5 Message-Digest Algorithm. Copyright 
©1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.
OpenEdge includes Sonic software, which includes software developed by Apache Software Foundation (http://www.apache.org/). Copyright © 1999-2000 The Apache Software Foundation. All rights reserved. The names “Ant”, “Axis”, “Xalan,” “FOP,” “The Jakarta Project”, “Tomcat”, “Xerces” and/or “Apache Software Foundation” must not be used to endorse or promote products derived from the Product without prior written permission. Any product derived from the Product may not be called “Apache”, nor may “Apache” appear in their name, without prior written permission. For written permission, please contact apache@apache.org.

OpenEdge includes Sonic software, which includes software Copyright © 1999 CERN - European Organization for Nuclear Research. Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. CERN makes no representations about the suitability of this software for any purpose. It is provided "as is" without expressed or implied warranty.

OpenEdge includes Sonic software, which includes software developed by ExoLab Project (http://www.exolab.org/). Copyright © 2000 Intalio Inc. All rights reserved. The names “Castor” and/or “ExoLab” must not be used to endorse or promote products derived from the Products without prior written permission. For written permission, please contact info@exolab.org. Exolab, Castor and Intalio are trademarks of Intalio Inc.

OpenEdge includes Sonic software, which includes software developed by IBM. Copyright © 1995-2003 International Business Machines Corporation and others. All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation. Software distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product. Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

OpenEdge includes Sonic software, which includes the JMX Technology from Sun Microsystems, Inc. Use and Distribution is subject to the Sun Community Source License available at http://sun.com/software/communitysource.

OpenEdge includes Sonic software, which includes software developed by the ModelObjects Group (http://www.modelobjects.com). Copyright © 2000-2001 ModelObjects Group. All rights reserved. The name “ModelObjects” must not be used to endorse or promote products derived from this software without prior written permission. Products derived from this software may not be called “ModelObjects”, nor may “ModelObjects” appear in their name, without prior written permission. For written permission, please contact djacobs@modelobjects.com.

OpenEdge includes Sonic software, which includes code licensed from Mort Bay Consulting Pty. Ltd. The Jetty Package is Copyright © 1998 Mort Bay Consulting Pty. Ltd. (Australia) and others.
OpenEdge includes Sonic software, which includes files that are subject to the Netscape Public License Version 1.1 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.mozilla.org/NPL/. Software distributed under the License is distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License. The Original Code is Mozilla Communicator client code, released March 31, 1998. The Initial Developer of the Original Code is Netscape Communications Corporation. Portions created by Netscape are Copyright 1998-1999 Netscape Communications Corporation. All Rights Reserved.

OpenEdge includes Sonic software, which includes software developed by the University Corporation for Advanced Internet Development http://www.ucaid.edu Internet2 Project. Copyright © 2002 University Corporation for Advanced Internet Development, Inc. All rights reserved. Neither the name of OpenSAML nor the names of its contributors, nor Internet2, nor the University Corporation for Advanced Internet Development, Inc., nor UCAID may be used to endorse or promote products derived from this software and products derived from this software may not be called OpenSAML, Internet2, UCAID, or the University Corporation for Advanced Internet Development, nor may OpenSAML appear in their name without prior written permission of the University Corporation for Advanced Internet Development. For written permission, please contact opensaml@opensaml.org.

OpenEdge includes UnixWare platform of Perl Runtime authored by Kiem-Phong Vo and David Korn. Copyright © 1991, 1996 by AT&T Labs. Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software. THIS SOFTWARE IS BEING PROVIDED “AS IS”, WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHORS NOR AT&T LABS MAKE ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

OpenEdge includes Vermont Views Terminal Handling Package software developed by Vermont Creative Software. Copyright © 1988-1991 by Vermont Creative Software.

OpenEdge includes XML Tools, which includes versions 8.9 of the Saxon XSLT and XQuery Processor from Saxonica Limited (http://www.saxonica.com/) which are available from SourceForge (http://sourceforge.net/projects/saxon/). The Original Code of Saxon comprises all those components which are not explicitly attributed to other parties. The Initial Developer of the Original Code is Michael Kay. Until February 2001 Michael Kay was an employee of International Computers Limited (now part of Fujitsu Limited), and original code developed during that time was released under this license by permission from International Computers Limited. From February 2001 until February 2004 Michael Kay was an employee of Software AG, and code developed during that time was released under this license by permission from Software AG, acting as a "Contributor". Subsequent code has been developed by Saxonica Limited, of which Michael Kay is a Director, again acting as a "Contributor". A small number of modules, or enhancements to modules, have been developed by other individuals (either written especially for Saxon, or incorporated into Saxon having initially been released as part of another open source product). Such contributions are acknowledged individually in comments attached to the relevant code modules. All Rights Reserved. The contents of the Saxon files are subject to the Mozilla Public License Version 1.0 (the "License"); you may not use these files except in compliance with the License. You may obtain a copy of the License at http://www.mozilla.org/MPL/ and a copy of the license can also be found in the
installation directory, in the c:/OpenEdge/licenses folder. Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

OpenEdge includes XML Tools, which includes Xs3P v1.1.3. The contents of this file are subject to the DSTC Public License (DPL) Version 1.1 (the "License"); you may not use this file except in compliance with the License. A copy of the license can be found in the installation directory, in the c:/OpenEdge/licenses folder. Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License. The Original Code is xs3p. The Initial Developer of the Original Code is DSTC. Portions created by DSTC are Copyright © 2001, 2002 DSTC Pty Ltd. All rights reserved.

OpenEdge includes YAJL software Copyright 2007, Lloyd Hilaiel. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. Neither the name of Lloyd Hilaiel nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
The Progress® OpenEdge® DataServer for ODBC allows you to develop and deploy OpenEdge ABL or WebSpeed® applications that access DB2 UDB® and Sybase® data sources. Throughout this manual, the term ODBC data sources, or simply data sources, refers specifically to these data sources.

The OpenEdge DataServer for ODBC is fully compliant with ODBC-3.5. However, the OpenEdge DataServer for ODBC only partially implements ODBC-2.0 and ODBC-3.0 features, such as stored procedures. For more information on stored procedures and other programming feature functionality, see Chapter 2, “Programming Considerations.”

This chapter introduces you to the DataServer for ODBC. It describes the DataServer components, utilities, and configurations. Specifically, this chapter discusses:

- DataServer for ODBC overview
- DataServer components
- DataServer utilities
- DataServer demonstration databases
- DataServer configurations
- Software requirements
- Guidelines for using the DataServer
DataServer for ODBC overview

The OpenEdge DataServer for ODBC allows you to access your ODBC data source with the OpenEdge ABL and develop applications within the OpenEdge Studio. The OpenEdge Studio is a set of tools that helps you to maintain data sources and develop applications with graphical user interfaces. When you develop applications with the OpenEdge Studio, you can design a separation of your user interface, business logic, and data sources. OpenEdge Studio provides user interface independence, whether you need a character UI, a Graphical Windows UI, or a Web interface. When you incorporate a DataServer with OpenEdge Studio, you are enabling the use of your OpenEdge ABL business logic to access a foreign data source.

OpenEdge provides you the tools for the development, deployment, management, and integration of your application. The DataServer allows you to implement the OpenEdge features and OpenEdge ABL expansions in applications that run with the supported data sources. Some of these tools and features are:

- **OpenEdge® Application Server** — Use an application server to partition your application across multiple servers
- **Progress Dynamics®** — Use Progress Dynamics to design and develop a repository-based application
- **Data Dictionary** — Use the Data Dictionary to modify database schema; create indexes; and define database triggers, validation expressions, and help messages
- **Data Administration** — Use Data Administration to manage database data and perform DataServer tasks
- **Database triggers** — Use a trigger to fire a block of ABL code whenever a specific data source event occurs; for example, when creating or deleting a record or assigning a value to a field

Note that using the DataServer to access an ODBC data source does not provide you with access to all OpenEdge ABL, WebSpeed, and database features. For details, see Chapter 2, “Programming Considerations.”

The OpenEdge DataServer for ODBC is a fully featured DataServer with a wide range of deployment flexibility and scalability. The following are some of the features:

- It is available for use by multiple users.
- It can be configured on remote servers.
- The ODBC drivers for the supported data sources are bundled with the product.
- It is available only on Professional Windows platforms. For specific versions supported, see the OpenEdge Release Notes and OpenEdge Getting Started: Installation and Configuration.

**Note:** To install and use an ODBC driver successfully, your system must meet the driver system requirements for connecting to data sources. For more information, see the appropriate DataDirect product documentation available at [http://www.datadirect.com](http://www.datadirect.com).
DataServer components

The DataServer is a set of software components that allows OpenEdge and WebSpeed applications to access data in a supported ODBC data source. The DataServer enables access by translating standard OpenEdge ABL and WebSpeed code into calls appropriate to the ODBC data source. The application can thus manipulate the data from all supported ODBC data sources as though it were OpenEdge-based data.

The DataServer consists of the components described in Table 1–1.

Table 1–1: DataServer Components

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataServer for ODBC (Windows platforms only)</td>
<td>An OpenEdge software module that allows you to use OpenEdge or WebSpeed with a supported ODBC data source.</td>
</tr>
<tr>
<td>ODBC Driver Manager (Windows platforms only)</td>
<td>A software module that manages the connection between an application and ODBC drivers. It is implemented on Windows by default as a Microsoft-supplied dynamic link library (DLL) module.</td>
</tr>
<tr>
<td>Schema holder</td>
<td>A repository for data definitions for one or more ODBC data sources.</td>
</tr>
<tr>
<td>ODBC driver (Windows platforms only)</td>
<td>A software module that implements the ODBC Application Programming Interface (API) specification. The driver interfaces between the ODBC Driver Manager and the associated data source. A separate ODBC Driver is required for each data source.</td>
</tr>
<tr>
<td>ODBC data source (Windows platforms only)</td>
<td>A name that identifies a specific set of data and how to obtain it. You must register a supported database as an ODBC data source.</td>
</tr>
<tr>
<td>ODBC Administrator (Windows platforms only)</td>
<td>A Microsoft program for registering and configuring data sources. You register a data source by providing information that the ODBC driver needs through the ODBC Administrator. Note that this registration method is used only in Windows platforms.</td>
</tr>
<tr>
<td>DataServer ODBC utilities</td>
<td>A set of utilities that allows you to perform certain tasks related to the DataServer. You access them from Data Administration.</td>
</tr>
</tbody>
</table>
Figure 1–1 illustrates how the DataServer components are organized.

As shown in Figure 1–1, in self-service mode, the components for the DataServer for ODBC are organized similarly to the client-server mode except that the DataServer is embedded in the client rather than executing separately.

In either case, an OpenEdge or WebSpeed application uses the DataServer to work with the ODBC Driver Manager, which accesses the desired ODBC driver. The driver then accesses the relevant ODBC data-source manager, which in turn accesses the data in the associated ODBC data source. Note that there is a separate ODBC driver for each supported data source.

Some of the DataServer components are linked with elements of the standard OpenEdge architecture, as described in the “DataServer components” section on page 3–2.
DataServer for ODBC logic

The flow chart in Figure 1–2 illustrates the DataServer’s internal logic.

1. The user runs an OpenEdge or WebSpeed application.

   - FOR EACH customer:
     - DISPLAY name.
     - END.

2. The DataServer translates a statement into SQL.

   - SELECT name
     - FROM customer.
   
3. The data source manager receives the SQL statements from the DataServer.

   - SELECT name
     - FROM customer.

4. The data source manager compiles the request and returns the results to the client.

   - "Second Skin Scuba"
   - "Match Point Tennis"
   - "Off The Wall"
   - "Pedal Power Cycles"

5. The client displays the returned results.

   - Name
     - Second Skin Scuba
     - Match Point Tennis
     - Off The Wall
     - Pedal Power Cycles

Figure 1–2: DataServer logic
As shown in Figure 1–2, when you execute an application that accesses a supported data source, the Compiler translates OpenEdge ABL or SpeedScript statements into their SQL equivalents. The DataServer then issues the SQL statements to the appropriate ODBC driver through the ODBC API. The driver, which provides the software mechanisms for accessing and controlling the data source, processes the SQL statements, transfers them to the data-source manager, and returns the results to the DataServer through the ODBC API.

**Note:** SQL statements presented in Figure 1–2 have been simplified for the purpose of focusing on the data flow.

In order to facilitate the flow of statements between client and server, the DataServer places OpenEdge equivalents for data definitions from a supported data source into a schema holder (a repository for data definitions for one or more ODBC data sources). When the OpenEdge client or WebSpeed Agent executes statements and retrieves information from the data source, it relies on data definitions in the schema holder.

**The schema holder**

As noted in the previous section, a *schema holder* contains information about one or more supported ODBC data sources. This information, called the *schema* of the data source, is a description of its structure, its tables, the fields within its tables, and its indexes.

An application accesses the schema holder only when it compiles procedures and at the beginning of a run-time session for *schema caching* (the loading of data definitions into memory). Typically, the schema holder is not accessed during a run-time session after the initial schema caching.

Before an OpenEdge client can access data in the relevant ODBC data source, you must create a schema holder and load the required data source definitions into the schema holder. You can then use the Data Dictionary to add OpenEdge database features such as validation expressions and messages.

Figure 1–3 illustrates the schema-loading process.

![Figure 1–3: The schema-loading process](image)

If you plan to use the DataServer to send SQL statements directly to the data source using only the `RUN STORED–PROCEDURE` syntax and you do not expect returned data, you need not load data definitions into the schema holder; however, you must do the following:

- Load the stored procedure into the schema holder
- Connect to a data source
For information on using RUN STORED-PROCEDURE, see the “Stored procedures” section on page 2–42.

Security

Using the OpenEdge DataServer for ODBC involves following the security guidelines required by both the OpenEdge database and the ODBC data source. By default, OpenEdge allows unrestricted access to data sources, so at minimum, you should follow the guidelines that the data source requires for your applications.

OpenEdge security

The OpenEdge database management system has no minimum security requirements. You can, however, impose security features on any OpenEdge database or schema holder. There are four levels of application security that you can impose:

- Database-connection security
- Schema security
- Compile-time security
- Run-time security

For more information about compile-time and run-time security, see OpenEdge Deployment: Managing ABL Applications. For general information about OpenEdge security, see OpenEdge Getting Started: Core Business Services.

ODBC data-source security

As noted previously, you should follow the security guidelines that your ODBC data source has established for your applications. Some ODBC data sources require that all users supply a valid login name and password to access them. In addition, ODBC data sources often provide security that controls access to data sources and objects within data sources. Data-source access security typically has four levels: system administrator, database owner, database object owner, and public:

- System administrators can grant or revoke permissions to other users to create or own a wide type of objects; for example, databases.
- Database owners can grant other users permission to access or modify a database or its objects.
- Database object owners can be the owner of objects in a database owned by another user.
- Public database objects can be accessed by any users without restriction.

For more information on database security, see the documentation for your data source.

Note: There are specific security requirements for accessing data with the DataServer that relate to creating a schema holder. For details, see the “Creating a schema holder” section on page 3–14.
DataServer utilities

OpenEdge provides a set of utilities that allows you to perform the following DataServer tasks:

- Creating a schema holder
- Updating a schema holder
- Verifying that the definitions in the schema holder match the current data source information
- Editing connection information for a schema holder
- Changing the code page for a schema holder
- Deleting the schema
- Migrating an existing OpenEdge database to a supported ODBC-compliant data source
- Adjusting the pulled schema to match an existing OpenEdge database

In addition, you can use the Data Dictionary to modify data definitions at the field level; for example, to change display formats, add help messages, or add validation expressions. You can also use Data Administration to manipulate data definition files for the schema holder.
DataServer demonstration databases

As part of the DataServer product, Progress Software Corporation provides you with the ability to create ODBC demonstration data that you can use to experiment with the DataServer. Note that you do not install the ODBC demonstration databases as part of the OpenEdge installation media. Instead, you create them by running the OpenEdge-to-ODBC utility, which migrates the OpenEdge Sports database—objects and data—to your ODBC data source. For instructions, see the “Demonstration databases for ODBC DataServers” section on page 5–2.
DataServer configurations

The DataServer for ODBC can run in a variety of configurations. Depending on its configuration, it is either a single OpenEdge executable or a set of OpenEdge executables that you can distribute across operating systems. There are two general types of configurations—local and remote—with variations on each type:

- **Local DataServer** — All of the DataServer software components, the schema holder, the ODBC software, and your data-source client software run on one machine.

- **Remote DataServer** — The DataServer software components run on different machines. Typically, the OpenEdge client component runs on one machine, while the OpenEdge server component runs on another machine called the host machine. Note that the ODBC software and any client software required by the data-source vendor run on the same machine as the DataServer software modules. In the remote DataServer configuration, this is the host machine. The host machine and the machine on which the client runs communicate through OpenEdge networking (TCP/IP). A remote configuration exists within the framework of a distributed application, whereby DataServers are spawned on the host machine using the OpenEdge ProBroker executable or broker in the Progress Explorer administration framework.

The local DataServer configuration

Figure 1–4 shows the local DataServer configuration.

![Diagram of the local DataServer for ODBC](image)

*Figure 1–4: The local DataServer for ODBC*
In the local DataServer configuration, all of the DataServer software components, the schema holder, and any ODBC data-source client software run on the same machine. Depending on the ODBC and data-source client software implementation, the actual target database may be local or remote to the machine where the local OpenEdge DataServer for ODBC executes. If the data-source client software supplies modules that manage networked database communications, a remote database configuration is possible. The DataServer cares only about the local ODBC data-source definition. Remote database access is transparent to the DataServer.

The remote DataServer configuration

Figure 1–5 shows the remote DataServer configuration.

![Diagram of the remote DataServer for ODBC](image)

**Figure 1–5: The remote DataServer for ODBC**

In the remote configuration, the OpenEdge client and schema holder run on a client machine and the OpenEdge server components and any ODBC data-source client software run on a remote server machine. Depending on the ODBC and data-source client software implementation, the actual target database may be local or remote to the machine where the OpenEdge DataServer for ODBC executes. If the data-source client software supplies modules that manage networked database communications, a remote database configuration is possible. The DataServer cares only about the data-source definition on its machine. Remote database access is transparent to the DataServer.

In remote DataServer configurations, OpenEdge handles the communication between the client and the server. The client and server processes that make up the DataServer adapt to a variety of network configurations.
Figure 1–5 shows a remote DataServer configuration. Table 1–2 lists the supported configurations. It contains possible client-server combinations and networking options.

**Table 1–2: Supported configurations**

<table>
<thead>
<tr>
<th>Client</th>
<th>Networking</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenEdge on all Windows platforms&lt;sup&gt;1&lt;/sup&gt;</td>
<td>None</td>
<td>The server, client, ODBC client, and data source are on the same machine.</td>
</tr>
<tr>
<td>OpenEdge on Windows, and UNIX</td>
<td>OpenEdge</td>
<td>The client (Windows or UNIX) is on the client machine; the server, ODBC driver, and data source are on a Windows server machine.</td>
</tr>
</tbody>
</table>

<sup>1</sup> See Release Notes and the OpenEdge Getting Started: Installation and Configuration Guide for specific details on supported platforms.

**Configuring distributed DataServer applications using ProBroker**

Figure 1–6 shows an example of a distributed DataServer configuration that uses ProBroker. The figure illustrates how to combine clients and servers that run on different platforms.
In Figure 1–6, the client runs on a client machine (either Windows or UNIX) and can access a remote server on any supported OpenEdge DataServer for ODBC platform. Multiple brokers may be running simultaneously on one server machine. The client connects initially to the ProBroker executable by locating the host and port of the executing broker. It is then automatically reconnected to a DataServer established for it by the broker. Each executing broker may spawn a multitude of DataServer processes. A spawned DataServer process uses ODBC drivers to locate and connect to a data source. Depending on the ODBC and data-source software implementation, the actual target database might be either local or remote to the host machine. Note that in this example, the schema holder also runs on the Windows client; however, you can optionally locate it on any host that is accessible on your network. Each spawned DataServer can service database requests for the same database or for a different database than those of other spawned servers.

In remote DataServer configurations, OpenEdge handles the communication between the client and the server. The client and server processes that make up the DataServer adapt to a variety of network configurations.

**Configuring distributed DataServer applications using the Unified Broker Framework**

The Unified Broker Framework is a system administration framework that provides a consistent interface in which specific OpenEdge products such as the DataServer for ODBC can be managed. It supports elements that allow you to perform such common administrative tasks as:

- Starting and stopping processes
- Managing, configuring, and validating property files

These elements include the OpenEdge Explorer and Progress Explorer configuration tool, mergeprop utility, and command-line utilities.

The framework also facilities activities that are fundamental to the ODBC DataServer’s broker-based technology. For example, the AdminService, the framework’s central element, enables supported products like the DataServer for ODBC in managing an application’s resources.

For details about the Unified Broker Framework, its elements, and the OpenEdge products that employ its features, see *OpenEdge Getting Started: Installation and Configuration*.

**A closer look at the OpenEdge Explorer and Progress Explorer tool**

The Progress Explorer tool is a graphical user interface that provides an easy way for you to manage OpenEdge servers. The Progress Explorer tool runs as a Windows client and works with another administration component, the AdminService, to provide a client/server framework for managing the following OpenEdge servers.

Using the Progress Explorer tool, you can:

- Create new instances of OpenEdge servers and configure their property settings
- Modify property settings of existing OpenEdge server instances
- Start and stop OpenEdge servers
- Monitor the status of OpenEdge servers
Introduction

For more information about working with the Progress Explorer tool, see the Progress Explorer online help.

**Using the mergeprop utility**

The `mergeprop` utility is a command-line utility that supports functionality similar to that supported by the OpenEdge Explorer and Progress Explorer configuration tool. It is an alternative approach to configuring and managing the content of the OpenEdge property files. Property files, such as the `ubroker.properties` file used by the DataServer, store configuration information that specifies and controls the behavior of various components.

**Typical configuration scenarios**

In a distributed client-server configuration, the OpenEdge client runs on a client machine (either in Windows or on UNIX) and accesses a remote server on a Windows server where the broker for the DataServer for ODBC is running. Multiple NameServers and/or brokers can be running simultaneously on one server machine. The DataServer client connects to a broker for the Progress Explorer either directly or through a controlling NameServer. (See the important caution that follows.) The DataServer client is then automatically reconnected to a DataServer established for it by the broker. Each executing broker can spawn a multitude of DataServer processes. A spawned DataServer process uses the ODBC driver to locate and connect to the data source. The actual target database might be either local or remote to the host machine. The schema holder can run on the client or you can locate it on any host that is accessible on your network. Each spawned DataServer can service database requests for the same database or for a different database than those of other spawned servers.

In remote DataServer configurations, OpenEdge handles the communication between the client and the server. The Progress Explorer supports only the TCP/IP network configuration.

**Caution:** In a run-time configuration, all DataServer clients should attach consistently either to a set of NameServers or to a set of brokers. **Do not** run brokers under controlling NameServers for one client while another client simultaneously attaches directly to a broker.

For more information about configuring and connecting the DataServer, see:

- The “Configuring with the OpenEdge Explorer and Progress Explorer” section on page 5–6
- The “Starting and stopping a broker process from the OpenEdge Explorer and Progress Explorer and connecting a client” section on page 6–3

For more information about the Unified Broker Framework, its elements, and how Unified Broker products such as the DataServer for ODBC work within the framework, see OpenEdge *Getting Started: Installation and Configuration*. 
Software requirements

However, the exact configuration of client drivers and ODBC software required to connect to a particular target database server can vary from one installation to another. Furthermore, the OpenEdge DataServer for ODBC is only “certified” to work with a select group and combination of drivers and target database servers. “Certification” means that the drivers and/or databases have been tested by OpenEdge to ensure ODBC compliance and that the ODBC DataServer has been adjusted internally to handle known cases where non-compliance exists. Certified drivers that are supplied by Data Direct Technologies are installed in OpenEdge when you purchase the DataServer product. There drivers are installed into the $DLC/bin/odbc directory.

For more information, refer to *OpenEdge Getting Started: Installation and Configuration*, the OpenEdge Product Availability Guide available at [www.progress.com](http://www.progress.com) and the OpenEdge Release Notes for specific versions and environments supported by OpenEdge. You should also always check further with vendor documentation from your database, driver and/or client software provider to ensure proper ODBC installation and database configuration.
Guidelines for using the DataServer

OpenEdge supports many capabilities not found in other database management systems, such as backward scrolling cursors and the ability to find the previous or last record in a table. The DataServer supports these and other programming and database features to ensure that your applications work with both OpenEdge databases and ODBC data sources.

The DataServer allows you to use OpenEdge database ABL features as extensions to your data source. Some of the OpenEdge ABL programming and OpenEdge database design techniques that you can implement on your ODBC data source using the DataServer are:

- ROWID function
- Arrays
- Backward and forward scrolling
- Case-insensitive indexes

For access to some of these features, you might have to make minor modifications to how your ODBC data source or application is organized. For a discussion of these issues and instructions for modifying your data source, see Chapter 2, “Programming Considerations.”

If you create an ODBC data source from an existing OpenEdge database with the OpenEdge-to-ODBC migration utility and select the Create Extended ABL Objects option, you can use the FIND PREV/LAST statements in addition to taking advantage of OpenEdge-like cursor behavior. If you select Create Shadow Columns, you will have case-insensitive indexes within the resulting ODBC data source. For data sources where creating the extended ABL objects is not supported, the toggle box for this feature is disabled. For data sources where indexes are case-insensitive by default, the toggle box for creating shadow columns is disabled.

How you use the DataServer depends on whether you plan to access information in a data source through an OpenEdge application, migrate an OpenEdge database to an ODBC data source, or upgrade to the OpenEdge DataServer for ODBC. The following sections summarize what you must do and point you to the information you need in this manual.

Using the DataServer for ODBC for the first time

Prior to using the DataServer for the first time, you must install and configure the software.

To prepare to use the DataServer:

1. Install the DataServer modules on the machines your configuration requires.
2. If you choose not to use the bundled ODBC drivers, install your ODBC driver software on the machine that will execute the server component.
3. Create a local schema holder on the client or server machine, as appropriate. Schema holders cannot be transferred between different host machines.

Chapter 3, “Configuring the DataServer,” provides information about installing DataServer modules and creating schema holders.
Migrating an OpenEdge database to an ODBC data source

A conversion utility is supplied to migrate an OpenEdge database to an ODBC data source.

To migrate an OpenEdge database to the DataServer:

1. Install the DataServer components on the machines that your configuration requires.
2. Run the OpenEdge-to-ODBC migration utility, selecting the appropriate supported data source for migration.

See the “Migrating an OpenEdge database to an ODBC data source” section on page 5–21 for specific instructions.

Upgrading to the Release 10 DataServer for ODBC

In order to take advantage of OpenEdge Release 10 features, you must upgrade your schema holder.

To upgrade an existing DataServer schema holder:

1. Install the Release 10 DataServer modules on the machines that your configuration requires.
2. Create an empty Release 10 local schema holder on the client machine or the server machine, as appropriate.

See Chapter 3, “Configuring the DataServer,” for information about where to install DataServer modules and how to create a schema holder. For more information on upgrading your DataServer, see Appendix A, “Upgrading DataServer Applications.”
Documentation resources

Table 1–3 suggests paths through this manual that accommodate different approaches to using the DataServer for ODBC.

Table 1–3: How to use this manual

<table>
<thead>
<tr>
<th>If you are . . .</th>
<th>Read . . .</th>
</tr>
</thead>
</table>
| New to the DataServer for ODBC | Chapter 2, “Programming Considerations”  
Chapter 3, “Configuring the DataServer”  
Chapter 4, “Connecting the DataServer”  
Chapter 5, “The DataServer Tutorial” |
| Migrating an OpenEdge database to an ODBC data source | “Running the OpenEdge DB-to-ODBC utility” section on page 5–21  
Chapter 2, “Programming Considerations”  
Chapter 3, “Configuring the DataServer”  
Chapter 4, “Connecting the DataServer”  
Chapter 5, “The DataServer Tutorial” |
| Upgrading the DataServer to OpenEdge Release 10 | Chapter 2, “Programming Considerations”  
Chapter 3, “Configuring the DataServer”  
Chapter 5, “The DataServer Tutorial”  
Appendix A, “Upgrading DataServer Applications” |
| Upgrading from an earlier version of your data source | Chapter 3, “Configuring the DataServer”  
Appendix A, “Upgrading DataServer Applications” |

Table 1–4 lists manuals from the OpenEdge documentation set that contain useful information on different aspects of DataServer usage.

Table 1–4: DataServer-related topics in OpenEdge documentation (1 of 2)

<table>
<thead>
<tr>
<th>Topic</th>
<th>OpenEdge manuals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting OpenEdge</td>
<td>OpenEdge Getting Started: Installation and Configuration</td>
</tr>
<tr>
<td>Using the Data Dictionary</td>
<td>OpenEdge Getting Started: ABL Essentials</td>
</tr>
<tr>
<td>Defining security for an OpenEdge database</td>
<td>OpenEdge Data Management: Database Administration</td>
</tr>
</tbody>
</table>
| Writing applications in the OpenEdge ABL | OpenEdge Getting Started: Progress OpenEdge Studio  
OpenEdge Getting Started: ABL Essentials |
### Table 1–4: DataServer-related topics in OpenEdge documentation

<table>
<thead>
<tr>
<th>Topic</th>
<th>OpenEdge manuals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connecting your data source with WebSpeed</td>
<td><em>OpenEdge Getting Started: Installation and Configuration</em></td>
</tr>
<tr>
<td>Writing WebSpeed applications</td>
<td><em>OpenEdge Application Server: Developing WebSpeed Applications</em></td>
</tr>
</tbody>
</table>
An application developed in the OpenEdge Studio can use the DataServer for ODBC to access data from multiple sources transparently. With the DataServer, your applications can access information in both OpenEdge databases and supported foreign ODBC data sources.

This chapter discusses the differences between OpenEdge databases and supported ODBC data sources that you must consider when you plan your applications and design your databases. The chapter also describes DataServer strategies for resolving these differences, which your application might need to accommodate. Specifically, the chapter discusses the following topics:

- Database design issues
- Record creation
- Data-source record locking
- Transactions
- Error handling
- Cursors
- ABL issues
- Stored procedures
- Enhancing DataServer performance
Follow these guidelines carefully when you develop your application to ensure that it can access OpenEdge databases and ODBC data sources transparently.

The material in this chapter is also of interest to users who plan to migrate an OpenEdge database to an ODBC data source. However, such a migration raises additional issues that you must consider when designing your application. For details, see the “Running the OpenEdge DB-to-ODBC utility” section on page 5–21.
Database design issues

When you create or modify the OpenEdge databases or ODBC data sources that your applications access, you must consider general database design issues such as OpenEdge and ODBC data-source objects, data types, naming conventions, code pages, and indexes. The following sections discuss how OpenEdge databases and ODBC data sources differ in such areas and describe how the DataServer resolves them.

OpenEdge database objects and ODBC data-source objects

OpenEdge databases and the ODBC data sources share the structural elements common to relational databases, but each system has its own elements. These elements, called database or data-source objects, are components of the OpenEdge database or ODBC data source’s logical structure. Table 2–1 lists OpenEdge database objects and notes the ODBC data-source equivalents.

<table>
<thead>
<tr>
<th>OpenEdge</th>
<th>DB2 UDB</th>
<th>Sybase</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Table</td>
<td>Table</td>
</tr>
<tr>
<td>Field</td>
<td>Column</td>
<td>Column</td>
</tr>
<tr>
<td>Record</td>
<td>Row</td>
<td>Row</td>
</tr>
<tr>
<td>Index</td>
<td>Index</td>
<td>Index</td>
</tr>
<tr>
<td>Unique index</td>
<td>Unique key</td>
<td>Primary key</td>
</tr>
<tr>
<td>Nonunique index</td>
<td>Nonunique key</td>
<td>Foreign key</td>
</tr>
<tr>
<td>Sequences¹</td>
<td>No equivalent</td>
<td>Native equivalent²</td>
</tr>
<tr>
<td>Validation expression</td>
<td>Check constraints</td>
<td>Integrity constraint</td>
</tr>
<tr>
<td>Validation message</td>
<td>No equivalent</td>
<td>No equivalent</td>
</tr>
<tr>
<td>Initial value</td>
<td>Default</td>
<td>Default</td>
</tr>
<tr>
<td>Schema trigger</td>
<td>Trigger</td>
<td>Trigger</td>
</tr>
<tr>
<td>No equivalent to stored procedures</td>
<td>Stored procedure</td>
<td>Stored procedure</td>
</tr>
<tr>
<td>View</td>
<td>View</td>
<td>View</td>
</tr>
</tbody>
</table>

1. Sequences in DB2 UDB become native objects depending upon the DB2 UDB version and platform. For instance, sequences only become native database objects in DB2/400 starting with iSeries OS version 5.3.

2. The DataServer supports Progress ABL sequences through native stored procedures and triggers.
Naming conventions

When planning for maximum transparency across OpenEdge databases and ODBC data sources, be sure to consider the restrictions that each places on naming data-source objects when you perform a OpenEdge-to-ODBC conversion. Table 2–2 describes these restrictions, and the notes that follow it discuss how the DataServer resolves differences between OpenEdge and ODBC naming conventions.

Table 2–2: OpenEdge database and ODBC data-source naming restriction

<table>
<thead>
<tr>
<th>Category</th>
<th>OpenEdge</th>
<th>DB2 UDB</th>
<th>Sybase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphanumeric characters</td>
<td>A-Z, a-z</td>
<td>A-Z, a-z</td>
<td>All alphanumeric characters from the character set that you defined for your Sybase database</td>
</tr>
<tr>
<td>Special characters</td>
<td>Allows $, &amp;, #, %, –, and _</td>
<td>Allows @, $, #, and _</td>
<td>Allows @, $, #, _, ¥, and £</td>
</tr>
<tr>
<td>Initial character</td>
<td>Alphabetic only</td>
<td>Alphabetic only</td>
<td>Alphabetic only</td>
</tr>
<tr>
<td>Maximum length of object name</td>
<td>32 characters</td>
<td>18 characters</td>
<td>24 characters</td>
</tr>
<tr>
<td>Keywords</td>
<td>Not allowed</td>
<td>Not allowed</td>
<td>Not allowed</td>
</tr>
<tr>
<td>Unique table names</td>
<td>Required</td>
<td>Required</td>
<td>Required</td>
</tr>
</tbody>
</table>

1. In a DB2 UDB schema, lowercase letters in object names are converted to uppercase letters.
2. Although OpenEdge allows you to use the ampersand (&) in database-object names, the OpenEdge Studio tools use this character internally to denote preprocessor directives.
3. When the DataServer encounters characters not supported by OpenEdge in a table, index, or field name, it converts them to underscores (_) in the schema holder. For example, the data-source table name emp_status becomes emp_status in the schema holder. If the unsupported character is at the beginning of an object name, the DataServer substitutes a lowercase z. Also, the DataServer strips @ symbols from stored procedure parameter names.
4. Object names in these data sources are limited by the data source to 30 characters. When you use the Progress-to-ODBC utility to perform a database conversion, however, the utility truncates object names to 24 characters in order to accommodate the suffix characters that it appends when it finds nonunique names, reused OpenEdge keywords, array fields, sequences, triggers, and so forth. If you intend to use the OpenEdge-to-ODBC utility, be sure to limit object names to 24 characters to accommodate this name resolution during conversion.
5. OpenCell databases and most ODBC data sources contain restrictions against using keywords as database object names. If a data source object name consists of an OpenEdge keyword, the DataServer appends an underscore character (_ ) to the name. For example, an object named “each” becomes “each_.”
6. DB2 UDB and Sybase qualify nonunique table names to make them unique (see your data-source documentation for details); however, OpenEdge still regards these qualified names as nonunique. The DataServer resolves nonunique table names for you. When OpenEdge encounters matching table names while creating or updating a schema holder, it names the second and subsequent tables table-1, table-2, etc. For example, if OpenEdge encounters a table named lee.employee (the employee table owned by the user lee) that is the second instance of the name in the data source, it names the corresponding table in the schema holder employee-1.

NOTE: When resolving naming conventions, the DataServer can append up to three characters to an object name: two in the case of a nonunique name and one in the case of an OpenEdge keyword. Therefore, you should limit your object names to 29 characters so that you do not exceed the OpenEdge 32-character limit for object names in the schema holder. Note that object-name length limit for some data sources (for example, DB2 UDB) is low enough that you are unlikely to exceed the OpenEdge 32-character limit.
7. DB2 UDB allows you to qualify an object name by adding the following:
   - Data-source instance name
   - Owner’s name
   - Table name (for columns)
8. Sybase allows you to qualify an object name by adding the following:
   - Database name
   - Owner’s name
   - Table name (for columns)
ODBC data source limits

ODBC data sources generally have upper limits on various components; for example, the number of columns per table, the number of columns per SELECT statement in a view, the maximum size of a data row, and the maximum length of character data types. These ODBC data source-specific restrictions might affect how OpenEdge applications run and port to various data sources. See your data source documentation for full details.

Code pages

Flexibility in managing code pages, also called character sets, is critical for international applications. When you access an ODBC data source through the DataServer, the DataServer retrieves character data as determined by the code page that the data source uses. OpenEdge allows you to designate code pages for the data source and for the client. For OpenEdge databases, the default code page is iso8859–1.

Figure 2–1 shows a possible configuration of code pages for the DataServer components and processes.

![Diagram of DataServer processes and code pages]

In the configuration shown in Figure 2–1, all components use the same code page. (For information on setting character sets for your data source, see the administration guide supplied by the vendor.) On the OpenEdge side, if the client and the schema holder use different code pages, a conversion takes place between them.

In order for DataServer applications to manipulate data from an ODBC data source accurately, you must specify the correct code page in the schema holder. For OpenEdge applications accessing the DataServer, the schema holder identifies the code page of the character data. The DataServer sends the data-source name for the code page to the data source to indicate the character set for the data that the data source returns.
Be sure to set the code page in the schema holder to match a code page that the ODBC data source supports. To minimize the number of translations, specify the default code page that the data source uses. If OpenEdge does not support the data source’s code page, you can specify instead a compatible code page that is available for your data source. The directory \%DLC\%\prolang\convmap contains conversion tables for all of the code pages that OpenEdge supports. Check to see whether any of them match your code page. The DataServer for ODBC does not support Unicode code pages.

The default code page setting in the schema holder is iso8859–1/iso_1. DB2/400 data sources use EBCDIC code pages. Its Latin-1 equivalent is not fully compatible with iso8859-1. The recommended default for a Latin-1 ODBC DataServer configuration is to use the ibm850 code page.

You can specify a different code page for the schema holder at the following times:

- When you create the DataServer schema for the ODBC data source.
- When you load a new schema with a specified code page into an existing schema holder. In this case, the newly loaded schema’s code page overrides the schema holder’s original code page.
- When you select DataServer → ODBC Utilities → Change DataServer Schema Code Page from the Data Administration screen.

Note: As the last option implies, you can change the code page of your schema holder at any time. However, because changing the code page does not affect the data in your database and because the code page is expected to match how that data is actually stored, changing the code page against a data source that has existing data is usually not recommended. Writing new data to your existing database using a different code page may become the cause of format inconsistencies and data corruption.

Keep in mind that your ODBC software configuration might have local requirements for defining the proper language interface between the ODBC drivers and the data source. See your ODBC and database documentation for details.

Client code page

The Internal Code Page (-cpinternal) startup parameter determines the code page that the OpenEdge client uses when it manipulates data in memory. If the OpenEdge client uses a different code page from the code page set in the schema holder, the DataServer translates between the two code pages, so you must verify that the convmap.cp file contains a conversion table for the client and the code page setting in the schema holder. Suppose, for example, that you set the schema holder to code page ibm850 and the client uses code page iso8859–1. The convmap.cp file must include a table that converts from ibm850 to iso8859–1 and from iso8859–1 to ibm850. If convmap.cp does not include the appropriate table, you can define your own conversion table.

OpenEdge also allows you to define your own collation tables; however, customized collation tables only take effect after data-source collation when you use the DataServer to access an ODBC data source. The data-source collation tables, not the OpenEdge collation tables, have first priority when you perform comparisons and sorts. After performing comparisons and sorts, the OpenEdge client may sort out records that do not conform to the requirements of your customized collation tables.
For example, if you use the default code page 1252/ISO and the default case-insensitive sort ordering with a Sybase data source, collation of the tilde character (~), which is ASCII character 126, sorts before all the alphanumeric characters (0–9, a–z). If you were to select data from a character column “name” with the following:

```
select * from table where name <=CHR(126) (the equivalent of ~)
```

Sybase returns records with columns that begin with a tilde character but no alphanumeric records, since ASCII character 126 is sorted below alphanumeric characters in this Sybase collation.

Normally, the default OpenEdge collation sorts a tilde character above all alphanumeric characters. Therefore, in order for the previous example to exhibit OpenEdge-like behavior and return alphanumeric records as well as records beginning with the tilde, the Sybase sort order for this code page would need to be modified accordingly.

Conversely, if you execute the opposite:

```
select * from table where name >=CHR(126)
```

Sybase returns records with columns that began with a tilde character followed by all that begin with alphanumeric characters. However, the default OpenEdge collation, which sorts the tilde higher than all the alphanumeric characters, would omit records beginning with alphanumeric characters and only return records beginning with the tilde character.

To get the full result set returned from Sybase from the OpenEdge client would require modifying the collation table associated with the OpenEdge code page and weighting it to match the Sybase sort order.

For a complete discussion of how OpenEdge handles code-page issues, see *OpenEdge Development: Internationalizing Applications*.

**Indexes and sorting**

You create and maintain all indexes from within the ODBC data source, using native data-source tools rather than the Data Dictionary. A data-source index uses a logical pointer to the physical locations of table rows in order to sequence data access. You can add and drop indexes but you cannot use their names in queries. The data source alone ultimately decides when and how to use indexes; its decisions are not affected by the DataServer.

Using index definitions in the ODBC data source, the DataServer builds index information in the schema holder. OpenEdge index definitions for the data source schema serve two purposes:

- They allow you to use the OF option in OpenEdge ABL with the FOR EACH and FIND statements. Using the OF option improves the readability of your code. The OF keyword is equivalent to the SQL WHERE clause. You can use OF only when you have a field of the same name in two tables and the field is an index in at least one of the tables. Therefore, since the custnum field is common to both the order and customer tables, you could write the following statement:

```
FOR EACH order OF customer:
```
• They support the ABL USE–INDEX option. USE–INDEX is translated to SQL ORDER BY for DataServer operations. For example, if you define city-dept as an ODBC data-source primary key on the city and department fields, it is a unique index in the schema holder. In this case, the following ABL statements are equivalent when accessing the data source:

```abl
FOR EACH employee USE-INDEX city-dept:
```

```abl
FOR EACH employee BY employee.city BY employee.department:
```

**Note:** If you do not specify USE–INDEX or ORDER–BY, your query will return records in an unpredictable order. Your application might not require predictable ordering, but if it does, be sure to include USE–INDEX or ORDER–BY in your query definition.

### USE-INDEX and BY clause considerations

The BY clause offers greater programming flexibility than the USE–INDEX clause, and should be used in most cases. If indexes are added or deleted, applications coded with the BY clause only require a recompile to adjust to the index changes. Applications coded with the USE-INDEX clause require code modifications to be current with the index changes. If the index specified by a USE–INDEX clause is dropped, the code will fail to compile until the code is modified. If there is a more efficient index than the one specified in the USE–INDEX clause, it will not be selected by ABL and it is unlikely to be used in the data source’s query plan.

### Dummy indexes for sort order

You can add dummy index definitions to your schema holder independent of the actual index definitions in your ODBC data source to provide a sort order. These dummy indexes can improve the readability of your ABL code. An index in the schema holder need not match an index in the ODBC data source. However, the absence of an index definition in the data source can impact performance when you retrieve data with selection on the dummy indexes.

### Unique indexes

If your ODBC data-source tables have at least one unique index, they can be used to support operations such as backward and forward scrolling and accurate cursor positioning through the FIND CURRENT, PREV, and LAST statements. If a table does not have a unique index, you can only scroll forward through its data.

If an ODBC data-source table does not have a unique index, you can designate an index to serve as the unique index for the schema holder. An index that you designate as unique in the schema holder must be unique with respect to the data in the data source; otherwise, you receive run-time errors. See Chapter 5, “The DataServer Tutorial,” for instructions on using the Data Dictionary to designate unique indexes.

ODBC data-source views and result sets from stored procedures do not have unique indexes. Just as for tables, you can use the Data Dictionary to create a unique index in the schema holder based on fields in a view or result set so that you can browse data accessed through views or stored procedures.
**Note:** Do not change the designated ROWID key of a record while an application is running. Suppose, for example, that custnum is a unique key and has been designated the ABL ROWID. If a user changes the value of custnum for a customer from 1 to 111, other users receive an error message when they try to access the record for customer 1.

**Large key entry support**

Updates in the OpenEdge database enable existing index key limits, currently set at approximately 200 bytes, to be widened. For example, updates enable databases with a block size of 4K or 8K to support a maximum index key of approximately 2000 bytes. Also, the index key width expansion extends the maximum key size supported in the OpenEdge client that can be used with databases exported to foreign data sources.

When you are migrating OpenEdge keys to a foreign data source, the key sizes cannot be larger than those supported by the foreign data source. Because the maximum OpenEdge key size is data-source dependent, you should consider your target data source’s capacity with respect to the maximum OpenEdge key-size capacity before you perform a migration.

**Note:** The key capacity for most foreign data sources is a function of the index page size used or allowed on the OS platform in which the data source is installed. Refer to documentation supplied by the data source vendor to determine the key size limit applicable to your target database.

**Case sensitivity**

By default, an OpenEdge database is case insensitive; however, you can set the attributes of a field to define it as either case sensitive or case insensitive. If you intend to do this, consider the following:

- Schema pull criteria is case sensitive against SQL database qualifiers.
- Pattern-matching literals in data-source access statements retrieve case-sensitive data.
- OpenEdge considers the user ID and password submitted at connection time to be case sensitive.

If an indexed field is case insensitive, OpenEdge does not distinguish between uppercase and lowercase letters for that index when sorting or matching data. In general, this flexibility in an application makes data entry easier for end users because they can enter lowercase or uppercase versions of an index. However, if you want to enforce an uppercase/lowercase distinction in your applications, set the attribute to case sensitive.

During a migration, the DataServer for ODBC tries to maintain compatibility with OpenEdge case-insensitivity through the use of shadow columns. Case-sensitivity is particularly useful for indexes that are migrated from OpenEdge applications to an ODBC data source. Shadow columns are an extra column added to record format created in the foreign database immediately before an indexed column. In most cases, this column is name `_S#_column (exceptions are noted in the following sections).

If you are going to use a case-insensitive code page in the foreign data source, then OpenEdge case-insensitivity will be maintained automatically by your code page configuration. In such cases, shadow columns are not necessary and should be turned off during migration.
Keep in mind that when schema from an ODBC data source is pulled back into a schema holder database and shadow columns do not exist, the columns are still case-insensitive, by default. If the code page used in the ODBC data source is case-sensitive and shadow columns were not added for case-insensitive columns during migration and/or do not exist in the ODBC data source, OpenEdge compatibility (without shadow columns) is maintained by applying the `UPPER` function to SQL. In some cases, use of the `UPPER` function can produce less optimized SQL. Therefore, if case-insensitivity is not required, you can simply turn on case-sensitivity in the Dictionary for that column to avoid applying the `UPPER` function. If case-insensitivity is required, Progress recommends adding shadow columns to your foreign database for columns that are case-insensitive and pulling those definitions back into your schema holder. See the “Adding extended ABL support” section on page 5–35 for instructions on maintaining case-insensitive compatibility in an existing foreign data source.

Specific ODBC data sources handle case sensitivity as described in the following sections.

**DB2 UBD**

DB2 UDB is case sensitive. Be sure to consider case sensitivity when you perform comparisons (equals or matches).

**Sybase**

The System Administrator sets case sensitivity in Sybase. Consider the affect of case-sensitivity on your OpenEdge application depending on how this is set in Sybase.

**Interaction of code page, collation, and case sensitivity**

Properly setting code page, collation, and case sensitivity values such that they complement each other, will yield the best scenarios for data access. To avoid conflict among code page, collation, and case sensitivity, set these characteristics at schema creation, and allow a schema pull to manage the DataServer integration. If any of these settings are changed, the schema holder should be regenerated.

**ODBC data-source views**

ODBC data-source schema objects include views. A view is a presentation of data in one or more tables. Views appear as tables in the Data Dictionary’s table list for the schema holder, not as views. In addition, the schema holder contains no unique index information for views. Because views do not have unique indexes, you cannot modify any of the data that a view contains; however, you can access a view with the FOR EACH, FIND NEXT, and GET NEXT ABL statements. Further, because views do not have index definitions, the DataServer cannot reposition the cursor to retrieve individual records. Thus you must be sure to get all of the data that you need in a single database request.

The following ABL code example retrieves data from a view:

```abl
OPEN QUERY query-name FOR EACH view-name.
REPEAT:
    GET NEXT query-name.
    DISPLAY view-name.
    . . .
END.
```
If a view has a unique combination of columns, you can simulate a unique index using the Data Dictionary. You can then access a view that has a simulated unique index just as you do a table; that is, you can scroll backward and forward, and update, create, and delete data, if the data source supports it. See the “Modifying field-level information” section on page 5–30 for information on how to do this.

Some views are the results of joins and contain data from more than one table. You can also provide unique index information for these views if they have a unique combination of columns. You can then scroll backward and forward, but the ODBC data source does not allow you to create or delete data in a multi-table view. You can, however, update data in some views.

The DataServer does not support access to columns in views that are the results of aggregates or computations unless the calculated column has a name associated with it. You assign a specific name to a calculated column when you define a data-source view. For example, the following SQL statement names a computed column in a view definition:

```
create view view-name as select cc = count(custnum) from customer
```

If your data source supports views, you can also access those views by using the RUN STORED–PROC send–sql–statement option to send a SQL statement to select the data from the view. In this case, you can access the view without adding index definitions for the view to the schema holder.

Although the schema holder contains your ODBC data-source views, the Data Dictionary’s SQL View Report does not list them, nor can you access them through the PRO/SQL menu functions.

**Triggers**

*Triggers* are code that an application associates with a data-source object and an action. For example, writing a record might cause code associated with that object or action to execute. The DataServer allows an application to execute triggers for both OpenEdge databases (including the schema holder) and ODBC data sources (if the data source supports triggers). In an application that executes both types, the OpenEdge trigger (CREATE, FIND, UPDATE, DELETE) executes first. If processing an OpenEdge trigger results in a data-source request, the DataServer passes the request to the appropriate ODBC data source and the operation (INSERT, UPDATE, DELETE) executes.

Triggers for OpenEdge databases and ODBC data sources are independent of each other. A data-source trigger that rolls back does not affect OpenEdge triggers. Defining a trigger in OpenEdge does not create a data-source trigger definition. An OpenEdge trigger that rolls back does so independently of the data source’s transaction scope. Note, however, that although triggers for OpenEdge databases and ODBC data sources are independent, they might affect each other based on the kind of transaction that your application is executing.

See the documentation for your ODBC data source to determine whether it supports triggers and, if so, for information on the kind of triggers it supports.
OpenEdge sequences

A sequence generator is a database object that provides incremental values within any integer range. (With an OpenEdge sequence, you can specify any positive or negative increment range). The DataServer for ODBC simulates support for sequences for the Sybase and MS SQL Server Version 6.5 data sources only using stored procedures. OpenEdge sequences are not migrated to foreign data sources for other target database servers. Even in cases where the foreign data source supports sequences, they cannot be pulled into the schema image of an OpenEdge DataServer’s schema holder.

Since the schema holder that ODBC DataServer uses to hold schema is also an OpenEdge database, it is possible to define sequences to a schema holder database itself and then use OpenEdge sequences in your DataServer application in lieu of native sequences at the data source.

Processing considerations for 32-bit and 64-bit data types

OpenEdge supports an existing or new 32-bit INTEGER data type defined in ABL. However, the internal bit size for an INTEGER data type will automatically expand from a 32-bit field size to a 64-bit field size to accommodate larger values. And, in most situations, this expansion also supports the option to convert a database column from a 32-bit INTEGER to a 64-bit INT64 without performing a dump and load. Refer to the following sources for additional language details about INT64:

- **OpenEdge Development: ABL Reference**

Processing activities that require additional action

The following processing activities require the defined action to successfully convert existing 32-bit INTEGER columns to 64-bit INT64 columns:

- When existing r-code references a column’s original 32-bit INTEGER definition, you must recompile the code and check the table’s Cycle Redundancy Check (CRC).

- Contrast the actions required to perform the following data definition changes:
  - If you change data definitions in your original database from INTEGER to INT64 and you had previously migrated your database to an ODBC DataServer, you must repeat the migration step to change the field’s physical storage on the foreign data source.
  - If you change data definitions in your original database from INT64 to INTEGER and you had previously migrated your database to an ODBC DataServer, you must repeat the migration step to change the field’s physical storage on the foreign data source, and then dump and reload the field’s data.

- If you changed your native definition to include a bigint data type, it is recommended that you repeat the step to pull the table into your schema holder. Otherwise, a conversion from the native bigint data type might occur at runtime if your schema holder defines a different data type. Also, if the schema holder definition of the native bigint is INTEGER, an overflow condition might occur when accessing data in the foreign data source.
Determining your application needs

Determine your application needs and goals before changing INTEGER data types to INT64 data types in applications. Not all INTEGER data types will necessarily require the additional bit size, nor will wholesale changes in data types necessarily work efficiently with other interfaces to ABL. For more information about determining when to use INT64 versus INTEGER, refer to the Release 10 ABL Data Types Web paper available at the Web paper category of the PSDN Web site located at http://communities.progress.com/pcom/community/psdn.

User-defined data types

DB2 UDB and Sybase data sources allow you to define your own data types that map to the data types for your data source. When the DataServer reads the schema information for a user-defined data type, it reads the data-source base data type and maps it to the equivalent OpenEdge data type. Suppose, for example, that you create a data type named phone_number and map it to the char data type. In the schema holder, the DataServer represents your phone_number data type as a CHARACTER data type. If you make any changes to a user-defined data type, you must update the schema holder to reflect those changes.

Arrays

The OpenEdge database allows you to define fields as arrays, also called field extents. OpenEdge interprets specially named data-source columns of the same data type as OpenEdge fields with the same number of array elements. You must name the data-source columns column-name##1, column-name##2, and so forth. In DB2 UDB, however, you must name these columns column-name__1, column-name__2, and so forth. (Some DB2 UDB OS environments have trouble translating the pound sign (hash sign) in object names.) The DataServer creates a single field definition in the schema holder for the field extents. See the “Adding extended ABL support” section on page 5–35 for instructions on adding these columns automatically with the Progress-to-ODBC utility.

Unknown values

The DataServer supports ODBC data-source null values. Procedures that use a null value behave exactly as they do when accessing an unknown value ("?") in an OpenEdge database, except for one difference—you cannot compare a field to the unknown value if the field is not allowed to hold the unknown value (i.e., is not null-capable). For example, if the custnum field is not allowed to hold the unknown value, the following statement fails at run time:

```
FIND customer WHERE customer.custnum <> ?
```

See the documentation for your data source to determine whether it supports null values.

A column that is not allowed to hold the unknown value is marked “mandatory” in the schema holder.
In a DataServer application, you assign the unknown value to a column in an ODBC data source by using the question mark operator (?), which the DataServer translates to the appropriate null-value representation for the data source. For example, the following procedure assigns the unknown value to the address2 field of the customer table:

```
FIND FIRST customer NO-ERROR.
IF AVAILABLE customer THEN
    customer.address2 = ?.
```

**Unknown values and sorts**

How unknown values are handled during sorts varies depending on the type of data source. If one of the components of your scrolling index is unknown and the data source does not provide a consistent sort order for unknown values, you can force scrolling by using the USE–INDEX phrase with another index whose unique components do not include the unknown value. See your data-source documentation for information on how your data source handles unknown values during sorts.

The ODBC DataServer performs data retrieval against a data source that will allow records in its result sets that have null values in any of its index components. To prevent records in result sets that have null values in their index components, use the OpenEdge startup switch -znnic when you start your OpenEdge client.

**Zero-length character strings**

When you use the unknown value in a WHERE clause with the DataServer, the unknown value satisfies only the equals (=) operator. You can also use a zero-length character string in a WHERE clause. The unknown value and zero-length character string are not the same. The unknown value translates to a NULL, which is a special marker in a data source that supports NULLs used to represent missing information. On the other hand, zero-length strings and blank columns contain actual values. Both of the following statements find the first customer record with a zero-length string in the address2 field.

Notice the space between the quotation marks in the first statement:

```
FIND FIRST customer WHERE customer.address2 = " ".
FIND FIRST customer WHERE customer.address2 = "".
```

Although "" and " " evaluate the same way in a WHERE clause, they have different results when you use them with the BEGINS function. For example, the following statement retrieves all customer names except those that have the unknown value:

```
FOR EACH customer WHERE customer.name BEGINS "":
```

The following statement uses " " to retrieve only those names that begin with a space:

```
FOR EACH customer WHERE customer.name BEGINS " ":
```
The following statement is not meaningful to an ODBC data source. It generates the error message “Illegal operator for unknown value or zero length character string:”

```
FIND FIRST customer WHERE customer.address2 > ?.
```

This restriction has been relaxed for columns of the DATE data type. For example, the following statement is valid:

```
FIND FIRST order WHERE order.orderdate > ?.
```

DB2 UDB considers all zero-length character strings as equal to a single space. Therefore, DB2 UDB considers "" and a string of blank spaces to be effectively the same thing.
Record creation

Record creation is handled differently for OpenEdge databases and ODBC data sources accessed through the DataServer. The difference occurs because ABL code run against an OpenEdge database follows different record-scoping rules than ABL code run against an ODBC data source.

Records are scoped to the outermost block in which they are used, or the smallest enclosing block that encompasses all references to a record. In the absence of enclosing blocks, a record is scoped to the procedure block. Outside of record scope, a buffer associated with a record is cleared by ABL.

The OpenEdge database writes a record to the database after all its indexed columns are assigned. The ODBC DataServer writes records or record changes out to the database at the end of record scope or at the end of a transaction boundary within that record scope. For this reason, the duration of record locks can vary between OpenEdge and an ODBC data source since the record lock will be held from the time of the write until the end of a transaction.

While OpenEdge and ODBC data sources both commit changes at a transaction boundary, OpenEdge is capable of downgrading the lock on a record within scope after a commit boundary has been traversed. An ODBC data source drops all record locks at the end of a transaction. See the “Transaction scoping and buffer management” section on page 2–24 for more information.

The following ABL statements will force a physical database write to an ODBC data source before the end of a transaction block or the end of record scope:

- `RELEASE`
- `VALIDATE`
- `RECID/ROWID`

While these statements can change lock status and force a write, they have no effect on transaction scope. Locks will be held from the time of the write until the time of a commit or rollback. For the `RELEASE` statement, the record is released from its buffer but is inaccessible and still locked awaiting a transaction boundary. After a `RELEASE`, a record would need to be reread in order to refill buffer content. The OpenEdge database waits for all the key values to be assigned or for the end of record scope to write a record regardless of the outcome of a `VALIDATE` statement. The ODBC DataServer writes a record at the time of the `VALIDATE` since a write is required in order to resolve the integrity of a record within the ODBC data source. OpenEdge can assign a `RECID` or `ROWID` without writing a record. For an ODBC data source, the generation of `RECID` and `ROWID` are contingent on a record write in order to produce its value.

The availability of database changes is dependent on when the record is written. The user writing the record can read back the in-memory copy of the record as soon as it has been written out to the database. Another user trying to access the same record may or may not see the changes written to the database, depending on the transaction isolation level of the foreign data source. While OpenEdge will show other users records modified but not yet committed, an ODBC data source will not report on the state of record content until a transaction boundary has been traversed. Therefore, while OpenEdge might indicate that a record already exists or has certain content before another user has committed such changes, an ODBC data source will report that the record is in use and block access until the transaction is complete. The code fragments in this section illustrate these differences.
Suppose that you have a table called customer with a field called custnum that is defined as an indexed field, and you write the following procedure:

```plaintext
DO TRANSACTION:
  CREATE customer.
  ASSIGN
customer.name = "SMITH"
customer.custnum = 10
customer.address = "1 Main St".
END.
```

When you run this procedure:

- The OpenEdge database does not create the record at the CREATE statement. Instead, it writes it to the database at the end of the record scope or when the index information is supplied, whichever occurs first. In this example, the OpenEdge database writes the record after executing the statement custnum = 10.
- The DataServer writes the record later, at the end of the record scope. In this example, it writes the record after executing the statement END.

The following procedure, which uses multiple buffers for the same record, illustrates the differences between the OpenEdge database and DataServer record creation:

```plaintext
DEFINE BUFFER bfCustomer FOR customer.

CREATE customer.
customer.custnum = 111.

FIND bfCustomer WHERE bfCustomer.custnum = 111.
DISPLAY bfCustomer.
```

In this procedure, the code creates a customer, sets custnum equal to 111, then finds and displays the customer record using custnum (the unique index). In this case:

- The OpenEdge database displays the customer 111 record.
- The DataServer fails to find customer 111 because it has not yet written the record for customer 111 to the data source.

To get a consistent response from the DataServer, use this procedure instead:

```plaintext
DEFINE BUFFER bfCustomer FOR customer.

CREATE customer.
customer.custnum = 111.
VALIDATE customer. /* or RELEASE customer. */

FIND bfCustomer WHERE bfCustomer.custnum = 111.
DISPLAY bfCustomer.
```
The VALIDATE or RELEASE statement causes the DataServer to write the customer record to the database before the FIND statement occurs. Using the VALIDATE or RELEASE statements forces a write to the data source even if the transaction has not yet completed. This makes the record available in the local database cache and it will be found if the record must be re-read before the end of the transaction.

**Note:** If you set the default value when creating a record, you must change the value before you create another record with the default value if the field is part of a unique key. Otherwise, the second record will cause a duplicate key error.

Record updates are handled similarly to record creation. A record is updated in an ODBC data source at the end of record scope or at the end of a transaction, whichever comes first. For example, when you run the following procedure, the newly updated record is not found:

```sql
FIND customer WHERE customer.custnum = 12.
DO TRANSACTION.
  ASSIGN customer.address = "14 Oak Park".
  FIND customer WHERE customer.address = "14 Oak Park".
END.

FIND customer WHERE customer.custnum = 12.
DO TRANSACTION:
  ASSIGN customer.address = "14 Oak Park".
  VALIDATE customer.
  FIND customer WHERE customer.address = "14 Oak Park".
END.
```

To send the record to the data source sooner, use the VALIDATE statement, as follows:

```sql
FIND customer WHERE customer.custnum = 12.
DO TRANSACTION:
  ASSIGN customer.address = "14 Oak Park".
  VALIDATE customer.
  FIND customer WHERE customer.address = "14 Oak Park".
END.
```

For more information about record scoping and transaction behavior, see *OpenEdge Getting Started: ABL Essentials.*
Data-source record locking

In a DataServer application, an ODBC data source handles all of its own locking issues. ABL locking rules do not apply when you access information from an ODBC data source.

Table 2–3 provides data-source-specific comparisons.

Table 2–3: ABL and data-source locking

<table>
<thead>
<tr>
<th>ABL</th>
<th>Data source</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO–LOCK</td>
<td>Can support the NO–LOCK option in a manner consistent with Progress when transaction isolation levels are set properly.¹,²</td>
</tr>
<tr>
<td>SHARE–LOCK</td>
<td>May support shared locks at the table, page, and record level. However, the scope and duration of Progress and data-source shared locks may differ depending on how data-source cursors behave at a transaction boundary and how isolation levels are set. For more information, see your data-source documentation.</td>
</tr>
<tr>
<td>EXCLUSIVE–LOCK</td>
<td>Can support exclusive locks at the table, page, and/or record level, depending on the data source.</td>
</tr>
</tbody>
</table>

¹. DB2 UDB does not support an equivalent lock type. In a DB2 UDB application, the NO–LOCK condition option is the equivalent of a SHARE–LOCK option. As a result, you might receive the message “record locked.”
². Sybase supports the NO–LOCK option in a manner consistent with Progress except for indexed columns. Record-level locking remains in effect when you access indexed columns.

Each data source uses locks or optimistic concurrency control automatically to isolate users from each other in a multi-user configuration. Your data source and ODBC driver may provide one or a number of transaction isolation levels. In a multi-user configuration, you can isolate users from each other in your data source by setting the isolation level (if the ODBC driver permits). In your OpenEdge schema holder, use the –Dsrv TXN_ISOLATION,n connection parameter (where n = 1, 2, 4, or 8) to set the isolation level in ODBC. See the Microsoft ODBC Programmer’s Reference for more information and the reference manuals provided by your data source vendor for supported isolation levels.

Note: Certain earlier versions of DB2 UDB and Sybase use page-level locking rather than record-level locking. This can affect data access when two or more users attempt to read or update different records that are on the same page. See your data-source documentation for details.

Table 2–4 shows the possible –Dsrv TXN_ISOLATION,n values with the respective meaning.

Table 2–4: TXN_ISOLATION values in the –Dsrv parameter

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Read uncommitted (default)</td>
</tr>
<tr>
<td>2</td>
<td>Read committed</td>
</tr>
<tr>
<td>4</td>
<td>Repeatable read</td>
</tr>
<tr>
<td>8</td>
<td>Serializable</td>
</tr>
</tbody>
</table>
Share locks

ODBC defines four isolation levels in the following order from least to most restrictive: read uncommitted, read committed, repeatable read, and serializable. These isolation levels will determine what kind of share locks will take effect against a given data source. Not all data sources support all levels.

ODBC data sources do not respond to the SHARE-LOCK option in the ABL FOR EACH and FIND statements. Instead, share locks are governed by the data source and the available ODBC isolation levels allowed by that data source. A SHARE-LOCK specification does affect how the ODBC DataServer accesses records however. For greatest DataServer efficiency, it is always recommended that the NO-LOCK option be specified in your ABL if you do not need to lock records for update. Record isolation will still be managed by the data source irrespective of the lock mode chosen. If you wish to change the share lock behavior of your data source, you may be able to do so by changing the isolation level at connection time using the –DSrv parameter TXN_Isolation.

When you read records with a FOR EACH or FIND statement, regardless of whether you include the SHARE-LOCK option, the ODBC data source typically behaves as follows:

- It puts some form of shared lock on the record, page, or table if the ODBC isolation level is anything other than read uncommitted. This occurs regardless of whether the share lock is specified in the ABL statement.

- After the data source reads the record, it releases the shared lock if the isolation level is read uncommitted or read committed. It may hold share locks until the completion of a transaction if the isolation level is repeatable read or serializable.

If a record has a shared lock on it, other users can usually access that record and apply a shared lock, but this is dependent on the isolation level and DataServer locking behavior. Refer to the transaction and locking references in the Microsoft ODBC Programmer’s Reference or data source reference manuals for more information.

Exclusive locks

When you update, delete, or create a record, an ODBC data source puts an exclusive lock on the record; however, the data source does not apply the exclusive lock to a record until all shared locks on it are released. Therefore, you cannot perform an update on a record until other users release it. If a record has an exclusive lock on it, no other user can access it until it is released at the end of a transaction. In a transaction block, the data source always holds an exclusive lock until the end of a transaction’s scope if the data-source driver supports commitment control boundaries and the ODBC autocommit feature is not turned on.

Exclusive locking significantly affects the performance of data sources that use page-level locking, especially if the transaction requires a large number of exclusive locks on non-contiguous records.

NO-LOCK

The DataServer can emulate the OpenEdge NO-LOCK behavior if the isolation level is set to read uncommitted at connection time and you specify the NO-LOCK syntax in your ABL statement.
Locking impact on queries

The DataServer processes query statements marked with SHARE-LOCK and EXCLUSIVE-LOCK queries very differently from NO-LOCK queries.

Note: There is an exception to the previous statement. Queries marked SHARE-LOCK when the record isolation level is read uncommitted are treated like a NO-LOCK.

The records of a NO-LOCK query are pre-fetched, which means that changes to the result set after the query has been executed are unknown. Records modified in the database after the query is established will not be reflected in the result set. SHARE-LOCK and EXCLUSIVE-LOCK queries do not pre-fetch records, but do preselect record keys. The query behavior is that of a “keyset-driven” cursor. Since the record keys are stored at the time an ABL query is first executed, new records added to the database since the key list was established are not included in the query result set. However, records that have been deleted or modified in the database will be reflected in the query results as records are accessed by the application.

Locking examples

The following example illustrates how the end of a transaction affects an OpenEdge database and ODBC data-source locks differently:

```ln
DO TRANSACTION:
  FIND customer WHERE customer.custnum = 10.
  UPDATE customer.
END.

• When you access an OpenEdge database with this procedure, the customer record is share-locked when the first transaction ends.

• When you access an ODBC data source with the DataServer, the customer record is released when the first transaction ends.

This example illustrates how an OpenEdge database and ODBC data-source shared locks differ in scope and duration:

```ln
FIND customer WHERE customer.custnum = 10 SHARE-LOCK NO-WAIT NO-ERROR.
IF AVAILABLE customer THEN DO:
  DISPLAY customer.
  PROMPT-FOR customer.
ELSE DO:
  ASSIGN customer.
  LEAVE.
END.
END.
```
In this example, the first record is only share-locked within the ODBC data source if the isolation level setting requires it. (Recall that a share lock specified in an ABL statement is ignored by the DataServer.) As a result, the first record might be updated before the second FIND statement executes, in which case the record that the second FIND statement fetches might be different from the record fetched by the first FIND statement. This procedure might cause update information to be lost because the procedure applies updates based on the first record, and these updates will overwrite the values in the second record.

Using the DataServer to access an ODBC data source ensures that locks are upgraded in the data source in the same way as in an OpenEdge database. For example, the following procedure causes the same behavior whether you access an OpenEdge database or an ODBC data source:

```abl
FIND customer WHERE customer.custnum = 10.
DISPLAY customer.
PROMPT-FOR customer.
DO TRANSACTION:
   ASSIGN customer.
END.
```

The record is share-locked when it is fetched. The DataServer upgrades the shared lock to an exclusive lock inside the transaction by locking the record, reading it, and checking whether the record has changed since it was first fetched. If it has changed, the lock upgrade fails and you receive an error message.

If your data source uses record-level locking, you might have to wait to access a record in the following circumstances:

- You try to update a record when another user is reading it (it is share-locked). This also depends on the isolation level.
- You try to read or update a record when another user is updating it (it is exclusive-locked).

When this happens, Progress uses a time-out loop, checking periodically to see whether the record is available. You can choose Cancel at any time to abort the request.

The ODBC data source notifies the DataServer if it cannot perform a requested operation within a given period of time. Under unusual system or network loads, the DataServer might receive notification that a request has not been completed. In this case, it returns a message that the record that the request was accessing is locked, even though no other user has a lock on the record.

One type of locking behavior that you might encounter is a deadlock, or “deadly embrace.” A deadlock occurs when two users want to access each other’s table, page, or record, and either the table, page, or record that they want has an exclusive lock on it or they need to put an exclusive lock on it. Neither table, page, or record will give up its lock until the other is available. When an ODBC data source detects this situation:

- The data source kills the transaction that has accumulated the least amount of CPU time and releases the table, page, or record for the other user.
- Progress displays a message that the transaction was killed.
- The system responds as if you had chosen Cancel.
Note: On DB2 UDB for AIX, the DBA must set the variable LOCKTIMEOUT to some number of seconds in the database configuration to get beyond a deadlock condition. The default value (-1) causes a user to wait indefinitely for a lock to be released when attempting to update a locked record.

For details on how Progress locks work, see *OpenEdge Getting Started: ABL Essentials*. See ODBC and data-source documentation for more information about ODBC and data-source locks.

### The NO-WAIT option

The ABL NO-WAIT option specifies not to wait for a record that is currently locked by another process. Some ODBC data sources support this option and some do not. The following sections provide details.

**DB2 UDB**

DB2 UDB does support the NO-WAIT option when the LOCKTIMEOUT option is set to zero.

**Sybase**

The NO-WAIT option works for DataServer applications in the same way that it works for OpenEdge database applications: the DataServer uses a time-out mechanism. If Sybase does not return a record within a set period of time, the DataServer considers the record to be locked. It then cancels the request to Sybase and sets the “locked” and “not available” conditions.

During a period of heavy demand, you might encounter situations where the “not available” condition is set although the record is not currently locked by a user. In this case, you might want to increase the time-out interval by using the `-Dsrv RESP_TIMEOUT` parameter.
Transactions

With DataServer operations, an ODBC data source handles its own transaction roll back and recovery operations. However, if the ODBC driver supports transaction processing, the transaction scoping rules apply: a transaction ends when the code exits the outermost block that performs an update. With the DataServer:

- When a transaction that updates an ODBC data source ends successfully, ABL sends a COMMIT to the data source.
- If you interrupt the transaction, ABL sends a ROLLBACK to the data source.

See OpenEdge Getting Started: ABL Essentials for details on how Progress handles transactions and error conditions.

Transaction scoping and buffer management

The DataServer and an OpenEdge database hold locks differently at transaction boundaries. The OpenEdge database will hold a lock past the transaction boundary where the DataServer will not. This causes different behavior in the use of buffers. Consider the following example:

```
DEFINE BUFFER st_buf1 FOR state.
DEFINE BUFFER st_buf2 FOR state.

DO TRANSACTION:  /* Transaction 1 */
   FIND st_buf1 WHERE st_buf1.state = "NH" EXCLUSIVE-LOCK.
   DISPLAY st_buf1.state-name.  /* state-name = "New Hampshire" */
END.

/* During this PAUSE, state-name is changed from "New Hampshire" to "Granite State" by an external user */
PAUSE MESSAGE " state.state-name ".

DO TRANSACTION:  /* Transaction 2 */
   FIND st_buf2 WHERE st_buf2.state = "NH" EXCLUSIVE-LOCK.
   DISPLAY st_buf1.state-name st_buf2.state-name.
END.
```

Executing the above code against an OpenEdge database will downgrade the EXCLUSIVE-LOCK in Transaction 1 to a SHARE-LOCK. This occurs at the end of Transaction 1 and remains in effect at the time of the PAUSE statement and prior to the start of Transaction 2. This SHARE-LOCK prevents another user from modifying the state-name value between the transactions. As an optimization, when Transaction 2 is executed, the client does not refresh the st_buf1 buffer since the SHARE-LOCK prevented its contents from becoming stale between transactions. Also, since st_buf2 will have the same data integrity as st_buf1, they share the same buffer content.
Transactions

Executing the previous code against the DataServer will have different results because the DataServer does not retain any lock conditions beyond the scope of the transaction boundary. The DataServer will release the EXCLUSIVE-LOCK on the record at the END statement of Transaction 1. This leaves the record exposed to modification by another client during the PAUSE statement. If another client modifies the state-name value to “Granite State” during the PAUSE, Transaction 2 will read the updated value during the FIND statement. However, because of OpenEdge buffering rules and record scoping, neither buffer is refreshed with the updated value and the DISPLAY statement displays “New Hampshire New Hampshire.”

To avoid this type of problem, the following workarounds are available:

- Change the lock status in Transaction 1 to NO-LOCK if an EXCLUSIVE-LOCK is not required. The lock upgrade in Transaction 2 will force the buffer to be updated in this transaction.

- If the lock can not be changed in Transaction 1, release the first buffer before the start of Transaction 2 with the statement “RELEASE st_buf1.” The RELEASE statement will force a refresh of the data when the FIND in Transaction 2 executes.
Error handling

One common data-entry error is attempting to add a duplicate record; that is, trying to create a record using a unique key that already exists in the database. Suppose, for example, that a user tries to add a customer with custnum = 1 (where custnum is a unique key), but a customer record with this custnum value already exists. The attempt fails and ABL generates an error.

When this type of error occurs, ABL tries to resolve it by working back through the procedure, looking at each block header until it finds the closest block with the error-handling property, and then undoing and retrying the block. (See *OpenEdge Getting Started: ABL Essentials* for more information about error handling.) However, because the DataServer is accessing a non-OpenEdge data source, OpenEdge cannot detect duplicate-key errors until the end of a transaction block. Therefore, if an error occurs in a subtransaction, ABL cannot detect it until the end of the entire transaction block and must perform default error handling for the entire transaction block.

The following example illustrates ABL and DataServer error handling:

```
rep-blk:
REPEAT:
  PROMPT-FOR customer.custnum. /* User input */
  FIND customer USING customer.custnum NO-ERROR.
  IF AVAILABLE customer THEN
    UPDATE customer.custnum customer.name customer.state. /* User input */
  END.
  DO ON ERROR UNDO do-blk, RETRY do-blk:
    FIND state WHERE st.state = customer.state.
    DISPLAY state.
    SET state. /* User input */
  END.
END.
```

This procedure displays the following screen, in which the user is prompted to enter data into the custnum field and then the state field:

<table>
<thead>
<tr>
<th>Cust-Num</th>
<th>Name</th>
<th>State</th>
<th>State-Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Lift Line Skiing</td>
<td>MA</td>
<td>Massachusetts</td>
</tr>
<tr>
<td>East</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OpenEdge prompts for the state abbreviation.

Suppose that the user enters an existing state (for example, NH) while Progress is processing the DO block. When this duplicate-key entry occurs for an OpenEdge database, Progress returns control to the DO block, displays a message that the record exists, and reprompts the user for a state abbreviation.
However, with the DataServer, if a duplicate key entry occurs in the \texttt{DO} block, Progress returns control to the \texttt{REPEAT} block rather than the \texttt{DO} block. As a result, the procedure reprompts the user for a customer number after the inner transaction completes:

\begin{verbatim}
CREATE customer NO-ERROR.
ASSIGN customer.custnum = 45 NO-ERROR.
ASSIGN customer.name = "Smith" NO-ERROR.

DEFINE VARIABLE ix AS INTEGER NO-UNDO.

DO TRANSACTION:
  CREATE customer NO-ERROR.
  ASSIGN customer.custnum = 45 NO-ERROR.
  VALIDATE customer.
  IF ERROR-STATUS:ERROR THEN DO:
    MESSAGE "error: number of messages = " ERROR-STATUS:NUM-MESSAGES.
    DO ix = 1 TO ERROR-STATUS:NUM-MESSAGES:
      MESSAGE "error" ERROR-STATUS:GET-NUMBER(ix)
      ERROR-STATUS:GET-MESSAGE (ix).
    END.
    UNDO, LEAVE.
  END.

  ASSIGN customer.name = "Smith" NO-ERROR.
  VALIDATE customer.
  IF ERROR-STATUS:ERROR THEN . . .
  END.
\end{verbatim}

This code returns data-source errors after the \texttt{VALIDATE} statement.

Another difference in behavior occurs when two users simultaneously attempt to create records with duplicate keys. Progress raises an error immediately, but the data source raises an error after the first transaction commits and only if the second transaction does not roll back. To avoid this difference, either change the scope of the transaction so that it completes more quickly or make the key nonunique and enforce uniqueness at the application level. Another technique is to use a \texttt{RELEASE} or \texttt{VALIDATE} statement when you check for the key’s uniqueness.

\begin{verbatim}
CREATE customer NO-ERROR.
ASSIGN customer.custnum = 45 NO-ERROR.
ASSIGN customer.name = "Smith" NO-ERROR.

DEFINE VARIABLE ix AS INTEGER NO-UNDO.

DO TRANSACTION:
  CREATE customer NO-ERROR.
  ASSIGN customer.custnum = 45 NO-ERROR.
  VALIDATE customer.
  IF ERROR-STATUS:ERROR THEN DO:
    MESSAGE "error: number of messages = " ERROR-STATUS:NUM-MESSAGES.
    DO ix = 1 TO ERROR-STATUS:NUM-MESSAGES:
      MESSAGE "error" ERROR-STATUS:GET-NUMBER(ix)
      ERROR-STATUS:GET-MESSAGE (ix).
    END.
    UNDO, LEAVE.
  END.

  ASSIGN customer.name = "Smith" NO-ERROR.
  VALIDATE customer.
  IF ERROR-STATUS:ERROR THEN . . .
  END.
\end{verbatim}
Overflow checking

Overflow checking is a process by which the DataServer evaluates whether the value defined for a data type exceeds the data type’s capacity. If the value is greater than the bit size defined for a data type, an error message occurs.

Beginning with OpenEdge Release 10.1B, all in-memory operations on INTEGER and INT64 data types are 64-bit, promoting potential conflicts when data is exchanged between OpenEdge Release 10.1A and earlier OpenEdge clients, and 10.1B and later ODBC DataServers. For example, if a networked 10.1B DataServer produces an INT64 data type that is sent to a pre-10.1B client, and the INT64 data type is mapped to a 32-bit data type that exceeds 32-bits of data storage, the ODBC data source will detect an overflow condition and an error message appears on the client machine.
Cursors

A cursor points to consecutive records in a table. ABL uses cursors to keep track of where it is in a table; for example, when it processes FOR EACH statements. Suppose that you are reading records from the customer table using the custnum index, and your current record is customer number 50. This means that ABL has a cursor positioned at custnum 50. Note that ABL maintains cursor positioning across queries.

The DataServer allows applications that access ODBC data sources to imitate OpenEdge cursor behavior for FIND cursors. FOR EACH and OPEN QUERY statements do not retain cursor position across other queries or against a FIND statement.

This section discusses the following cursor-related activities:

- Forward and backward scrolling
- Block cursors

For details about firehose and fast forward-only cursors, see the “Firehose and Fast Forward-Only Cursors” section on page 4–5.

Forward and backward scrolling

A query is scrolling if you specify SCROLLING in the DEFINE QUERY statement or if you define a browse for the query. You can use the REPOSITION statement to change your current position with the result set. For a non-scrolling query, the DataServer allows you to only move sequentially forward through the rows by using the FIRST and options of the GET statement.

Scrolling basically means that the result set is cached on the client for backward and forward scrolling.

With the DataServer, forward and backward scrolling works with an ODBC data source just as it does with an OpenEdge database except when the data source fails to find a record. In ABL, the cursor position can change after a failed search; however, with the DataServer, a failed search does not affect the cursor. For example, if a cursor is positioned at custnum 50 and a request for the next customer from an OpenEdge table fails, the cursor moves to the next sequential record beyond custnum 50. The same failed request from a data source leaves the cursor at custnum 50. Your DataServer applications should not assume a certain cursor position after a failed find.

You can, however, still use the AVAILABLE function to determine whether a record is available. The AVAILABLE function behaves consistently whether it accesses an OpenEdge database or an ODBC data source.
Impact of MAX-ROWS

The MAX-ROWS qualifier on a query determines the number of rows returned. The MAX-ROWS calculation is applied before any sorting is applied to the query, causing the results to be random. Consider the following example:

```
DEFINE QUERY qCustomer FOR customer SCROLLING.
OPEN QUERY qCustomer FOR EACH customer MAX-ROWS 15.
GET LAST qCustomer.
DISPLAY customer.custnum.
REPEAT:
    GET PREV qCustomer.
    DISPLAY customer.custnum.
END.
```

When this example is run with the OpenEdge sports database, the first record return has custnum 54, and the repeat loop displays other custnums in descending order from there. When this example is run with the sports database migrated to an ODBC data source, the first record returned has custnum 15, and the repeat loop displays values in descending order from there. This difference in results is a caused by the MAX-ROWS being applied prior to the sorting.

Block cursors

Block cursors are available as a performance enhancement. They allow blocks of records to be bound and fetched together from a single database request, thus reducing the network overhead and minimizing record copies for ABL queries and finds, and minimizing record copies.

Additional details about block cursor behavior

DataServer connections that run at the read uncommitted isolation level will also execute queries that specify the SHARE-LOCK condition as NO-LOCK queries and will also utilize block cursors for their result sets. You can shut off block cursor behavior and revert back to look-ahead cursors for NO-LOCK queries by setting the -Dsrv PRGRS_BLOCK_CURS,0. If you leave block cursors on but wish to turn off the block and/or look-ahead cursor optimizations for a specific query, you can set the QUERY-TUNING option QUERY-TUNING(NO-LOOKAHEAD) on your ABL statement.

While block cursors increase efficiency, they might also require more memory than look-ahead cursors. Block cursors will try to reuse memory from previous result sets whenever possible. You can adjust the memory available to block cursors according to your specifications. Adjust the cache size of an individual block with the same connection-level parameter you used to set the look-ahead cache size. The -Dsrv QT_CACHE_SIZE option allows you to specify at connect time how large the block cache should be for individual query results. When block cursors are active, the default QT_CACHE_SIZE is set to 10,000 bytes. When look-ahead cursors are active, this size defaults to 30,000 bytes.

You can override the connection-level QT_CACHE_SIZE at the statement level by setting the QUERY-TUNING option called CACHE-SIZE on the query itself. Determining the proper block size should be based on the maximum length of your returned records multiplied by the expected number of resultant rows and then compared to your available memory. Whenever an existing block is not reused, the block of memory allocated for the query will be adjusted downward to an exact multiple of the number of rows that can fit into the allocated area.
To prevent large applications from potentially overutilizing memory for block cursors, two other settings can be adjusted to limit the amount of memory available to block cursors. The first is the maximum block memory usage per table. If you have multiple queries open against the same table simultaneously, each query uses its own query block. The `-Db iv
PRGRS_TABLE_BLOCKSIZE` option puts an upper limit on the total memory available to query blocks of an individual table. This number should normally be set larger than the `QT_CACHE_SIZE` value. It can be set as high as two or three times the `QT_CACHE_SIZE`. If the maximum block memory available to the table will be exceeded by allocating space for the current `NO-LOCK` query in your ABL, the query is instead executed with a look-ahead cursor. The default maximum block memory area per table is set at 65,000 bytes.

The second adjustment switch available is the `-Db iv PRGRS_MAX_BLOCKSIZE` option. This value sets an upper limit to the overall memory provided to block cursors, irrespective of per table allocations. The default maximum is 1048576 bytes (or 1MB). Your accumulated total memory allocated for block cursors will not exceed this value. If allocating block cursor memory for a `NO-LOCK` query in ABL will cause this limit to be exceeded, the query would instead be executed with a look-ahead cursor. The `PRGRS_MAX_BLOCKSIZE` value should be set with respect to the amount of memory available on your machine. If you are running in client-server mode, the number should be set with respect to both the available memory on the server machine as well as the number of clients that will be connecting.

The following example uses the state table in the demo database, which has a maximum record size of 84 bytes:

```sql
FOR EACH state NO-LOCK QUERY-TUNING(CACHE-SIZE 850):
    DISPLAY state.
END.
```

In this example, the `QUERY-TUNING CACHE-SIZE` value overrides the `-Db iv QT_CACHE_SIZE` query block size default of 10,000 bytes. Ten records at 84 bytes per record totals 840 bytes in the query block. If a previously established block is located for that table that is greater than or equal to the requested size, it will be reused. However, if a new block is established instead, its memory allocation will be adjusted downward to 840 bytes in order to fit an exact multiple of rows in the block. Then, if the result set contains 30 records, the query block will get refilled three times before the end of the result set if read sequentially.

**Block cursors on versus off**

Block cursors are the default over look-ahead cursors when `PRGRS_BLOCK_CURS` is enabled and a server-side cursor is needed. The switch `-Db iv PRGRS_BLOCK_CURS,0` will turn off block cursors on ODBC data sources. You can always disable block cursors for a specific query by disabling lookahead with a query tuning option: `QUERY-TUNING(NO-LOOKAHEAD)`. Block cursors is not a one-size-fits-all feature. There are application environments where the default values for block allocation sizes can result in either under-utilization or over-utilization of memory. See the “Using the block cursor switches” section on page 8–11 for information on tuning the amount of memory that block cursors utilizes.
For any given query, the use of a block cursor or a look-ahead cursor is mutually exclusive. In certain circumstances, block cursors will downgrade to look ahead. These are the criteria:

- When `setrv BINDING,0` is set. When binding is off, block cursors are disabled.
- Predictable single record result sets—`FOR FIRST/FOR LAST` or dynamic find operations.
- The `PRGRS_TABLE_BLOCKSIZE` has been exceeded and all existing block table space is in use by open cursors.
- The accumulation of memory allocated to all block cursors has reached `PRGRS_MAX_BLOCKSIZE`.
- The `QT_CACHE_SIZE` query-tuning (`CACHE-SIZE`) value is not larger than two times the maximum row size. At least two rows need to fit in the result block to use a block cursor.

**Note:** The macro `BLOCKBUF_AVAILABLE` can be used to determine if the current query is or is not using a block cursor.

---

**Block versus lookahead cursors**

Block cursors and lookahead cursors are both associated with `NO-LOCK` queries. These are queries where the lock status has been explicitly set to `NO-LOCK` or where the lock status has been explicitly set to `SHARE-LOCK` and the transaction isolation level is set to read-uncommitted. Block cursors and lookahead cursors are mutually exclusive.

Block cursors are more efficient than lookahead cursors for the following reasons:

- The results of a query are bound directly to the area from which they are copied into the record buffer on the client. Lookahead cursors copy out of a common bind area into the lookahead cache and then are copied into client record buffer resulting in multiple copies of the data.
- The fetch process used by block cursors fetches multiple rows at a time, reducing the number of driver calls and potentially reducing the total number of network round trips to the server.
- Block cursor memory is preallocated prior to fetching a result set, whereas lookahead cursors post allocate memory as records are fetched from the result set.
- Block cursors are directly linked to the result set binding feature whereas lookahead cursors have no particular dependency on whether binding or late-binding is active.
ABL issues

The following sections describe how to use various ABL statements in DataServer applications and highlights the ABL statements and functions that the DataServer does not support.

ROWID function

The ROWID data type provides a unique record identifier that is compatible across OpenEdge databases and ODBC data sources. Applications that use the ROWID function behave in the same way whether they access records in an OpenEdge database or in an ODBC data source. The ROWID function is more consistent than RECID across data sources. Progress Software Corporation recommends that you replace the RECID function with ROWID in existing applications.

The DataServer supports the ROWID function for ODBC data-source tables that have a unique index. The Data Dictionary uses an index that meets this criterion to provide values for the ROWID function. If you build your schema holder using OpenEdge compatibility from the Progress-to-ODBC utility, the Data Dictionary automatically designates a ROWID index; however, you can select a different unique index in a data-source table to support ROWID. See the “Defining the ROWID” section on page 5–31 for instructions.

The ROWID value in an ODBC data source differs from the ROWID value in an OpenEdge database in the following ways:

- In ABL, you can create a ROWID without creating a record. In DataServer applications, creating a ROWID creates a record. The following statement illustrates the difference in behavior:

  ```ABL
  CREATE customer.
  a = ROWID(customer).
  ```

  The DataServer creates a customer record using default values. After the user assigns values to the fields in that record, the DataServer updates it. When you UNDO the transaction, the DataServer deletes the record.

- The ROWID changes if the value of the unique keys in the designated index changes.

- The DataServer considers a single instance of the unknown value to be unique, so it is possible to have a ROWID with the unknown value. However, the ROWID function fails if you search for an unknown value, because a ROWID will never have the unknown value. For example, the following FIND statement fails even if there is a row with the unknown value in the designated column:

  ```ABL
  FIND customer WHERE ROWID(customer) = ?.
  ```

- If you force the creation of a record before entering the value for the designated column (for example, by committing a transaction or releasing or validating a record), the creation fails if the column cannot have NULL values. If the column can have NULL values, the DataServer assigns the new record a ROWID of NULL. However, if the column has an initial value, the DataServer creates the row with that initial value as the ROWID.
Follow these guidelines when using ROWID in applications that you want to deploy across multiple OpenEdge databases and/or ODBC data sources:

- Do not try to get a record’s ROWID value before the user assigns values to the unique keys of the record. Some DataServers use the unique key to generate a ROWID value.
- Refresh the ROWID value if a value of a unique key might have changed.
- Refresh the ROWID value after you undo a DELETE. The ROWID value might be different after the record is re-created.
- ROWID values are stable for a session, but you cannot rely on them to be the same across sessions.

For a complete description of the ROWID function, see its reference entry in OpenEdge Development: ABL Reference.

**RECID function**

For backward compatibility, the DataServer supports the RECID function for ODBC data source tables that have a unique four-byte integer column defined as the key for the ROWID/RECID index of a given table in the schema holder. Whenever the ROWID index selection for a schema holder table in the Dictionary has a multi-component key composite or is a single-component key but not a single unique integer component, the RECID function is not supported and the compiler will disallow the use of the RECID function in a WHERE clause.

**Note:** The ROWID function does not have this same restriction and is the recommended alternative for this limitation.

You can make an existing application that includes RECID behave more consistently across data sources by replacing RECID with ROWID. For more information, see “ROWID function” section on page 2–33.

**Automated ROWID and RECID support**

When working with Sybase or DB2/400 foreign data sources, you can enable ROWID and RECID support when using the OpenEdge-to-ODBC migration utility. For Sybase, selecting the option to create a RECID column will cause an integer column with unique values for each row, called PROGRESS_RECID, to be used as the ROWID/RECID. For DB2/400, you can choose to either create a RECID column for all tables (like the Sybase RECID support option does by default) or for only those tables that do not have a unique key constraint. Other ODBC data sources do not allow setting ROWID/RECID compatibility with a migration switch.

However, you can add a PROGRESS_RECID column to ODBC tables manually and when the table is updated into the schema holder it will select the added PROGRESS_RECID column for ROWID/RECID support assuming it is set up properly. This is acceptable for three possible scenarios:

1. When creating a schema holder for a data source other than Sybase or DB2/400 and the OpenEdge database you are migrating that does not have a natural unique integer key that would serve as a logical designation for RECID/ROWID.
2. Any time foreign schema is migrated to an OpenEdge schema holder, compatibility is switched off, and the criteria in the first scenario cannot be met.
3. When you are using an existing native data source whose definitions you are pulling back into an OpenEdge schema holder but did not migrate to the server originally from an OpenEdge database.

For more information on adding a PROGRESS_RECID column, see the “Modifying tables to support ROWID function for Sybase” section on page 5–33.

You can make an existing application that includes RECID behave more consistently across data sources by replacing RECID with ROWID. See the “ROWID function” section on page 2–33 for more information.

**DEFINE BROWSE statement**

The DEFINE BROWSE statement relies on a unique record identifier for forward and backward scrolling. If your ODBC data-source table does not support the ROWID function (through either a PROGRESS_RECID column or an indexed NUMBER column with unique values), you can write code that explicitly requests the default browse scrolling behavior, similar to this:

```
DEFINE VARIABLE iRow AS INTEGER NO-UNDO.
DEFINE QUERY qCustomer FOR customer FIELDS (custnum name address) SCROLLING.
DEFINE BROWSE b QUERY qCustomer DISPLAY custnum name address WITH 10 DOWN.
DEFINE BUTTON upd.

OPEN QUERY qCustomer FOR EACH customer.
ENABLE upd b WITH FRAME x.
ON CHOOSE OF upd DO:
  iRow = CURRENT-RESULT-ROW("q").
  GET PREV qCustomer.
  GET NEXT qCustomer EXCLUSIVE-LOCK.
  IF CURRENT-RESULT-ROW("qCustomer") = iRow THEN
    UPDATE customer.address WITH FRAME z VIEW-AS DIALOG-BOX.
    /* else, indicate that an error occurred: the record was deleted in the meantime. */
    DISPLAY customer.address WITH BROWSE b.
END.
WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

**Field lists**

The DataServer fully supports the use of field lists in queries (DEFINE QUERY, FOR EACH, PRESELECT, and SQL SELECT statements). Using FIELD and EXCEPT clauses can greatly improve performance for NO-LOCK queries, particularly if the DataServer is a remote configuration where the query results must be passed over a network. When a field list is used, unneeded data is not returned. The field list does not guarantee that it will restrict the data to the specified fields. It can be necessary to return additional fields such as those required to accommodate the values of a selected index.

For example, the following statement returns the same results for an OpenEdge database and an ODBC data source:

```
DEFINE QUERY myquery FOR customer FIELDS (custnum name) SCROLLING.
OPEN QUERY myquery FOR EACH customer NO-LOCK
  WHERE customer.custnum < 6 BY customer.name.
```
Include the SCROLLING option to enable GET PREVIOUS. You must include the NO–LOCK option when you open queries that are defined with field lists.

Similarly, you must include the NO–LOCK option in FOR EACH statements that include field lists, as in the following example:

```
FOR EACH customer FIELDS (custnum name) NO-LOCK:
```

Field lists are effective only when you also specify the NO–LOCK option. This option ensures that the DataServer does not have to refetch rows, which can slow performance. If a lock upgrade is required, the field list is ignored and all fields are retrieved.

Use field lists to retrieve only those fields that your application requires. For performance reasons, the DataServer retrieves the first index field even if you do not include it in the field list. In cases where the DataServer can predict that a query will require a refetch, it retrieves the entire record. The DataServer allocates memory based on the maximum size defined for a field in a record. Omitting larger fields from a query can enhance performance. In addition, combining lookahead cursors and field lists greatly improves a query’s performance.

When you specify a field that has an extent, the query returns the entire array.

When the DataServer processes a query with a field list, it caches the fields that are part of the field list and any other fields that the query specified, which you can then access without making another call to the data source. For example, the DataServer fetches the name and the zip field to process the following query:

```
FOR EACH customer FIELDS (name) WHERE customer.postalcode = 01730 NO-LOCK:
```

**Note:** Cached fields might have performance implications if you modify the record later, as the DataServer must refetch the record to place a lock on it.

If you specify a field list in a join, you might have to adjust the cache size for lookahead cursors, either with the CACHE–SIZE option in a QUERY–TUNING phrase or at the session level with the -Dsrv qt_cache_size startup parameter.

Any performance gained through field lists is lost if you use nonlookahead cursors. Lookahead and block cursors gain performance by prebinding the fields of your result set. For maximum efficiency, any LOB type should be explicitly excluded from your field list if possible because ODBC data sources do not allow those fields to be pre-bound.

Programmers are responsible for coding their applications to restrict the use of their query buffers to the fields specified by the field list. References to fields outside the field list are not caught by a compile time error. Sometimes such a reference will return a run-time error, but that is not guaranteed. The following code will return a run-time error reporting that the st field is missing from the customer buffer:

```
FOR EACH customer FIELDS (custnum name)
  WHERE customer.name = "Off the Wall" NO-LOCK:
    FIND FIRST order WHERE order.st = customer.st NO-LOCK.
END.
```
The following code will not return a run time error because the CAN-FIND expression resolves to FALSE, masking the fact that there was in fact no customer.st value to compare:

```
FOR EACH customer FIELDS (custnum name)  
    WHERE customer.name = "Off the Wall" NO-LOCK:  
    CAN-FIND(FIRST order WHERE order.st = customer.st NO-LOCK).  
END.
```

See the “Record Phrase” entry in *OpenEdge Development: ABL Reference* for more information on the FIELDS option.

### FIND statements

The DataServer fully supports the FIND FIRST, CURRENT, LAST, PREV, and NEXT statements for MS SQL Server data source tables that have unique indexes. If a table has no unique index, only the FIND FIRST and FIND NEXT statements are available for that table. The following sections provide additional information about specific FIND statements.

#### FIND NEXT statement

The FIND NEXT statement can generate different results with the DataServer when compared to the same statement executed against an OpenEdge database. A FIND NEXT statement refers to a previous FIND statement only if the WHERE clauses of each statement are identical. If the WHERE clauses are different, or if one of the statements does not have a WHERE clause, the FIND NEXT will behave like a FIND FIRST statement.

#### FIND PREV and FIND LAST statements

Applications that use the FIND PREV or FIND LAST statements work on tables in a manner consistent with the OpenEdge database. The only exception occurs when the FIND PREV or FIND LAST statement fails. In OpenEdge, the cursor is located after the last record that was scanned. In an ODBC data source, the cursor behaves as if the failed FIND had never occurred. To support these statements, a table must include support for the ROWID function (through either a PROGRESS_RECID column or an indexed NUMBER column with unique values). See the “ROWID function” section on page 2–33 for more information.

For example, the procedure `find.p` accesses OpenEdge and ODBC data sources using the same FIND and FIND PREV statements in each case, as shown:

```
/* This code accesses an OpenEdge database. */
FIND demo.customer WHERE demo.customer.custnum = 3.
FIND PREV demo.customer.
DISPLAY demo.customer.custnum
    LABEL "PROGRESS DATABASE RESULT" WITH FRAME new-frame.

/* This code accesses a MS SQL Server data source. */
FIND mssdemo.customer WHERE mssdemo.customer.custnum = 3.
FIND PREV mssdemo.customer.
DISPLAY mssdemo.customer.custnum
    LABEL "MSS DATA SOURCE RESULT" WITH COL 29.
```
When you run \texttt{find.p} with an OpenEdge table and a MS SQL Server table, you get the following results (assuming that the database has records for customer numbers 1 through 4):

\begin{table}[h]
\begin{tabular}{|c|c|}
\hline
\textbf{OPENEDGE DATABASE RESULT} & \textbf{MSS DATA SOURCE RESULT} \\
\hline
2 & 2 \\
\hline
\end{tabular}
\end{table}

If the \texttt{FIND PREV} statement fails, the cursor remains located after \texttt{customer.custnum 3} in the OpenEdge table, which was the last record scanned. In the data source table, the cursor is positioned at \texttt{custnum 2}. Failed finds do not affect cursor position in data source tables.

\section*{Compiling ABL procedures}

The \texttt{COMPILE} statement lets you compile ABL procedures and save the compilations to disk. This speeds up your application execution time since ABL does not have to recompile every time you want to run a procedure.

To compile procedures that access an ODBC data source, start up an OpenEdge client and connect to the schema holder for your target data source using the schema holder’s logical database name, then use the \texttt{COMPILE} statement. If you change the name of the schema holder after you compile a procedure, you must connect to the renamed schema holder and recompile the procedure. For more information, see the \texttt{COMPILE} Statement reference entry in \textit{OpenEdge Development: ABL Reference}.

\textbf{Note:} You do not have to connect to the target data source to compile a procedure. The schema holder contains all the information that the Compiler requires.

\section*{R-code}

R-code is generated when an ABL procedure is compiled. The compiled r-code is portable among machines. For example, r-code that you compile on a Sun machine can run on any other UNIX machine.

R-code is also \textbf{not} portable among windowing systems; that is, r-code compiled for a character application will not run under Windows and r-code compiled for Windows will not run under a character application.

R-code is also \textbf{not} portable among database management systems. ABL generates calls that are specific to a database. For example, code that you compile for a database named \texttt{Sports} will not run with a database named \texttt{mysports}.

See \textit{OpenEdge Deployment: Managing ABL Applications} applications for more details on r-code portability.

The size of r-code grows when you compile procedures against an ODBC data source as compared to compiling against an OpenEdge database. The r-code for a DataServer application contains as text the portions of SQL statements that the DataServer passes to the data source.
FOR statements

FOR FIRST and FOR LAST statements that cause the DataServer to sort data before returning it to the client behave differently when returning data from an ODBC data source than when returning data from an OpenEdge database.

BEGIN function

This section is specific to DB2 UDB data sources.

With one exception, DB2 UDB considers all zero-length character strings as equal to a single space. Therefore, it considers "" and a string of blank spaces to be the same thing.

The exception is as follows: DB2 UDB considers "" and " " to be different when you use them with the BEGINS function. For example, when you use the following statements to access DB2 UDB through the DataServer, the first statement retrieves all records from the customer table, but the second retrieves only those records in which the name begins with a blank character:

```
FOR EACH customer WHERE customer.name BEGINS "";
```

```
FOR EACH customer WHERE customer.name BEGINS " ";
```

Unsupported ABL statements

The DataServer supplies you with complete OpenEdge functionality when accessing ODBC data sources. Nearly all ABL language elements (statements, functions, and so forth) and Data Dictionary features work the same whether your application accesses a data source through the DataServer or an OpenEdge database.

Use the DBRESTRICTIONS function to find out which OpenEdge features your ODBC data sources or specific tables in those data sources do not support. For the DataServer for ODBC, DBRESTRICTIONS can return SETUSERID, SET–CURRENT–VALUE, and COUNT–OF. In certain circumstances, it can return additional values:

- It can return READ–ONLY if:
  - You connect to the schema holder for a DB2 UDB or Sybase data source in read-only mode.
  - You connect to a Sybase logical data source in read-only mode.
- If your ODBC data-source table does not include the columns necessary for supporting the RECID function, DBRESTRICTIONS also returns RECID.

See the DBRESTRICTIONS function reference entry in OpenEdge Development: ABL Reference for information on syntax.
Table 2–5 summarizes the ABL differences between OpenEdge databases and ODBC data sources.

### Table 2–5: ABL differences for OpenEdge databases and ODBC data sources

<table>
<thead>
<tr>
<th>OpenEdge feature</th>
<th>ODBC data-source difference</th>
</tr>
</thead>
</table>
| **BEGINS operator**<br>Adds a 0 to the left of the name | When you use these ABL elements to access data in an ODBC data source, you might get different results than from an OpenEdge database. Suppose, for example, that you have customers named SI and SIM and that you issue this FIND statement:  

```
FIND customer WHERE name BEGINS "SI".
```

ABL returns the customer named SI. With the DataServer, the FIND fails because the FIND statement is ambiguous. You receive the same results when you use an abbreviated index or the USING option in your query. |
| **CONTAINS operator**<br>Adds a 0 to the left of the name | This operator relates to word indexing, which the DataServer does not support. |
| **COUNT–OF function**<br>Adds a 0 to the left of the name | The DataServer does not support this function. |
| **CREATE statement**<br>Adds a 0 to the left of the name | Records that you create after opening a cursor might be invisible to that cursor. |
| **CURRENT–VALUE statement**<br>Adds a 0 to the left of the name<br>**NEXT–VALUE statement**<br>Adds a 0 to the left of the name<br>**SET–CURRENT–VALUE statement**<br>Adds a 0 to the left of the name<br>**DYNAMIC–CURRENT–VALUE statement**<br>Adds a 0 to the left of the name<br>**DYNAMIC–NEXT–VALUE statement**<br>Adds a 0 to the left of the name | The DataServer can only use these features natively at the data source for Sybase and MS SQL Server Version 6.5. Other data sources do not support compatibility with OpenEdge sequence objects. However, you can always use sequences and these corresponding ABL features from an OpenEdge database in conjunction with your DataServer application. |
| **FIND statements**<br>Adds a 0 to the left of the name<br>**FOR EACH statements**<br>Adds a 0 to the left of the name<br>**OPEN QUERY statement**<br>Adds a 0 to the left of the name | To reduce the number of records included in the results set, qualify your FIND statements and queries with a WHERE clause. |
| **MATCHES function**<br>Adds a 0 to the left of the name<br>**BEGINS function**<br>Adds a 0 to the left of the name | The DataServer does not support using the percent (%) or underscore (_) character with the MATCHES and BEGINS functions. Do not use these functions with a pattern that is not an expression, but is stored in an ODBC data source. It is theoretically possible to do this with an OpenEdge database, but using this kind of criteria results in poor performance. |
| **NO–LOCK option**<br>Adds a 0 to the left of the name<br>**SHARE–LOCK option**<br>Adds a 0 to the left of the name<br>**EXCLUSIVE–LOCK option**<br>Adds a 0 to the left of the name | An ABL EXCLUSIVE–LOCK is emulated in the ODBC DataServer. An ABL NO–LOCK can be emulated in the ODBC data source when the isolation level is set to read-uncommitted. SHARE–LOCK behavior is data-source and isolation-level dependent. See the “Data-source record locking” section on page 2–19 for more information. |
Table 2–5: ABL differences for OpenEdge databases and ODBC data sources (2 of 2)

<table>
<thead>
<tr>
<th>OpenEdge feature</th>
<th>ODBC data-source difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>ODBC data source NULL = Progress unknown value(&quot;?&quot;). Progress NULL (empty string) = one space in ODBC data source. The ODBC data source does not distinguish between a zero-length character string and a string that contains only blank spaces.</td>
</tr>
<tr>
<td>Record creation</td>
<td>In an ODBC data source, a record is created at the end of a record’s scope and <strong>not</strong> when the required index information becomes available.</td>
</tr>
<tr>
<td>SETUSERID function</td>
<td>You cannot use this function to change the login name and password of an ODBC data source.</td>
</tr>
</tbody>
</table>

1. For more information, see the “Data-source record locking” section on page 2–19.
**Stored procedures**

From the viewpoint of an ODBC-compliant data source, a *stored procedure* is a predefined procedure or set of statements or program that is defined for a specific data source according to that data source’s rules. By executing the stored procedure, you execute these statements or programs without having to enter their individual statements or code at each execution.

**Note:** Even for data sources that do not support stored procedures, you can send statements directly to your data source using the `RUN STORED-PROCEDURE send-sql-statement` syntax. See the “Sending SQL statements directly to the data source” section on page 2–52 for details.

**Defining native stored procedures to ABL**

The first time you run a stored procedure, the data-source management system creates an execution plan for it and stores the plan in the database. The next time you run the stored procedure, it runs as a precompiled procedure. This makes access to the database much quicker and more efficient than when you access it with new queries each time.

The DataServer allows you to use the ABL to run stored procedures written for an ODBC-compliant data source. The procedures must be defined according to the rules of the particular data source. All stored procedures are executed on the server machine by the native data source. See the documentation for your data source to determine if it supports stored procedures.

When you create or update your schema image, the stored procedures, functions, and packages appear in the list of accessible objects along with tables, view, and sequences. Progress allows you to run the stored procedures that you create in data sources from within Progress procedures. If your data source supports stored procedures, see your data-source documentation for complete information about creating and using stored procedures.

**Note:** In DB2 UDB, the same stored procedure can be defined with different signatures, that is, the same procedure name can be used for two different procedure definitions, called signatures, with different parameter construction. If you pull a stored procedure definition into your schema holder, which has overloaded functions, i.e., multiple signatures, all parameters defined to all signatures of that procedure will be pulled into the same table definition in your schema. You must use the Data Dictionary to manually remove parameters that are pulled from signatures you did not intend to use in ABL. The schema holder will only allow one of the procedure signatures to be defined per procedure name.
Stored procedures called from within ABL cannot return Boolean types. Table 2–6 lists issues that occur when you pass other data types as parameters.

### Table 2–6: Argument data types for stored procedures

<table>
<thead>
<tr>
<th>Progress</th>
<th>ODBC data source</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECIMAL</td>
<td>The DataServer represents all three data types as the OpenEdge INTEGER data type in the schema image. To preserve the scale and precision of these data types, you must manually update the information in the schema image for these parameters. Use the Data Dictionary to update the data type and format information in the field property sheet for the parameter.</td>
</tr>
<tr>
<td>FLOAT</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>CHAR</td>
<td>The data source represents this type as a VARCHAR parameter. Its size cannot exceed the VARCHAR size limit for the associated data source. If they exceed this limit, they cause an ODBC data-source error.</td>
</tr>
<tr>
<td>DATE</td>
<td>If you pass a DATE data type as an input parameter and use it in an equality test, the test might fail. In this case, use the trunc function in the stored procedure. For example:</td>
</tr>
</tbody>
</table>

```plaintext
    procedure x_date (indate in date, outdate out date)
    as begin
        select date_terminate into outdate from datetbl
        where trunc(hire_date) = trunc(indate);
    end;
```

If you are running several stored procedures, run them serially and process all the results from one stored procedure before you run a second one. The DataServer allows only one active request for running a stored procedure. However, you can process results from several stored procedures concurrently if you specify the DataServer startup parameter (-Dsrv PRGRS_PROC_TRAN) when you start your client session. When you run stored procedures concurrently, the DataServer uses one connection to the data source per procedure. If the stored procedures attempt to update the same record from a single client’s requests, the connections could block each other or possibly create a deadlock.

**Note:** In your DB2 UDB stored procedure, be sure to code parameters passed into your stored procedure as variable, not fixed, character strings (with a two-byte length prefixing the character value).

You run stored procedures from within Progress procedures by using the ABL RUN STORED–PROCEDURE statement. A stored procedure that you run with this statement can return three types of values:

- An integer return code, which could be a success code or a value returned by the stored procedure (defined by the data source)
- Values of output parameters that you define when you create the procedure
- Results from the database
Progress has statements and functions that allow you to use the return codes and the values of the output parameters. Table 2–7 lists these statements and functions.

### Table 2–7: Returning values from stored procedures

<table>
<thead>
<tr>
<th>ABL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOSE STORED–PROCEDURE statement</td>
<td>Retrieves the values from the output parameters you defined for the stored procedure and tells ABL that the stored procedure has ended</td>
</tr>
<tr>
<td>PROC–HANDLE function</td>
<td>Allows you to specify a handle to identify a stored procedure</td>
</tr>
<tr>
<td>PROC–STATUS function</td>
<td>Reads the return value</td>
</tr>
<tr>
<td>RUN STORED–PROCEDURE statement</td>
<td>Executes a stored procedure</td>
</tr>
</tbody>
</table>

Note that you can substitute the abbreviations CLOSE STORED–PROC and RUN STORED–PROC for the full names CLOSE STORED–PROCEDURE and RUN STORED–PROCEDURE, respectively. The remainder of this guide generally uses the abbreviated form.

See Appendix B, “Stored Procedure Reference,” for reference entries for the statements and functions described in Table 2–7.

The ABL provides two techniques for accessing the results returned from the data source by the stored procedure. You can:

- Use the proc-text-buffer that Progress supplies for the rows of results
- Define a buffer for the rows of results

After you read the values into the buffer, you can operate on them in a variety of ways. You can use the database data just as you would use information from an OpenEdge database—format it and use it for calculations.

The following sections describe how to do the following:

- Running a stored procedure
- Retrieving return codes
- Retrieving output parameter values
- Retrieving data-source results
- Defining a view to use as a buffer
The following example uses a stored procedure created in Transact-SQL for Sybase:

```
CREATE PROCEDURE pcust (@num INT, @orders INT OUT, @states INT OUT) AS
  SELECT customer.custnum, customer.name, order.ordernum FROM customer, order
  WHERE customer.custnum = order.custnum AND customer.custnum > @num
  SELECT @orders = @@rowcount
  SELECT customer.custnum, state.st FROM customer, state
  WHERE customer.st = state.st AND customer.custnum > @num
  SELECT @states = @@rowcount
  RETURN 0
GO
```

**Note:** For other ODBC data sources, use your vendor-specific SQL syntax.

This Transact-SQL code creates the stored procedure `pcust` and defines three parameters: `num`, which is an input parameter, and `orders` and `states`, which are output parameters. The procedure returns values for the output parameters to the caller after processing the results of the `pcust` `SELECT` statements. You can think of output parameters as temporary fields; that is, you can access the data in these columns using the standard notation of `tablename.fieldname`. (Note that although `pcust` is a stored procedure, its syntax is that of a table and it is stored in a table definition.) For example, you can access the data in the `orders` and `states` fields by specifying `pcust.orders` and `pcust.states`. All the parameters in the example have an integer data type.

**Note:** DB2 UDB uses external modules compiled in 3GL languages (for example, C) to implement stored procedures. Because these languages are often case sensitive, you must specify in your `CREATE PROCEDURE` statement a case-sensitive procedure name that matches the module name exported from the 3GL stored procedure. In the previous example, the procedure name `pcust`, which is stored in your dictionary as a file name that represents the stored procedure to the schema holder, is case sensitive and must match exactly the case-sensitive module name exported from your DLL.

If you perform a schema pull to retrieve a stored procedure from an existing DB2 UDB database, the procedure name will be imported into your schema holder as all uppercase characters because the DB2 UDB server is not case sensitive and rolls all object names to uppercase. After inputting the stored procedure, you should go into the Data Dictionary and modify the procedure name to the appropriate case that matches your 3GL module name. Note that the DataServer for ODBC caches the schema, which means you will need to disconnect from the schema holder and reconnect for your procedure name change to take effect.
Programming Considerations

Running a stored procedure

The ABL statement RUN STORED-PROC allows you to run a stored procedure in the ODBC data source. You must also indicate the end of a stored procedure by using the CLOSE STORED-PROC statement.

This is the syntax for the RUN STORED-PROC statement:

**Syntax**

```
- RUN STORED-PROC procedure [ NO-ERROR ]
- [ ( [ OUTPUT ] parameter, ... [ OUTPUT ] parameter ) ]
```

This is the partial syntax for the CLOSE STORED-PROC statement:

**Syntax**

```
CLOSE STORED-PROC procedure
```

For example, the following ABL code runs the stored procedure pcust:

```
DEFINE VARIABLE handle1 AS INTEGER NO-UNDO.
RUN STORED-PROC pcust handle1 = PROC-HANDLE (20, OUTPUT 0, OUTPUT 0).
FOR EACH proc-text-buffer:
    DISPLAY proc-text-buffer.
END.
CLOSE STORED-PROC pcust WHERE PROC-HANDLE = handle1.
DISPLAY pcust.orders pcust.states.
```

This code first defines an integer variable named handle1 that serves as a handle for identifying the stored procedure. If you have only one active stored procedure, you do not have to specify a handle. However, it is good programming practice to use handles to identify all of your stored procedures.

It then runs the stored procedure as follows:

- The pcust stored procedure passes the values 20, 0, and 0 to the three parameters (specifying orders and states as output parameters).
- Using a FOR EACH statement, it reads the results into the Progress-supplied buffer proc-text-buffer.
- It displays the results.

The Progress procedure next uses the CLOSE STORED-PROC statement to fetch the orders and states output parameters and then displays them. Note that the stored procedure does not return output parameter values unless you request them with the keyword OUTPUT or INPUT-OUTPUT when you execute the procedure.
You can close all stored procedures at once with the following statement:

```sql
RUN STORED-PROC closeallprocs.
```

**Note:** For Sybase, the DataServer typically maintains one connection. If your application requires that you process other queries while a stored procedure is open, use the `-dsrv qt_separate_connection` parameter or the `QUERY–TUNING (SEPARATE–CONNECTION)` option to specify that the DataServer use a separate connection for each statement that requires a cursor.

See Appendix B, “Stored Procedure Reference,” for a description of the complete syntax for the Progress statements and functions that support running stored procedures.

### Retrieving return codes

A stored procedure might return a code that provides information. For example, it might indicate whether the stored procedure was successful or whether it encountered an error condition. The following example of ABL code runs the stored procedure `pcust`. It uses the `PROC–STATUS` function and the `CLOSE STORED–PROC` statement to retrieve the return code and assign the value to the variable `stat`. The meaning of this return code is defined by the underlying data source:

```abl
/* Return status */

DEFINE VARIABLE iStat AS INTEGER NO-UNDO.

RUN STORED–PROCEDURE pcust (20, OUTPUT 0, OUTPUT 0).
FOR EACH proc-text-buffer:
    DISPLAY proc-text-buffer.
END.
CLOSE STORED–PROC pcust iStat = PROC–STATUS.
DISPLAY pcust.orders pcust.states stat.
```

**Note:** When the DataServer sends a request to execute a native stored procedure, ODBC assumes that a return code will be received. However, DB2 UDB native stored procedures do not directly support return codes. Instead, DB2 UDB simulates the use of a return code for ODBC by inserting an additional output parameter as the first parameter in your SQLDA structure parameter list. This additional parameter serves as a return value to the caller. The ODBC DataServer can read a return value properly as long as the additional parameter is accommodated for in your DB2 UDB native stored procedure. Place the value you would like to have output as your return code in this first parameter. It can then be received in Progress with the `PROC–STATUS` option just as you would for any other native stored procedure environment. You do not need to register your stored procedure in the DB2 UDB database with an additional parameter and doing so would elicit a signature violation at execution time.
Retrieving output parameter values

When you call a stored procedure, you can specify the ordered list of positional parameters or you can name them individually. To retrieve output parameter values from a stored procedure, request them with the keyword OUTPUT or INPUT-OUTPUT when you execute the procedure. When you run a stored procedure in a DataServer application, the parameters are supplied and passed using OpenEdge data types.

The following ABL procedure uses the second option for passing parameters—it passes them by name with the PARAM option:

```abl
/* Parameters by name */
RUN STORED-PROC pcust (PARAM num = 20, OUTPUT PARAM states = 0, OUTPUT PARAM orders = 0).
FOR EACH proc-text-buffer:
  DISPLAY proc-text-buffer.
END.
CLOSE STORED-PROC pcust.
DISPLAY pcust.orders pcust.states.
```

When you use PARAM to specify parameter names, you do not have to specify all parameters for the stored procedure. Instead, you can include only those parameters you want to use, in any order you choose. If the stored procedure names a default value for the parameter, you do not have to name that parameter at run time. However, you must explicitly name parameters that do not have defaults or name them when you want to pass values that are different from the default.

Retrieving data-source results

OpenEdge provides two techniques for accessing results returned from a data source by the stored procedure. You can:

- Use the proc-text-buffer that OpenEdge supplies for the rows of results.
- Define a special ODBC view on the data source to use as a buffer for the rows of results.

The following ABL procedure reads the database results from the stored procedure into the proc-text-buffer supplied by OpenEdge:

```abl
RUN STORED-PROC pcust (20, OUTPUT 0, OUTPUT 0).
FOR EACH proc-text-buffer:
  DISPLAY proc-text-buffer.
END.
CLOSE STORED-PROC pcust.
DISPLAY pcust.orders pcust.states.
```

The ABL-defined buffer, proc-text-buffer, has one character field named proc-text. The buffer accepts the returned database results, converts them to CHARACTER data type, and concatenates them into one string. The advantage of using the proc-text-buffer is that you do not have to worry about what kind of data the procedure returns. The buffer accepts any type of data, in any order. The disadvantage is that it is much more difficult to manipulate the data after you receive it. To act on anything but CHARACTER data, you must extract the data from the buffer and convert it to its original data type before you can use it.
Another benefit of the proc-text-buffer is that it holds the results from all of the SQL statements included in a stored procedure. However, a buffer that you create can hold the results of only one SQL statement.

This is the partial syntax for the DEFINE BUFFER statement that you use to create a buffer with the same characteristics of the proc-text-buffer:

**Syntax**

```
DEFINE BUFFER buffer-name FOR proc-text-buffer
```

For a complete description, see the DEFINE BUFFER entry in *OpenEdge Development: ABL Reference*.

### Defining a view to use as a buffer

If you do not use the buffer proc-text-buffer defined by the Progress ABL, you must define your own. Defining a view in the data source that can serve as a buffer allows you to retrieve database results in their original data types.

While a stored procedure can include multiple SQL statements, a buffer that you define holds the results of only a single SQL statement.

**To define a buffer:**

1. Define a view in the ODBC data source with:
   - The naming convention _BUFFER_buffername for all data sources except DB2 UDB. For DB2 UDB, use the naming convention P_BUFFER_buffername because DB2 UDB does not allow object names to start with an underscore character.
   - The same number of columns and data types that the stored procedure returns in the results set.
   - The columns in the order that the stored procedure returns them.

   For example, to return two columns with two types of values, an integer and a character string, use an SQL utility to define the following view in the data source:

   ```
   CREATE VIEW _BUFFER_custlist AS SELECT customer.custnum, customer.name FROM customer WHERE 1 = 0
   ```

   Notice that these views are defined to ensure that they never return any results. This indicates that you will not use the views as views, but as buffers. It is not necessary to define views that you will use as buffers this way, but it does allow you to distinguish quickly between views and buffers.

2. Update your schema image using the Update/Add Table Definitions DataServer utility. The utility adds the view to the list of accessible objects in the schema holder. The DataServer defines the view as a buffer that Progress can use.

   See the “Creating a schema holder” section on page 5–8 for instructions on using this utility.
This buffer defines two returned values for a stored procedure—an INTEGER and a CHARACTER value—in that order. If the data types do not match those returned by the stored procedure, the procedure returns more than two types of values, or returns the values in a different order than you specified, you receive a run-time error.

The easiest way to create a buffer that accepts data from stored procedures is to use the text of the SQL SELECT statement from the stored procedure. This ensures that you define your data types correctly and in the correct order. Use a native process such as sp_help_text to view the stored procedure from Sybase, or view procedures in the system tables appropriate for your data source.

The next example does not use the supplied buffer. Instead, it defines buffers by creating views in the data source, using the following syntax:

**Syntax**

```
CREATE VIEW _BUFFER_ buffer-name
```

These are examples of the views, created in your ODBC data source, that you can use as buffers to store the results from the stored procedure `pcust`:

```
CREATE VIEW _BUFFER_pcust_orders AS SELECT customer.custnum, 
    customer.name, order.ordernum FROM customer, order WHERE 1 = 0
```

```
CREATE VIEW _BUFFER_pcust_states AS SELECT custnum, state.st 
    FROM customer, state WHERE 1 = 0
```

The following ABL procedure shows the results of the stored procedure `pcust` being written into the new buffers `pcust_orders` and `pcust_states`:

```
/* Typed buffers */

RUN STORED-PROC pcust (20, OUTPUT 0, OUTPUT 0).
FOR EACH pcust_orders:
    DISPLAY pcust_orders.
END.
FOR EACH pcust_states:
    DISPLAY pcust_states.
END.
CLOSE STORED-PROC pcust.
    DISPLAY pcust.orders pcust.states.
```

Because two different buffers have been defined, the returned values maintain their data types instead of being converted to character strings and stored in the ABL-defined buffer `proc-text-buffer`. You can then use the returned values in calculations without first converting them back to their original data types. In addition, the two separate buffers make your output look cleaner, allowing OpenEdge to build a new default frame for the two different types of output. Reading your results into an explicitly defined buffer also allows you to manipulate the data just as you would manipulate data from an OpenEdge database; for example, with Frame phrases and FORM statements.
The next example accesses the stored procedure `pcust` twice; procedure handles (through the `PROC-HANDLE` function) identify the different results from your data source:

```abl
/* Procedure handles */
DEFINE VARIABLE handle1 AS INTEGER NO-UNDO.
DEFINE VARIABLE handle2 AS INTEGER NO-UNDO.
RUN STORED-PROCEDURE pcust handle1 =
    PROC-HANDLE (20, OUTPUT 0, OUTPUT 0).
RUN STORED-PROCEDURE pcust handle2 =
    PROC-HANDLE (20, OUTPUT 0, OUTPUT 0).
FOR EACH pcust_orders WHERE PROC-HANDLE = handle1:
    DISPLAY pcust_orders.
END.
FOR EACH pcust_states WHERE PROC-HANDLE = handle1:
    DISPLAY pcust_states.
END.
FOR EACH pcust_orders WHERE PROC-HANDLE = handle2:
    DISPLAY pcust_orders.
END.
FOR EACH pcust_states WHERE PROC-HANDLE = handle2:
    DISPLAY pcust_states.
END.
CLOSE STORED-PROC pcust WHERE PROC-HANDLE = handle1.
CLOSE STORED-PROC pcust WHERE PROC-HANDLE = handle2.
```

In this example, the results look the same as in the previous example. However, because you are running a stored procedure twice, ABL uses the procedure handles to identify the different instances. If you run more than one stored procedure in your application, you must explicitly define procedure handles for each one.

The next example shows how to use standard ABL syntax to join the results of the stored procedures with other tables in the database:

```abl
/* Join with procedure results */
RUN STORED-PROC pcust (20, OUTPUT 0, OUTPUT 0).
FOR EACH pcust_orders, EACH orderline
    WHERE pcust_orders.ordernum = orderline.ordernum:
        DISPLAY orderline.ordernum orderline.itemnum.
END.
FOR EACH pcust_states:
    DISPLAY pcust_states.
END.
CLOSE STORED-PROC pcust.
```

This example joins the order information returned from the stored procedure with the order-line information in the same database.
Sending SQL statements directly to the data source

ABL uses stored-procedure syntax to allow you to send extended SQL statements directly to a data source. The DataServer uses the `RUN STORED-PROCEDURE` statement with the `send-sql-statement` option to pass SQL statements to the data source. Although OpenEdge SQL allows you to use SQL statements, this option gives you access to your data source vendor’s SQL syntax and its extensions to SQL, such as Transact-SQL for Sybase. For example, you can issue SQL statements from within Progress procedures to modify the data definitions of your data source. You can send multiple SQL statements by concatenating them and passing the string as a single parameter to the `send-sql-statement` option.

You use the `RUN STORED-PROC` statement with the `send-sql-statement` option and pass the SQL statements as a parameter. The syntax of the statement must be valid SQL syntax for the underlying database and must follow SQL naming and case conventions. For example, this code passes a `SELECT` statement as a parameter:

```
DEFINE VARIABLE handle1 AS INTEGER NO-UNDO.
RUN STORED-PROC send-sql-statement handle1 = PROC-HANDLE
   ("SELECT name, address, city, state, postalcode
    FROM customer WHERE creditlimit >= 500").
FOR EACH proc-text-buffer WHERE PROC-HANDLE = handle1:
   DISPLAY proc-text-buffer.
END.
CLOSE STORED-PROC send-sql-statement WHERE PROC-HANDLE = handle1.
```

This example returns the name, address, city, state, and postal code for all customers whose `max-credit` is greater than or equal to $500. You must read the results into a buffer as you would with a stored procedure called by an ABL procedure. You can read the results into the `proc-text-buffer`, as in the example above, or you can define your own buffer from within your data source that accepts a data type other than the `CHARACTER` data type.

The following example illustrates returning database results into the `proc-text-buffer` and converting the results to the `INTEGER` data type:

```
DEFINE VARIABLE handle1 AS INTEGER NO-UNDO.
DEFINE VARIABLE iMax AS INTEGER NO-UNDO.
RUN STORED-PROC send-sql-statement handle1 = PROC-HANDLE
   ("SELECT MAXIMUM(custnum) FROM customer").
FOR EACH proc-text-buffer:
   iMax = INTEGER(SUBSTRING(proc-text, 1, 3)).
   DISPLAY iMax.
END.
CLOSE STORED-PROC send-sql-statement WHERE PROC-HANDLE = handle1.
```

The DataServer passes the SQL statement directly to the ODBC data source. The compiler does not process it, so errors occur only at run time and not when you compile a procedure.
Handling errors from stored procedures

The RUN STORED-PROC statement supports the NO-ERROR option. The following example shows how to trap errors within a procedure:

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO.

RUN STORED-PROC send-sql-statement NO-ERROR
("select count(*) from xxx.customer where name between 'A' and 'Z' ").

IF ERROR-STATUS:ERROR THEN DO:
   DO ix = 1 TO ERROR-STATUS:NUM-MESSAGES:
      MESSAGE "error" ERROR-STATUS:GET-NUMBER(ix)
      ERROR-STATUS:GET-MESSAGE(ix).
   END.
END.

FOR EACH proc-text-buffer.
   DISPLAY proc-text-buffer.
END.

CLOSE STORED-PROC send-sql-statement.
```
Enhancing DataServer performance

When you develop a DataServer application, you can design it either to emphasize portability across data sources or to optimize the strengths of the DataServer’s interaction with a particular aspect of the target ODBC data source. For example, you might write a query that gives you consistent results across databases or one that takes advantage of your data-source cursor management functionality.

In addition to influencing how DataServer applications perform through queries, you can control how the DataServer processes queries on a statement-by-statement basis. Some of the DataServer’s default behavior might not be optimal for the application you are designing. The QUERY–TUNING phrase and startup and connection parameters give you the ability to control query processing.

Information on query tuning appears in the following locations:

- The remaining sections of this chapter, which document the QUERY–TUNING phrase.
- For information on tuning queries at compile time and runtime, see the “Query tuning with connection and startup parameters” section on page 4–16.

Progress query types

The DataServer provides several ways to submit an ABL query to an ODBC data source:

- **OpenEdge ABL** — This approach applies to the DEFINE QUERY and FOR EACH statements. The DataServer generates SQL for each of these statements. You can use the QUERY–TUNING option to customize the queries that the DataServer passes to ODBC.

- **OpenEdge SQL SELECT** — This approach applies to the SQL SELECT statement. When you use this statement in a Progress procedure, the DataServer passes the SQL directly to the data source. This can improve performance, especially when counting records, and can also allow you to access certain types of data more effectively, such as aggregates.

- **Vendor-specific SQL** — This approach applies to stored procedures. If you want to use specialized query syntax supported only by a vendor’s SQL extensions, you can use RUN–STORED–PROC send–sql–statement to send the syntax to your ODBC data source. For example, you can use this approach to modify the data definitions of your data source from the OpenEdge client. You might also use a stored procedure to include BEGINS as a search criterion; this SQL query can result in better performance. For more information, see the “Sending SQL statements directly to the data source” section on page 2–52.

Whether your application can take advantage of the strengths of a particular approach depends on the kind of query you are writing and the kind of data you are accessing. Another factor to keep in mind when you decide which technique to use for issuing queries is whether a query is better served by being processed by the client or by the server. ABL queries are processed by the client (except in the cases of most joins); SQL SELECT statements and vendor-specific SQL extensions are processed by the server (the ODBC data-source manager).
Query tuning

How you structure a query determines how efficiently you access a database. Using your ODBC data source efficiently enhances the performance of DataServer applications. The standard approach to enhancement is using selection criteria to refine access to data, but you can further optimize the DataServer’s execution of a query by specifying the ABL QUERY–TUNING phrase.

You can include the QUERY–TUNING phrase in these ABL statements:

- **FOR EACH**
  
  Syntax
  
  ```
  FOR EACH table QUERY-TUNING(query-tuning-option
  query-tuning-option...)
  ```

- **OPEN QUERY**
  
  Syntax
  
  ```
  OPEN QUERY query QUERY-TUNING(query-tuning-option
  query-tuning-option...)
  ```

- **DO PRESELECT**
  
  Syntax
  
  ```
  DO PRESELECT table QUERY-TUNING(query-tuning-option
  query-tuning-option...)
  ```

- **REPEAT PRESELECT**
  
  Syntax
  
  ```
  REPEAT PRESELECT table QUERY-TUNING(query-tuning-option
  query-tuning-option...)
  ```

You must place the QUERY–TUNING phrase after the last record phrase. For example, place it near the end of the statement where you also place block modifier phrases such as BREAK, ON ERROR, and TRANSACTION.

You can include multiple query-tuning options in a single statement; simply separate each option from the previous one by a single space.
Table 2–8 describes the query-tuning options.

**Table 2–8: Query-tuning options**  
(1 of 2)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARRAY–MESSAGE</td>
<td>Specifies whether the DataServer sends multiple result rows in a single logical network message, thereby reducing network traffic. Default: ARRAY–MESSAGE, if the query uses a lookahead cursor.</td>
</tr>
<tr>
<td>NO–ARRAY–MESSAGE</td>
<td></td>
</tr>
</tbody>
</table>
| CACHE–SIZE integer      | Specifies the size in bytes of the cache used by lookahead cursors. A larger cache size can improve performance for queries that return a large number of records because the DataServer might need fewer SQL statements to get the results.  
Minimum: The DataServer always caches at least one record.  
Maximum: None  
Default: 30000                                                                 |
| DEBUG SQL               | Specifies whether the DataServer should print to the dataserv.lg file the debugging information that it generates for a query. Default: NO–DEBUG                                                                 |
| DEBUG EXTENDED          | Specify DEBUG SQL to print the SQL that the DataServer executes against the ODBC data source.  
Specify DEBUG EXTENDED to print the generated SQL plus additional information, such as cursor statistics.  
**Note:** In addition to EXTENDED and SQL, there are other options that can assist you in analyzing performance. |
| NO–DEBUG                |                                                                                                                                                                                                          |
| JOIN–BY–SQLDB           | Specifies whether the DataServer allows an ODBC data source to perform a join (this usually improves performance). JOIN–BY–SQLDB implies SEPARATE–CONNECTION queries that include joins.  
Default: JOIN–BY–SQLDB                                                                                                           |
| NO–JOIN–BY–SQLDB        | JOIN–BY–SQLDB is a compile-time option. A query must be compiled to use or not use this option. You can turn off the JOIN–BY–SQLDB default globally at compile time by specifying the Server Join (-nojoinbysqldb) startup parameter when you start a client session. This parameter does not override the explicit use of JOIN–BY–SQLDB in the QUERY–TUNING phrase. |
| LOOKAHEAD               | Specifies whether the DataServer uses lookahead or standard cursors. Lookahead cursors fetch as many records as fit in the allocated cache (see the CACHE–SIZE entry in this table). This reduces the number of SQL statements and network messages that are required, thereby improving performance.  
Using lookahead cursors results in behavior that is different from Progress because changes made to the records in the cache might not be immediately visible. Specify NO–LOOKAHEAD for behavior that is consistent with Progress.  
Default: LOOKAHEAD when statements use NO–LOCK or SHARE–LOCK with transaction isolation level equal to read uncommitted. |
| NO–LOOKAHEAD            |                                                                                                                                                                                                          |
All but two of the QUERY–TUNING options take effect at both compile time and runtime. The exceptions are JOIN–BY–SQLDB and NO–JOIN–BY–SQLDB, which apply only at compile time. You can override query-tuning defaults (except JOIN–BY–SQLDB) at run-time by specifying the appropriate startup parameters.

The following example shows how to use the QUERY–TUNING phrase to enhance performance, including a join that the DataServer instructs the ODBC data source to perform by default:

```
FOR EACH customer, EACH order OF customer WHERE order.ordnum > 20
  BY customer.custnum QUERY–TUNING(NO–LOOKAHEAD DEBUG EXTENDED) TRANSACTION:
```

The QUERY–TUNING options in this example specify the following:

- Lookahead cursors are not used (the NO–LOOKAHEAD option).
- The DataServer writes an extended report on the SQL statements it executes (the DEBUG EXTENDED option).
When the DataServer constructs queries for an ODBC data source, it uses the QUERY–TUNING options that you specify as guidelines. This is because there might be syntax considerations that prevent the DataServer from applying the QUERY–TUNING options as specified. In such a case, the DataServer executes the query using the most appropriate options.

**Note:** The DataServer does not issue errors or warnings if it does not apply the QUERY–TUNING options that you specify.

## Caching records

The DataServer caches results sets from an ODBC data source to enhance performance. It caches as much data as fits in its allocated cache size. Depending on what kind of cursor a query is using, the DataServer caches row identifiers or records:

- **Standard cursors** — The DataServer caches row identifiers (a PROGRESS_RECID column or other index) for the results set. Each identifier requires four bytes of cache; therefore, a results set of 100 records requires 400 bytes of cache.

- **Lookahead cursors** — The DataServer caches complete records or partial records as specified by a field list. It uses the maximum length allowed for a row as defined in the ODBC data source to calculate the record length, not the actual contents of the record. In addition to the defined row length, the record consists of a row identifier field; therefore, a row with a defined maximum length of 100 bytes requires 104 bytes of cache. If a column is longer than 256 bytes, the DataServer refetches it.

In the case of joins, each record in the cache is a result of the fields selected in the join. In addition to the record, there is a row identifier field (four bytes) for each table involved in the join. For example, a three-way join adds 12 bytes to the cache for each record.

You can affect the performance of a query by controlling the size of the cache. As queries generate different results, they benefit from different cache sizes. Generally, the larger the cache, the faster the performance. However, you must balance cache size against other memory requirements for your system. Consider also that continually adjusting cache size in an application might decrease performance as each adjustment requires the DataServer to make several calls to the data source.

To determine the optimal cache size for a query, experiment with different values for CACHE–SIZE and use DEBUG EXTENDED to generate cursor statistics in the dataserv.lg file that you can examine. Aim for minimal cursor activity. The following example sets an optimal cache size for a particular query against the Sports database:

```
FOR EACH customer, EACH order OF customer WHERE order.ordnum > 20
  QUERY–TUNING(CACHE–SIZE 20 DEBUG EXTENDED):
```
**Join by SQL DB**

For queries that include joins issued in **FOR EACH** and **OPEN QUERY** statements, the DataServer evaluates the queries and in some cases instructs the ODBC data source to perform the joins. A join performed by a data source, called a **join by SQL DB**, can improve performance; however, you receive the associated query results in an order consistent with the data source, not with ABL. To get results that are consistent with ABL, turn off **JOIN-BY-SQLDB**, either with the **QUERY-TUNING** phrase at the query level or with the Server Join (**-nojoinbysqldb**) startup parameter when you compile. If the order of returned records is important to your application, specify a sort order on the query.

For each join, the DataServer evaluates whether the ODBC data source can perform it and estimates whether doing so improves performance. To determine whether a join by SQL DB is possible, the DataServer assesses whether the following criteria are true:

- All tables in the join are in the same logical OpenEdge database; that is, they are contained in the same DataServer schema.
- Every table, except the innermost one, has a unique record identifier (ROWID or RECID support).
- The query does not include a **USING** phrase for any of the inner tables. For example, a join by SQL DB will not occur for this query:

  ```plaintext
  FOR EACH customer, EACH order OF customer USING order.ordernum:
  ```

- The query does not include a **BY** phrase that contains expressions or array fields.
- The query does not include a request for an **EXCLUSIVE-LOCK** on any of the tables in the join.
- The join does not exceed 10 levels.

To estimate whether performing a join by the data source might improve performance, the DataServer assesses whether these additional criteria are true:

- The join uses an **OF** clause or a **WHERE** clause for each of the inner table loops. For example, the following query requires a field-to-field correspondence between two tables:

  ```plaintext
  FOR EACH customer, EACH order OF customer:
  ```

- The **WHERE** clause includes either an operator or the **AND** option. The following example includes the equals (=) operator:

  ```plaintext
  FOR EACH customer, EACH order
  WHERE customer.custnum = order.custnum:
  ```
The DataServer also performs a join by SQL DB for the following query:

```
FOR EACH customer, EACH order
  WHERE customer.custnum = order.custnum AND customer.custnum > 100:
```

For the following query, however, the DataServer instructs the client to perform the join because of the OR option:

```
FOR EACH customer, EACH order
  WHERE customer.custnum = order.custnum OR customer.custnum > 100:
```

By default, the DataServer instructs an ODBC data source to perform a join when possible and when desirable. However, you can control the default behavior by using either the `QUERY–TUNING NO–JOIN–BY–SQLDB` phrase or the Server Join (`-nojoinbysqldb`) startup parameter. The `QUERY–TUNING` phrase controls the behavior for a single query. The `-nojoinbysqldb` parameter controls it at the session level.

Table 2–9 describes how these controls interact and affect the behavior.

<table>
<thead>
<tr>
<th>QUERY–TUNING</th>
<th>Startup parameter</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOIN–BY–SQLDB</td>
<td>-nojoinbysqldb</td>
<td>The ODBC data source performs the join if possible</td>
</tr>
<tr>
<td>JOIN–BY–SQLDB</td>
<td>None</td>
<td>The ODBC data source performs the join if possible</td>
</tr>
<tr>
<td>NO–JOIN–BY–SQLDB</td>
<td>-nojoinbysqldb</td>
<td>The client performs the join</td>
</tr>
<tr>
<td>NO–JOIN–BY–SQLDB</td>
<td>None</td>
<td>The client performs the join</td>
</tr>
<tr>
<td>None</td>
<td>-nojoinbysqldb</td>
<td>The client performs the join</td>
</tr>
<tr>
<td>None</td>
<td>None</td>
<td>The ODBC data source performs the join if possible and the join contains the selection criteria</td>
</tr>
</tbody>
</table>

A join by SQL DB does not occur by default for the following query because the DataServer determines that it does not increase performance:

```
FOR EACH customer, EACH order:
```

You receive a warning if you specify `JOIN–BY–SQLDB` when the ODBC data source cannot perform the join and the DataServer performs the join instead. You receive a warning at compile time if you specify `JOIN–BY–SQLDB` when the data source can perform the join but it is not optimal for it to do so.
Writing queries for performance

This section provides a collection of tips and guidelines to follow when writing queries. For example, a query that processes a large number of rows performs best if it uses NO-LOCK, lookahead cursors, a large cache size, and a small field list.

These tips and guidelines might improve the performance of your DataServer applications. To assess the usefulness of a particular suggestion, apply it, then use the DEBUG diagnostic options to gather statistics on how your application runs.

- Use FOR EACH, GET, and OPEN QUERY statements rather than FIND statements, which generally perform more slowly. Consider using the FOR FIRST statement instead of FIND LAST.

  The only exception is that FIND LAST is faster than GET LAST. This is because GET LAST causes the client to process all of the records; the FIND LAST statement allows the server to retrieve the last record.

- Use field lists.
- Use the QUERY–TUNING options.
- Use lookahead cursors.
- Use NO–LOCK where possible.
- Avoid specifying lock upgrades. Instead, allow the DataServer and the ODBC data source to handle lock upgrades.
- Do not ask for a particular ordering of results with USE–INDEX or BY clauses unless your application requires it. Instead, allow the DataServer and the ODBC data source to determine the most efficient index (if any) for processing a query and avoid the overhead of sorting results.
- If you use a BY clause that will sort a large amount of data, make sure a corresponding index exists in your data source to make sorting efficient. In some cases it may also be desirable to have indexes over columns used in WHERE clause selection criteria.
- For aggregates, use either the RUN–STORED–PROCEDURE send–sql–statement syntax or a Progress SQL statement. If you use a Progress SQL statement with a cursor, declare the cursor read-only.
- Avoid using the RECID function. Instead, use the ROWID function.

See the “Analyzing performance” section on page 4–19 for more information on collecting statistics.
Skip schema verification

When r-code runs (each time a table, view, or buffer is opened), the DataServer checks the data definitions of the ODBC data source to ensure the data definitions match the schema definitions in the schema holder. If the definitions do not match, the DataServer returns an error.

Unmatched definitions can cause corruption of the data source. For this reason, checking the integrity of data definitions at run time ensures the data corruption due to unmatched definitions will not occur. The skip schema check feature can be used to bypass this check at run time. Because definition verification is time consuming in a production environment, you might consider using the -Dsrv skip-schema-check startup parameter if your environment allows. You might consider using this option to increase performance, but only if you are certain that the data definitions in the data source match your schema holder definitions.

Note: The dataserv.log log file denotes when the DataServer skips the schema check.

The following example shows how to use the -Dsrv parameter with the skip schema check option in the CONNECT statement:

```
CONNECT data-source-name -ld logical-name -dt ODBC
-Dsrv skip-schema-check.
```

Caution: If you use the skip schema check option, the DataServer skips the schema check and does not detect discrepancies between the OpenEdge schema definitions and the data-source data definitions. If there are discrepancies and the DataServer continues to process inserts, and deletions, your data source may become corrupted. Progress Software Corporation recommends that you weigh carefully the performance benefit against the risk to your database before deciding to use -Dsrv skip-schema-check.

Replacing FIND FIRST for performance

When coding ABL applications for the DataServer, FIND FIRST statements can often be replaced with other ABL statements to greatly improve performance. OPEN QUERYs and dynamic FINDs are significantly more efficient. Likewise, using FOR FIRST in place of FIND FIRST can improve performance when retrieving a single record with NO-LOCK. For example, if your application uses the following FIND FIRST code:

```
FIND FIRST tablename WHERE where-clause NO-LOCK.
```

The code can be replaced with the following for significant performance gains:

```
FOR FIRST tablename WHERE where-clause NO-LOCK:
END.
```

In the case of the FOR FIRST, the record is available beyond the end of the FOR FIRST loop. Similar performance advantages can be gained when retrieving last record as well.
Configuring the DataServer

Configuring the DataServer for ODBC involves starting executables for several processes. This chapter provides step-by-step instructions for initially setting up the DataServer. Specifically, it describes:

- DataServer components
- Configuring an ODBC driver and registering the data source
- Configuring a local DataServer
- Configuring the remote DataServer
- Creating a schema holder
- Maintaining a schema holder
- Typical configuration for a remote client to connect to a remote DataServer

Before you configure a DataServer, make sure that you have installed all of the required software. For details, see the “Software requirements” section on page 1–15.
DataServer components

The DataServer for ODBC can run in a variety of configurations. Some configurations involve a single process running on one machine. Others involve multiple processes running on different machines across multiple platforms.

DataServer configuration prerequisites

Before you configure a DataServer, you must register your data source as an ODBC data source. For details, see the “Registering your data source” section on page 3–3.

To set up a DataServer configuration, determine which components you need on which platforms, then set up the appropriate executables on those platforms. Table 3–1 lists the possible combinations and describes which executables you must set up on each machine. In this table, the term local indicates that the DataServer component runs on the same machine as the client, while the term remote indicates that the component runs on a different machine than the client.

Table 3–1: Installing the DataServer components

<table>
<thead>
<tr>
<th>Client</th>
<th>DataServer</th>
<th>Installing and configuring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Local Windows</td>
<td>Use the default OpenEdge client executable (prowin32.exe). The server component is embedded in the client executable.</td>
</tr>
<tr>
<td>Windows</td>
<td>Remote Windows</td>
<td>On the client machine, use the default OpenEdge client executable (prowin32.exe). On the host machine, use the default broker executable (_probrkr.exe) or configure a broker executable using the Progress Explorer.</td>
</tr>
<tr>
<td>UNIX</td>
<td>Remote Windows</td>
<td>On the client machine, use the OpenEdge client executable (_progres). On the host machine, use the default broker executable (_probrkr.exe) or configure a broker using the Progress Explorer.</td>
</tr>
</tbody>
</table>

For instructions on setting up your DataServer configuration, see the sections that apply to the platforms that you will use. For example, if you are configuring a local DataServer, see the “Configuring a local DataServer” section on page 3–5. If you are building a remote DataServer configuration host, see the “Configuring the remote DataServer” section on page 3–6.
Configuring an ODBC driver and registering the data source

Before you can configure the DataServer:

- You must install your ODBC drivers and any additional software required by the data source vendor.
- You must set up your data sources.

The OpenEdge DataServer for ODBC automatically installs the branded DataDirect ODBC drivers for the supported data sources. If you wish to use a different ODBC driver, you must purchase the drivers from the vendor and install them separately.

**Note:** Configuring the client for an ODBC connection might require installing software required by the non-OpenEdge data source vendor. You must completely install any additional client software required by the non-OpenEdge data-source vendor before you can register your data sources. For details, see the “Software requirements” section on page 1–15.

Configuring the ODBC driver

The DataServer relies on the ODBC driver to complete the connection to the data source, so it is important that you configure the driver as specified by the vendor. Refer to the configuration information provided by your ODBC driver vendor.

Registering your data source

On Windows, before you can use your data source with the DataServer for ODBC, you must register it as an ODBC data source using the ODBC administration tool.

**To register your data source as an ODBC data source using the ODBC administration tool:**

1. Start the ODBC administration tool for your data source. This can be either the administration tool provided by Microsoft or a repackaging of that tool by a non-OpenEdge data-source vendor. Also, some vendors might provide a similar administration tool.
2. Select the appropriate ODBC driver.
3. Select your ODBC data source.
4. Specify a name for your data source. The name that you use to register a data source (often abbreviated as DSN for data-source name) is also the name by which OpenEdge recognizes the data source.
5. Set other configuration options required or optionally allowed through the driver vendor’s configuration utilities against the target database.

6. Test connect.

7. Exit the ODBC administration tool.

You can now configure the DataServer, as described in the following sections.
Configuring a local DataServer

In a local configuration, all DataServer components—client and server—run on the same machine. (The server component is embedded in the client executable.) This section describes how to configure a local DataServer.

To configure the local DataServer in Windows:

1. Install ODBC software and any client software required by the data source on the system where your local DataServer resides.

2. Make sure that you registered the data source with the ODBC driver correctly on the appropriate machine.

Once you have set up your environment, you can build the schema holder for your ODBC data source and connect using the client executable. See the “Creating a schema holder” section on page 3–14 for instructions.
Configuring the remote DataServer

In configurations that include a remote host, two DataServer processes run on the host machine:

- **Broker** — The DataServer broker (\_probrkr.exe) or the broker for the Progress Explorer on the host machine determines the types of requests coming over the network and starts (spawns) the appropriate DataServer (\_odbsrv.exe) for the client process.

- **DataServer** — The DataServer (\_odbsrv.exe) on the host machine accesses the ODBC data source and communicates with the client process.

**Notes:** The ODBC DataServer component is a client with respect to the ODBC data-source configuration even though from the OpenEdge point-of-view it resides on the server machine. As a result, your ODBC software installation and any client software required by the data source must reside on the machine from which the DataServer and the broker processes will execute. In the remote DataServer configuration, the OpenEdge ODBC client component requires no special software or configuration; it requires only a standalone OpenEdge Windows or UNIX client. OpenEdge has no restrictions regarding the location of the actual database to which your ODBC data-source name connects.

Before you can run the server components, you must configure the DataServer by setting the required environment variables on the host machine. In Windows you use the Progress Explorer tool or command line tools. See the “Configuring with Progress Explorer” section on page 3–6 or the “Configuring from the command line” section on page 3–7 for details.

**Configuring with Progress Explorer**

In Windows, you can use the Progress Explorer tool to configure the DataServer for ODBC.

**Note:** This section describes how to access Progress Explorer to configure the DataServer for ODBC on Windows. It does not explain how to connect the DataServer. For connection instructions, see the “Starting and stopping a broker process from the Progress Explorer and connecting a client” section on page 4–3. For information about Progress Explorer, see the Progress Explorer online Help.

To access the Progress Explorer:

1. Make sure that the AdminServer is running.
2. Start Progress Explorer.
   - From the Windows Desktop choose Start→Programs→OpenEdge→Progress Explorer Tool. The Progress Explorer appears in the MMC framework.
3. Connect to localhost.
4. From the Progress Explorer’s left pane, select the ODBC DataServer folder and double-click. The list of existing DataServer brokers for ODBC appears in the right pane.
5. Select the DataServer instance whose properties you want to create or edit, and right-click. A pop-up menu appears.
Note: The DataServer for ODBC installation provides one predefined DataServer Broker (oddbroker1) and one predefined NameServer (NS1). Each broker is referred to as an instance. See the Progress Explorer online Help for more information. You can use these predefined components as a starting point for creating and configuring additional DataServer Brokers, and, if needed, NameServers. (See OpenEdge Getting Started: Installation and Configuration for information about the NameServer’s role in a configuration.)

6. Choose the Properties option from the pop-up menu. The Properties dialog box for that instance appears.

7. Configure the DataServer broker by setting general DataServer properties, owner information properties, application service properties, logging settings, environment variables, and others. For details on these settings, see the Progress Explorer online Help.

Note: By default, your DataServer for ODBC broker instances are defined with a controlling NameServer and are provided with a default Data Service. Progress Software Corporation recommends using a NameServer configuration at all times. In such cases, the DataServer client’s initial connection is to the NameServer. However, you can alternatively connect the DataServer directly to the broker instance by setting the -DataService value to none in the connection parameters of your schema holder or by adding the -DirectConnect parameter, which will override your -DataService parameter. If you will always bypass the controlling NameServer, you should remove the controlling NameServer from your broker instance definition. See the “Starting and stopping a broker process from the Progress Explorer and connecting a client” section on page 4–3 for more information about connecting the DataServer to the NameServer and the broker.

It is not recommended that you simultaneously run some DataServers for ODBC under brokers with controlling NameServers and others directly under brokers (that is, without controlling NameServers). This defeats the purpose of using a NameServer to control brokers. If you do this, the benefits of the NameServer are lost and load balancing is ineffective. Progress Software Corporation recommends that you always use a NameServer, with one exception: you can choose initially to connect directly to a broker to simplify confirming an initial connection. Once you establish a connection, Progress recommends that you reintroduce the NameServer into your configuration.

Once you have completely set up your environment, you can build the schema holder for your ODBC data source. See the “Creating a schema holder” section on page 3–14 for instructions.

Configuring from the command line

You may configure a remote DataServer broker process from the command line. You may use the _probrkr.exe executable directly or use the odbman.bat Progress Explorer utility, the command line equivalent of using the Progress Explorer.

Before you start the configuration tasks, make sure that you have installed your ODBC software and any data-source-specific client software on your host machine. Also make sure that you have registered all of your data sources. See the “Configuring an ODBC driver and registering the data source” section on page 3–3 for details.
To configure from the command line:

- When using the OpenEdge broker (_probrkr.exe), you set the environment variables described in this section from the command line using environment-variable commands at the DOS shell prompt.

- When using the odbman Progress Explorer utility, set the environment variables described in this section in the environment section of your broker properties file for the specific broker instance definition.

- You must set and export environment variables in the same environment (DOS shell) from which you plan to run the DataServer.

Table 3–2 describes the environment variables that you must set.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSLOGDIR</td>
<td>The pathname of the log file that OpenEdge uses to keep track of DataServer processes and error messages. By default, OpenEdge writes to %DSLOGDIR%\dataserv.lg where %DSLOGDIR% is resolved to directory path of the log file &quot;dataserv.lg&quot;.</td>
</tr>
<tr>
<td>ODBSRV</td>
<td>The pathname of the OpenEdge server executable. This variable is required only when the DataServer is spawned using the OpenEdge broker (_probrkr.exe). When using odbman, this setting is satisfied by the srvrExecFile setting in the broker properties file.</td>
</tr>
<tr>
<td>PROBRKR</td>
<td>The pathname of the broker executable. When using the OpenEdge broker (_probrkr.exe), set this variable on the host machine to the broker executable name. When using odbman, this setting is satisfied by the broker instance definition in the broker properties file.</td>
</tr>
<tr>
<td>PROSTARTUP</td>
<td>The pathname of your default OpenEdge startup (.pf) file. To set the Message Buffer Size (-Mm) startup parameter for the broker to a value different from the 1024 default buffer size, you must set the PROSTARTUP environment variable equal to the path and name of the .pf file that contains the -Mm parameter. This approach ensures that your value is recognized and therefore used by the broker when it starts. Only Windows clients must use this technique. <strong>Note:</strong> If you use the command line utility and intend to set the -Mm startup parameter to a value other than 1024, you must use the .pf file with the PROSTARTUP variable set.</td>
</tr>
</tbody>
</table>
Once you have completely set up your environment, you can build the schema holder for your ODBC data source and connect using the client executable. See the “Creating a schema holder” section on page 3–14 for instructions.

The ubroker.properties file

When using the Progress Explorer administration framework, you configure the DataServer for ODBC by editing the %DLC%\properties\ubroker.properties file. This file stores configuration definitions for instances of many OpenEdge products. For a complete list of products and a detailed discussion of the Unified Broker Framework, see OpenEdge Getting Started: Installation and Configuration.

When you use this file to configure the DataServer for ODBC, you provide information that enables the host to start a broker that spawns the appropriate DataServer process (_odbsrv.exe).

Each configuration definition contains the environment variable and property settings for a broker instance. The command-line utilities use this file to store, validate, and manage the configurations for these brokers. A single copy of this file maintains all supported broker configurations for each OpenEdge installation.

Editing the ubroker.properties file

The simplest way to make configuration changes to the ubroker.properties file is to copy an existing broker or NameServer definition and then modify the copied values as required for your configuration. To generate a new universally unique identifier (uuid), use the genuuid script found in the bin directory of your installation. For more information, see the ubroker.properties.README file. You must ensure that all related properties and sections of the file are properly specified for each broker or NameServer instance.

Note: Be sure to preserve the original %DLC%\properties\ubroker.properties file. Rename the original file and work with a copy of the file. You must name the copy of the file ubroker.properties.

Table 3–3 describes the sections in the ubroker.properties file that apply to the DataServer for ODBC. The file configures a default NameServer named NameServer.NS1 and a default broker named odbbroker1, which you can use either as they are or as templates for your own configuration specifications.

<table>
<thead>
<tr>
<th>Product</th>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All products</td>
<td>ParentGroup</td>
<td>Defines the name of each NameServer and product broker parent entity.</td>
</tr>
<tr>
<td></td>
<td>Environment</td>
<td>Default environment variable settings for all NameServers and product brokers.</td>
</tr>
<tr>
<td></td>
<td>UBroker</td>
<td>Defines default property settings for all product brokers.</td>
</tr>
</tbody>
</table>
### Table 3–3: DataServer for ODBC sections of ubroker.properties file (2 of 3)

<table>
<thead>
<tr>
<th>Product</th>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NameServer</td>
<td>NameServer</td>
<td>The parent entity of NameServers. It defines default property settings for all NameServer instances.</td>
</tr>
<tr>
<td>NameServer.NS1</td>
<td></td>
<td>A sample property entity of a NameServer instance. It defines property settings for this NameServer instance.</td>
</tr>
<tr>
<td>Environment.NS1</td>
<td></td>
<td>A sample environment entity of a NameServer instance. It defines environment variable settings for this NameServer instance.</td>
</tr>
</tbody>
</table>
Configuring the remote DataServer

The parent entity of DataServer for ODBC brokers. It defines default property settings for all of these broker instances.

UBroker.OD.oddbroker1
A sample property entity of a DataServer for ODBC broker instance. It defines default property settings for the broker instance named odbbroker1.

Note that although many of the settings in this section can work in your environment, some of them are for demonstration purposes only. Property settings may be changed dynamically for brokers started after the DataServer is started. This is not the case for brokers that are running.

You must specify the appropriate settings for the following:

- `srvrExecFile`: Specify the pathname of the DataServer executable that the broker runs. The default is @{DLC}/bin/_odbsrv.exe, which runs the default DataServer from your OpenEdge install path.
- `srvrMinPort/srvrMaxPort`: Specify the minimum and maximum port numbers available to the DataServer.
- `srvrStartupParam`: Specify the OpenEdge startup parameters for the DataServer. Do not modify the default parameters, although you can add parameters if necessary. The `-svub` switch is required for a broker connection in the Progress Explorer administration framework.
- `srvrStartupTimeout`: Specify the number of seconds the broker waits for aserver/agent to become available before starting a new agent.

Environment.oddbroker1
A sample environment entity of a DataServer for ODBC broker instance. It defines environment variable settings that apply to the broker instance named odbbroker1.

Be sure to set the variables to the appropriate values; the values in the file are for demonstration purposes only. Add to this section any environment variables that you want to apply to the DataServer’s environment. See the “Configuring the remote DataServer” section on page 3–6 for the environment variable settings required to configure the DataServer for ODBC.

<table>
<thead>
<tr>
<th>Product</th>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataServer</td>
<td>UBroker.OD</td>
<td>The parent entity of DataServer for ODBC brokers. It defines default property settings for all of these broker instances.</td>
</tr>
<tr>
<td></td>
<td>UBroker.OD.oddbroker1</td>
<td>A sample property entity of a DataServer for ODBC broker instance. It defines default property settings for the broker instance named odbbroker1.</td>
</tr>
<tr>
<td></td>
<td>Environment.oddbroker1</td>
<td>A sample environment entity of a DataServer for ODBC broker instance. It defines environment variable settings that apply to the broker instance named odbbroker1.</td>
</tr>
</tbody>
</table>

Table 3–3: DataServer for ODBC sections of ubroker.properties file (3 of 3)
The following example illustrates the DataServer sections of the `ubroker.properties` file:

```
# # Default properties for broker instances serving ODBC DataServers #
# [UBroker.OD]
  srvrExecFile="@{Startup\DLC}\bin\odsrv.exe"
  srvrStartupParam=-svub -S X -N TCP -U X -P X -hs 0 -s 40
  operatingMode=State-aware
  classMain=com.progress.ubroker.broker.ubroker
  portNumber=4444
  defaultService=0
  initialSrvrInstance=0
  minSrvrInstance=0
  maxSrvrInstance=256
  brkrLoggingLevel=3
  description=ODBC DataServer Broker
#
# Sample ODBC DataServer Broker definition #
#
# [UBroker.OD.odbbroker1]
  srvrExecFile="@{Startup\DLC}\bin\odsrv.exe"
  srvrStartupParam=-svub -S X -N TCP -U X -P X -hs 0 -s 40
  srvrLogFile=@{WorkPath}\oddbroker1.server.log
  brokerLogFile=@{WorkPath}\oddbroker1.broker.log
  portNumber=4444
  defaultService=1
  appserviceNameList=oddbroker1
  controllingNameServer=NS1
  environment=oddbroker1
  uuid=172.18.103.53:1f415c:d6330e5d24:-7f1e
  description=A sample ODBC DataServer Broker
#
# Environment for ODBC Dataserver Broker: odbbroker1 #
#
[Environment.oddbroker1]
  DSLOGDIR=@{WorkPath}
```

For a complete description of the parameters included in each of these sections, see the comments in the `%DLC%\properties\ubroker.properties` file.

The `ubroker.properties` file is read on startup of the AdminServer process. For changes in any used environment variables to take effect, the AdminServer must be restarted.

**Validating `ubroker.properties` content**

Whenever you create your own version of the `ubroker.properties` file, you should use the relevant validation utility to validate your changes and make sure that there are no syntax errors or conflicts. When configuring the DataServer for ODBC, you validate the file by using the `odbconfig` utility. See the “ODBCONFIG utility” section on page C–7 for more information.

**Configuring multiple brokers**

You can configure multiple brokers by adding more `UBroker.OD.broker-name` and `Environment.broker-name` sections. Each broker instance must have a unique name. The broker instances inherit the properties that you define in the `UBroker.OD` (parent entity) section, but can override them.

If you want to access multiple ODBC data sources and configure one or more brokers, each broker must run in a separate directory and a distinct environment.
Using the NameServer

By default, your DataServer for ODBC broker instances are defined with a controlling NameServer and are provided with a default Data Service. Progress Software Corporation recommends using a NameServer configuration at all times. In such cases, the DataServer client’s initial connection is to the NameServer. However, you can alternatively connect the DataServer directly to the broker instance by setting the -DataService value to none in the connection parameters of your schema holder. If you will always use a -DataService value of none, you should remove the controlling NameServer from your broker instance definition. See the “Starting and stopping a broker process from the Progress Explorer and connecting a client” section on page 4–3 for more information about connecting the DataServer to the NameServer and the broker. See OpenEdge Getting Started: Installation and Configuration for more information about the NameServer’s role in a configuration.

Note:  Do not simultaneously run some DataServers for ODBC under brokers with controlling NameServers and others directly under brokers (that is, without controlling NameServers). This defeats the purpose of using a NameServer to control brokers. If you do this, the benefits of the NameServer are lost and load balancing is ineffective. Progress Software Corporation recommends that you always use a NameServer, with one exception. You can choose initially to connect directly to a broker to simplify confirming an initial connection. Once you establish a connection, Progress recommends that you reintroduce the NameServer into your configuration.
Creating a schema holder

A schema holder contains a schema image of the ODBC data source that your DataServer application accesses. The schema image contains the schema for your data source—that is, its data definitions—in an OpenEdge format. Although applications can access a data source directly by sending SQL statements through the DataServer, they cannot directly access its data definitions and must use the schema image information instead.

Before you begin

Consider your business needs and deployment strategy when you decide where to build your schema holder. The schema holder can reside on the client machine or on any host in an enterprise configuration. Client schema holders improve performance for some applications. Host schema holders give you the flexibility of having them run as Windows services. Additionally, host schema holders might be easier to maintain. In a WebSpeed configuration, placing the schema holder on the Transaction Server machine optimizes performance.

General schema holder creation tasks

In general, you must do the following to create a schema holder:

1. Establish the appropriate permissions for pulling the schema image into the schema holder.
2. Perform some preparatory steps.
3. Create and then connect an empty OpenEdge database. This database becomes your schema holder and contains your schema image.
4. Create the schema holder, which involves specifying connection parameters and pulling the schema from the data source.

The following sections describe these steps in detail.

Establishing permissions

When you use the DataServer to create a schema image in a schema holder, it accesses the associated ODBC data source. This section describes the data-source permissions required when you create, update, or connect to a schema image.

Permissions for creating or updating a schema holder

You must have select privileges on specific system tables or database catalog files in your ODBC data source to create or update a schema image for the data source. This is because the Data Dictionary must access the system tables in the data source to create or update the schema image, but it cannot do so without select privileges. However, this is the only time that you need this level of privilege when using the DataServer.
The following data-source permissions are required to create or update a schema holder:

- **DB2 UDB**—You must have at least select permission on the following system objects: systables, syscolumns, sysindexes, and syskeys.
  
  On DB2/MVS, you must grant BINDADD permission and CREATE IN COLLECTION NULLID permission to the host user.

- **Sybase**—You must have at least select permission on the following system objects: sycolumns, syindexes, syobjects, and syusers.

### Permissions for connecting a schema holder

When you connect to a schema holder, you can optionally also connect to an ODBC data source. To connect to the data source, you must provide a valid data-source login name and password combination at connection time (if required by the non-OpenEdge data source). Use the User ID (-U) parameter to provide the user ID and the Password (-P) parameter to provide the password.

Your DataServer application must have the minimum data-source permission required to access data-source tables to validate schema information and allow multiple users to access tables. For example, for Sybase data sources, your application must have at least select permission on the sycolumns, syobjects, and syusers system objects. For details, see the documentation for your data source.

### Application-specific permissions

In addition to the ODBC data-source permissions required by the DataServer, the required permissions for users depend on the applications that they are using. For example, a user who is running an application that queries but does not update the employee table in a data source must connect to the data source with a login name and password combination that provides at least select privileges for the employee table. For users who will manipulate data at runtime, the appropriate read, insert, update, and delete permissions must be granted as administered by the foreign (target) data source.

In summary, the login name (or user ID) and password combination required to run a particular application depends on the following:

- The data-source tables that the application accesses
- The type of access required to those tables

**Note:** The system administrator for your ODBC data source must establish all login name and password combinations with the appropriate data-source commands and procedures. See the documentation for your data source for information on granting user permissions.
Preparing to create the schema holder

You must follow specific steps to create a schema holder.

To create a schema holder:

1. Verify that your non-OpenEdge data source is accessible and that you can connect to it. Non-OpenEdge data-source vendors often provide interactive SQL tools that can serve as a test for connectivity.

2. Verify that you have installed ODBC drivers and configured your data source appropriately. Also, ensure that any client software required by the non-OpenEdge data source is installed and configured properly.

3. Once you have configured your data source (DSN), make sure that you can establish a connection independent of using the DataServer. ODBC driver vendors often provide connectivity tools that test whether you can establish a connection to the data source through an ODBC interface.

4. Start the DataServer as described in either the “Starting a local DataServer” section on page 4–2 or the “Starting a remote DataServer” section on page 4–3, whichever is appropriate for your configuration.

5. Open Data Administration or the character Data Dictionary.

Creating an empty OpenEdge database

The DataServer uses an empty OpenEdge database as a holder for the schema for your ODBC data source. The simplest way to create an empty OpenEdge database is to do it from Data Administration, as described in this section. For information on other ways to create an empty OpenEdge database, see OpenEdge Data Management: Database Administration.

To create and connect an empty OpenEdge database from Data Administration:

1. Start OpenEdge with no database connected and access Data Administration.

2. Select the Database→Create Database option. The Create Database dialog box appears:
3. Type the schema-holder name (for example, odbholder) in the **New Physical Database Name** field.

4. Select the **An EMPTY Database** option.

5. Click **OK**. The following dialog box appears:

By default, the name of the newly created data source appears in the **Physical Name field**. You do not have to provide any additional connection information. You can add connection parameters when you create the data source or edit connection information later. See the online Help for a complete description of the **Connect Database** dialog box.

6. Click **OK** to connect the empty OpenEdge database and return to the **Data Administration** main window.

### Creating a schema holder

You can create a schema holder next.

1. **To create a schema holder:**

   1. From the **Data Administration** main menu, select **DataServer → ODBC Utilities → Create DataServer Schema**. The following dialog box appears:

   ![Create Modify Database Record for DataServer Schema](image)

2. In the **Logical Database Name** field, type the name that you will use to connect to your ODBC data source and refer to it in your programming applications. This name must be different from the schema holder name. For more information on database names, see the database access chapter in *OpenEdge Getting Started: ABL Essentials*. 
Note: If you place the schema from a second ODBC data source into a schema holder, the second schema must have a different logical database name from the first schema. The schema holder has one physical name, but each schema that it contains must have a different logical name.

3. In the Code Page field, type the name of the code page for the schema holder. The name must be the OpenEdge name for the code page that the data source uses. The default is iso8859–1.

Table 3–4 lists the most common ODBC data-source code pages and the equivalent OpenEdge names.

Table 3–4: ODBC data-source and OpenEdge code pages

<table>
<thead>
<tr>
<th>ODBC data-source code page</th>
<th>OpenEdge equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>iso_1</td>
<td>iso8859–1</td>
</tr>
<tr>
<td></td>
<td>(default schema-holder code page)</td>
</tr>
<tr>
<td>cp850</td>
<td>ibm850</td>
</tr>
</tbody>
</table>

If you use a code page that OpenEdge does not support, you must supply a conversion table that translates between the OpenEdge client code page and the code page that your data source uses. For a complete discussion of code pages, see OpenEdge Development: Internationalizing Applications.

4. In the Collation field, enter the name of the collation rule to use. The default is Basic. See the “Client code page” section on page 2–6 for a discussion of collation issues to consider.

5. Type the connection parameters in the Connection Parameters field.

See Chapter 4, “Connecting the DataServer,” for a description of the required and optional connection parameters.

6. In the ODBC Data Source Name field, type the name that you used when you registered the data source with the ODBC administration tool.

7. Click OK. The utility prompts you for your data-source user ID and password. If they are required by the non-OpenEdge data source and you did not provide them in the Connection Parameters field (see Step 5), enter a data-source user ID and password combination that has select privileges for the system objects listed in the “Establishing permissions” section on page 3–14 and read access to other database objects that the schema holder will include.
8. Click OK. When the DataServer connects to the ODBC data source, it reads information about data-source objects. The following dialog box appears:

![Pre-Selection Criteria For Schema Pull dialog box]

**Note:** The Owner field in the Pre-Selection Criteria dialog is labeled Collection/Library when the foreign DBMS type is DB2/400.

You can select tables based on the object name, owner name, and qualifier. For example, you can specify A* in the Object Name field to list all the tables whose names begin with A. DB2 UDB for iSeries uses a SQL “Collection” or native “Library” to qualify data object selection. Note that the Owner label is replaced with Collection/Library when the foreign DBMS type is DB2/400. The Collection/Library default value is derived from the Collection/Library name specified during migration. If you did not migrate your foreign database from OpenEdge and are instead pulling objects from an existing foreign data source into an OpenEdge schema holder, the Collection/Library default value is derived from the Data Source Name (DSN) properties of the DB2 UDB driver configuration. You can specify any valid native “Library” in the Collection/Library parameter, however OpenEdge expects and Progress recommends that data objects in the foreign data source are defined to the SQL “Collection” specified in the DSN configuration. “Collections” are SQL-compliant so objects defined to them are also SQL-compliant by definition. Data object definitions retrieved from native iSeries libraries are expected to be SQL-compliant as well. Any native attributes of database object definitions retrieved from objects in DB2/400 for iSeries “libraries” are not inclusive in their schema holder definitions if those attributes are purely native, i.e., are not SQL-compliant attributes. Progress recommends defining all database objects from all native libraries you wish to define to OpenEdge in a single collection that you also specify in your DSN configuration.

Alternatively, the iSeries Access for Windows native driver for iSeries may provide some native feature mapping not available with the branded DB2 UDB driver provided with OpenEdge. When the native driver is used, if not specified, the default “Collection/Library” name originates from the 'SQL Default Library' specified by the DSN. If 'SQL Default Library' is not specified but a 'Library List' is designated by the DSN, then the first library in the library list is designated as the default 'Collection/Library'. If neither of these values are specified by the DSN, then the user id of the user performing the dictionary action is used for the default 'Collection/Library' value.

**Note:** Progress Software Corporation recommends that you do not specify an entry that consists exclusively of wild cards for each of the three entry fields in the dialog box. An entry that consists exclusively of wild cards might degrade the performance of the database when you perform a schema pull. (It will include system catalog files from the data source not typically included in user databases.)
9. Click **OK**. OpenEdge displays a list of the data-source objects that you can include in the schema holder:

If you specified all wild cards as your table-selection criteria, the list might also include system-owned objects, which you do not have to include in the schema holder.

10. Click the option appropriate to the action you want to perform:

   - **Select Some** — Displays the Select by Pattern Match dialog box on which you can specify object information used to select objects
   
   - **Deselect Some** — Displays the Deselect by Pattern Match dialog box on which you can specify object information used to deselect objects

   You can also select and deselect individual objects by clicking and double-clicking on an object. An asterisk appears next to an object that has been selected; double-click an object to remove the asterisk and identify that the object is now deselected.

11. Click **OK** after you have identified all the objects you want to include in the schema holder. The DataServer reads information about the objects that you select and loads their data definitions into the schema holder. The time that this process takes depends on the size and number of objects that you select.

   For each data-source table, the DataServer attempts to select an index to support the Progress ROWID. If an appropriate index does not exist, the DataServer issues the warning, “Please check warnings and messages in the file ds_upd.e.” The ds_upd.e file lists the objects that do not support ROWID. You can change the DataServer’s selection of an index to support ROWID by using the Data Dictionary. See the “Defining the ROWID” section on page 5–31 for instructions. For additional information, see the “Indexes and sorting” section on page 2–7 and the “ROWID function” section on page 2–33.
Maintaining a schema holder

Data Administration provides a set of DataServer utilities that you can use to maintain a schema holder. Chapter 5, “The DataServer Tutorial,” describes these utilities. Recall that you must have select privileges on certain data-source objects to update a schema holder. See the “Establishing permissions” section on page 3–14 for details.

If you make changes to an ODBC data source, make sure to update the associated schema holder to reflect those changes if you want them to be accessible to a DataServer application. Note that you do not need to update the schema holder if the application will never access data objects affected by the change. For example, if you add a table object that a DataServer application will never access, you do not need to update the schema holder.

Each time that you update the schema holder, you must recompile your DataServer application (.p and .w files) to generate new r-code.

Updating a deployed schema holder

The guidelines and techniques that apply to deploying an OpenEdge database also apply to deploying a schema holder for an ODBC data source. However, if you make any changes to the data-source schema, make sure to update its schema holder. There are two techniques for updating a deployed schema holder:

- Allow your users to use the DataServer Update/Add Table Definitions utility.
- Send a new data definition file for the schema holder. Your users can use the DataServer Delete Schema utility to empty the original schema holder. They can then load the new data-definition file into the schema holder.

To update a deployed DataServer application:

1. Run an SQL script to make changes to the ODBC data source.
2. Run the Update/Add Table Definitions utility on one schema holder.
3. Recompile code against the updated schema holder to build new r-code.
4. Send out copies of the new .r files to your users.
5. Distribute copies of the new schema holder .db and .bi files to your users. You must use the OpenEdge PROCOPY utility to distribute them.
Typical configuration for a remote client to connect to a remote DataServer

A typical configuration to connect a remote client to a remote DataServer requires the following general tasks:

- Configuring an ODBC data source
- Starting a broker
- Starting the schema holder to accept remote client connections
- Connecting remote clients to an ODBC DataServer broker

This section highlights the basic procedure associated with each of these tasks.

Configuring an ODBC data source

An ODBC data source allows you to store information about how to connect to a data provider. You use the ODBC Data Source Administrator to configure an ODBC source. Note that the DataServer license must be installed on the machine where the ODBC data source is configured.

To configure an ODBC data source:

1. In Windows, choose Start → Control Panel → Administrative Tools → Data sources (ODBC) and select Systems DSN. The System DSN Data Source Administrator appears.

2. Choose Add to add a data source. Proceed through the windows as they are presented to configure the data source, ensuring that the final test connection is successful.

Starting a broker

You can start the DataServer broker, using various methods supported by the Unified Broker Framework, including:

- The OpenEdge Explorer and Progress Explorer tool or mergeprop utility (for a Unified Broker)
- The command line (for a probroker)

The following example shows how to perform this step from the command line. For information about using the mergeprop utility, see OpenEdge Getting Started: Installation and Configuration. For information about using the OpenEdge Explorer and Progress Explorer tool, see the “Starting and stopping a broker process from the OpenEdge Explorer and Progress Explorer and connecting a client” section on page 6–3, and OpenEdge Getting Started: Installation and Configuration.
To start a DataServer broker from the command:

1. Choose All Programs → OpenEdge → Proenv. The proenv prompt appears.

2. Type the following example command line to start the DataServer broker:

   ```
   _probrkr -H hostname -S brokerservice -N tcp
   ```

   The values for _probrkr and brokerservice must be unique.

   For more details about using either the OpenEdge Explorer and Progress Explorer or the command line to start a unified broker process for the DataServer, see the “Configuring the remote DataServer” section on page 3–6.

Starting the schema holder to accept remote client connections

The connection between the schema holder and a remote client allows you to access data definitions stored in the schema holder. The following example shows how to service the schema holder to accept a remote client connection.

Using the same Progress PROENV shell used in Step 4 of the “Starting a broker” section on page 3–22, type the following command to start the DataServer connection:

```proserve schemaholdername -H hostname -S databaseservice -N tcp```

The value for databaseservice must be unique to the connection you are making to the schema holder.

Connecting remote clients to an ODBC DataServer broker

This section describes a typical configuration for a remote client to connect to a remote DataServer.

**Note:** For a description of each parameter identified in this section, see Table 6–1.

Use the following command line entries for a remote character or a remote UNIX client machine:

```
_progres schemaholdername -H hostname -S databaseservice -N tcp -db dsn_name -dt ODBC -ld logical-datasrc-name -H hostname -S brokerservice -N tcp -U userID -P password
```
Use the following command line entries for a remote GUI client machine:

```
prowin32 schemaholdername -H hostname -S databaseservice -N tcp -db dsn_name -dt ODBC -ld logical-datasrc-name -H hostname -S brokerservice -N tcp -U userID -P password
```

For details about the database connection parameters noted in each example in this procedure, see the “Connecting a schema holder at startup” section on page 6–9.
Connecting the DataServer

You can start and connect a DataServer using one of the following methods:

- Progress Explorer tool (Windows platforms only)
- Progress Explorer Command line utility (ODBMAN)
- Command line statements (_probrkr.exe)

This chapter describes:

- Starting a local DataServer
- Starting a remote DataServer
- Connection guidelines
- Connecting a schema holder
- Connection failures and OpenEdge responses
Starting a local DataServer

Starting the OpenEdge client starts the local DataServer. You include information about the ODBC data source, user ID, and password in the startup command.

To start a local DataServer for ODBC from the startup command:

1. Make sure that your ODBC drivers, as well as any client software required by the non-OpenEdge data source vendor, are installed and configured properly.

2. Start the supporting ODBC data source.

3. Set any environment variables required for your configuration as described in the relevant section of Chapter 3, “Configuring the DataServer” (for example, $DSLOGDIR). You must set them in the environment from which you are starting the DataServer.

   Note: If you change the values of any environment variables such as $ODBC_HOME, you must shut down the DataServer processes and restart them.

4. Enter the following command:

   ```
   prowin32 schema-holder-name -1 -db datasrc-name -dt ODBC -ld logical-datasrc-name -U userID -P password
   ```

   For example, the following command starts OpenEdge with the local DataServer, connects a local schema holder named odbholder in single-user mode, and connects the ODBC data source demo with the user bob whose password is bobpass:

   ```
   prowin32 odbholder -1 -db demo -dt ODBC -ld demo -U bob -P bobpass
   ```
Starting a remote DataServer

Starting a remote DataServer involves starting the processes that make up the remote DataServer.

To start and stop a remote DataServer for ODBC:

1. Start the DataServer broker process on your host machine. For details, see one of the following sections:
   - The “Starting and stopping a broker process from the Progress Explorer and connecting a client” section on page 4–3
   - The “Starting and stopping a broker process using ODBMAN” section on page 4–5
   - The “Starting and stopping a broker process from the command line” section on page 4–6

2. Start an OpenEdge client process on a UNIX machine or on a PC running Windows that connects to the schema holder and the ODBC data source.

Starting and stopping a broker process from the Progress Explorer and connecting a client

You can use the Progress Explorer to administer the server module (_odbsrv.exe) of the DataServer on Windows.

On the Windows host

Before you attempt to start the DataServer in the Explorer, be sure that you have configured it completely. After starting the broker from the Progress Explorer, you start your OpenEdge client as you would in any remote DataServer configuration.

To start and stop the DataServer from the Explorer, see the Progress Explorer online Help.

On the client

After you start the broker on the host machine from the Progress Explorer, you can connect your UNIX or Windows client. Use the same parameters that you would use to connect to the schema holder and ODBC data source in a standard probrkr configuration. In addition, you must:

- Include the -Dsrv SVUB,1 parameter. This parameter allows you to connect to the broker administered by the Explorer.

- Include the -DataService data–service parameter to connect through a NameServer to the broker. The value for data–service must specify a valid name from the DataService list registered to this NameServer as defined by your appServiceNamelist entry in the broker properties file. If a default DataService has been defined for your broker instance, you can omit this parameter and connect using the default service.
For diagnostic purposes, it is acceptable to bypass the NameServer connection and connect directly to the broker instance. To do this, specify the reserved DataService name “none” or add the -DirectConnect parameter as follows:

-DataService none

or

-DirectConnect

- Set the -S parameter to one of the following:
  - The port number assigned to the controlling NameServer (when the -DataService value is not “none”) or the port number of the broker instance that you started in the Explorer (when the -DataService value is “none”).
  - The service name in your \windows\install-dir\system32\drivers\etc\services file whose associated port matches the port of the controlling NameServer (when the -DataService value is not “none”) or the broker instance that you started in the Explorer (when the -DataService value is “none”).

- Set the -H parameter to the name of the machine where the NameServer and/or broker instance are running.

If you do not set the required -Dsrv SVUB,1 and optional -DataService data-service connection parameters as described in this section, the client is assumed to be configured for a standard OpenEdge broker and the -H and -S parameters are used to locate a probrkr executable on the appropriate host machine. By setting the SVUB parameter on, you redirect the -H and -S parameters to locate the appropriate NameServer and/or broker on the host machine. The following example illustrates how to use these connection parameters for a client that connects to a NameServer:

CONNECT odbholder -db odbdemo -dt ODBC -U bob -P bobpass -H host1 -S odsrv -DataService odbbroker1 -Dsrv SVUB,1
Starting and stopping a broker process using ODBMAN

You can use the ODBMAN utility to start a DataServer broker in a Windows host.

To start and shut down the DataServer:

1. Once you ensure the AdminService is running, to start the DataServer broker, enter this command at the system prompt on the machine where the broker will run:

   odbman -name broker-name -start

   In this command, broker-name is the name that you specified for the broker when you configured it. Optionally, you can indicate a user account by specifying -user user-name.

   If you want to run the broker from a remote machine, you must specify additional options that identify the remote host, as follows:

   odbman -name broker-name -start -host host-name -user user-name

   In this command:

   • broker-name is the name that you specified for your DataServer broker instance when you configured your ubroker.properties file.

   • host-name is the name of the host machine on which you want the broker to run.

   • user-name is the user ID of the system account under which the broker will run.

2. To stop the DataServer broker, enter this command at the system prompt on the machine where the broker will run:

   odbman -name broker-name -stop

   You can stop a broker on a remote machine by adding the -host and -user options.

   See the “ODBMAN utility” section on page C–9 for a description of all the command options.
Starting and stopping a broker process from the command line

You must follow specific steps to start and stop a broker process from the command line.

To start and stop a broker process from the command line at the DOS shell prompt:

1. Set the environment variable ODBSRV to the name of the executable (including the path) of the DataServer for ODBC. Be sure to set this variable on the host machine.

2. To start the DataServer broker process, enter the following command at the system prompt on your Windows host machine. Select a value for service-name from the list of available services in your \windows-install-dir\system32\drivers\etc\services file:

   ```
   brokername -H host-name -S service-name
   ```

   For example, the following command uses the default broker executable. The service name demosv is a service listed in that file:

   ```
   _probrkr -H paris -S demosv
   ```

3. To stop the broker process, enter this command:

   ```
   proshut -Gw -H host-name -S service-name
   ```

After you start the Windows broker process, you are ready to start an OpenEdge client on a PC running Windows or on a UNIX machine. See the “Starting the Windows client process” section on page 4–6 or the “Starting the UNIX client process” section on page 4–7 for instructions.

Starting the Windows client process

Start the OpenEdge client process on your Windows machine by running the prowin32 executable. The Windows executable includes support for the DataServer.

You can create a program icon for the OpenEdge client process. On the command line for the program icon, enter the following information:

1. The executable

2. The schema holder name. If you have multiple schema holders, create a program icon for each schema holder

3. The connection parameters required by the remote DataServer configuration
For example, a command line for an OpenEdge Windows client process that you use to access an ODBC data source might look like this:

```
prowin32 odbholder -RO -db demo -dt ODBC -ld demo -H host1 -S oserviceA -U bob -P bobpass
```

See the “Connecting a schema holder at startup” section on page 4–9 for command-line information and more examples.

**Starting the UNIX client process**

After starting the remote broker, you start the OpenEdge client process on your UNIX machine by running this executable:

```
pro
```

You can supply the connection parameters required by the DataServer when you start the client process, or you can include them in the **Connection Parameters** field when you create a schema holder. For example, this command starts the OpenEdge client, connects a read-only schema holder named odbholder, and connects the ODBC data source demo with the user bob whose password is bobpass:

```
pro odbholder -RO -db demo -dt ODBC -ld odbdemo -H host1 -S oserviceA -N TCP -U bob -P bobpass
```

See the “Connecting a schema holder at startup” section on page 4–9 for descriptions of the required command line.
Connection guidelines

When you create a schema holder, you specify certain connection information, such as the physical and logical names for your ODBC data source and the user and password information. OpenEdge stores this information so that you do not have to provide it each time you connect. After you consider your security requirements, you connect a schema holder and then an ODBC data source.

Connections and security in Windows

When you configure an ODBC data source in Windows, you can choose one of three levels of security:

- **Standard** — Requires that a client pass a user ID and password that the ODBC data source validates against the list of users in the system login table. The request typically comes from a untrusted connection, such as through TCP/IP. The OpenEdge client or WebSpeed agent passes this information with the User ID (-U) and Password (-P) connection parameters.

- **Integrated** — Relies on the Windows operating system to validate user information. The ODBC data source accepts user information from the process that requests a connection. The request must come from a trusted connection, such as one using Named Pipes. The ODBC data source performs no additional validation.

- **Mixed** — Accepts requests from trusted or nontrusted connections. For example:
  - If the connection is trusted and the client provides no user ID, a user ID that consists entirely of spaces, or a user ID that matches the user that started the process, the ODBC data source accepts the connection.
  - If the connection is untrusted, the OpenEdge client must provide the user ID and password.

Progress Software Corporation recommends the following guidelines for working with an ODBC data source and Windows security:

- Configure an ODBC data source to use Standard or Mixed security if you are using remote OpenEdge clients.

- If you are using Mixed security, always have the clients specify the -U and -P connection parameters.
Connecting a schema holder

As described previously, connection can take place automatically using the information that you enter when you create a schema holder. In addition, OpenEdge provides the following techniques for connecting to a schema holder:

- Use the Data Dictionary or Data Administration. From the main menu, select **Database → Connect** and supply the schema holder’s physical name and the appropriate connection parameters. You connect to the data source when you select it as your working database. You cannot use the **Utilities → Auto-Connect** option to connect to an ODBC data source.

- Use the ABL **CONNECT** statement (see its reference entry in *OpenEdge Development: ABL Reference*). A CONNECT statement must first list the schema holder and related connection parameters, then the ODBC data source and related parameters.

  For example, this command connects a schema holder named holder and an ODBC-supported data source named odbdemo:

  ```
  CONNECT odbholder -db odbdemo -dt ODBC.
  ```

  You can use combinations of different connection techniques. For example, you can connect the schema holder at startup, then connect to the DataServer using the CONNECT statement. Any combination of connection techniques works, as long as you first connect to the schema holder and then the ODBC data source.

  If you are not updating the schema holder, you can specify the Read-only (\-RO) connection parameter to enhance DataServer performance.

  If you connect to the schema holder and the ODBC data source in a single startup command or connection statement, be sure to specify parameters that affect the schema holder **before** the Database Name (\-db) parameter. Specify only those parameters that affect the ODBC data source connection after the \-db parameter.

  The following section explains how to connect both a schema holder and an ODBC data source when you start up OpenEdge.

**Connecting a schema holder at startup**

OpenEdge supports connection parameters that you can use to connect both the OpenEdge schema holder and an ODBC data source at startup. These parameters control how your system connects to a database. When the DataServer runs in a remote configuration, your startup command or parameter file must always include parameters that control networking options.
Table 4–1 describes the database connection parameters that you use when you connect to a schema holder and an ODBC data source through the DataServer.

### Table 4–1: DataServer connection parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Type ODBC (-dt ODBC)</td>
<td>Optional</td>
<td>Specifies that the type of the target data source is ODBC. Note that the data source must be a supported ODBC data source.</td>
</tr>
<tr>
<td>Physical Database Name (-db)</td>
<td>Required</td>
<td>Indicates the name by which OpenEdge recognizes the ODBC data source to which you want to connect. This name must match the name that you used when you registered the data source as an ODBC data source.</td>
</tr>
<tr>
<td>Logical Database Name (-ld)</td>
<td>Optional</td>
<td>Specifies the logical name of the ODBC data source. This is the name that you use to refer to the data source in your applications. You must use this parameter only when the logical data-source name differs from its physical name. This name should match the logical database name that you defined in your schema holder. For example, your applications might refer to the Sybase demo database as mydemo. In this case, the physical name is demo, and the logical name is mydemo.</td>
</tr>
<tr>
<td>Host Name (-H)</td>
<td>Required for remote DataServer</td>
<td>Indicates the name of the host machine in the network.</td>
</tr>
<tr>
<td>Service Name (-S)</td>
<td>Required for remote DataServer</td>
<td>Indicates the name of the service that you are calling. If you use the NameServer with Progress Explorer, specify the service name or IP address of the host machine where the NameServer resides. If you are using _probrkr.exe or the Progress Explorer without a NameServer, specify the service name or IP address of the host machine where the broker resides.</td>
</tr>
<tr>
<td>User ID (-U)</td>
<td>Required if the ODBC data source requires it</td>
<td>Supplies the login name that the DataServer uses to log into the ODBC data source.</td>
</tr>
<tr>
<td>Password (-P)</td>
<td>Required if the ODBC data source requires it</td>
<td>Supplies the password that the DataServer uses to log into the ODBC data source. Different login name and password combinations allow for different levels of user privileges.</td>
</tr>
<tr>
<td>Data Service (-DataService)</td>
<td>Required for Progress Explorer connections</td>
<td>Specifies the data service the NameServer uses. This must be used in conjunction with the -Dsrv option SVUB, 1. For more information, see the “Starting and stopping a broker process from the Progress Explorer and connecting a client” section on page 4–3.</td>
</tr>
<tr>
<td>Explorer Redirection (-SVUB)</td>
<td>Required for Progress Explorer connections</td>
<td>Redirects connection logic to the Progress Explorer instead of ProBroker. For more information see the “Starting and stopping a broker process from the Progress Explorer and connecting a client” section on page 4–3.</td>
</tr>
</tbody>
</table>
Connecting a schema holder

Table 4–1:  DataServer connection parameters  

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single–User Mode (-1)</td>
<td>Optional</td>
<td>Specifies that a schema holder is used in single-user mode. Single–user mode is the default unless a server is started for the schema holder.</td>
</tr>
<tr>
<td>Read-Only (-R0)</td>
<td>Optional</td>
<td>Specifies that a schema holder is read-only. Connecting a schema holder as read-only increases processing speed at client startup time. It also allows multiple client processes on the same machine to access the schema holder without starting additional server processes.</td>
</tr>
<tr>
<td>Local Cache (-cache)</td>
<td>Optional</td>
<td>Specifies that you are using a local cache file for the schema holder. Create the cache file with the SAVE CACHE COMPLETE statement.</td>
</tr>
<tr>
<td>DataServer (-Dsrv)</td>
<td>Optional</td>
<td>Specifies options with which you control your ODBC Driver and DataServer environment. See the “Query tuning with connection and startup parameters” section on page 4–16 and the “ODBC options” section on page 6–2 for more information. <strong>Note:</strong> When you specify a list of -Dsrv parameters, be sure not to include any spaces anywhere in this list.</td>
</tr>
<tr>
<td>Direct Connect (-DirectConnect)</td>
<td>Optional</td>
<td>Instructs the client to bypass the controlling NameServer and connect directly to the ODBC DataServer Broker.</td>
</tr>
<tr>
<td>Server Join (-nojoinbysqldb)</td>
<td>Optional</td>
<td>Specifies that the client evaluates and performs queries that have joins. This might slow performance, but it provides results that are consistent with OpenEdge behavior. Use -nojoinbysqldb at startup time.</td>
</tr>
</tbody>
</table>
You can create a parameter file for each database:

- For a local DataServer, the parameter file must contain the -db parameter and can optionally contain the -Dsrv, -U, and -P connection parameters, depending on the requirements of the data service.

- For a remote DataServer, the same parameter conditions apply as for a local DataServer. In addition, a remote connection must contain the -H and -S connection parameters.

You can add more startup and connection parameters than the ones listed—these are the typical parameters. For a complete list of parameters and for information on how to create a parameter file, see *OpenEdge Deployment: Startup Command and Parameter Reference*.

**Using a local DataServer configuration: examples**

Use the following syntax to start OpenEdge:

- In single-user mode
- In a local DataServer configuration
- With a local schema holder connected
- With an ODBC data source connected

**Syntax**

```
prowin32 -1 schema-holder-name -db datasrc-name -dt ODBC -ld logical-datasrc-name -U userID -P password -Dsrv qt_debug,EXTENDED,PRGRS_CONNECT,server=server-name
```

You can type these commands on the command line of a program item property box.

The following syntax starts OpenEdge in a local DataServer configuration:

**Syntax**

```
prowin32 -1 odbholder -RO -db sports -dt ODBC -ld mysport -U bob -P bobpass -Dsrv qt_debug,EXTENDED,PRGRS_CONNECT,server=sqlserv1
```

In this case, OpenEdge starts with the following:

- The schema holder’s physical name is odbholder and it is read-only
- The physical data-source name (and the ODBC data-source name) is sports
- The data-source type is ODBC
- The logical data-source name is mysport
- The user ID is bob
- The password is bobpass
- Assorted -Dsrv options are specified
Connecting a schema holder

**Using a remote DataServer configuration: examples**

A remote connection differs from a local connection in that it requires the Host (-H) and Service (-S) parameters.

Use the following command syntax to start OpenEdge:

- In single-user mode
- In a remote-DataServer configuration
- With a local schema holder connected
- With an ODBC data source connected

**On a Windows client:**

**Syntax**

```
prowin32 schema-holder-name -db datasrc-name -dt ODBC
    -ld logical-datasrc-name -H hostname -S service-name
    -U userID -P password -Dsrv
    qt_debug,EXTENDED,PRGRS_CONNECT,server=server-name
```

**On a UNIX client:**

**Syntax**

```
pro schema-holder-name -db datasrc-name -dt ODBC
    -ld logical-datasrc-name -H hostname -S service-name
    -U userID -P password -Dsrv
    qt_debug,EXTENDED,PRGRS_CONNECT,server=datasrc-name
```
The following examples start OpenEdge in a remote DataServer configuration. In these examples:

- The schema holder’s physical name is odbholder and it is read-only
- The data-source name is sports
- The data-source type is ODBC
- The logical data-source name is mydemo
- The host name is host1
- The service name is odbsrv
- The network type is TCP
- The user ID is bob
- The password is bobpass

**On a Windows client:**

```
prowin32 odbholder -RO -db sports -dt ODBC -ld mydemo -H host1
-S odbsrv -U bob -P bobpass
-Dsrv qt_debug,EXTENDED,PRGRS_CONNECT,server=sqlserv1
```

**On a UNIX client:**

```
pro odbholder -RO -db sports -dt ODBC -ld mydemo -H host1
-S odbsrv -N TCP -U bob -P bobpass
-Dsrv qt_debug,EXTENDED,PRGRS_CONNECT,server=sqlserv1
```

**Note:** This configuration assumes you started the remote DataServer broker using the command line interface. To connect to a DataServer broker started through Progress Explorer you must add the SVUB,1 setting to the -Dsrv parameter and add the -DataService name parameter.

**Unsupported connection parameters**

You cannot use the following OpenEdge connection parameters when connecting to an ODBC data source through the DataServer.

- Blocks in Database Buffers (-B)
- Lock-table Entries (-L)
- Number of Users (-n)
- Buffered I/O (-r)
- Version 6 Query (-v6q)
- Raw I/O (-R)
Special connection issues

The DataServer (-Dsrv) parameter uses the PRGRS_CONNECT option to allow you to pass ODBC-specific information to the ODBC driver. A DataServer connection string contains all of the information needed to establish a connection. It consists of a series of keywords/value pairs separated by semicolons. The DataServer for ODBC passes the connection string specified by the PRGRS_CONNECT option directly through to the ODBC driver without modification. For more information, refer to the Microsoft ODBC Programmer’s Reference.

The DataServer parameter has the following syntax:

**Syntax**

```
-Dsrv PRGRS_CONNECT,connection-string;
```

ODBC-specific connection information is passed in `connection-string`. The connection string is separated from the option by a comma (,) and ends with a semicolon (;).

Value pairs within the connect string are delimited by a semicolon (;).

**Note:** PRGRS_C0 is a valid abbreviation for PRGRS_CONNECT.

Use the PRGRS_CONNECT parameter in the following cases:

- To establish complex connections that require more than the Physical Database Name (-db), User ID (-U), and Password (-P) parameters, as follows:

  ```
  CONNECT datasrc-name -ld logical-datasrc-name -dt ODBC
  -Dsrv PRGRS_C0,DSN=datasrc-name;server=servername;.
  ```

  For `datasrc-name`, supply the name of the ODBC data source. Server is a driver-specific keyword. The -Dsrv connection string is passed directly to the data source. The DataServer does not modify this value.

- To connect to an ODBC data source whose name has a blank space, which is not allowed by OpenEdge, substitute the characters &^ for the illegal characters in the data-source name. OpenEdge ignores `datasrc-name` when you use PRGRS_CONNECT; however, you must supply it to pass syntax validation. Supply the name as part of the connection string for PRGRS_CONNECT, as follows:

  ```
  CONNECT datasrc-name -ld logical-datasrc-name -dt ODBC
  -Dsrv PRGRS_C0,datasrc-name;.
  ```

- To connect to the ODBC data source using the ODBC driver as a guide, specify an empty PRGRS_CONNECT, which tells the ODBC driver to handle the entire connection process interactively. This technique is useful if you are not sure about what connection parameters to use for a particular ODBC data source. For example:

  ```
  CONNECT datasrc-name -ld logical-name -dt ODBC
  -Dsrv PRGRS_C0,;.
  ```
Query tuning with connection and startup parameters

You can control aspects of DataServer query handling not only programmatically within ABL statements, but also through startup and connection parameter options as described in this section.

**Note:** Startup and connection parameters override query-tuning defaults. However, options set in the QUERY–TUNING phrase take precedence over startup and connection parameters. For example, if you specify NO–DEBUG for a query within your application, specifying qt_debug,SQL at connection time overrides the default application behavior but does not override the NO–DEBUG option that you specified for the query. See the “Query tuning” section on page 2–55 for more information.

You override query-tuning defaults with the DataServer (-Dsrv) connection parameter when you connect to an ODBC data source. This is the syntax:

**Syntax**

```
CONNECT data-source-name -dt ODBC
   -Dsrv query-tuning-option1,value1
   -Dsrv query-tuning-option2,value2.
```

An alternate syntax for the -Dsrv parameter is:

**Syntax**

```
CONNECT data-source-name -dt ODBC -Dsrv
     query-tuning-option1,value1,query-tuning-option2,value2.
```
Table 4–2 describes the query-tuning options that you can specify with the -Dsrv parameter.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qt_no_debug</td>
<td>Specifies whether the DataServer prints debugging information that it</td>
</tr>
<tr>
<td></td>
<td>generates for the query to the dataserv.lg file. The default is qt_no_debug,</td>
</tr>
<tr>
<td></td>
<td>to supply no debugging information. To override the default, specify</td>
</tr>
<tr>
<td></td>
<td>qt_debug,option as follows:</td>
</tr>
<tr>
<td></td>
<td>• Specify qt_debug,SQL to print the SQL statements that the DataServer</td>
</tr>
<tr>
<td></td>
<td>executes against the ODBC data source.</td>
</tr>
<tr>
<td></td>
<td>• Specify qt_debug,EXTENDED to print information such as cursor</td>
</tr>
<tr>
<td></td>
<td>statistics in addition to the SQL statements executed by the DataServer.</td>
</tr>
<tr>
<td></td>
<td>• Specify qt_debug,CURSOR to print information about the cursors that the</td>
</tr>
<tr>
<td></td>
<td>DataServer uses for internal calls and for opening queries.</td>
</tr>
<tr>
<td></td>
<td>• Specify qt_debug,PERFORMANCE to print information on the amount of</td>
</tr>
<tr>
<td></td>
<td>time that certain operations take.</td>
</tr>
<tr>
<td></td>
<td>• Specify qt_debug,CALL_SUMMARY to print information on cursors and</td>
</tr>
<tr>
<td></td>
<td>timing.</td>
</tr>
<tr>
<td></td>
<td>• Specify qt_debug,VERBOSE to print all of the information gathered by</td>
</tr>
<tr>
<td></td>
<td>the other qt_debug options.</td>
</tr>
<tr>
<td></td>
<td>For more detailed descriptions of these options, see Table 4–4.</td>
</tr>
<tr>
<td>qt_debug,SQL</td>
<td>Specifies whether the DataServer uses lookahead or standard cursors. To</td>
</tr>
<tr>
<td></td>
<td>generate efficient queries, qt_lookahead is the default in the following</td>
</tr>
<tr>
<td></td>
<td>cases:</td>
</tr>
<tr>
<td></td>
<td>• Statements that use NO-LOCK</td>
</tr>
<tr>
<td></td>
<td>• Statements that use SHARE-LOCK with transaction isolation level set to</td>
</tr>
<tr>
<td></td>
<td>read uncommitted</td>
</tr>
<tr>
<td>qt_no_lookahead</td>
<td>Specify qt_no_lookahead for query behavior that is consistent with an</td>
</tr>
<tr>
<td></td>
<td>OpenEdge database.</td>
</tr>
<tr>
<td>qt_separate_connection</td>
<td>Specifies whether each cursor should use a separate connection to the ODBC</td>
</tr>
<tr>
<td></td>
<td>data source. The default is qt_no_separate_connection, which provides</td>
</tr>
<tr>
<td></td>
<td>behavior that is consistent with OpenEdge.</td>
</tr>
<tr>
<td></td>
<td>Specify qt_separate_connection to use a separate connection. Executing</td>
</tr>
<tr>
<td></td>
<td>cursors in separate connections can improve performance because the</td>
</tr>
<tr>
<td></td>
<td>DataServer does not have to restart the cursors.</td>
</tr>
<tr>
<td>qt_cache_size,integer</td>
<td>Specifies the size in bytes of the cache used by lookahead cursors. A</td>
</tr>
<tr>
<td></td>
<td>larger cache size can improve performance for queries that return a large</td>
</tr>
<tr>
<td></td>
<td>number of records because the DataServer might need fewer SQL statements</td>
</tr>
<tr>
<td></td>
<td>to get the results.</td>
</tr>
<tr>
<td></td>
<td>Minimum: The DataServer always caches at least one record.</td>
</tr>
<tr>
<td></td>
<td>Maximum: None</td>
</tr>
<tr>
<td></td>
<td>Default: 30000</td>
</tr>
</tbody>
</table>
The following example shows how to use the query-tuning options to enhance performance. The DataServer opens a separate connection to ODBC for each cursor and writes an extended report on the SQL statements it executes:

```
CONNECT holder -db infdb -dt ODBC -ld demo -U user -P password -Dsrv qt_separate_connection,qt_debug,EXTENDED.
```

OpenEdge provides a startup parameter called Server Join (-nojoinbysqldb) that controls the default JOIN–BY–SQLDB behavior. You specify this parameter in the startup command for your OpenEdge session. It overrides the JOIN–BY–SQLDB default so that the client evaluates and performs joins. Using this parameter might slow performance, but it provides results that are consistent with OpenEdge behavior. See Chapter 2, “Programming Considerations,” for more information.

**Binding with connection and startup parameters**

Binding settings determine how data is pulled from the data source by the DataServer. When binding is turned on, query results are copied directly into the DataServer’s memory, eliminating the need to do post-query calls to retrieve the data. When binding is not enabled, queries run independent of the data retrieval, and subsequent calls to the data source are required to retrieve the data.

The default setting is to use binding for both NO-LOCK and EXCLUSIVE-LOCK queries. You can override the default using the -Dsrv BINDING, n connection parameter. The syntax is as follows:

**Syntax**

```
-Dsrv BINDING,n;
```

**Table 4–3** describes the BINDING options that you can specify with the -Dsrv parameter.

**Table 4–3: Connection BINDING options**

<table>
<thead>
<tr>
<th>BINDING value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Indicates binding is disabled</td>
</tr>
<tr>
<td>1</td>
<td>Use binding for NO-LOCK queries only</td>
</tr>
<tr>
<td>2</td>
<td>Use binding for EXCLUSIVE-LOCK queries only</td>
</tr>
<tr>
<td>3</td>
<td>Use binding for both NO-LOCK and EXCLUSIVE-LOCK queries. This value is the default.</td>
</tr>
</tbody>
</table>
Analyzing performance

The qt_debug option of the DataServer (-Dsrv) startup parameter (and the QUERY–TUNING DEBUG phrase) instructs the DataServer to print information on the queries it generates to the dataserv.lg log file. The qt_debug option provides extended diagnostic capabilities that you can use to determine which parts of your application might be causing additional network traffic or extra processing by the ODBC data source. Table 4–4 lists the diagnostic capabilities of qt_debug. Note that the log file uses the numeric values (in parentheses) to identify information that a specific option generates.

Table 4–4: Diagnostic options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qt_no_debug</td>
<td>Supplies no debug information. This is the default.</td>
</tr>
<tr>
<td>qt_debug,SQL</td>
<td>Prints the SQL statements that the DataServer executes against the data source.</td>
</tr>
<tr>
<td>qt_debug,EXTENDED</td>
<td>Prints the SQL statements that the DataServer executes against the data source plus additional information such as cursor statistics.</td>
</tr>
<tr>
<td>qt_debug,CURSOR</td>
<td>Prints information about the cursors that the DataServer uses for internal calls and for opening queries. It tracks when cursors open and close and when the DataServer reuses them. It also summarizes each cursor’s activity. These diagnostics are especially helpful when determining OpenEdge maximum cursor settings or cursor “leaks” that your application might have.</td>
</tr>
</tbody>
</table>
| qt_debug,PERFORMANCE | Prints information on the amount of time that certain operations take. These statistics are available only for some platforms.  

**Note:** Any time differences between what the DataServer reports and what a data source reports might be due to network performance issues rather than to DataServer or data-source behavior. |
| qt_debug,CALL_SUMMARY | Prints information on cursors and timing. This information is supplied in summary form as an OpenEdge data (.d) file. Call Progress Software Corporation Consulting for assistance with this file.  

**Note:** This option is not available as a QUERY–TUNING phrase option. |
| qt_debug,VERBOSE     | Prints all of the information gathered by the other qt_debug options.  

**Note:** This option generates a very large log file. Be sure to clear your log file before using this option to test a procedure. |

**Note:** Turning on debugging options decreases DataServer performance. Be sure to turn off debugging options when you run DataServer applications in production mode.
This connection statement causes the DataServer to report on the time that ODBC operations take:

```
CONNECT odbcdb -U password
-Dsrv qt_cache_size,32000,qt_debug,PERFORMANCE.
```

**Local schema caching**

By using a local file to store schema definitions, you can access them more quickly. Once you create a local schema cache, you connect to the schema holder only when you compile applications. You no longer rely on having a local schema holder for maximum performance. Running DataServer applications with a local schema cache instead of a schema holder also results in better performance in networked environments.

The ABL `SAVE CACHE COMPLETE` statement creates a binary file that contains the entire schema for an OpenEdge database. Use this statement to create a cache file for a connected schema holder:

```
SAVE CACHE COMPLETE schema-holder-name TO filename.
```

For example, the following statement creates a cache file named `sqlcache` for the `sqlhold` schema holder:

```
SAVE CACHE COMPLETE sqlhold TO sqlcache.
```

To use the cache file for a schema holder, specify the Schema Cache File (`-cache`) startup parameter and the cache filename when you connect to the schema holder. For example, the following `CONNECT` statement connects an ODBC-supported database whose data source name is `sqlserv1` with the schema `sqlhold` and tells OpenEdge to use the cache file:

```
CONNECT sqlhold -RO -cache sqlchache -db sqlbdb -dt ODBC -ld sqldemo
-U bob -P bobpass -Dsrv qt_debug,EXTENDED,PRGRS_CONNECT,server=sqlserv1.
```

If you make any changes to a schema holder, you must create a new cache file for it. For more information, see *OpenEdge Getting Started: ABL Essentials* and the `SAVE CACHE` Statement reference entry in *OpenEdge Development: ABL Reference*. 
Connection failures and OpenEdge responses

Table 4–5 lists circumstances under which a connection might fail and describes how OpenEdge responds.

<table>
<thead>
<tr>
<th>Failure circumstance</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>During startup</td>
<td>The system displays an error message and returns to the operating system prompt.</td>
</tr>
<tr>
<td>During a CONNECT statement</td>
<td>The system aborts the remainder of the CONNECT statement as though the remainder never existed, and a run-time error condition occurs. Any connections that you made prior to the failed connection remain in effect. You can use the NO-ERROR option with the CONNECT statement to trap run-time errors. If you use the NO-ERROR option and it fails, you see the same failure behavior as you do with an unsuccessful CONNECT statement. However, run-time error conditions do not occur.</td>
</tr>
<tr>
<td>During an attempted auto-connect</td>
<td>The system aborts the remainder of the connect as though the remainder never existed, and a run-time error condition occurs. Any connections that you made prior to the failed connection remain in effect. You cannot trap auto-connect run-time error conditions.</td>
</tr>
<tr>
<td>During an attempt to connect using the Data Dictionary</td>
<td>The Data Dictionary displays an error message and returns to the main window.</td>
</tr>
<tr>
<td>During an attempt to connect a connected ODBC data source with a different logical name</td>
<td>The system responds with a run-time error condition but you can continue. You can use the NO-ERROR option to suppress the error.</td>
</tr>
<tr>
<td>During an attempt to connect a connected ODBC data source with the same logical name</td>
<td>The system responds with a warning but you can continue. You can use the NO-ERROR option to suppress the warning.</td>
</tr>
<tr>
<td>During an attempt to connect an unconnected ODBC data source whose logical name is already in use by a connected ODBC data source</td>
<td>The system responds with a run-time error condition and you cannot connect to the second database.</td>
</tr>
</tbody>
</table>
Connection troubleshooting

The following reasons might cause a connection attempt to an ODBC data source to fail:

- The schema holder is not connected.
- The OpenEdge or ODBC-required environment variables are not set correctly when using the DataServer and a broker. For environment variable information, see Chapter 3, “Configuring the DataServer.”
- The data-source is not registered properly for ODBC client connectivity.
- You have an outdated version of an ODBC DLL; for example, ODBC16.DLL, which runs on 16-bit machines only. This prevents OpenEdge from accessing the data source, though you might still be able to access the data source through your ODBC driver using another product, such as MS Query.
- The data source has not been started or is not running correctly. Use the data-source utilities to check the status of the data source and the ability to connect to it.
- You omitted a -Dsrv parameter that is required for the data source to which you are attempting to connect. See the “Connecting a schema holder at startup” section on page 4–9 and the “ODBC options” section on page 6–2 for details.
- The login name and password combination that you provided during connection is invalid for the data source.
- You specified an incorrect ODBC data-source name when you created the schema holder.

For more information, see Chapter 6, “Troubleshooting.”
Managing connections to an ODBC data source

Typically, the DataServer maintains one connection to an ODBC data source. In some instances, such as for joins and catalog queries, the DataServer automatically starts separate connections. However, there are circumstances in which a DataServer application might require more than one connection. For example, the DataServer cannot send a query to a data source while a stored procedure is still open unless you specify that the DataServer uses separate connections for each request. Depending on your ODBC driver, the following cases might require a connection to accommodate additional cursors:

- Running a stored procedure, including using the send_sql_statement option
- Performing a join on the server
- Creating or updating the schema image for the data source

In the first case, additional connections are necessary only if your application executes additional database requests while a cursor on a stored procedure is still open.

You can use the -Dsrv qt_separate_connection parameter or the corresponding QUERY–TUNING option (SEPARATE–CONNECTION) to specify that the DataServer uses a separate connection for each statement that requires a cursor. However, if you want to use the main connection when performing joins on the server, use the -Dsrv qt_no_separate_connection parameter when you connect. Note that using a separate connection allows only read-only access to the database. You must issue transactions that require update access to your database from your main connection.

For example, the following statement specifies that the DataServer use a separate connection for the FOR EACH customer query:

```
FOR EACH customer QUERY–TUNING (SEPARATE–CONNECTION):
  DISPLAY customer.
```
Accessing the DataServer log

OpenEdge supports a log file named `dataserv.lg` that is dedicated to tracking information related to DataServers. This single file contains information about the processes for all DataServers operating on your host machine. It provides a useful record of connection and disconnection processes and error messages that you can use to diagnose problems or failures. Each new connection to the same log file appends to the end of the existing file. If you output a detailed log over multiple connections that do a lot of database I/O, the log can quickly take up disk space. For each process, the log provides the following information:

- Physical database name
- Database type
- User ID
- ODBC function calls
- SQL statements

Specifying the `-Dsrv qt_debug` option causes the DataServer to write to the `dataserv.lg` file information about the SQL ODBC calls that it generates as well.

To obtain access to the DataServer log file, using the host machine:

1. Before starting up the broker process, set the `DSLOGDIR` environment variable to the name of the directory where you want to place the log file.
   
   If you set the environment variable, OpenEdge writes the information to the `dataserv.lg` file. If OpenEdge cannot open this file or `%DSLOGDIR%` is unset, it writes the information to the `dataserv.lg` file in the process’ current directory, and appends to it with each subsequent process that uses it.

2. Open the `dataserv.lg` file to read the DataServer log. For information on debug options that affect DataServer log output, see the “Analyzing performance” section on page 4–19.
This chapter presents step-by-step instructions for tasks associated with the DataServer. Some of these exercises relate to maintaining the schema holder. Along with providing an overview of the ODBC demonstration databases, the tutorial describes:

- Preparing to create demonstration databases
- Updating a schema holder
- Changing connection information in a schema holder
- Changing a code page in a schema holder
- Deleting a schema holder
- Migrating an OpenEdge database to an ODBC data source
- Using the Adjust Schema utility
- Modifying a schema holder
- Adding extended ABL support
Demonstration databases for ODBC DataServers

The demonstration databases for the DataServer for ODBC allow you to run the tutorials from this chapter, code examples, or your own procedures.

The demonstration databases are not part of the OpenEdge installation media. To create and initialize a demonstration data source, you run the OpenEdge DB-to-ODBC utility, which migrates the OpenEdge Sports database to your ODBC data source and then creates a schema holder. (See the “Preparing to create demonstration databases” section on page 5–3 for detailed instructions.) After you do this, you are ready to run the tutorial exercises or your own OpenEdge applications against the data source.

Before you create a demonstration database, be sure that your system meets the following prerequisites:

- Sufficient ODBC data-source privileges to create a database, add users, and create tables
- Sufficient disk space for your ODBC data source
Preparing to create demonstration databases

Subsequent sections contain tutorials that illustrate using the DataServer utilities to maintain your schema holder. Before you use these tutorials, you must perform some preparatory steps, including creating the demonstration database and schema holder and accessing the DataServer’s Utilities menu.

The following lists identifies the basic preliminary steps you must complete to create demonstration databases:

- Install and started your data source. This step depends on your unique environment. Refer to your data source documentation for details.
- Install your DataServer drivers.
- Install the OpenEdge client. Refer to the OpenEdge client-related documentation for details.

To create the demonstration database and schema holder using the OpenEdge DB to ODBC utility:

1. Start the ODBC Data Source Administrator tool for your data source. Refer to the “Configuring an ODBC driver and registering the data source” section on page 3–3.
2. Select the appropriate ODBC driver for the schema holder.
3. Select the target ODBC data source for the schema holder. Refer to the “Configuring a local DataServer” section on page 3–5 and the “Configuring the remote DataServer” section on page 3–6.
4. Make any additional selections required by your data source.
5. Specify mysport as the data source name. OpenEdge recognizes the name that you use to register a data source.

   You are not required to specify the data source name as mysport. However, identifying the data source name, using this suggested data source name or another one you choose, can help you to distinguish your tutorial data source from all other data sources you set up.
6. From the Start menu on the task bar, select OpenEdge and either select the specific product you want to use to begin your OpenEdge session or type the following command line in Windows:

   ```
   OpenEdge-install-path\dic\bin\prowin32
   ```

   Starting a local OpenEdge session also automatically starts the local DataServer.
7. Access Data Administration. Create a copy of the Sports database and connect to it.
8. Select DataServer → ODBC Utilities → Schema Migration Tools → OpenEdge DB to ODBC to start the OpenEdge DB-to-ODBC Utility. The conversion utility dialog box opens.
9. Specify OpenEdge parameter values as shown in Table 5–1.

<table>
<thead>
<tr>
<th>Table 5–1: OpenEdge DB-to-ODBC conversion values (1 of 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interface element</strong></td>
</tr>
<tr>
<td>Original OpenEdge Database</td>
</tr>
<tr>
<td>Connect parameters for OpenEdge</td>
</tr>
<tr>
<td>Name of Schema holder Database</td>
</tr>
<tr>
<td>ODBC Data Source Name</td>
</tr>
<tr>
<td>Foreign DBMS Type</td>
</tr>
<tr>
<td>ODBC Username</td>
</tr>
<tr>
<td>ODBC User’s Password</td>
</tr>
<tr>
<td>ODBC connect parameters</td>
</tr>
<tr>
<td>Codepage for Schema Image</td>
</tr>
<tr>
<td>Collation Name</td>
</tr>
<tr>
<td>Collection/Library</td>
</tr>
</tbody>
</table>
Table 5–1: OpenEdge DB-to-ODBC conversion values

<table>
<thead>
<tr>
<th>Interface element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create RECID Column</td>
<td>Enabled for Sybase and DB2/400 data sources only. Leave checked to create RECID column. When migrating to DB2/400 data source, options are enabled to create a RECID column for all tables or for only those tables that do not have a unique key constraint. Checked by default but only for those tables that do not have a unique key constraint by default when data source is DB2/400.</td>
</tr>
<tr>
<td>Load SQL</td>
<td>Leave this toggle box checked.</td>
</tr>
<tr>
<td>Create Shadow Columns</td>
<td>If your ODBC data source is case sensitive, and you wish to maintain OpenEdge-compatible, case insensitive behavior, check this box; otherwise, leave blank.</td>
</tr>
<tr>
<td>Move Data</td>
<td>Check this toggle box to dump and load data from the OpenEdge database to the target database. Copying data from a large database can take a long time. You can uncheck this toggle box if you want to dump and load data at a more convenient time. This toggle box is available only if the Load SQL toggle box is checked.</td>
</tr>
<tr>
<td>Include Defaults</td>
<td>Check to allow default values defined for fields of OpenEdge database.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This option is available only for DB2/400 data sources. If an OpenEdge DATE field has an initial value other than TODAY, then the value will not be pushed to the foreign data source.</td>
</tr>
<tr>
<td>For fields width use:</td>
<td>When pushing fields to a foreign data source, you can select one of two primary field format options:</td>
</tr>
<tr>
<td></td>
<td>• Width — Uses the value of the _width field in the _field record.</td>
</tr>
<tr>
<td></td>
<td>• ABL Format — Compiles with the current default width specified. (default)</td>
</tr>
<tr>
<td></td>
<td>If you select the ABL Format option, you have an additional setting to define:</td>
</tr>
<tr>
<td></td>
<td>• Expand x(8) to 30 — This setting is on by default to indicate that the format for the character fields defined as x(8) will be created as 30 characters.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> You cannot use the Expand x(8) to 30 setting with the Width option.</td>
</tr>
</tbody>
</table>

**Note:** For a complete description of these Progress-to-ODBC parameters, see Table 5–5.
Running the utility creates and connects a schema holder and the ODBC data source. It operates as follows:

a. SQL script is generated.
b. SQL that creates the schema is sent to the foreign data manager.
c. The schema is pulled back to the schema holder.
d. The foreign schema holder and the OpenEdge database are compared and all information needed by OpenEdge is applied to the schema holder.
e. If the **Move Data** toggle box was selected, the data is loaded.
f. The schema holder is disconnected.
g. A message is displayed that tells the user which startup procedure to use to connect.
DataServer utilities

OpenEdge supports a number of utilities that allow you to create and maintain an ODBC DataServer. Choose DataServer → ODBC  Utilities to see the available DataServer utilities, described in Table 5–2.

Table 5–2: DataServer utilities

<table>
<thead>
<tr>
<th>ODBC utility</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create DataServer Schema...</td>
<td>Creates a schema image in the schema holder for an ODBC data source</td>
</tr>
<tr>
<td>Update/Add Table Definitions...</td>
<td>Updates the schema holder to reflect any changes that you make to data-source data definitions</td>
</tr>
<tr>
<td>Verify Table Definition...</td>
<td>Makes sure that the data definitions in the schema holder match your data-source data definitions</td>
</tr>
<tr>
<td>Edit Connection Information...</td>
<td>Changes connection information or the logical database name for a data source</td>
</tr>
<tr>
<td>Change DataServer Schema Code Page...</td>
<td>Changes the code page in the schema holder associated with the ODBC data source</td>
</tr>
<tr>
<td>Delete DataServer Schema...</td>
<td>Deletes a schema image from the schema holder</td>
</tr>
<tr>
<td>Schema Migration Tools</td>
<td>Accesses utilities for migrating or comparing an OpenEdge database to an ODBC data source</td>
</tr>
</tbody>
</table>

When you access a DataServer utility (as you will do in the tutorials that follow this section), the User ID and Password dialog box might appear before the utility opens:

![User ID and Password dialog box](image)

In the User ID and Password dialog box, click OK if you are satisfied with the user ID and password combination that you have already supplied. If you want to change them, or they were never specified, enter a user ID and password with the privileges required for creating and updating a schema holder. See the “Establishing permissions” section on page 3–14 for more information on the required privileges.
Creating a schema holder

To create a schema holder:

1. From the Data Administration main menu, select DataServer → ODBC Utilities → Create DataServer Schema. The Create/Modify Database Record for DataServer Schema dialog box appears:

   ![Create/Modify Database Record for DataServer Schema dialog box]

   - Logical Database Name field
   - Code Page field

2. In the Logical Database Name field, type the name that you will use to connect to your ODBC data source and refer to in your programming applications. This name must be different from the schema holder name. For more information on database names, see the database access chapter in OpenEdge Getting Started: ABL Essentials.

   **Note:** If you place the schema from a second ODBC data source into a schema holder, the second schema must have a different logical database name from the first schema. The schema holder has one physical name, but each schema that it contains must have a different logical name.

3. In the Code Page field, type the name of the code page for the schema holder. The name must be the OpenEdge name for the code page that the data source uses. The default is iso8859–1.

   **Table 5–3** lists the most common ODBC data-source code pages and the equivalent OpenEdge names.

   **Table 5–3:** ODBC data-source and OpenEdge code pages

<table>
<thead>
<tr>
<th>ODBC data-source code page</th>
<th>OpenEdge equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>iso_1</td>
<td>iso8859–1</td>
</tr>
<tr>
<td>(default schema-holder code page)</td>
<td></td>
</tr>
<tr>
<td>cp850</td>
<td>ibm850</td>
</tr>
</tbody>
</table>

   If you use a code page that OpenEdge does not support, you must supply a conversion table that translates between the OpenEdge client code page and the code page that your data source uses. For a complete discussion of code pages, see OpenEdge Development: Internationalizing Applications.
4. In the **Collation** field, enter the name of the collation rule to use. The default is Basic. See the “Client code page” section on page 2–6 for a discussion of collation issues to consider.

5. Type the connection parameters in the **Connection Parameters** field.

   See Chapter 4, “Connecting the DataServer,” for a description of the required and optional connection parameters.

6. In the **ODBC Data Source Name** field, type the name that you used when you registered the data source with the ODBC administration tool.

7. Click **OK**. The utility prompts you for your data-source user ID and password. If they are required by the non-OpenEdge data source and you did not provide them in the **Connection Parameters** field (see Step 5), enter a data-source user ID and password combination that has **select** privileges for the system objects listed in the “Establishing permissions” section on page 3–14, and also has read access to other database objects that the schema holder will include.

8. Click **OK**. When the DataServer connects to the ODBC data source, it reads information about data-source objects. The **Pre-Selection Criteria for Schema Pull** dialog box appears:

![Pre-Selection Criteria for Schema Pull dialog box](image)

**Note:** The **Owner** field in the Pre-Selection Criteria dialog is labeled **Collection/Library** when the foreign DBMS type is DB2/400.

You can select tables based on the object name, owner name, and qualifier. For example, you can specify A* in the Object Name field to list all the tables whose names begin with A.

DB2 for iSeries uses a SQL “Collection” or native “Library” to qualify data object selection. Note that in the above screen capture, the **Owner** label is replaced with **Collection/Library** when the foreign DBMS type is DB2/400. The **Collection/Library** default value is derived from the **Collection/Library** name specified during migration. If you did not migrate your foreign database from OpenEdge and are instead pulling objects from an existing foreign data source into an OpenEdge schema holder, the **Collection/Library** default value is derived from the Data Source Name (DSN) properties of the DB2 driver configuration. You can specify any valid native “Library” in the **Collection/Library** parameter, however OpenEdge expects and Progress recommends that data objects in the foreign data source are defined to the SQL “Collection” specified in the DSN configuration. “Collections” are SQL-compliant so objects defined to them are also SQL-compliant by definition. Data object definitions retrieved from native iSeries libraries are expected to be SQL-compliant as well. Any native attributes of database object definitions retrieved from objects in DB2/400 for iSeries “libraries” are not inclusive in their schema holder definitions if those attributes are purely native, i.e., are not SQL-compliant attributes. Progress recommends defining all database objects...
from all native libraries you wish to define to OpenEdge in a single collection that you also specify in your DSN configuration.

Alternatively, the iSeries Access for Windows native driver for iSeries may provide some native feature mapping not available with the branded DB2 UDB driver provided with OpenEdge. When the native driver is used, if not specified, the default "Collection/Library" name originates from the 'SQL Default Library' specified by the DSN. If 'SQL Default Library' is not specified but a 'Library List' is designated by the DSN, then the first library in the library list is designated as the default 'Collection/Library'. If neither of these values are specified by the DSN, then the user id of the user performing the dictionary action is used for the default 'Collection/Library' value.

**Note:** Progress Software Corporation recommends that you do not specify an entry that consists exclusively of wild cards for each of the three entry fields in the dialog box. An entry that consists exclusively of wild cards might degrade the performance of the database when you perform a schema pull. (It will include system catalog files from the data source not typically included in user databases.)

9. Click **OK**. OpenEdge displays a list of the data-source objects that you can include in the schema holder:

![Select ODBC Objects dialog box]

If you specified all wild cards as your table-selection criteria, the list might also include system-owned objects, which you do not have to include in the schema holder.

10. Click the option appropriate to the action you want to perform:

   - **Select Some** — Displays the **Select by Pattern Match** dialog box on which you can specify information used to select objects
   
   - **Deselect Some** — Displays the **Deselect by Pattern Match** dialog box on which you can specify information used to deselect objects

You can also elect to select and deselect individual objects by clicking and double-clicking on an object. An asterisk appears next to an object that has been selected; double-click an object to remove the asterisk and identify that the object is now deselected.
11. Click **OK** after you have identified all the objects you want to include in the schema holder. The DataServer reads information about the objects that you select and loads their data definitions into the schema holder. The time that this process takes depends on the size and number of objects that you select.

For each data-source table, the DataServer attempts to select an index to support the Progress **ROWID**. If an appropriate index does not exist, the DataServer issues the warning, “Please check warnings and messages in the file ds_upd.e.” The ds_upd.e file lists the objects that do not support **ROWID**. You can change the DataServer’s selection of an index to support **ROWID** by using the Data Dictionary. See the “Defining the **ROWID**” section on page 5–31 for instructions. For additional information, see the “Indexes and sorting” section on page 2–7 and the “**ROWID function**” section on page 2–33.
Updating a schema holder

The Update/Add Table Definitions utility allows you to update a schema holder to reflect changes that you make to the data definitions in your ODBC data source.

**Note:** Using this utility to update the schema holder is the only way to make sure that your applications can access modifications or additions to an ODBC data source.

The Update/Add utility allows you to:

- Add object definitions from the ODBC data source to a schema holder. Use this option if you add a new table or view to the data-source data definitions and want the schema holder to reflect the change.
- Update existing object definitions in a schema holder to reflect a change in the supporting data-source object definitions.

To update a schema holder:

1. Access **Data Administration**, if you are not already there, and select **DataServer → ODBC Utilities → Update/Add Table Definitions**. The **Pre-Selection Criteria For Schema Pull** dialog box appears:

   ![Pre-Selection Criteria For Schema Pull dialog box]

2. Type preselection criteria values into the fields as required. These values preselect the data-source objects that the utility uses to update the schema holder. By default, the wildcard symbol (*) appears; it specifies that the utility uses all of the objects in the data source.

   Note that for a DB2/400 data source, you must type the name of the library in the **Owner/Library** field. For all other DB2 UDB data sources, you must type the name of the DB2 UDB database in this field. If the DB2 UDB database name is different from the authorization ID for the tables being selected, type the authorization ID in the **Owner/Library** field.

   **Note:** For each of the three entry fields in the dialog box, if you enter a value that consists only of wild cards, you might degrade the performance of the database when you perform a schema pull. (It will include system catalog files that are not typically included in user databases.)
3. Click OK. The Select ODBC Objects dialog box lists the objects and table information that you have preselected, for example:

![Select ODBC Objects dialog box]

4. Select the objects that you want to update, then click OK. When the update completes, OpenEdge returns to the Data Administration main window.

When the update completes, OpenEdge reminds you to check the ds_upd.e file. This file contains information about the tables that did not support record IDs as well as other warnings.

When you update a definition, OpenEdge overwrites the old definition with the new one based on the current data-source object. It also preserves the OpenEdge-specific table information. As a result, if you want to add a new column to a table in your ODBC data source and then update the definition, you do not have to reenter all of the OpenEdge-specific information for the previously existing columns (fields) in the definition.

**Note:** When you update a table in the schema holder with the Update/Add Table Definitions utility, the information for the user-defined ROWID is lost. You must reselect an index to support the ROWID.
Verifying a schema holder

The Verify Table Definition utility allows you to verify that the schema holder for your ODBC data source matches the data definitions in the data source. For example, if you delete the Customer table from the data source but not from the schema holder, this utility reports that the schema holder contains an orphaned object. You can verify the schema information in a single table or from multiple tables, and then choose to update the tables so that the schema information matches the data-source definitions.

The Verify utility reads the definitions in the ODBC data source and compares them to the information in the schema holder. It reports the differences that it finds and their degree of severity. These are the categories of differences and how they impact your database applications:

- **Minor** — These differences have no impact on the usability of your application.
- **Retained** — These differences cannot be corrected by the Update/Add utility, hence the term “retained.” You must determine how severely they impact your application and change the data definitions either in the schema holder using the Data Dictionary or in the data source.
- **Severe** — These differences might cause your application to malfunction. When the Verify utility detects severe differences, it automatically updates the schema holder to solve the discrepancies by adjusting the schema-image information in the schema holder to match the data-source definitions. Severe differences in definitions that the DataServer uses internally also cause the schema holder to be updated.

Table 5–4 lists the differences that this utility detects.

<table>
<thead>
<tr>
<th>Database object</th>
<th>Difference</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>Description</td>
<td>Retained</td>
</tr>
<tr>
<td>Table</td>
<td>Foreign type</td>
<td>Severe</td>
</tr>
<tr>
<td>Table</td>
<td>Name in OpenEdge</td>
<td>Retained</td>
</tr>
<tr>
<td>Table</td>
<td>Package name</td>
<td>Severe</td>
</tr>
<tr>
<td>Table</td>
<td>ROWID index</td>
<td>Retained</td>
</tr>
<tr>
<td>Index</td>
<td>Active</td>
<td>Minor</td>
</tr>
<tr>
<td>Index</td>
<td>Description</td>
<td>Retained</td>
</tr>
<tr>
<td>Index</td>
<td>Name in OpenEdge</td>
<td>Retained</td>
</tr>
<tr>
<td>Index field</td>
<td>Abbreviated</td>
<td>Minor</td>
</tr>
<tr>
<td>Index field</td>
<td>Ascending</td>
<td>Severe</td>
</tr>
<tr>
<td>Index field</td>
<td>Order</td>
<td>Severe</td>
</tr>
</tbody>
</table>

Table 5–4: Verify utility report (1 of 2)
To verify a table:

1. Access the Data Administration tool, if you are not already there, and select DataServer → ODBC Utilities → Verify Table Definition. The Pre-Selection Criteria For Schema Pull dialog box appears:

![Pre-Selection Criteria For Schema Pull dialog box]

Note: The Owner field in the Pre-Selection Criteria dialog is labeled Collection/Library when the foreign DBMS type is DB2/400.

2. Type preselection criteria values into the fields if desired. These values preselect the data-source objects that the utility uses to update the schema holder. By default, the wild card symbol (*) appears; it specifies that the utility uses all of the objects in the data source.

1. When you update an index, it is flagged as unique if it was defined as unique in the ODBC data source or the schema holder.

2. If the corresponding information in the ODBC data source is incompatible with the information in the schema holder, the affected fields are not updated. For example, if the data-source data type is decimal and the OpenEdge data type is CHARACTER, the data-type information is not updated.

<table>
<thead>
<tr>
<th>Database object</th>
<th>Difference</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Case sensitivity</td>
<td>Retained</td>
</tr>
<tr>
<td>Field</td>
<td>Decimals</td>
<td>Retained²</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td>Retained</td>
</tr>
<tr>
<td>Field</td>
<td>Extent</td>
<td>Severe</td>
</tr>
<tr>
<td>Field</td>
<td>Initial value</td>
<td>Retained²</td>
</tr>
<tr>
<td>Field</td>
<td>Mandatory</td>
<td>Retained</td>
</tr>
<tr>
<td>Field</td>
<td>Name in OpenEdge</td>
<td>Retained</td>
</tr>
<tr>
<td>Field</td>
<td>Order</td>
<td>Retained</td>
</tr>
<tr>
<td>Field</td>
<td>OpenEdge data type</td>
<td>Retained²</td>
</tr>
<tr>
<td>Field</td>
<td>OpenEdge format</td>
<td>Retained²</td>
</tr>
<tr>
<td>Field</td>
<td>Shadow-column Name</td>
<td>Severe</td>
</tr>
</tbody>
</table>
3. By default, the utility verifies objects in the schema holder that match objects in the ODBC data source. To check whether there are objects in the data source that are not represented in the schema holder, deselect the **Verify only objects that currently exist in the schema holder** toggle box.

4. Select **Output differences to file** if you anticipate a large volume of differences, or wish to save the report. There is a 32K limit on the contents of the verify report. The file that is produced will be named, `<dbname>+.vfy` and will be written to the working directory.

5. Click **OK**. A dialog box lists the objects and table information that you preselected. For example:

![Select ODBC Objects](image)

6. Select the objects that you want to verify, then click **OK**.

7. To select tables by matching a pattern, click **Select Some**. The **Select Objects by Pattern Match** dialog box appears:
8. Type the pattern that you want to match, then click **OK** to start the verification. If you did not choose to output the differences to a file, dialog boxes that list the objects and the verification results appear:

![Schema Verify - Detected Differences dialog box](image)

9. When reading the text of the report, **SH** indicates the value in the schema image; **NS** indicates the value in the ODBC database:

![Verify - Report dialog box](image)

10. Click **Close** to return to the **Schema Verify** dialog box.

The utility automatically selects objects with severe differences for updating. You can select or deselect all other objects as you wish. Note that you **must** resolve retained differences manually. Retained differences appear in subsequent reports until you resolve them.

11. Click **OK** to start the update or **Cancel** to quit the utility without updating the schema holder.
Changing connection information in a schema holder

The Edit Connection Information utility allows you to change ODBC data-source connection information or logical name in the associated schema.

To change connection information for a schema holder:

1. Access Data Administration, if you are not already there, and select DataServer→ODBC Utilities→Edit Connection Information. The Create/Modify Database Record for DataServer Schema dialog box appears:

![Create/Modify Database Record for DataServer Schema](image)

2. Make changes to Connection Parameters as required. When you are done, click OK to return to the Data Administration main window.

The changes do not take effect until you disconnect and reconnect the schema holder. When you reconnect, OpenEdge uses the new connection parameters.

For details on connection parameters, see Chapter 4, “Connecting the DataServer,” and OpenEdge Deployment: Startup Command and Parameter Reference.

Using the Edit utility to change the logical name of an ODBC data source causes the tool to close; this is because you started the tool when you were connected to the data source under another name. To continue working with the utilities, simply restart the tool.

**Note:** If you change the ODBC Data Source Name (DSN), do not select a DSN that uses a different ODBC driver than the original DSN. Configuration switches residing in the schema holder are dependent on driver name. You will receive only a warning if you do use a different driver, but the schema holder configuration may no longer match the characteristics of the driver and cause unpredictable run-time results.
Changing a code page in a schema holder

Using the Change DataServer Schema Code Page utility, you can change the code page in a schema holder at any time. As a result, you can create a schema holder even if you do not have the correct code-page information, then add or correct the code-page information at a later date. However, if you have been writing 8-bit character data to your ODBC data source with the DataServer and then change the code page, the data is unaffected by a code-page change. Note that your database might be corrupted if you start writing data with the DataServer and a schema holder that uses a new code page.

To change a code page in a schema holder:

1. Access **Data Administration**, if you are not already there, and select **DataServer → ODBC Utilities → Change DataServer Schema Code Page**. The utility displays a message about the possibility of corrupting your database by using the wrong code page.

2. Choose **OK** to continue. The **Change Code Page of Non Progress DB** dialog box appears:

3. Either accept the current value or type the OpenEdge and ODBC data-source names separated by a slash (/) for a code page that the data source supports.

   If you are using a code page that the table does not list, see *OpenEdge Development: Internationalizing Applications* for a complete list of code pages that OpenEdge supports.

   If you are using an unsupported code page, OpenEdge allows you to create your own conversion tables.

4. Click **OK** to change the code page and return to the **Data Administration** main window.

   If you were connected to the schema holder and the ODBC data source when you chose to change the code page, OpenEdge disconnects you to make the change. The **Connect Database** dialog box appears to allow you to reconnect.
Deleting a schema holder

The Delete DataServer Schema utility allows you to delete the schema holder for an ODBC data source.

To delete a schema holder:

1. Access Data Administration and select DataServer → ODBC Utilities → Delete DataServer Schema. A dialog box appears, prompting you to verify the deletion.

2. Click Yes to verify your selection. After OpenEdge deletes the schema holder, it displays a confirmation message.

3. Click OK to return to the Data Administration main window.
Migrating an OpenEdge database to an ODBC data source

The DataServer for ODBC supports the OpenEdge DB-to-ODBC utility that allows you to migrate an OpenEdge database to an ODBC data source. While the DataServer typically makes an ODBC data source conform to an OpenEdge database, this utility provides compatibility in the opposite direction. It copies an existing OpenEdge database schema into a target ODBC data source.

The OpenEdge DB-to-ODBC utility performs the following tasks:

- Creates objects in the target ODBC data source
- Creates the schema holder and schema image
- Optionally populates the ODBC data source by dumping and loading the data from the OpenEdge database

The ODBC data source that you create with this utility is a basis for an application database. Before deploying your new ODBC data source, you might want to make manual adjustments to take advantage of additional ODBC-compliant features that are not supported by the migration utility.

The OpenEdge DB-to-ODBC utility requires a local OpenEdge database.

Preparing a data source for the utility

The OpenEdge DB-to-ODBC migration utility does not literally translate definitions for OpenEdge fields into columns in your target data source. It automatically makes adjustments in the target data source in order to provide the functionality of the OpenEdge and target systems.

If your foreign data source supports all the functions of being compatible, the OpenEdge DB-to-ODBC utility will create the objects in the ODBC data source. It converts arrays by using one column in the ODBC data source for each extent of the OpenEdge array. The utility generates names for the columns that it adds to a table in a target data source using the format field-name##extent-number. For example, an OpenEdge field called monthly-amount with an extent of 12 will have 12 columns in the data source with names such as MONTHLY_AMOUNT#1 through MONTHLY_AMOUNT#12. In DB2 UDB, however, you must name these columns column-name_1, column-name_2, and so forth. (Some DB2 UDB OS environments have trouble translating the pound sign (#) in object names.)

Running the OpenEdge DB-to-ODBC utility

The OpenEdge DB-to-ODBC utility runs in Windows with a local DataServer accessing an ODBC target data source through an ODBC driver.

You can run the utility interactively from UNIX with a remote DataServer configuration, on Windows, or in batch mode.
Running the utility interactively

You can run the OpenEdge-to-ODBC utility interactively.

To run the Progress-to-ODBC utility interactively:

1. Create a target data source. You must use an empty target data source when you run the Progress-to-ODBC utility.

2. Configure your ODBC driver to connect to your new target data source and register the target data source.

3. Start the OpenEdge client and connect to the OpenEdge database that you want to migrate to the target data source.

Note: For a DBE (double-byte enabled) DataServer application, you must specify the Internal Code Page (-cpinternal) and Stream Code Page (-cpstream) parameters when you start the OpenEdge client. The values that you specify for these parameters must match the code page that the target data source uses.

4. From Data Administration, choose DataServer→ODBC Utilities→Schema Migration Tools→OpenEdge DB to ODBC.

5. The OpenEdge DB to ODBC Conversion dialog box appears:

The dialog box prompts you for information described in Table 5–5.
### Table 5–5: OpenEdge DB-to-ODBC utility

<table>
<thead>
<tr>
<th>Interface elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original OpenEdge Database</td>
<td>Accept the name of the connected source database or type the name of a database to which to connect.</td>
</tr>
<tr>
<td>Connect parameters for OpenEdge</td>
<td>If you did not specify a new value for the name of the Original OpenEdge Database parameter, do not modify the Connect parameters for OpenEdge parameter. If you did specify a new value, type any additional connect parameters that are necessary.</td>
</tr>
<tr>
<td>Name of Schema holder Database</td>
<td>Type the name of the schema holder. The utility creates the schema holder if it does not exist.</td>
</tr>
<tr>
<td>ODBC Data Source Name</td>
<td>Type the data source name (This is the name you specified when registering the data source in step 2 of the “Running the OpenEdge DB-to-ODBC utility” section on page 5–21). This is the name of the schema image and the name that you will use to refer to the target database in applications. The data-source name must be different from the name that you typed for the schema holder and different from the name of any other schema image existing in that schema holder.</td>
</tr>
<tr>
<td>Foreign DBMS type</td>
<td>Select the foreign data-source type to which the ODBC driver is connecting. OpenEdge provides the following choices: Sybase, DB2, and Other. Select Other if you are accessing a target data source other than the ones listed here. Note that you will get only the generic ODBC SQL functionality if you access databases other than the ones listed here.</td>
</tr>
<tr>
<td>ODBC Username</td>
<td>Type the user ID.</td>
</tr>
<tr>
<td>ODBC User’s Password</td>
<td>Type the password.</td>
</tr>
<tr>
<td>ODBC connect parameters</td>
<td>Type additional connection parameters for the schema holder. The utility provides the required -U and -P parameters, but you might want to specify others.</td>
</tr>
<tr>
<td>Codepage for Schema Image</td>
<td>Type the OpenEdge name for the code page that the ODBC data source uses. By default, the code page for a schema holder is ISO8859-1. You can leave this field blank and use the Change Code Page utility to add the code page information for the schema holder later.</td>
</tr>
<tr>
<td>Collation name</td>
<td>Enter the OpenEdge name for the collation that your ODBC data source will use.</td>
</tr>
<tr>
<td>Collection/Library</td>
<td>Provide the name of the target location for database objects generated by the migration. This value also becomes the qualifier for retrieving object definitions into the schema holder during the pull phase of the migration. For more information, see the “Migration considerations for DB2” section on page 5–25.</td>
</tr>
</tbody>
</table>
If you want a complete migration of your OpenEdge database to a target data source, you must enter information in all fields and check all toggle boxes.

**Create RECID Column**
- Enabled for Sybase and DB2/400 data sources only. Leave checked to create RECID column. When migrating to DB2/400 data source, options are enabled to create a RECID column for all tables or for only those tables that do not have a unique key constraint. Checked by default and second radio button is selected by default when data source is DB2/400.

**Load SQL**
- If enabled, check this toggle box to execute the .sql file that contains the data definition for your OpenEdge database and load these definitions into the target data source.
- Uncheck this toggle box to generate only the SQL script.

**Create Shadow Columns**
- If your ODBC data source is case sensitive, and you wish to maintain OpenEdge-compatible, case insensitive behavior, check this box; otherwise, leave blank. This option is disabled for data sources that are case-insensitive by default.

**Move Data**
- Check this toggle box to dump and load data from the OpenEdge database to the target database. Copying data from a large database can take a long time. You can uncheck this toggle box if you want to dump and load data at a more convenient time.
- This toggle box is available only if the Load SQL toggle box is checked.

**Include Defaults**
- Check to allow default values defined for fields of OpenEdge database.

**Notes:**
- This option is available only for DB2/400 data sources. If an OpenEdge DATE field has an initial value other than TODAY, then the value will not be pushed to the foreign data source.

**For fields width use:**
- When pushing fields to a foreign data source, you can select one of two primary field format options:
  - **Width** — Uses the value of the _width field in the _field record.
  - **ABL Format** — Compiles with the current default width specified. (default)
- If you select the ABL Format option, you have an additional setting to define:
  - **Expand x(8) to 30** — This setting is on by default to indicate that the format for the character fields defined as x(8) will be created as 30 characters.

**Note:** You cannot use the Expand x(8) to 30 setting with the Width option.
The utility creates a schema holder, updates the empty target data source that you created to contain the objects stored in your OpenEdge database, and creates a startup procedure that you can use to connect your schema holder. The startup procedure derives its name from the ODBC name for your target database. For example, if you specified “sports” as the ODBC data-source name, the utility creates the csports.p startup procedure.

Migration considerations for DB2

DB2 for iSeries uses a SQL “Collection” or native “Library” to qualify data object selection. This is in contrast to most SQL data sources that use the owner name of the user performing the migration to determine how to qualify objects migrated to the ODBC data source. Note that the user is prompted for an additional interface element, Collection/Library when the foreign DBMS type is DB2/400. The default value given to the Collection/Library is derived from the Data Source Name (DSN) properties of the DB2 UDB driver configuration. You can specify any valid native “Library” in the Collection/Library parameter however OpenEdge expects and Progress recommends that data objects in the foreign data source are defined to the SQL “Collection” specified in the DSN configuration. “Collections” are SQL-compliant so objects defined to them are also SQL-compliant by definition. Data object definitions retrieved from native iSeries libraries are expected to be SQL-compliant as well. Any native attributes of database object definitions retrieved from objects in DB2/400 for iSeries “libraries” are not inclusive in their schema holder definitions if those attributes are purely native, i.e., are not SQL-compliant attributes. Progress recommends defining all database objects from all native libraries you wish to define to an OpenEdge schema holder in a single collection that you also specify in your DSN configuration.

Alternatively, the iSeries Access for Windows native driver for iSeries may provide some native feature mapping not available with the branded DB2 UDB driver provided with OpenEdge. When the native driver is used, if not specified, the default "Collection/Library" name originates from the 'SQL Default Library' specified by the DSN. If 'SQL Default Library' is not specified but a 'Library List' is designated by the DSN, then the first library in the library list is designated as the default 'Collection/Library'. If neither of these values are specified by the DSN, then the user id of the user performing the dictionary action is used for the default 'Collection/Library' value.

Running the utility in batch mode

To run the Progress-to-ODBC utility in batch mode:

1. Create a target ODBC data source. You must use an empty target data source when you run the Progress-to-ODBC utility.
2. Configure your ODBC driver to connect to your new target data source.
3. On your client machine, pass parameters to the utility by setting the environment variables listed in Table 5–6.
Table 5–6: Progress-to-ODBC utility batch parameters

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLLECTION</td>
<td>Used only when the foreign DBMS type is DB2/400. Allows you to specify the SQL “Collection” or native “Library” for qualifying the data object target location. It also becomes the selection qualifier during the migration phase where objects are pulled from the foreign data source. If a value is not specified, a default value will be derived from the Data Source Name (DSN) configuration defined for the driver in the ODBC Administrator. For more information, refer to the “Migration considerations for DB2” section on page 5–25.</td>
</tr>
<tr>
<td>CRTDEFAULT</td>
<td>Used only when the foreign DBMS type is DB2/400. Default values are pushed to the foreign database during migration, and during a schema pull, default values from the foreign database are stored in the schema holder. Specify YES to enable this behavior.</td>
</tr>
<tr>
<td>PRODBNAME</td>
<td>Specifies the source OpenEdge database name.</td>
</tr>
<tr>
<td>PROCONPARMS</td>
<td>Specifies parameters for the connection to the source OpenEdge database.</td>
</tr>
<tr>
<td>SHDBNAME</td>
<td>Specifies the new schema-holder name.</td>
</tr>
<tr>
<td>ODBCDBNAME</td>
<td>Specifies the logical name of the target data source. The logical name of the data source can be the same as its physical name, but it must be different from the name that you enter for the schema holder.</td>
</tr>
<tr>
<td>ODBCUSERNAME</td>
<td>Specifies the user name for the target data source.</td>
</tr>
<tr>
<td>ODBCPASSWORD</td>
<td>Specifies the password of the user for the target data source.</td>
</tr>
<tr>
<td>ODBCCONPARMS</td>
<td>Specifies additional connection parameters for the schema holder.</td>
</tr>
<tr>
<td>ODCBCODEPAGE</td>
<td>Specifies the OpenEdge name for the code page that the ODBC data source uses. By default, the code page for a schema holder is ibm850. You can leave this field blank and use the Change Code page utility to add the code page information for the schema holder later.</td>
</tr>
<tr>
<td>ODBCCOLLNAME</td>
<td>The Progress name for the collation that your ODBC data source will use.</td>
</tr>
<tr>
<td>ODBCTYPE</td>
<td>Selects the foreign data-source type to which the ODBC driver is connecting. OpenEdge provides the following choices: Sybase, DB2, and Other. Select Other if you are accessing a target data source other than those listed here. Note that you will get only the generic ODBC SQL functionality if you access databases other than those listed here.</td>
</tr>
<tr>
<td>LOADSQL</td>
<td>Allows you to specify whether you want the utility to create the schema in your empty ODBC data source. Specify YES to enable this behavior.</td>
</tr>
</tbody>
</table>
4. Enter the following commands to set and export environment variables at the system prompt, then run protoodb.p:

```
PRODBNAME=db-name; export PRODBNAME
PROCONPARMS=-1 -i
SHDBNAME=schema-holder-name; export SHDBNAME
...
pro -b -p product/odb/protoodb.p
```
Using the Adjust Schema utility

The Adjust Schema utility allows you to compare your ODBC Schema holder information with the original OpenEdge database, updating any OpenEdge attributes in the schema holder. These attributes include field names, labels, help, validation expressions and messages. You must be connected to the OpenEdge database and the schema holder that you want to compare to before running this utility.

To adjust your schema:

1. From Data Administration, select DataServer → ODBC Utilities → Schema Migration Tools → Adjust Schema. The Adjust Schema dialog box appears:

   ![Adjust Schema dialog box]

   If you leave the **all** in the Files To Compare section, all objects are compared and the changes are made. If you only want to compare certain objects, you must enter the information in the following format:

   `<comma-separated list of tables>;<comma-separated list of sequences>;<comma-separated list of views>`

   For example:

   `customer,order;next-order-num,next-cust-num;custview,ordview`

2. Click OK.

   All of the objects that are compared will be displayed on screen as they are processed.
Modifying a schema holder

You can begin using the DataServer as soon as you load your ODBC data-source data definitions into the schema holder. However, you might want to use OpenEdge features such as labels, validation expressions, or validation messages. You also might want to change the default data type provided for fields in the schema image tables.

You can define OpenEdge information at the table and field levels in the schema holder. The following sections describe how to enter OpenEdge information at both levels. An additional section describes how to select an index to support the ROWID function.

Modifying table-level information

You must follow specific steps to modify table-level information.

To modify information in the schema holder at the table level:

1. Access the Data Dictionary.
2. Select a table from the Tables list; for example, the Customer table.
3. Click Table Properties. The Table Properties dialog box appears:

![Table Properties dialog box]

4. Click Validation. The Table Validation dialog box appears. You can change either the validation expression or the message by typing new text in the fields.
5. Click OK to return to the Table Properties dialog box.
6. Click OK to return to the Data Dictionary main window.
Modifying field-level information

You can modify field-level information.

To modify information in the schema holder at the field level:

1. Access the Data Dictionary, if you are not already there, and click Fields. The Fields list appears.
2. Select a table from the Tables list.
3. Select a field from the Fields list.
4. Click Field Properties. The following dialog box appears:

You can enter OpenEdge information at the field level, such as a validation expression or a validation message.

The Data Dictionary displays the standard ODBC SQL names for data types and not the native ODBC data-source names. Using the Data Dictionary, you can make the following changes:

- Change the data type or the format in which OpenEdge displays data. For example, the Sybase smallint data type maps to the ODBC SQL SQL_SMALLINT data type, which in turn maps to the INTEGER data type. However, you can change the SQL_SMALLINT mapping to the DECIMAL or LOGICAL data type instead.

- For CHARACTER fields that are not indexed, you can change the case sensitivity.

Note: You cannot create fields or add mandatory or extent properties to them.
5. Choose **DataServer** to view the field name and position as stored in the ODBC data source. A dialog box similar to the following appears:

![ODBC Specific Fields](image)

**Note:** You cannot change ODBC data-source information using the Data Dictionary. For example, the `custnum` field is named `custnum` in the demonstration database.

6. Click **OK**.

7. When you are done making changes, click **OK** to return to the **Data Dictionary** main window.

**Note:** You can override field-level validation expressions in your application by including the appropriate ABL statement.

### Defining the ROWID

When you create or update a schema holder, the DataServer uses the following guidelines to select a column in an ODBC data-source table to support the **ROWID** function:

- If the data-source table has a `PROGRESS_RECID` column, the DataServer selects that column. A column of this type provides optimal support for the **ROWID** function; you cannot select an alternative to it.

- If the data-source table does not have a `PROGRESS_RECID` column, the DataServer evaluates the available indexes and selects one according to the following preferences:
  
  a. Unique, single-component, mandatory, integer
  
  b. Unique, single-component, integer
  
  c. Unique, single-component, mandatory, any data type except float
  
  d. Unique, multi-component, any data type

If more than one index in the data-source table meets the second level—unique, single-component, integer—the DataServer selects the first such index that it encounters to support the **ROWID** function. Note that the indexes in this class are **not** mandatory, hence it is essential that you enforce the column supporting **ROWID** as mandatory at least through code if not through definitions. If your application handles an index in such a way as to make it a better support for the **ROWID** function, you can designate it in the Data Dictionary.

**Note:** An index that you select as a **ROWID** must be defined as a unique index. It must also be mandatory, if not by definition, then by means of the application code.
You can select an index to support the ROWID function, in the Data Dictionary with the schema holder connected. (You do not have to connect to the ODBC data source.)

**To select an index to support the ROWID function:**

1. Click the **Tables** mode button.
2. Select the table whose ROWID you want to modify.
3. Click the **Table Properties** button.
4. Click the **DataServer** button. The **ROWID Choices** dialog box appears:

![ROWID Choices dialog box](image)

5. Double-click an index to see detailed information on its attributes. The following dialog box appears:

![Detail Information dialog box](image)

6. Click **OK** to return to the **ROWID Choices** dialog box.
7. Select the index that you want to use to support ROWID.
8. Click **OK** to return to the **Table Properties** dialog box.
Modifying tables to support ROWID function for Sybase

If you want to use the ROWID function with an ODBC data source, you must make certain changes to your data-source table. The following example uses Sybase to illustrate the necessary changes.

To modify tables to support the ROWID function, using a Sybase database:

1. Add a column of the integer data type named PROGRESS_RECID. The new column must be able to contain null. For example:

   ```sql
   alter table table
   add PROGRESS_RECID integer null
   ```

2. Add a column with identity characteristics named PROGRESS_RECID_IDENT_. The new column must have the numeric data type. For example:

   ```sql
   alter table table
   add PROGRESS_RECID_IDENT_ numeric(10,0) identity
   ```

3. Create a trigger to maintain the PROGRESS_RECID column, as shown:

   ```sql
   create trigger _TI_table on table for insert as
   begin
     if (select max(inserted.PROGRESS_RECID) from inserted) is NULL
     begin
       update table set PROGRESS_RECID = @@identify
       where PROGRESS_RECID is null
       select convert (int, @@identity)
     end
   end
   ```

4. Change the nonunique indexes so that they include a PROGRESS_RECID column as the last component, as shown:

   ```sql
   create index table##index on table (column, PROGRESS_RECID)
   ```

5. If you have already created your schema holder, update it to reflect your changes to the data-source table.

If your data source supports stored procedures, you may modify the steps above to implement the ROWID function using a similar syntax.
To modify tables to support the ROWID function, using a DB2/400 database:

1. Add a column with identify characteristics named PROGRESS_RECID. The new column must be able to contain null. For example:

```
alter table table
    add PROGRESS_RECID bigint GENERATED ALWAYS AS IDENTITY
    (START WITH 1 INCREMENT BY 1)
```

DB2/400 data sources do not require a trigger or stored procedure to generate the identity value since a value is generated by the system for the identity column automatically.

2. Change the nonunique indexes so that they include a PROGRESS_RECID column as the last component. For example:

```
create index table##index on table (column, PROGRESS_RECID)
```

3. If you have already created your schema holder, update it to reflect your changes to the data source table.
Adding extended ABL support

The DataServer provides support for arrays, case-insensitive indexed fields, and the ABL `ROWID` function when certain objects exist in an ODBC data source. You can modify your data source to support these extended features. You can also add support for these features when you create new tables.

The DataServer maintains the columns that you add to your ODBC data-source tables to support these extended features. If non-OpenEdge applications are updating those data-source tables, these applications must be aware of the additional columns and update them appropriately. You can use triggers, if your data source supports them, to make sure that the columns are updated correctly.

The following sections describe how to modify an existing ODBC data source so that it supports arrays, case-insensitive indexes, and OpenEdge record identifiers. See the “ROWID function” section on page 2–33 for alternate ways to support this function.

Modifying tables to support arrays

The DataServer allows you to extend OpenEdge’s ability to support database arrays to your ODBC data source. To have access to this functionality, you must make certain changes to your data-source table.

To modify tables to support arrays:

1. Name the columns of a data-source table that you want the DataServer to roll into an array `column##1, column##2`, etc. The columns must be adjacent and in sequence.

2. Make sure that these columns are of the same data type. For example, if you want the schema holder to include an array named `MONTH` with 12 elements, the ODBC data-source table must have 12 adjacent columns of the same data type named `month##1, month##2, month##3`, and so forth. OpenEdge names the corresponding field in the schema holder `month`. In your applications, refer to each element of the array as `month[1], month[2], month[3]`, and so forth.

3. If you have already created your schema holder, update it to reflect your changes to the data-source table.

Note: For Informix and DB2/400 arrays, enter the text "__" (two underscores) instead of "##" (two number signs).
**Modifying tables to support case-insensitive indexes**

You can use case-insensitive indexes with an ODBC data source, after making certain changes to your data-source table.

***To modify tables to support case-insensitive indexes:***

1. Add a column of the same data type before the indexed column. The new column must immediately precede the indexed column.

2. Name the column `_S#_column1`.
   
   For example, if your table has an indexed column named `emp_id`, name the new column `_S#_emp_id`. The new column accommodates the uppercase version of the index.

3. Set the `_S#_` column to the uppercase value of the original column.

4. Recreate the index with the `_S#_` column as a component in place of the original column.

5. If you have already created your schema holder, update it to reflect your changes to the data-source table.

---

**Note:** For Informix case-insensitive indexes, provide "_S__" (underscore followed by "S" followed by two "_") instead of "_S#_" (underscore followed by "S" followed by number sign followed by underscore).
Troubleshooting

This chapter describes common problems and how to work around them. Specifically, it explains troubleshooting techniques for:

- Tuning your environment with the -Dsrv startup parameter
- ODBC driver problems in Windows platforms

For information on troubleshooting DataServer connections, see the “Connection failures and OpenEdge responses” section on page 4–21 and the “Accessing the DataServer log” section on page 4–24.
Troubleshooting

Tuning your environment with the -Dsrv startup parameter

The DataServer (-Dsrv) startup parameter allows you to use special ODBC or DataServer options to tune your environment. You pass these options as arguments to -Dsrv when connecting to an ODBC data source.

There are two versions of the syntax, as follows:

Syntax

```
CONNECT data-source-name -ld logical-name -dt ODBC
-Dsrv arg1,val1 -Dsrv arg2,val2 -Dsrv arg3,val3...
```

Syntax

```
CONNECT data-source-name -ld logical-name -dt ODBC
-Dsrv arg1,val1, arg2,val2, arg3,val3...
```

In this syntax:

- The schema holder is already connected.
- The `data-source-name` argument is the name of the data source and the `logical-name` argument is its logical name, which is defined when you created your schema image.
- You pass the options as the `argn,valn` pairs.

Here is an example of how to use the CONNECT statement with the -Dsrv parameter:

```
CONNECT mssql_db -ld mydb -Ux -Py
-Dsrv qt_debug,EXTENDED
-Dsrv PRGRS_IDBUF,100
-Dsrv PRGRS_MINBUF,10000
-Dsrv MAX_R,10000.
```

Note that MAX_R is an abbreviation of MAX_ROWS. You can abbreviate option names as long as they identify parameters uniquely.

Both the syntax statements and the example show the use of the -Dsrv startup parameter in CONNECT statements. You can also specify -Dsrv options in a parameter file, on a program item command line, or in the Connection Parameters field in the Database Connect dialog box.

ODBC options

The DataServer allows you access to selected options defined in the ODBC interface by providing corresponding options that you can use with the -Dsrv startup parameter. Each OpenEdge-supplied ODBC option has a name of the form `option-name` and corresponds with a startup option in the ODBC interface having the name `SQL_option-name`. Table 6–1 lists the OpenEdge-supplied ODBC startup options and the corresponding startup options defined in the ODBC interface.
Tuning your environment with the -Dsrv startup parameter

Refer to any good ODBC application developer’s guide for information on the ODBC-defined options.

When you specify an OpenEdge-supplied ODBC option with the -Dsrv startup parameter, the DataServer sends the option to the ODBC driver for processing by the ODBC interface.

The following example of the -Dsrv startup parameter tells the ODBC driver to return no more than 1,000 rows to the OpenEdge application:

```
-Dsrv MAX_ROWS,1000
```

**Table 6–1: ODBC options**

<table>
<thead>
<tr>
<th>OpenEdge-supplied option</th>
<th>ODBC-defined option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS_MODE</td>
<td>SQL_ACCESS_MODE</td>
</tr>
<tr>
<td>ASYNC_ENABLE</td>
<td>SQL_ASYNC_ENABLE</td>
</tr>
<tr>
<td>AUTOCOMMIT</td>
<td>SQL_AUTOCOMMIT</td>
</tr>
<tr>
<td>LOGIN_TIMEOUT</td>
<td>SQL_LOGIN_TIMEOUT</td>
</tr>
<tr>
<td>MAX_LENGTH</td>
<td>SQL_MAX_LENGTH</td>
</tr>
<tr>
<td>MAX_ROWS</td>
<td>SQL_MAX_ROWS</td>
</tr>
<tr>
<td>NOSCAN</td>
<td>SQL_NOSCAN</td>
</tr>
<tr>
<td>OPT_TRACE</td>
<td>SQL_OPT_TRACE</td>
</tr>
<tr>
<td>PACKET_SIZE</td>
<td>SQL_PACKET_SIZE</td>
</tr>
<tr>
<td>QUERY_TIMEOUT</td>
<td>SQL_QUERY_TIMEOUT</td>
</tr>
<tr>
<td>RESP_POLLCT</td>
<td>SQL_RESP_POLLCT</td>
</tr>
<tr>
<td>RESP_TIMEOUT</td>
<td>SQL_RESP_TIMEOUT</td>
</tr>
<tr>
<td>TXN_ISOLATION</td>
<td>SQL_TXN_ISOLATION</td>
</tr>
</tbody>
</table>

Refer to any good ODBC application developer’s guide for information on the ODBC-defined options.

When you specify an OpenEdge-supplied ODBC option with the -Dsrv startup parameter, the DataServer sends the option to the ODBC driver for processing by the ODBC interface.

The following example of the -Dsrv startup parameter tells the ODBC driver to return no more than 1,000 rows to the OpenEdge application:

```
-Dsrv MAX_ROWS,1000
```

**Note:** The DataServer generally sets the correct value automatically. Therefore, you should reserve use of the ODBC options for troubleshooting and fine-tuning purposes only.
DataServer options

In addition to the ODBC-defined options, the DataServer provides a number of -srv options that are defined on the DataServer side of a connection. Each DataServer option has a name of the form PRGRS_option-name, to reflect its origin. Table 6–2 lists and describes these options.

Table 6–2: DataServer options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRGRS_ALWAYS_INDEX</td>
<td>The value is either 1 for ON or 0 for OFF. The default value is 0. This option determines whether or not the DataServer should always order result sets. If this option is set to OFF, then queries that do not explicitly describe a desired ordering or that do not require ordering by OpenEdge will return unordered results. Note: The driver data source must be able to preserve cursors beyond a transaction boundary in order for you to be able to select the PRGRS_ALWAYS_INDEX option.</td>
</tr>
<tr>
<td>PRGS_CACHE_CONN</td>
<td>Specifies a maximum value to indicate the number of ODBC connections that can be cached without having to close existing connections. The default is 5.</td>
</tr>
<tr>
<td>PRGRS_CONNECT</td>
<td>Passes a connection string to the ODBC driver.</td>
</tr>
<tr>
<td>PRGRS_IDBUF</td>
<td>Specifies the number of keys in the scrolling buffer. The value must be greater than 0. The default value is 25 keys.</td>
</tr>
<tr>
<td>PRGRS_LOCK_ERRORS</td>
<td>Specifies that when an application gets the stated error message, standard error handling behavior occurs (that is, OpenEdge waits and retries rather than halting). Valid values are native ODBC data-source error message numbers.</td>
</tr>
<tr>
<td>PRGRS_MAPFIXEDCHAR</td>
<td>For foreign data types that map to the ODBC SQL_TYPE_CHARACTER fixed length data type, this parameter specifies that white space data should be trimmed from the right or trailing side of a non-white space character value in a result set. Since the OpenEdge CHARACTER data type to which the SQL_TYPE_CHARACTER fixed length type maps is variable in length, it makes sense to trim the padded white space at the end of the fixed area, truncating the server value and NULL terminating that truncated value. Progress recommends you specify this value if the trailing white space in the fixed length character field is not significant to your server's data values or the value you need to receive into the OpenEdge client application.</td>
</tr>
</tbody>
</table>
The following example of the -Dsrv startup parameter sets the number of keys in the scrolling buffer to 100:

```
-Dsrv PRGRS_IDBUF,100
```

The "Using ODBC and DataServer options" section on page 6–6 discusses when and how to use these options.
Using ODBC and DataServer options

This section provides information on using various ODBC and DataServer options.

DataServer connection options

Some data sources (for example, Sybase) support transactions with a degree of granularity that might cause error conditions when executing complex transactions. For example, a transaction might be rolled back unsuccessfully.

Avoid using the data-source transaction manager in such cases. Instead, specify 
-Dsrv AUTOCOMMIT,1 to use the OpenEdge client’s local before-image mechanism to connect to the data source. This startup parameter creates a transaction from each SQL statement that the DataServer issues. OpenEdge emulates the logical transaction on behalf of the application using the local before-image mechanism. This connection technique is useful in multi-database applications.

Notes: OpenEdge does not guarantee crash recovery when it uses the local before-image mechanism to emulate transactions.

The Sybase ODBC drivers and the Informix Wire Protocol ODBC driver require -Dsrv AUTOCOMMIT,1.

Query result order: The PRGRS_ALWAYS_INDEX option

Some queries can gain a performance advantage when the result set from the query does not use an index to order the results. In older versions of the ODBC DataServer, results sets were always indexed because queries and their results were gathered in segments with new cursors being generated for each query segment. However, most advanced drivers and data sources now allow a single cursor to be retained for the life of a given query, even across a transaction boundary. Drivers that can preserve cursors are allowed to execute unindexed queries if the PRGRS_ALWAYS_INDEX option is set to 0 (off). When this option is off and the user does not specify an index or BY clause for the results of their ABL statement and the particular ABL being executed does not require scrolling capability in the OpenEdge client, the query result will remain unindexed by the DataServer.

Note: In an unindexed query, the DataServer will not send an ORDER BY clause to the data source. However, the DataServer has no control over whether or not the actual data source utilizes an index in order to generate a result set.

Concurrent query execution: The PRGRS_CACHE_CONN option

It is possible to run read-only queries in separate connections through the driver to the data source. Opening a separate connection to run a query or stored procedure might provide better performance; although, having too many open connections might also degrade performance. The PRGRS_CACHE_CONN option allows you to set a limit for the maximum number of server connections available in the DataServer session. If the session attempts to exceed the maximum threshold, a session will need to wait for an existing connection to complete first before an additional connection can be made.
Connection problems: The PRGRS_CONNECT option

The PRGRS_CONNECT option allows you to pass ODBC-specific information to the ODBC driver. It has the following syntax:

**Syntax**

```
-Dsrv PRGRS_CONNECT,connection-string;
```

The connection string is separated from the option by a comma (,) and ends with a semicolon (;).

Use the PRGRS_CONNECT option in the following cases:

- To connect to an ODBC data source whose name is not allowed by OpenEdge; for example, a name that includes blank spaces, ampersands (&), commas (,), and/or carets (^). In the connection string, pass the following characters rather than the unallowed characters. The driver resolves the passed characters to the unallowed character:
  - Pass the ampersand (&) character as two ampersand (&&) characters.
  - Pass the caret (^) character as two caret (^^) characters.
  - Pass the blank space character as an ampersand and a caret (&^).
  - Pass the comma (,) character as a caret and ampersand (^&).

- To establish complex connections that require more than the Physical Database Name (-db), User ID (-U), and Password (-P) parameters. In all cases, the values must not be space delimited and must be passed in a single connection string. For example, the following connection string sets the user ID and password for the server and user ID and password for the data source:

```
DSN=sports;UID=engine-login-name;PWD=engine-loginpass;
UIDDBMS=dblogin-name;PWDDBMS=dblogin-pass
```

For more information and syntax examples, see the “Special connection issues” section on page 4–15.

Key-buffer size: The PRGRS_IDBUF option

The PRGRS_IDBUF option sets the size of the key-buffer. Generally, a default of 25 keys is sufficient. If the driver does not have preserved cursors across a transaction boundary, the key-buffer is used with all non-lookahead cursors. If the driver does have preserved cursors across a transaction boundary, use the PRGRS_IDBUF option.
Troubleshooting

Locking error messages: The PRGRS_LOCK_ERROR option

Some data sources do not distinguish between lock problems and errors. The PRGRS_LOCK_ERROR option lets you control how your application reacts if it encounters an error that is actually a lock problem when accessing a data source. Use this option to pass the native error number to the DataServer so that it handles this error as it would a Progress lock problem; that is, the DataServer waits and retries, rather than halting the application:

CONNECT data source name -ld logical-name -dt ODBC
-Dsrv PRGRS_LOCK_ERROR,error-number1,error-number2.

Large rows: The PRGRS_MINBUF option

Some data rows can be very large; for example, in a Sybase data source, rows often have large fields such as IMAGE and MEMO. The ODBC protocol specifies a dynamic buffer allocation process for handling large rows that do not initially fit into clients’ buffers; however, some drivers do not yet follow the correct ODBC protocol and do not handle these large rows correctly. Use the -Dsrv PRGRS_MINBUF, size option to force a minimum buffer size. For example, -Dsrv PRGRS_MINBUF,15000 enables the DataServer to handle 15K rows even with drivers that fail to follow the ODBC protocol.

Note: It is often difficult to determine when there is a buffer size problem and how to choose the correct value for PRGRS_MINBUF. Be careful when using this option.

Schema import: The PRGRS_NO_INDEX option

For ODBC drivers that are not fully ODBC Level 1 compliant, one potential trouble area is data-source index information. The DataServer requires a unique index for each table and view in order to scroll, update, delete, and create rows. If the process of fetching schema information fails, you can reconnect to the data source using the -Dsrv PRGRS_NO_INDEX,1 startup parameter to specify that index information is not automatically imported. Instead, you create a unique index definition manually by using the Data Dictionary. This guarantees that you can create a unique index for a combination of fields that truly have a unique index in the data source. Note that creating the index does not create an index in the ODBC data source; it marks the DataServer for the unique key for that table.

SQL statements cache: The PRGRS_PREPCACHE option

The DataServer keeps a cache of prepared SQL statements that it reuses with new parameters. This mechanism improves the DataServer performance. You can use the PRGRS_PREPCACHE option to manage this cache in two ways:

- Some drivers might fail in reusing a prepared statement or in getting the second record for a particular query. Using -Dsrv PRGRS_PREPCACHE,0 instructs the DataServer to re-prepare each SQL statement.

- Use the PRGRS_PREPCACHE option to control the size of the cache. The default cache size is 20 statements. You can increase the size for large applications that reuse many queries. The maximum size depends on the amount of resources you have available.
Concurrent procedure results: The PRGRS_PROC_TRAN option

The DataServer allows only one active request for running a sorted procedure before you run a second one. However, you can process results from several stored procedures concurrently if you set the PRGRS_PROC_TRAN switch to 1 (on). When switched on, this option will cause a separate connection to be used for each stored procedure request (assuming the PRGRS_CACHE_CONN threshold has not been exceeded).

Caution: When procedures run in separate connections of the same DataServer session, the scopes of their respective transactions are isolated from one other. If one running procedure attempts to update the same record used by another running procedure in the same DataServer session, a lock time out or even a deadlock could occur.

Cursor characteristics: The PRGRS_STABLE_CURS option

The ODBC driver and its data source ultimately determine whether a cursor is stable during the commit or rollback of a transaction and if the cursor can persist beyond a single transaction scope. The DataServer normally resolves cursor characteristics by interrogating the driver and setting the run-time environment accordingly. OpenEdge does not recommend bypassing normal operation under most circumstances. However, if your driver and/or data source do not provide stable cursors, then under very specific circumstances, you might improve performance by setting the PRGRS_STABLE_CURS option to 1 (on). If all the ABL you run in your OpenEdge DataServer session is without transactions or if all of its queries and resultant data sets are fully processed on one or the other side of any existing transaction boundaries, setting this option is possible.

Note: This is a session-level switch which means all ABL run in that session must comply with the above requirements.

Wait time for asynchronous connections: The PRGRS_WAIT_DELAY option

The PRGRS_WAIT_DELAY option is ignored unless you are running in ODBC asynchronous mode. Check your driver and OpenEdge connection requirements to determine if you are allowed to run in asynchronous mode. This option allows you to determine the number of seconds you would want the DataServer to delay further execution while waiting for the driver to respond to a request that might initiate a lock on database resources. The delay is initiated when the NO-WAIT option is used in ABL and the DataServer is awaiting a response to a database request made through the ODBC driver.

RECID Buffer size—the ZPRGRS_RECID_BUF_SIZE,nnn Option

The DataServer for MS SQL Server tries to select a unique index over each table defined in your schema holder to support the ProgressRECID/ROWNID functions. RECID functionality enables backward and forward scrolling in the DataServer product. The RECID buffer is used to sort key information about the unique index selected to support RECID. The default RECID buffer size is 245 bytes. The space is used dynamically to allocate the RECID buffer for a given record. The RECID buffer size needs to be large enough to contain all the key information regarding the unique index. If the unique index selected is a composite of many columns or contains large column names, the RECID key information might exceed the RECID buffer size and issue error message 2090. If you exceed the RECID buffer limit, Progress Software Corporation first recommends that you try to find an alternative unique index with a smaller number of key fields. This should help to improve performance during record access as well. You can change the RECID index in the Data Dictionary by selecting the DataServer button from the Table Properties of the table in your schema holder.
If it is not possible to change the selected RECID unique index for the table that is causing the buffer overflow, you can expand the area by setting the startup option as follows:

```
-Dsrv ZPRGRS_RECID_BUF_SIZE, nnn
```

Where `nnn` is the new size of the area in bytes.

You can expand the RECID buffer size to a maximum of 1000 bytes. The minimum size is 44 bytes. The algorithm to determine minimum adjusted size is as follows:

- 9 + Length of first Index column name + Length of first column data + 17
- + Length of second Index column name + Length of second column data + 22
- + Length of third Index column name + Length of third column data + 22
- + ...
ODBC driver problems in Windows platforms

The ODBC drivers use the MS-Windows DLL technology. A driver problem can leave the driver’s DLL in an unstable state. In this case, the next attempt to connect to the driver might fail with a GPF or with an unexpected error message. If this occurs, you might need to unload the driver’s DLL by using Windows utilities or by restarting Windows.

In addition, drivers utilized by ODBC can get overlaid with incompatible DLL versions as other products are installed on your system. You might need to reinstall ODBC drivers and products if such incompatibilities occur.
Appendix A describes how to upgrade a DataServer application. You can perform either of the following types of upgrades:

- Upgrading schema holders
- Upgrading from one data source version to another
Upgrading schema holders

If you have prior version schema holders and want to take advantage of current OpenEdge version features, you must prepare the schema holder before using the Release 10 DataServer.

To prepare to update a schema holder:

1. Start the version of OpenEdge with which the schema holder was created, then connect to the schema holder.

2. Dump the data definitions (.df file) from the schema holder and move the .df file to the new client machine, if necessary. Dumping and loading a .df file is the only way to preserve any information that you might have added to the schema, such as display formats, help strings, and validation expressions.

3. Start the current version of OpenEdge.

4. Create and connect to an empty OpenEdge database.

5. From the Data Administration main menu, choose **Admin** → **Load Data and Definitions** → **Data Definitions (.df)**.

6. Type the name of your .df file, then click **OK**.

   The utility loads the .df file into the schema holder.

7. Use the Verify Table Definitions DataServer utility to verify the data definitions in the schema holder.

8. Recompile your r-code against the new schema holder.
Upgrading from one data source version to another

Depending on the current version of your ODBC data source, you might need to upgrade it to a more recent version that is supported by the DataServer for ODBC. (For data-source version requirements, see the Release Notes, OpenEdge Getting Started: Installation and Configuration, or the OpenEdge Platform and Product Availability Guide available at www.progress.com.) Note that there is no path for upgrading from an earlier version of your ODBC data source through the DataServer.

To upgrade your ODBC data source to a new driver and data source name (DSN):

1. Upgrade the earlier driver version for connecting your data source to the newer one. See the documentation from your driver vendor for instructions. Documentation for OpenEdge branded drivers can be found in $DLC/odbc.

2. Using the Data Administration Dump utility, dump a data definitions file (.df) from the schema holder you used to access the ODBC data source.

3. Create a new empty OpenEdge database.

4. Using a new DSN and driver, connect to the ODBC data source.

5. Using the Data Administration Load utility, load the .df file into the empty OpenEdge database. This results in a new schema image populated for your data source.

6. Recompile your ABL r-code against the new schema holder.
Appendix B contains reference entries for the following ABL extensions, which support sending SQL statements from within ABL procedures:

- **CLOSE STORED-PROCEDURE statement**
- **PROC-HANDLE function**
- **PROC-STATUS function**
- **RUN STORED-PROCEDURE statement**
CLOSE STORED-PROCEDURE statement

For a non-ABL stored procedure, indicates that the procedure has completed execution and retrieves any return status. For a send-sql-statement stored procedure, closes the SQL cursor used by the procedure.

Syntax

```
CLOSE STORED-PROCEDURE procedure
   [ integer-field = PROC-STATUS ]
   [ WHERE PROC-HANDLE = integer-field ]
```

**procedure**

The name of the stored procedure that you want to close or the built-in procedure name, send-sql-statement.

**integer-field = PROC-STATUS**

Assigns the return value from a stored procedure to the specified integer field or variable (integer-field).

**WHERE PROC-HANDLE = integer-field**

An integer field or variable whose value uniquely identifies the stored procedure that produces the results returned from the data source or the SQL cursor of a send-sql-statement stored procedure.

Notes

- If you specified a PROC-HANDLE when you ran a stored procedure, you must specify the PROC-HANDLE when you close the stored procedure.

- If you do not specify a PROC-HANDLE, the CLOSE STORED-PROCEDURE statement closes the procedure if there is only one stored procedure running. If there is more than one stored procedure running, an error is returned.

- You cannot close a send-sql-statement procedure until you have retrieved all row results.

- You can close all stored procedures at once with the following statement:

  ```
  RUN STORED-PROC closeallprocs.
  ```

See also

PROC-HANDLE function, PROC-STATUS function, RUN STORED-PROCEDURE statement
PROC–HANDLE function

Returns the appropriate return value data type (usually integer) that acts as a unique identifier for an ODBC stored procedure.

Syntax

```
PROC–HANDLE
```

Example

This procedure runs the stored procedure pcust and writes the procedure handle to the variable iHandle. It writes the results of the stored procedure identified by this procedure handle into the ABL buffer, proc–text–buffer, and displays it:

```
DEFINE VARIABLE iHandle AS INTEGER NO-UNDO.
RUN STORED-PROCEDURE pcust iHandle = PROC-HANDLE (10, OUTPUT 0, OUTPUT 0).
FOR EACH proc-text-buffer WHERE PROC-HANDLE = iHandle:
    DISPLAY proc-text.
END.
CLOSE STORED-PROCEDURE pcust WHERE PROC-HANDLE = iHandle.
```

Notes

- Progress Software Corporation recommends that you specify a procedure handle for each stored procedure that you run.
- You do not have to specify a handle if there is only one active stored procedure and you do not include SQL statements in the application.

See also

CLOSE STORED–PROCEDURE statement, PROC–STATUS function, RUN STORED–PROCEDURE statement
PROC–STATUS function

Returns the return status from an ODBC stored procedure. The return status is an integer value that indicates whether a stored procedure failed and why it failed. See your ODBC data-source documentation for descriptions of the possible values for the return status.

Syntax

```plaintext
PROC–STATUS
```

Example

This procedure runs the stored procedure `pcust` and writes the results of the stored procedure into the ABL buffer, `proc–text–buffer`. The CLOSE STORED–PROCEDURE statement then retrieves the output parameters. The return status is written to the variable `iStat` and is displayed:

```plaintext
DEFINE VARIABLE iStat AS INTEGER NO-UNDO.

RUN STORED-PROCEDURE pcust (10, OUTPUT 0, OUTPUT 0).
FOR EACH proc-text-buffer:
   DISPLAY proc-text-buffer.
END.
CLOSE STORED-PROCEDURE pcust iStat = PROC-STATUS.
DISPLAY stat.
```

Note

For descriptions of the possible values for the return status of a non-ABL stored procedure, see your data-source documentation.

See also

CLOSE STORED–PROCEDURE statement, PROC–HANDLE function, RUN STORED–PROCEDURE statement
RUN STORED–PROCEDURE statement

Runs a non-ABL stored procedure or allows you to send SQL to an SQL-based data source using a DataServer.

Syntax

```
RUN STORED–PROCEDURE procedure
  [ integer-field = PROC–HANDLE ]
  [ NO–ERROR ]
  [ ( parameter [ , parameter ] ... ) ]
```

```
procedure

The name of the stored procedure that you want to run or the ABL built-in procedure name, send–sql–statement, to send SQL to an SQL-based data source.
```

```
integer-field = PROC–HANDLE

Assigns a value to the specified integer field or variable (integer-field) that uniquely identifies the stored procedure returning results from the non-OpenEdge database or that uniquely identifies the SQL cursor used to retrieve results from an ODBC-compliant data source.
```

```
NO–ERROR

Specifies that any ERROR conditions that the RUN STORED–PROCEDURE statement produces are suppressed. Before you close a stored procedure, check the ERROR–STATUS handle for information on any errors that occurred. You receive an error when you attempt to close a stored procedure that did not start. The NO–ERROR option must appear before any run-time parameter list.
```

```
parameter

A run-time parameter to be passed to the stored procedure. A parameter has the following syntax:
```

Syntax

```
[ INPUT | OUTPUT | INPUT–OUTPUT ]
[ PARAM parameter-name = ] expression
```

An expression is a constant, field name, variable name, or expression. INPUT is the default. OUTPUT and INPUT–OUTPUT parameters must be record fields or program variables.

If you run send–sql–statement for an SQL-based data source, you must pass a single character expression parameter containing the SQL statement you want the data source to execute.

If you do not specify parameter-name (the name of a keyword parameter defined by the stored procedure), you must supply all of the parameters in correct order. If you do specify parameter-name, you must precede your assignment statement with the keyword PARAM. If you do not supply a required parameter, and no default is specified in the stored procedure, you receive a run-time error.
Examples

This procedure runs the stored procedure `pcust` and writes the results of the stored procedure into the ABL buffer, `proc-text-buffer`:

```
DEFINE VARIABLE iHandle AS INTEGER NO-UNDO.
RUN STORED-PROCEDURE pcust iHandle = PROC-HANDLE (10, OUTPUT 0, OUTPUT 0) NO-ERROR.
IF ERROR-STATUS:ERROR THEN
  MESSAGE "Stored Procedure failed to run".
END.
FOR EACH proc-text-buffer WHERE PROC-HANDLE = iHandle:
  DISPLAY proc-text-buffer.
END.
CLOSE STORED-PROCEDURE pcust WHERE PROC-HANDLE = iHandle.
```

This procedure uses the `send-sql-statement` option of the `RUN STORED-PROCEDURE` statement to send SQL to an ODBC-compliant data source. It writes the results of the stored procedure into the ABL buffer, `proc-text-buffer`:

```
DEFINE VARIABLE iHandle AS INTEGER NO-UNDO.
RUN STORED-PROC send-sql-statement iHandle = PROC-HANDLE ("SELECT name, custnum FROM customer").
FOR EACH proc-text-buffer WHERE PROC-HANDLE = iHandle:
  DISPLAY proc-text_buffer.
END.
CLOSE STORED-PROC send-sql-statement WHERE PROC-HANDLE = iHandle.
```

This code example shows how to trap errors from the non-OpenEdge RDBMS within a procedure:

```
DEFINE VARIABLE iHandle AS INTEGER NO-UNDO.
DEFINE VARIABLE ix AS INTEGER NO-UNDO.
RUN STORED-PROC send-sql-statement iHandle = PROC-HANDLE NO-ERROR ("select count (*) from pcust.customer where name between 'A' and 'Z' ").
IF ERROR STATUS:ERROR THEN DO:
  DO ix = 1 TO ERROR-STATUS:NUM-MESSAGES:
    MESSAGE "error" ERROR-STATUS:GET-NUMBER(ix)
    ERROR-STATUS:GET-MESSAGE(ix).
  END.
END.
FOR EACH proc-text-buffer.
  DISPLAY proc-text-buffer.
END.
CLOSE STORED-PROC send-sql-statement WHERE PROC-HANDLE = iHandle.
```

Note

The `RUN STORED-PROCEDURE` statement starts a transaction with the same scope as transactions started with the `UPDATE` statement.

See also

`CLOSE STORED-PROCEDURE` statement, `PROC-HANDLE` function, `PROC-STATUS` function
Appendix C describes utilities and parameters you use to configure, manage, start, and stop the DataServer host and client. This appendix contains the following sections:

- Progress Explorer command line utilities for the DataServer
- DataServer startup parameters
Progress Explorer command line utilities for the DataServer

This section describes the utilities you use to configure, manage, start, and stop a DataServer. The utilities are presented in alphabetical order. This section discusses the purpose, syntax, and primary parameters for each operating system. See *OpenEdge Getting Started: Installation and Configuration* for additional information about the utilities and their role and relationship to other system administration facilities.
NSCONFIG utility

Use the NSCONFIG utility to help you debug existing NameServer configurations defined in a properties file, such as the ubroker.properties file. This utility displays the property settings associated with a NameServer configuration, and checks that the syntax and values are valid.

The NSCONFIG utility runs locally, on the machine on which the AdminServer is running. The utility does not run across the network.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>nsconfig [ [ -name name-server ] [ -propfile path-to-properties-file ] [ -validate ] ] [ -help ]</td>
</tr>
</tbody>
</table>

Parameters

- **-name name-server**

  Specifies which existing NameServer configuration to examine. The name must match the name of an existing NameServer configuration in the specified properties file. If you do not specify a NameServer, the NSCONFIG utility analyzes all NameServer configurations defined in the properties file specified by the -propfile parameter.

- **-propfile path-to-properties-file**

  Specifies a filename or pathname to a file that contains the property settings to be validated, for example test.properties. If a filename or pathname is not specified, it defaults to the installation version of the ubroker.properties file, such as %DLC%\properties\ubroker.properties in Windows.

- **-validate**

  Checks the syntax and values of property settings defined in the specified properties file.

- **-help**

  Displays command-line Help.

Notes

- A single NameServer can simultaneously support all of the AppServer, WebSpeed and DataServer products using Progress Explorer.

- The ubroker.properties file stores all the configuration definitions for each instance of the NameServer, AppServer, DataServer and WebSpeed Transaction Server products. Each configuration definition contains environment variables, registry entries if Windows, and property settings for each product instance. Progress Explorer and certain command-line utilities, such as NSCONFIG, use this file to store, validate and manage the configurations for the products.
The `ubroker.properties` file is installed in the `properties` subdirectory of the OpenEdge installation directory. For example, `%DLC%\properties\ubroker.properties` in Windows.

The file consists of a hierarchical structure of configuration entities, where parent entities provide configuration information that you can override or extend in each child entity. Each configuration entity has a name that begins the entity definition, and the definition contains configuration settings for one or more product instances.

For example, the NameServer configurations in `ubroker.properties` may include:

<table>
<thead>
<tr>
<th>Configuration entity name</th>
<th>Configuration entity function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[UBroker]</code></td>
<td>Defines default property settings for all NameServer, AppServer, DataServer, and WebSpeed Transaction Server brokers.</td>
</tr>
<tr>
<td><code>[NameServer]</code></td>
<td>Defines default property settings for all instances of a NameServer.</td>
</tr>
<tr>
<td><code>[NameServer.product-instance-name]</code></td>
<td>Defines property settings for this instance of a NameServer. The <code>ubroker.properties</code> file may contain several of these entities each with a unique <code>product-instance-name</code>.</td>
</tr>
</tbody>
</table>

Parent entities provide default values for all of their child entities. For example, the parent `[UBroker]` contains a set of definitions that can be inherited by its child `[NameServer]`, and then again by its child `[NameServer.product-instance-name]`. However, at any child level, a redefinition of any value supersedes the default value of its parent. All children from the redefinition level down inherit this new value.

Optionally, you may edit the `ubroker.properties` file using a text editor such as Notepad. If you want to manually edit this file to create or modify a product configuration, begin by making a backup copy from the installed `ubroker.properties` file (and naming it for example, `test.properties`). Once you edit the properties file, use the relevant validation utility, such as ODBCONFIG for the DataServer properties and NSCONFIG for the Name Server properties, to validate the changes and make sure there are no syntax errors or conflicts.
NSMAN utility

Use the NSMAN utility to control the operation of a configured NameServer. The utility allows you to start a NameServer, query its status, and shut down a NameServer.

Syntax

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>nsman</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>-name name-server</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>-kill</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>[</td>
</tr>
<tr>
<td></td>
<td>-host host-name -user user-name</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>]</td>
</tr>
<tr>
<td></td>
<td>[</td>
</tr>
<tr>
<td></td>
<td>-port port-number</td>
</tr>
<tr>
<td></td>
<td>]</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>-help</td>
</tr>
</tbody>
</table>

Parameters

- **-name name-server**
  
  This parameter is required. It specifies the name of the NameServer.

- **-kill**

  Stops and removes the NameServer from memory, no matter what it is doing.

- **-start**

  Starts the NameServer.

- **-stop**

  Tells the NameServer to stop itself.

- **-query**

  Queries the NameServer for its status.

- **-host host-name**

  Specifies the name of the machine where the AdminServer is running. If a host name is not specified, it defaults to the local host name.
-user user-name

Specifies a user name and prompts for a password. A user name and password are required only when you use the -host parameter and specify a remote host name. If you specify a remote host name with the -host parameter, but do not specify a user name with the -user parameter, you receive a prompt for a username and password.

-port port-number

Specifies the port number of the machine on which the AdminServer is running. If a port number is not specified, it defaults to 20931.

-help

Displays command-line Help.

Notes

• A single NameServer can simultaneously support all of the AppServer, WebSpeed and DataServer products.

• When you specify a user name with the -user parameter, Windows supports three different formats:
  – A user name as a simple text string, such as “mary,” implies a local user whose user account is defined on the local Windows server machine, which is the same machine that runs the AdminServer.
  – A user name as an explicit local user name, in which the user account is defined on the same machine that runs the AdminServer except the user name explicitly references the local machine domain, for example “\mary”.
  – A user name as a user account on a specific Windows domain. The general format is Domain\User, in which the User is a valid user account defined within the domain and the Domain is any valid Windows Server, including the one where the AdminServer is running.
ODBCONFIG utility

Use the ODBCONFIG utility to help you debug existing DataServer for ODBC configurations defined in a properties file, such as the `ubroker.properties` file. This utility displays the property settings associated with a DataServer for ODBC configuration, and checks that the syntax and values are valid.

The ODBCONFIG utility runs locally, on the machine on which the AdminServer is running. The utility does not run across the network.

```
odbconfig path-to-properties-file -validate
```

### Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
</table>
| Windows          | odbconfig
|                  | [     |
|                  | [     |
|                  | [     |
|                  | [-name DataServer-name ] |
|                  | [ -propfile path-to-properties-file ] |
|                  | [ -validate ] |
|                  | ] |
|                  | [ -help ] |
|                  | ] |

### Parameters

- **-name DataServer-name**
  Specifies which existing DataServer for ODBC configuration to examine. The name must match the name of an existing DataServer for ODBC configuration defined in the specified properties file. If you do not specify a DataServer by name, the ODBCONFIG utility analyzes all DataServer for ODBC configurations defined in the properties file specified by the -propfile parameter.

- **-propfile path-to-properties-file**
  Specifies a filename or pathname to a file that contains the property settings to be validated, for example `test.properties`. If a filename or pathname is not specified, it defaults to the installation version of the `ubroker.properties` file, such as `%DLC%\properties\ubroker.properties` on Windows.

- **-validate**
  Checks the syntax and values of property settings defined in the specified properties file.

- **-help**
  Displays command-line Help.
Notes

- The ubroker.properties file stores all the configuration definitions for each instance of the NameServer, AppServer, DataServer and WebSpeed Transaction Server products. Each configuration definition contains environment variables, registry entries if Windows, and property settings for each product instance. Progress Explorer and certain command-line utilities, such as ODBCONFIG, use this file to store, validate and manage the configurations for the products.

- The ubroker.properties file is installed in the properties subdirectory of the OpenEdge installation directory. For example, %DLC%\properties\ubroker.properties on Windows.

- The file consists of a hierarchical structure of configuration entities, where parent entities provide configuration information that you can override or extend in each child entity. Each configuration entity has a name that begins the entity definition, and the definition contains configuration settings for one or more product instances.

For example, the DataServer for ODBC configurations in ubroker.properties might include:

<table>
<thead>
<tr>
<th>Configuration entity name</th>
<th>Configuration entity function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[UBroker]</td>
<td>Defines default property settings for all NameServer, AppServer, DataServer, and WebSpeed Transaction Server brokers.</td>
</tr>
<tr>
<td>[UBroker.OD]</td>
<td>Defines default property settings for all instances of DataServers for ODBC.</td>
</tr>
<tr>
<td>[UBroker.OD.product-instance-name]</td>
<td>Defines property settings for this instance of a DataServer for ODBC. The ubroker.properties file may contain several of these entities each with a unique product-instance-name.</td>
</tr>
</tbody>
</table>

- Parent entities provide default values for all of their child entities. For example, the parent [UBroker] contains a set of definitions that can be inherited by its child [UBroker.OD], and then again by its child [UBroker.OD.product-instance-name]. However, at any child level, a redefinition of any value supersedes the default value of its parent. All children from the redefinition level down inherit this new value.

- Optionally, you may edit the ubroker.properties file using a text editor such as Notepad. If you want to manually edit this file to create or modify a product configuration, begin by making a backup copy from the installed ubroker.properties file (and naming it for example, test.properties). Once you edit the properties file, use the relevant validation utility, such as ODBCONFIG for the DataServer properties and NSCONFIG for the Name Server properties, to validate the changes and make sure there are no syntax errors or conflicts.
ODBMAN utility

Use the ODBMAN utility to control the operation of a configured DataServer for ODBC. The utility allows you to start a broker, query its status, start and stop additional DataServer servers, and shut down the broker.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>odbman</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>-name DataServer-name</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>-kill</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-start</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-stop</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-query</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>[</td>
</tr>
<tr>
<td></td>
<td>-host host-name -user user-name</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-user user-name</td>
</tr>
<tr>
<td></td>
<td>]</td>
</tr>
<tr>
<td></td>
<td>[</td>
</tr>
<tr>
<td></td>
<td>-port port-number</td>
</tr>
<tr>
<td></td>
<td>]</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-help</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

Parameters

- **-name DataServer-name**

  This parameter is required. It specifies the name of a broker.

- **-kill**

  Stops and removes the DataServer from memory, no matter what it is doing.

- **-start**

  Starts the DataServer.

- **-stop**

  Tells the DataServer to stop itself.

- **-query**

  Queries the DataServer for its status.

- **-host host-name**

  Specifies the name of the machine where the AdminServer is running. If a host name is not specified, it defaults to the local host name.
-user user-name

Specifies a user name and prompts for a password. A user name and password are required only when you use the -host parameter and specify a remote host name. If you specify a remote host name with the -host parameter, but do not specify a user name with the -user parameter, you receive a prompt for a user name and password.

-port port-number

 Specifies the port number of the machine on which the AdminServer is running. If a port number is not specified, it defaults to 20931.

-help

Displays command-line Help.

Notes

• When you specify a user name with the -user parameter, Windows supports three different formats:
  – A user name as a simple text string, such as “mary,” implies a local user whose user account is defined on the local Windows server machine, which is the same machine that runs the AdminServer.
  – A user name as an explicit local user name, in which the user account is defined on the same machine that runs the AdminServer except the user name explicitly references the local machine domain, for example “.\mary”.
  – A user name as a user account on a specific Windows domain. The general format is Domain\User, in which the User is a valid user account defined within the domain and the Domain is any valid Windows Server, including the one where the AdminServer is running.
PROBRKR command

Starts the DataServer broker. To use the DataServer from a remote client, you must first start the broker. Once you start the broker, it can receive the client requests and spawn the appropriate DataServer.

Syntax

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>_probrkr.exe dbname -S service-name [ -H host-name ]</td>
</tr>
</tbody>
</table>

Parameters

dbname

Specifies the name of the database where you are connecting to.

service-name

Specifies the name of the broker process on the host machine.

host-name

Specifies the name of the machine where the DataServer broker is installed. The default value is the current host.

Notes

- See OpenEdge Deployment: Startup Command and Parameter Reference for more details on the Server Name (-S) and Host Name (-H) startup parameters.

- You can use any of the startup parameters with the PROBRKR command. See OpenEdge Deployment: Startup Command and Parameter Reference for details.

- You must start the remote broker in the same environment in which your ODBC data source names (DSNs) are defined because the servers spawned by the broker inherit the setup of the environment from the broker. For example, set the environment variable ODBSRV to the name of the executable (including the path) of the DataServer for ODBC. Be sure to set this variable on the host machine. Also, in the same environment, make sure you have set all ODBC environment variables required to connect to the data source. See Chapter 3, “Configuring the DataServer,” for examples of required variables.

- Start the broker on a node that is locally connected to the disk containing the data source.

- To shut down the broker, use the PPROSHUT command with the Gateway parameter (-Gw).
# DataServer startup parameters

Startup commands and parameters for UNIX and Windows allow you to start and manage DataServer clients. See *OpenEdge Deployment: Startup Command and Parameter Reference* for additional information about syntax and usage.

Table C–1 lists the parameters that you use with the command line utilities to start a DataServer for ODBC.

## Table C–1: DataServer parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataServer</td>
<td>-Dsrv keyword,value</td>
</tr>
<tr>
<td>Database Type</td>
<td>-dt db-type</td>
</tr>
<tr>
<td>Host Name</td>
<td>-H host-name</td>
</tr>
<tr>
<td>Service name</td>
<td>-S service-name</td>
</tr>
<tr>
<td>Data Source User Name</td>
<td>-U user-name</td>
</tr>
<tr>
<td>Data Source User Name Password</td>
<td>-P password</td>
</tr>
<tr>
<td>Server Join</td>
<td>-nojoinbysqldb</td>
</tr>
</tbody>
</table>
ODBC data-source data types differ from OpenEdge data types. However, each data-source data type supported by the DataServer has at least one OpenEdge equivalent. Appendix D provides details on DB2 UDB, SQL-ODBC and OpenEdge data types.
The DataServer translates ODBC data-source data types into OpenEdge equivalents and places the mapping into the schema holder. You can access this mapping information using the Data Dictionary. Note, however, that the Data Dictionary lists the ODBC SQL equivalents for data-source data types rather than the actual data-source data types. For example, the Data Dictionary lists the Sybase datetime and smalldatetime data types as timestamp, which is the equivalent ODBC SQL data type and not the Sybase timestamp data type. There may also be some restrictions in data-source compatibility with OpenEdge. For example, the OpenEdge database cannot support a numeric or decimal field defined in Sybase with greater than 28 digits.

You can also modify these definitions using the Data Dictionary. For example, the DataServer maps the Sybase tinyint data type to the OpenEdge equivalent, INTEGER. Suppose, however, that your application uses the tinyint field in such a way that the LOGICAL data type is a more suitable equivalent. In this case, you would change the data type from INTEGER to LOGICAL in the schema holder. If you do change a data-type mapping, be sure to select a data type that accommodates the data in the column, otherwise conversion errors might occur at run time. Also, remember to specify a display format that is appropriate for the new data type. See the “Changing a code page in a schema holder” section on page 5–19 for an explanation of how to use the Data Dictionary to change OpenEdge data types in the schema holder.

The tables in the following sections list, for each ODBC data source, the data types supported by the DataServer, the ODBC SQL equivalents, and the default OpenEdge equivalents. The notes that follow some tables provide additional information.

Table D–1 lists the DB2 UDB data types supported by the DataServer, their ODBC SQL equivalents, and the default OpenEdge equivalents. The data types in parentheses are alternative data types that you can specify in the schema holder for your DB2 UDB data source. You cannot change the list of default data types in a schema holder for a DB2 UDB data source.

### Table D–1: DB2 UDB data type equivalencies

<table>
<thead>
<tr>
<th>DB2 UDB Data type</th>
<th>SQL-ODBC data type</th>
<th>OpenEdge default</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>SQL_TYPE_INTEGER</td>
<td>INTEGER (DECIMAL)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(INTEGER)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(INT64)</td>
</tr>
<tr>
<td>bigint</td>
<td>SQL_BIGINT</td>
<td>INT64 (DECIMAL)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(INTEGER)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(LOGICAL)</td>
</tr>
<tr>
<td>smallint</td>
<td>SQL_TYPE_SMALLINT</td>
<td>INTEGER (DECIMAL)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(INTEGER)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(INT64)</td>
</tr>
<tr>
<td>decimal</td>
<td>SQL_TYPE_DECIMAL</td>
<td>DECIMAL (INTEGER)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(INT64)</td>
</tr>
<tr>
<td>real/float</td>
<td>SQL_TYPE_REAL</td>
<td>DECIMAL (INTEGER)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(INT64)</td>
</tr>
<tr>
<td>double precision/float</td>
<td>SQL_TYPE_FLOAT</td>
<td>DECIMAL (INTEGER)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(INT64)</td>
</tr>
<tr>
<td>date</td>
<td>SQL_TYPE_TIMESTAMP</td>
<td>CHARACTER (DATE)</td>
</tr>
</tbody>
</table>
Table D–1: DB2 UDB data type equivalencies (2 of 2)

<table>
<thead>
<tr>
<th>DB2 UDB Data type</th>
<th>SQL-ODBC data type</th>
<th>OpenEdge default</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>SQL_TYPE_TIMESTAMP</td>
<td>CHARACTER$^{3,4,5}$</td>
</tr>
<tr>
<td>timestamp</td>
<td>SQL_TYPE_TIMESTAMP</td>
<td>DATETIME or DATETIME-TZ (CHARACTER) (DATE)</td>
</tr>
<tr>
<td>char(n)</td>
<td>SQL_TYPE_CHARACTER</td>
<td>CHARACTER$^6$</td>
</tr>
<tr>
<td>varchar(n)</td>
<td>SQL_TYPE_VARCHAR</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>long varchar(n)</td>
<td>SQL_TYPE_LONGVARCHAR</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>graphic</td>
<td>SQL_TYPE_BINARY</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>vargraphic(n)</td>
<td>SQL_TYPE_VARBINARY</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>long vargraphic</td>
<td>SQL_TYPE_LONGBINARY</td>
<td>CHARACTER</td>
</tr>
</tbody>
</table>

1. The DataServer truncates values in DB2 UDB decimal or numeric columns defined with a scale greater than 10. In the case of float columns, it reliably translates up to 10 places only.

2. Do not use the float or real data types in joins, in primary indexes, or with the equality operator.

3. When you change the default mapping of the DB2 UDB timestamp or time data types to the OpenEdge DATE data type, OpenEdge truncates the time portion of the date.

4. The DB2 UDB timestamp data type contains both date and time information. The DataServer maps this to the OpenEdge CHARACTER data type; however, you can change the CHARACTER data type to DATE in the schema holder. If you do, remember to change the format to match the new data type. For example, if you change the data type to DATE, specify a date format, such as 99/99/99.

5. The default format used for a TIME data type is “99:99:99”. A TIME column mapped to a CHARACTER type in the ODBC schema holder can be received on update into that TIME column as a 6-byte, undelimited value, such as “101112” (meaning 10 hours, 1 minutes and 12 seconds). Or, it can be received as a 5-8 byte, colon-delimited, value, for example, “1:2:3” or “10:11:12”. The TIME value received must be in hours, minutes and then seconds. No white space is allowed in the string format. The colon (:) is the only supported delimiter character.

6. SQL_TYPE_CHARACTER is a fixed length data type in a foreign database whereas the OpenEdge CHARACTER data type is variable length. If you retrieve data from a fixed length character data type from a foreign database, the value will be padded with blanks for the length of the fixed length data. If it is desirable to have the OpenEdge database trim white space data from the right or trailing side of a fixed length character value and then null terminated for the OpenEdge data conversion, you should specify the PRGRS_MAPFIXEDCHAR -Dsrv switch in your connection parameters to the foreign schema.
Sybase

Table D–2 lists the Sybase data types, their ODBC SQL equivalents, and their default OpenEdge equivalents. The data types in parentheses are alternative data types that you can specify in the schema holder for your Sybase data source.

Table D–2: Sybase data type equivalencies

<table>
<thead>
<tr>
<th>Sybase data type</th>
<th>SQL-ODBC data type</th>
<th>OpenEdge default</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>SQL_INTEGER</td>
<td>INTEGER (DECIMAL)</td>
</tr>
<tr>
<td>smallint</td>
<td>SQL_SMALLINT</td>
<td>INTEGER (DECIMAL or LOGICAL)</td>
</tr>
<tr>
<td>tinyint</td>
<td>SQL_TINYINT</td>
<td>INTEGER (DECIMAL or LOGICAL)</td>
</tr>
<tr>
<td>decimal</td>
<td>SQL_DECIMAL</td>
<td>DECIMAL (INTEGER)</td>
</tr>
<tr>
<td>numeric$^3$</td>
<td>SQL_DECIMAL</td>
<td>DECIMAL (INTEGER)</td>
</tr>
<tr>
<td>float$^2$</td>
<td>SQL_FLOAT</td>
<td>DECIMAL (INTEGER)</td>
</tr>
<tr>
<td>double precision</td>
<td>SQL_DOUBLE</td>
<td>DECIMAL (INTEGER)</td>
</tr>
<tr>
<td>real</td>
<td>SQL_REAL</td>
<td>DECIMAL (INTEGER)</td>
</tr>
<tr>
<td>char$^3$</td>
<td>SQL_CHAR</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>varchar$^5$</td>
<td>SQL_VARCHAR</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>nchar$^4$</td>
<td>SQL_CHAR</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>nvarchar$^4$</td>
<td>SQL_VARCHAR</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>text, ntext</td>
<td>SQL_LONGVARCHAR</td>
<td>CHARACTER$^5$</td>
</tr>
<tr>
<td>money</td>
<td>SQL_DECIMAL</td>
<td>DECIMAL (INTEGER)</td>
</tr>
<tr>
<td>smallmoney</td>
<td>SQL_DECIMAL</td>
<td>DECIMAL (INTEGER)</td>
</tr>
<tr>
<td>datetime</td>
<td>SQL_TIMESTAMP</td>
<td>CHARACTER$^6$ (DATE$^7$)</td>
</tr>
<tr>
<td>smalldatetime</td>
<td>SQL_TIMESTAMP</td>
<td>CHARACTER$^6$ (DATE$^7$)</td>
</tr>
<tr>
<td>binary</td>
<td>SQL_BINARY</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>varbinary</td>
<td>SQL_VARBINARY</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>image</td>
<td>SQL_LONGVARBINARY</td>
<td>CHARACTER$^5$</td>
</tr>
<tr>
<td>bit</td>
<td>SQL_BIT</td>
<td>LOGICAL</td>
</tr>
<tr>
<td>timestamp$^8$</td>
<td>SQL_VARBINARY</td>
<td>Unsupported</td>
</tr>
<tr>
<td>identity</td>
<td>NA</td>
<td>You can only display these values$^9$</td>
</tr>
<tr>
<td>bigint</td>
<td>SQL_BIGINT</td>
<td>INT64 (INTEGER, DECIMAL, or LOGICAL)</td>
</tr>
</tbody>
</table>

1. The DataServer truncates values in Sybase decimal or numeric columns defined with a scale greater than 10. In the case of float columns, it reliably translates up to 10 places only.
2. Do not use the float or real data types in joins, in primary keys, or with the equality operator.

3. When you define a binary or char column to allow nulls, Sybase stores the data type definitions as varbinary and varchar respectively. The equivalent SQL data types that the Data Dictionary displays for these are SQL VARBINARY, SQL VARCHAR, and SQL VARCHAR. This does not affect how the DataServer maps the Sybase data types to OpenEdge data types.

4. You can access nchar and nvarchar data types as Sybase objects and bypass their conversion to CHARACTER by running a Sybase stored procedure or using the send–sql–statement option supported by the DataServer.

5. Although the Sybase text and image fields can hold up to 2MB, OpenEdge retrieves only up to 32K. If you are using an OpenEdge Format phrase, there might be additional limits on the size of text and image fields. See the Format Phrase entry in the OpenEdge Development: ABL Reference. You can use the DataServer (-Dsrv MAX_LENGTH) startup parameter to limit the amount of returned text data.

6. By default, the initial value of a Sybase datetime or smalldatetime column is unknown ("?"). The default initial values for binary and varbinary are also unknown ("?"). The Sybase datetime and smalldatetime data types contain both date and time information. The DataServer maps these to the OpenEdge CHARACTER data type; however, you can change the CHARACTER data type to INTEGER or DATE in the schema holder. If you do this, remember to change the format to match the new data type. For example, if you change the data type to DATE, specify a date format, such as 99/99/99.

7. When you change the default mapping of Sybase datetime or smalldatetime data types to the OpenEdge DATE data type, OpenEdge truncates the time portion of the date.

8. The DataServer considers a timestamp data type to be a hidden value. It is not visible to the user, but you can still access a Sybase table that contains a timestamp column.

9. You can display values in identity columns, but you cannot insert or update them.
Index

A
Abbreviated index 2–40
Aggregates
   in views 2–11
Analyzing performance 4–19
Arrays 2–13
   in joins 2–59
   manually creating support 5–35
   supporting in data source 5–35
Auto-commit 6–6
Auto-connect
   failure response 4–21

B
BEGINs function 2–40
   empty character strings 2–39
Binding 4–18
Brokers
   DataServer C–11
   starting from MS-DOS 3–7, 4–6
   Unified Broker 3–22
Buffers
   allocating dynamically 6–8
   defining for stored procedures 2–49
BY option
   FOR EACH statement 2–8
   joins 2–59

C
Cache files
   changes to schema holders 4–20
   creating 4–11
   local schema cache 4–20
Caching
   schema 1–6
Case-insensitive indexes 2–9
   manually creating support 5–36
   supporting in data source 5–35
Character sets 2–5
   changing in a schema holder 5–19
CLOSESTORED-PROCEDURE statement
   2–46, B–2
   retrieving parameter values 2–48
Code pages 2–5
   changing in a schema holder 5–19
   collation table 2–6
   conversion table 2–6
   setting in schema holder 3–18, 5–8
COMPILE statement 2–38
Configuration examples
   local DataServer 4–12
   remote DataServer 4–13
CONNECT statement
   failure response 4–21
   NO-ERROR option 4–21
Connecting schema holders 4–9
   at startup 4–9
Connection BINDING Options 4–18
Connection to data source
  blank space in data source name 4–15
  DataServer opens multiple connections 4–23
  failure response due to existing connection 4–21
  failure response due to logical name 4–21
  failures with SQL Server 4–21
  guidelines 4–8
  interactive connection 4–15
  non-standard connections 4–15
  security on NT 4–8
  troubleshooting 4–22, 6–6, 6–7, 6–9
CONTAINS operator 2–40
COUNT-OF function 2–40
CREATE statement 2–17, 2–40
Creating OpenEdge databases 3–16
CURRENT-VALUE statement 2–40
Cursor
  defined 2–29
Cursors 2–29
  lookahead 2–56
  multiple 4–23

D

Data Dictionary
  connection failure response 4–21
  modifying schema holders 5–29

Data sources 1–3
  adding objects to schema holder 3–20, 5–10
  code page 3–18, 5–8
  data types D–2
  database objects 2–3
  keys 2–7
  locking behavior 2–19
  logical name 3–17, 5–8
  login name 2–41, 3–15
  naming conventions 2–4
  null value 2–13
  password 2–41, 3–15
  permissions 3–14
  physical name 3–18, 5–9
  registering 3–3
  security levels 1–7
  stored procedures 2–42
  triggers 2–11
  upgrading from earlier version 1–18
  upgrading versions A–3
  views 2–10
  zero-length character strings 2–14

Data types
  changing 5–31
  display format 5–31
  overflow checking 2–28
  stored procedures 2–43
  supported by DataServer D–2
  user-defined 2–13

Database conversion
  Progress-to-ODBC utility 5–21

Database Type (-dt) startup parameter 4–10

Databases
  ODBC data-source objects 2–3
  OpenEdge objects 2–3
  triggers 2–11

DataServer
  architecture 1–4
  broker
    on host machine 3–6
    compiling applications 2–38
    components 3–2
    configurations 1–10
    configuring for distributed applications using probrkr 1–12
    configuring with Progress Explorer on NT 3–6
    connection failure 4–21
    creating schema holders 3–17, 5–8
    diagnostic options 4–19
    enhancing performance 2–54
    error handling 2–26
    internal logic 1–6
    local configuration 1–10
    log file 3–8, 4–24
    ODBC utilities 1–3
    on host machine 3–6
    options 6–4
    record creation 2–16
    remote configuration 1–11
    scrolling 2–29
    security considerations 1–7
    starting
      in Progress Explorer 4–3
      MS Windows client 4–6
      UNIX broker processes 4–6
      UNIX client 4–7
    upgrading to OpenEdge Release 10 1–17
    utilities 1–8

DataServer (-Dsrv) startup parameter 4–11
  alternate syntax 4–16

DataServer options 6–5
  ODBC options 6–3
  PRGRS_CONNECT option 4–15
  qt_debug option 4–17
  qt_lookahead option 4–17
  qt_no_debug option 4–17
  qt_no_lookahead option 4–17
  qt_no_separate_connection option 4–17
  qt_separate_connection option 4–17
  query-tuning 4–16
stored procedures 2–43
  tuning your environment 6–2
DataServer for ODBC 1–3
DataServers
  starting a broker C–11
DB2
  case sensitivity 2–10
DBE DataServer
  startup parameters 5–22
DBRESTRICTIONS function 2–39
Deadlock 2–22
  stored procedures 2–43
DEFINE BROWSE statement 2–35
DEFINE BUFFER statement 2–49
Deleting schema holders 5–20
Demonstration databases 1–9, 5–2
  requirements for use 5–2
Diagnostic options
  qt_debug 4–19
DirectConnect 4–11
Display formats 5–31
Distributed DataServer
  configured using probrkr 1–12
Documentation
  suggested reading list 1–18
DSLOGDIR environment variable 3–8, 4–24
Dummy indexes
  for non-indexed data sources 2–8

E
Enterprise DataServer for ODBC 1–2, 3–2
Environment variables
  DSLOGDIR 3–8
  ODDBRV 3–8
  PROBRKR 3–8
Error handling 2–26
  stored procedures 2–53
Error messages
  connection failures 4–21
  locking problems 6–8
Exclusive locks 2–20
EXCLUSIVE-LOCK option 2–40
  in joins 2–59
  lookahead cursors 2–56
  record locking 2–19
Extended 4GL
  adding support to data source 5–35
  supported features 1–16

F
Field extents 2–13
Field formats 5–31
Field lists 2–35
Field properties
  modifying 5–30
FIELDS option 2–35
FIND statement 2–7, 2–37, 2–40
  LAST option 2–37
  PREV option 2–37
FOR statement 2–39
  EACH option 2–7, 2–40
Foreign keys 2–7

H
Holding locks 2–20
Host Name (-H) startup parameter 4–10

I
Indexes 2–7
  abbreviated 2–40
  case-insensitive 2–9
  dummy 2–8
  for stored procedure results 2–11
  for views 2–11
  problems importing 6–8
  selecting for views 2–8
  simulated 2–8
  support for large key entries 2–9
  unique 2–8
    FIND statements 2–8
    for browsing 2–8
INPUT option
  stored procedures B–5
INPUT-OUTPUT option
  retrieving parameter values 2–48
  returning parameter values 2–46
  stored procedures B–5
Index

Internal Code Page (-cpinternal) startup parameter 2–6
PROGRESS-to-ODBC utility 5–22
Isolation levels 2–20

J
JOIN-BY-SQLDB option 2–59
Joins 2–10
controlling by SQLDB behavior 2–60

K
Key buffer
troubleshooting 6–7
Keys
foreign 2–7
primary 2–7

L
Local Cache (-cache) startup parameter 4–11
Local DataServer 1–10
starting 4–2
Locks
deadlock 2–22
page-level 2–20
stored procedures 2–43
table-level 2–20
Log files 3–8
accessing 4–24
Logical database name 3–17, 5–8
changing 5–18
Logical Database Name (-ld) startup parameter 4–10
Login names 3–15, 4–10, 4–22
SETUSERID function 2–41
Lookahead cursors 2–56

M
Maintaining schema holders 5–3
recompiling code 3–21
MATCHES function 2–40
mergeprop utility 1–14
Modifying schema holders 5–29

N
Naming conventions 2–4
NO-JOIN-BY-SQLDB option 2–59
NO-LOCK option 2–40
record locking 2–19
NO-WAIT option 2–23
NSCONFIG C–3
NSMAN C–5
NULL search condition 2–41
Null value
in query 2–13
ROWID 2–33

O
ODBC
Administrator 1–3
Application Programming Interface (API) 1–6
architecture 1–4
compliance 1–1
creating schema holders 3–17, 5–8
data source 1–3
driver 1–3
interactive connections 4–15
NT 3–3
troubleshooting 6–11
unstable DLLs 6–11
Windows 3–3
Driver Manager 1–3
options 6–2
ODBCONFIG C–7
ODBMAN C–9
odbman utility
DataServer processes on UNIX 4–5
ODBSRV environment variable 3–8
OF option
FIND statement 2–7
FOR EACH statement 2–7
OPEN QUERY statement 2–40
OpenEdge
database objects 2–3
database security levels 1–7
naming conventions 2–4
triggers 2–11
unknown value 2–13
zero-length character strings 2–14
OUTPUT option
- retrieving parameter values 2–48
- returning parameter values 2–46
- stored procedures B–5

Overflow checking
- defined 2–28
- potential conflicts 2–28

P

PARAM option
- retrieving parameter values 2–48
- stored procedures B–5

Parameter (.pf) files
- connection requirements for DataServer 4–12

Parameters
- abbreviating names 6–2

Passing ODBC-specific information
- PRGRS_CONNECT option 4–15

Password (-P) startup parameter 4–10

Passwords 3–15, 4–22
- SETUSERID function 2–41

Performance
- analyzing 4–19
- caching records 2–58
- enhancing on DataServer 2–54
  - joins 2–59
- Progress query tips 2–54
- query tuning 2–55
- writing queries 2–61

Permissions
- data source security 1–7
- establishing 3–14

Physical database name 3–18, 5–9

Physical Database Name (-db) startup parameter 4–10

PRGRS_CONNECT option
- DataServer (-Dsrv) startup parameter 6–6, 6–7, 6–9

PRGRS_IDBUF option
- DataServer (-Dsrv) startup parameter 6–7

PRGRS_LOCK_ERROR option
- DataServer (-Dsrv) startup parameter 6–8

PRGRS_MNBUF option
- DataServer (-Dsrv) startup option 6–8

PRGRS_NO_INDEX option
- DataServer (-Dsrv) startup parameter 6–8

PRGRS_PREPCACHE option
- DataServer (-Dsrv) option 6–8

Primary keys 2–7

PROC-HANDLE function 2–51, B–3

PROC-STATUS function 2–47, B–3

PROC-HANDLE function 2–46
- database results 2–48
- returns from stored procedures 2–44

Progress
- data types D–2
- locking behavior 2–19

Progress Explorer
- configuring the DataServer on NT 3–6
- starting DataServer on NT 4–3
- stopping DataServer on UNIX 4–5

Progress Explorer Framework 3–22
- and the mergeprop utility 1–14
- and the Progress Explorer tool 1–13
- configuring DataServer applications using 1–13
- using the mergeprop utility 3–22
- using the Progress Explorer tool 3–22

Progress Explorer tool definition 1–13

PROGRESS-to-ODBC utility 5–22, 5–25, 5–27
- migrating Progress database to ODBC
data source 5–21
- parameters 5–4, 5–7, 5–14
- running in batch mode 5–22
- running interactively 5–21
Index

Q

qt_debug
CALL_SUMMARY option
DataServer (-Dsrv) startup parameter
4–17
CURSOR option
DataServer (-Dsrv) startup parameter
4–17
EXTENDED option
DataServer (-Dsrv) startup parameter
4–17
PERFORMANCE option
DataServer (-Dsrv) startup parameter
4–17
SQL option
DataServer (-Dsrv) startup parameter
4–17
VERBOSE option
DataServer (-Dsrv) startup parameter
4–17
qt_debug option
CALL SUMMARY option 4–19
CURSOR option 4–19
EXTENDED option 4–19
PERFORMANCE option 4–19
SQL option 4–19
VERBOSE option 4–19
qt_lookahead option
DataServer (-Dsrv) startup parameter
4–17
qt_no_debug option 4–19
DataServer (-Dsrv) startup parameter
4–17
qt_no_lookahead option
DataServer (-Dsrv) startup parameter
4–17
qt_no_separate_connection option
DataServer (-Dsrv) startup parameter
4–17
managing data sources 4–23
Query tuning
precedence of methods 4–16
startup parameters 4–16, 4–18
Query-tuning options 2–56
ARRAY-MESSAGE 2–56
CACHE-SIZE 2–56
DEBUG SQL 2–56
DEBUG-EXTENDED 2–56
JOIN-BY-SQLODBC 2–56
LOOKAHEAD 2–56
NO-ARRAY-MESSAGE 2–56
NO-DEBUG 2–56
NO-JOIN-BY-SQLODBC 2–56
NO-LOOKAHEAD 2–56
NO-SEPARATE-CONNECTION 2–57
SEPARATE-CONNECTION 2–57
QUERY-TUNING phrase 2–55

R

R-code 2–38
size 2–38
Read-only (-RO) startup parameter 4–9, 4–11
Read-only schema holders 4–11
RECID function 2–34
replacing with ROWID 2–34
Recompiling code
schema changes 3–21
Record caching
lookahead cursors 2–58
standard cursors 2–58
Record creation 2–16, 2–41
Record locking
data sources 2–19
exclusive locks 2–20
EXCLUSIVE-LOCK option 2–19
NO-LOCK option 2–19
shared locks 2–20
SHARE-LOCK option 2–19
Recovery
on data sources 2–24
Registering
data sources 3–3
RELEASE statement 2–18
Remote databases
starting a DataServer broker C–11
Remote DataServer 1–11
starting 4–3
Requirements
software 1–15
Roll back
on data sources 2–24
ROWID function 2–33
defining supporting index 5–31
listing objects not supported 3–20, 5–11
manually creating support 5–33
supporting in data source 5–35
Index

Rows
  troubleshooting large rows 6–8

RUN STORED-PROCEDURE statement
  2–43, B–5
  retrieving parameter values 2–48
  send-sql-statement option 2–52
  syntax 2–46

S

SAVE CACHE COMPLETE statement
  4–11, 4–20

Schema
  loading process 1–6
  Schema caching 1–6
  Schema holder
    before you begin 3–14
  Schema holders 1–3, 1–6
    changing logical name 5–18
    connecting 4–9
    connection failure 4–21
    connection permissions 3–15
    creating 3–14
    deleting 5–20
    maintaining 3–21, 5–3
    modifying 5–29
    setting code page 3–18, 5–8
    system administration permissions 5–7
    updating 3–21, 5–12
    upgrading version 7 to 8 A–2
    upgrading version 8 to 9 A–2
    verifying 5–14
    where best to build 3–14
  Schema images
    creating and updating 3–14
    security 3–14
  Schema import
    troubleshooting 6–8
  Scrolling
    backward 2–29
    forward 2–29
  Security
    guideline considerations 1–7

send-sql-statement option
  RUN-STORED-PROCEDURE
    statement 2–52

Sequence generator 2–12

Server Join (-nojoinbysqldb) startup parameter
  2–56, 2–59, 4–18
  Service Name (-S) startup parameter 4–10

SETUSERID function 2–41

Shared locks 2–20

SHARE-LOCK option 2–40
  record locking 2–19

Simulated indexes
  for nonindexed data sources 2–8

Single-user Mode (-1) startup parameter
  4–11

Skip schema verification 2–62

smalldatetime data type
  on Microsoft SQL Server D–2

Software requirements 1–15

SQL ORDER BY option
  Progress USE-INDEX option 2–8

SQL statement cache
  troubleshooting 6–8

SQL statements
  sending to data source 2–52

Starting a DataServer broker C–11

Startup parameters
  Database Type (-dt) 4–10
  DataServer (-Dsrv) 4–11, 4–16
  Host Name (-H) 4–10
  Internal Code Page (-cpinternal) 2–5
  Local Cache (-cache) 4–11
  Logical Database Name (-ld) 4–10
  Password (-P) 4–10
  Physical Database Name (-db) 4–10
  Read-only (-RO) 4–9, 4–11
  Server Join (-nojoinbysqldb) 4–11, 4–18
  Service Name (-S) 4–10
  Single-user Mode (-1) 4–11
  unsupported 4–14
  User ID (-U) 4–10

Stored procedures
  browsing results 2–8
  closing all 2–47
  data types 2–43
  defining 2–42
  defining buffers 2–49
  error handling 2–53
  INPUT option B–5
  INPUT-OUTPUT option B–5
  joining database results 2–51
  OUTPUT option B–5
  PARAM option B–5
  retrieving database results 2–48
  retrieving parameters 2–46, 2–48
  return status 2–47
  returned values 2–44
  returning status values B–2
running 2–42, 2–46
unique indexes for results 2–8
Stream Code Page (-cpstream) startup parameter 5–22
SYBASE
case sensitivity 2–10
T
Table properties
modifying 5–29
ROWID 5–32
Table validation 5–29
Terminology
differences between data sources 2–3
Transaction scoping
Progress 2–24
Transactions 2–24
Transact-SQL 2–52
Triggers 2–11
Troubleshooting 6–1
U
ubroker.properties file 3–9
hierarchy C–4, C–8
Unique indexes 2–8
stored procedures results 2–8
Unknown value
in query 2–13
ROWID 2–33
Unsupported 4GL statements 2–39
Updating schema holders 5–12
Upgrading
ODBC data sources A–3
schema holders A–2
to OpenEdge Release 10 DataServer 1–17
USE-INDEX option
FOR EACH statement 2–8
User ID 3–15
SETUSERID function 2–41
startup parameter (-U) 4–10
User-defined data types 2–13
USING
option 2–40
USING phrase
joins 2–59
V
VALIDATE statement 2–18
Validation
expressions 5–29, 5–31
messages 5–29
Verifying schema holders 5–14
Views 2–10
aggregates 2–11
browsing 2–11
W
Word index
CONTAINS operator 2–40
Z
Zero-length character strings 2–14, 2–39