PROGRESS
OPENEDGE® 10

OpenEdge Data Management:
DataServer for Microsoft SQL Server
Third party acknowledgements — See the “Third party acknowledgements” section on page Preface–9.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Preface</strong></td>
<td>Preface–1</td>
</tr>
<tr>
<td>1. <strong>Introduction.</strong></td>
<td>1–1</td>
</tr>
<tr>
<td>MS SQL Server DataServer overview</td>
<td>1–2</td>
</tr>
<tr>
<td>DataServer components</td>
<td>1–3</td>
</tr>
<tr>
<td>DataServer for MS SQL Server logic</td>
<td>1–5</td>
</tr>
<tr>
<td>Schema holder</td>
<td>1–6</td>
</tr>
<tr>
<td>Security</td>
<td>1–7</td>
</tr>
<tr>
<td>DataServer utilities</td>
<td>1–9</td>
</tr>
<tr>
<td>DataServer demonstration databases</td>
<td>1–10</td>
</tr>
<tr>
<td>DataServer configurations</td>
<td>1–11</td>
</tr>
<tr>
<td>The local DataServer configuration</td>
<td>1–11</td>
</tr>
<tr>
<td>The remote DataServer configuration</td>
<td>1–12</td>
</tr>
<tr>
<td>Configuring distributed DataServer applications using ProBroker</td>
<td>1–13</td>
</tr>
<tr>
<td>Configuring distributed DataServer applications</td>
<td></td>
</tr>
<tr>
<td>using the Unified Broker Framework</td>
<td>1–14</td>
</tr>
<tr>
<td>Software requirements</td>
<td>1–16</td>
</tr>
<tr>
<td>Guidelines for using the DataServer</td>
<td>1–17</td>
</tr>
<tr>
<td>Some functionality highlights</td>
<td>1–17</td>
</tr>
<tr>
<td>Using the DataServer for MS SQL Server for the first time</td>
<td>1–17</td>
</tr>
<tr>
<td>Migrating an OpenEdge database to an MS SQL Server data source</td>
<td>1–18</td>
</tr>
<tr>
<td>Documentation resources</td>
<td>1–19</td>
</tr>
<tr>
<td>2. <strong>Initial Programming Considerations.</strong></td>
<td>2–1</td>
</tr>
<tr>
<td>Database design issues</td>
<td>2–3</td>
</tr>
<tr>
<td>OpenEdge database objects and MS SQL Server database objects</td>
<td>2–3</td>
</tr>
<tr>
<td>Naming conventions</td>
<td>2–4</td>
</tr>
<tr>
<td>MS SQL Server DataServer data source limits</td>
<td>2–5</td>
</tr>
<tr>
<td>Code pages</td>
<td>2–5</td>
</tr>
<tr>
<td>Support for Unicode</td>
<td>2–9</td>
</tr>
<tr>
<td>Indexes and sorting</td>
<td>2–11</td>
</tr>
<tr>
<td>Case sensitivity</td>
<td>2–13</td>
</tr>
<tr>
<td>Interaction of code page, collation, and case sensitivity</td>
<td>2–13</td>
</tr>
<tr>
<td>MS SQL Server data source views</td>
<td>2–13</td>
</tr>
<tr>
<td>Triggers</td>
<td>2–14</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>OpenEdge sequences</td>
<td>2–15</td>
</tr>
<tr>
<td>Data types</td>
<td>2–17</td>
</tr>
<tr>
<td>Working with unsupported data types</td>
<td>2–19</td>
</tr>
<tr>
<td>Working with non-updatable data types</td>
<td>2–20</td>
</tr>
<tr>
<td>Processing considerations for 32-bit and 64-bit data types</td>
<td>2–20</td>
</tr>
<tr>
<td>DataServer for MS SQL Server support for datetime data types</td>
<td>2–21</td>
</tr>
<tr>
<td>Using datetime data types in a WHERE clause</td>
<td>2–29</td>
</tr>
<tr>
<td>Support for OpenEdge ABL BLOB data type</td>
<td>2–30</td>
</tr>
<tr>
<td>User-defined data types</td>
<td>2–31</td>
</tr>
<tr>
<td>Arrays</td>
<td>2–31</td>
</tr>
<tr>
<td>Unknown value (?)</td>
<td>2–31</td>
</tr>
<tr>
<td>Zero-length character strings</td>
<td>2–32</td>
</tr>
<tr>
<td>Record creation</td>
<td>2–33</td>
</tr>
<tr>
<td>Data source record locking</td>
<td>2–36</td>
</tr>
<tr>
<td>Share locks</td>
<td>2–37</td>
</tr>
<tr>
<td>Exclusive locks</td>
<td>2–37</td>
</tr>
<tr>
<td>Handling lock timeouts</td>
<td>2–37</td>
</tr>
<tr>
<td>Additional record locking details</td>
<td>2–39</td>
</tr>
<tr>
<td>NO–LOCK</td>
<td>2–39</td>
</tr>
<tr>
<td>Locking impact on queries</td>
<td>2–39</td>
</tr>
<tr>
<td>Locking examples</td>
<td>2–40</td>
</tr>
<tr>
<td>The NO–WAIT option</td>
<td>2–42</td>
</tr>
<tr>
<td>Transactions</td>
<td>2–43</td>
</tr>
<tr>
<td>Transaction scoping and buffer management</td>
<td>2–43</td>
</tr>
<tr>
<td>Error handling</td>
<td>2–45</td>
</tr>
<tr>
<td>Overflow checking</td>
<td>2–47</td>
</tr>
<tr>
<td>Cursors</td>
<td>2–48</td>
</tr>
<tr>
<td>Forward and backward scrolling</td>
<td>2–48</td>
</tr>
<tr>
<td>Block cursors</td>
<td>2–49</td>
</tr>
<tr>
<td>ABL issues</td>
<td>2–52</td>
</tr>
<tr>
<td>ROWID function</td>
<td>2–52</td>
</tr>
<tr>
<td>RECID function.</td>
<td>2–53</td>
</tr>
<tr>
<td>DEFINE BROWSE statement</td>
<td>2–54</td>
</tr>
<tr>
<td>Field lists</td>
<td>2–54</td>
</tr>
<tr>
<td>FIND statements</td>
<td>2–56</td>
</tr>
<tr>
<td>Compiling OpenEdge procedures</td>
<td>2–57</td>
</tr>
<tr>
<td>FOR statements</td>
<td>2–58</td>
</tr>
<tr>
<td>Features and language differences</td>
<td>2–58</td>
</tr>
<tr>
<td>RDBMS stored procedures</td>
<td>2–60</td>
</tr>
<tr>
<td>3. RDBMS Stored Procedure Details</td>
<td>3–1</td>
</tr>
<tr>
<td>Overview</td>
<td>3–2</td>
</tr>
<tr>
<td>Defining native stored procedures to ABL</td>
<td>3–2</td>
</tr>
<tr>
<td>RDBMS stored procedure basics</td>
<td>3–3</td>
</tr>
<tr>
<td>Run Stored-Procedure details</td>
<td>3–6</td>
</tr>
<tr>
<td>Complete syntax reference for Run Stored-Procedure</td>
<td>3–6</td>
</tr>
<tr>
<td>Run Stored-Procedure statement</td>
<td>3–7</td>
</tr>
<tr>
<td>Closed Stored-Procedure statement</td>
<td>3–9</td>
</tr>
<tr>
<td>Run Stored-Proc statement execution</td>
<td>3–10</td>
</tr>
<tr>
<td>without the LOAD-RESULT-INTO phrase</td>
<td>3–10</td>
</tr>
<tr>
<td>Run Stored-Proc statement execution</td>
<td>3–11</td>
</tr>
<tr>
<td>with the LOAD-RESULT-INTO phrase</td>
<td>3–11</td>
</tr>
<tr>
<td>Run Stored-Proc statement execution using</td>
<td>3–11</td>
</tr>
<tr>
<td>the send-sql-statement option</td>
<td>3–11</td>
</tr>
<tr>
<td>Data output and retrieval options</td>
<td>3–13</td>
</tr>
<tr>
<td>Return codes</td>
<td>3–13</td>
</tr>
<tr>
<td>Contents–2</td>
<td></td>
</tr>
</tbody>
</table>
Values of output parameters defined when creating a procedure . . . . . . . . . 3–13
Results retrieved from a database ............................................. 3–13
Loading results into a temp-table ........................................... 3–14
Interfacing with RDBMS stored procedures .................................. 3–15
Retrieving return codes ......................................................... 3–15
Retrieving output parameter values ........................................... 3–16
Retrieving result sets using proc-text-buffer .............................. 3–16
Defining a view to use as a buffer ............................................ 3–18
Loading result sets into temp-tables ........................................ 3–21
Handling errors ................................................................. 3–30
Trapping errors when using Load-Result-Into .............................. 3–30
Error messages related to using the Load-Result-Into phrase .......... 3–30
ROWID Support ..................................................................... 3–32
Understanding the ROWID implementation .............................. 3–32
ROWID characteristics ......................................................... 3–32
Using ROWID with RUN STORED-PROCEDURE and LOAD-RESULT-INTO 3–34
Additional ProDataSet support ................................................ 3–36

4. Additional Features to Enhance DataServer Performance ............... 4–1
   Enhancements overview .................................................... 4–2
   Connection pooling .......................................................... 4–2
   Main components ............................................................ 4–2
   ODBC Connection Pooling ............................................... 4–3
   DataServer connection management .................................... 4–4
   Firehose and Fast Forward-Only Cursors .............................. 4–5
   Monitoring cursor and connection use .................................. 4–6
   OpenEdge query types ..................................................... 4–8
   Query tuning ................................................................. 4–9
   Caching records ............................................................. 4–13
   Join by SQLDB ............................................................... 4–14
   Writing queries for performance ......................................... 4–16
   Modifying the run-time schema check ................................... 4–17
   Skip schema verification ................................................... 4–17
   Replacing FIND FIRST for performance ................................. 4–18

5. Configuring the DataServer ..................................................... 5–1
   DataServer components .................................................... 5–2
   DataServer configuration prerequisites .................................. 5–2
   Configuring an ODBC driver and registering the data source ....... 5–4
   Configuring the ODBC driver ............................................. 5–4
   Registering your data source .............................................. 5–4
   Configuring a local DataServer ........................................... 5–5
   Configuring a remote DataServer ....................................... 5–6
   Configuring with the OpenEdge Explorer and Progress Explorer . 5–6
   Configuring from the command line .................................... 5–9
   Creating a schema holder ................................................... 5–15
   Before you begin ............................................................ 5–15
   Permission requirements ................................................... 5–15
   Preparing to create the schema holder .................................. 5–16
   Creating a new empty OpenEdge database ............................. 5–17
   Creating a schema holder .................................................. 5–18
   Maintaining a schema holder ............................................. 5–22
   Updating a deployed schema holder .................................... 5–22
   Typical configuration for a remote client to connect to a remote DataServer .... 5–23
   Configuring an ODBC data source ..................................... 5–23
6. Connecting the DataServer .............................................. 6–1
   Starting a local DataServer .............................................. 6–2
   Starting a remote DataServer ............................................ 6–3
      Starting and stopping a broker process from the OpenEdge Explorer and Progress Explorer and connecting a client ........................................... 6–3
      Starting and stopping a broker process using MSSMAN ..................... 6–5
      Starting and stopping a broker process from the command line ........... 6–6
      Starting the Windows client process ..................................... 6–6
      Starting the UNIX client process ......................................... 6–7
   Connection guidelines ..................................................... 6–8
      Connections and security in Windows ..................................... 6–8
   Connecting a schema holder ............................................... 6–9
      Connecting a schema holder at startup ................................... 6–9
      Unsupported connection parameters ..................................... 6–14
      Special connection issues ............................................... 6–14
      Binding with connection and startup parameters .......................... 6–15
      Query tuning with connection and startup parameters .................... 6–16
      Analyzing performance with the Enhanced Logger ........................ 6–18
      Local schema caching .................................................... 6–26
   Connection failures and OpenEdge responses ................................ 6–28
      Connection troubleshooting ............................................... 6–29
      Managing connections to an MS SQL Server database .................... 6–29
      Accessing the DataServer log ............................................ 6–30

7. The DataServer Tutorial ................................................ 7–1
   Demonstration databases for DataServer for MS SQL Server .................. 7–2
   Preparing to create demonstration databases .................................. 7–3
   DataServer utilities ....................................................... 7–6
   Creating a schema holder ................................................ 7–8
   Updating a schema holder ................................................ 7–11
   Verifying a schema holder ............................................... 7–13
   Changing connection information in a schema holder ........................ 7–17
   Changing a code page in a schema holder .................................. 7–18
   Deleting a schema holder ................................................ 7–19
   Migrating an OpenEdge database to MS SQL Server .......................... 7–20
      Preparing a data source for the utility ................................... 7–20
      Running the OpenEdge DB to MS SQL Server utility ...................... 7–20
   Adjust schema utility ..................................................... 7–26
   Delta df to MS SQL Server Incremental Schema Migration utility ............ 7–27
      Updating the MS SQL Server database ................................... 7–31
   Modifying a schema holder ............................................... 7–32
      Modifying table-level information ....................................... 7–32
      Modifying field-level information ....................................... 7–33
      Defining the ROWID ...................................................... 7–35
   Adding extended ABL support ............................................. 7–37
      Modifying tables to support arrays ...................................... 7–37
      Using triggers to enable PROGRESS_RECID support of the ROWID function .............................................................. 7–37
      Using computed columns to enable PROGRESS_RECID support of the ROWID function .............................................................. 7–39
      Migrating RECID-trigger mechanism to RECID computed column mechanism .............................................................. 7–39
## Contents

### Chapter 8: Troubleshooting
- Tuning your environment with the --Dsrv startup parameter ........................................ 8–2
- ODBC options ....................................................................................................................... 8–3
- DataServer options ................................................................................................................ 8–4
- Using MS SQL Server and DataServer options ................................................................. 8–6
- Using the block cursor switches ......................................................................................... 8–11
- ODBC driver problems ........................................................................................................ 8–13

### Appendix A: Migration Issues
- Creating a schema holder ..................................................................................................... A–2
- Modifying your application ................................................................................................. A–3
- Modifying your insert trigger .............................................................................................. A–4

### Appendix B: Server Related Command Line Utilities and Startup Parameters
- Progress Explorer command line utilities for the DataServer ........................................... B–2
  - MSSCONFIG utility ............................................................................................................ B–2
  - MSSMAN utility ................................................................................................................ B–4
  - NSCONFIG utility .............................................................................................................. B–6
  - NSMAN utility ................................................................................................................... B–8
- Non-Progress Explorer command line utilities for the DataServer ................................... B–10
  - PROBRKR command ...................................................................................................... B–11
  - PROSHUT command ...................................................................................................... B–12
- DataServer startup parameters ......................................................................................... B–13

### Appendix C: Data Type Details
- Data type conversion details ............................................................................................. C–2

### Appendix D: Using qt_debug to Analyze Performance
- Using qt_debug to analyze performance ........................................................................... D–2

### Index

Index–1
Figures

Figure 1–1: Architecture for DataServer for MS SQL Server ......................... 1–4
Figure 1–2: DataServer logic ........................................................................ 1–5
Figure 1–3: The schema-loading process .................................................... 1–6
Figure 1–4: The local DataServer for MS SQL Server ................................ 1–11
Figure 1–5: The remote DataServer for MS SQL Server ............................ 1–12
Figure 1–6: The distributed DataServer for MS SQL Server using ProBroker . 1–13
Figure 2–1: DataServer processes and code pages ..................................... 2–5
Figure 5–1: User ID and Password dialog box ........................................... 5–16
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1–1</td>
<td>DataServer architecture components</td>
<td>1–3</td>
</tr>
<tr>
<td>1–2</td>
<td>Supported configurations</td>
<td>1–12</td>
</tr>
<tr>
<td>1–3</td>
<td>How to use this manual</td>
<td>1–19</td>
</tr>
<tr>
<td>1–4</td>
<td>DataServer-related topics</td>
<td>1–19</td>
</tr>
<tr>
<td>2–1</td>
<td>OpenEdge Database objects and MS SQL Server database objects</td>
<td>2–3</td>
</tr>
<tr>
<td>2–2</td>
<td>Database naming restrictions</td>
<td>2–4</td>
</tr>
<tr>
<td>2–3</td>
<td>Case sensitivity interactions</td>
<td>2–8</td>
</tr>
<tr>
<td>2–4</td>
<td>Supported Unicode data types in MS SQL Server</td>
<td>2–10</td>
</tr>
<tr>
<td>2–5</td>
<td>MS SQL Server data type equivalents in OpenEdge</td>
<td>2–17</td>
</tr>
<tr>
<td>2–6</td>
<td>MS SQL Server 2000 and 2005 datetime data types</td>
<td>2–21</td>
</tr>
<tr>
<td>2–7</td>
<td>MS SQL Server 2008 datetime data types</td>
<td>2–22</td>
</tr>
<tr>
<td>2–8</td>
<td>ABL datetime data types</td>
<td>2–23</td>
</tr>
<tr>
<td>2–9</td>
<td>Converting ABL datetime types to MS SQL Server data types</td>
<td>2–24</td>
</tr>
<tr>
<td>2–10</td>
<td>Converting ABL datetime types to MS SQL Server data types</td>
<td>2–25</td>
</tr>
<tr>
<td>2–11</td>
<td>BLOB data type in schema migration</td>
<td>2–30</td>
</tr>
<tr>
<td>2–12</td>
<td>BLOB data type in schema pull</td>
<td>2–31</td>
</tr>
<tr>
<td>2–13</td>
<td>OpenEdge database and data source locking</td>
<td>2–36</td>
</tr>
<tr>
<td>2–14</td>
<td>TXN_ISOCLUSION values in the –Dsrv parameter</td>
<td>2–36</td>
</tr>
<tr>
<td>2–15</td>
<td>ABL differences between OpenEdge Databases and MS SQL Server data sources</td>
<td>2–58</td>
</tr>
<tr>
<td>3–1</td>
<td>Stored procedure language elements</td>
<td>3–3</td>
</tr>
<tr>
<td>3–2</td>
<td>Argument data types for stored procedures</td>
<td>3–4</td>
</tr>
<tr>
<td>3–3</td>
<td>Returning result sets and loading the data into temp-tables</td>
<td>3–23</td>
</tr>
<tr>
<td>3–4</td>
<td>Options to plan the temp-table layout for result sets</td>
<td>3–24</td>
</tr>
<tr>
<td>4–1</td>
<td>Query-tuning options</td>
<td>4–10</td>
</tr>
<tr>
<td>4–2</td>
<td>Controlling join by SQLDB behavior</td>
<td>4–15</td>
</tr>
<tr>
<td>5–1</td>
<td>Installing the DataServer components</td>
<td>5–2</td>
</tr>
<tr>
<td>5–2</td>
<td>Environment variables for the remote DataServer</td>
<td>5–10</td>
</tr>
<tr>
<td>5–3</td>
<td>DataServer for MS SQL Server sections of the ubroker.properties file</td>
<td>5–11</td>
</tr>
<tr>
<td>5–4</td>
<td>MS SQL Server and OpenEdge code pages</td>
<td>5–19</td>
</tr>
<tr>
<td>6–1</td>
<td>DataServer connection parameters</td>
<td>6–10</td>
</tr>
<tr>
<td>6–2</td>
<td>Connection BINDING options</td>
<td>6–16</td>
</tr>
<tr>
<td>6–3</td>
<td>Connection query-tuning options</td>
<td>6–17</td>
</tr>
<tr>
<td>6–4</td>
<td>LOG-MANAGER system handle attributes</td>
<td>6–18</td>
</tr>
<tr>
<td>6–5</td>
<td>LOG-MANAGER system handle methods</td>
<td>6–19</td>
</tr>
<tr>
<td>6–6</td>
<td>DSLOG-MANAGER system handle attributes</td>
<td>6–20</td>
</tr>
<tr>
<td>6–7</td>
<td>DSLOG-MANAGER system handle methods</td>
<td>6–21</td>
</tr>
<tr>
<td>6–8</td>
<td>Client DataServer Log entry types</td>
<td>6–21</td>
</tr>
<tr>
<td>6–9</td>
<td>Server Log entry types</td>
<td>6–22</td>
</tr>
<tr>
<td>6–10</td>
<td>Setting logging levels and log entry types</td>
<td>6–23</td>
</tr>
<tr>
<td>6–11</td>
<td>Logging levels</td>
<td>6–24</td>
</tr>
<tr>
<td>6–12</td>
<td>PRGRS_LOG_ABL connect option values</td>
<td>6–26</td>
</tr>
<tr>
<td>6–13</td>
<td>Failure responses</td>
<td>6–28</td>
</tr>
<tr>
<td>7–1</td>
<td>OpenEdge DB to MS SQL Server utility</td>
<td>7–4</td>
</tr>
<tr>
<td>7–2</td>
<td>DataServer utilities</td>
<td>7–6</td>
</tr>
<tr>
<td>7–3</td>
<td>MS SQL Server and OpenEdge code pages</td>
<td>7–8</td>
</tr>
<tr>
<td>7–4</td>
<td>Verify utility report</td>
<td>7–13</td>
</tr>
<tr>
<td>7–5</td>
<td>OpenEdge-to-MS SQL Server Conversion utility</td>
<td>7–21</td>
</tr>
<tr>
<td>7–6</td>
<td>OpenEdge-to-MS SQL Server Conversion utility batch parameters</td>
<td>7–23</td>
</tr>
<tr>
<td>7–7</td>
<td>Generate Delta.sql OpenEdge to MSS utility</td>
<td>7–28</td>
</tr>
<tr>
<td>7–8</td>
<td>MS SQL Server equivalents of OpenEdge objects</td>
<td>7–31</td>
</tr>
<tr>
<td>8–1</td>
<td>ODBC options</td>
<td>8–3</td>
</tr>
<tr>
<td>8–2</td>
<td>DataServer options</td>
<td>8–4</td>
</tr>
<tr>
<td>Table B–1:</td>
<td>DataServer parameters</td>
<td>B–13</td>
</tr>
<tr>
<td>Table C–1:</td>
<td>MS SQL Server data type details</td>
<td>C–2</td>
</tr>
<tr>
<td>Table D–1:</td>
<td>Diagnostic options</td>
<td>D–2</td>
</tr>
</tbody>
</table>
## Examples

| Example 3–1: | Passing a SELECT statement as a parameter | 3–12 |
| Example 3–2: | Using the send-sql-statement with the LOAD-RESULT-INTO option | 3–12 |
| Example 3–3: | A stored procedure with multiple result sets, output parameters, and a single return value | 3–15 |
| Example 3–4: | Return status example | 3–16 |
| Example 3–5: | Passing parameters by name using the PARAM option | 3–16 |
| Example 3–6: | pcust stored procedure | 3–17 |
| Example 3–7: | Returning database results into the proc-text-buffer and results conversion | 3–17 |
| Example 3–8: | First view created in your MS SQL Server data source | 3–19 |
| Example 3–9: | Second view created in your MS SQL Server data source | 3–19 |
| Example 3–10: | Result set of pcust - typed buffers | 3–19 |
| Example 3–11: | Procedure handles | 3–20 |
| Example 3–12: | Joining stored procedure results with other database tables | 3–20 |
| Example 3–13: | RUN STORED-PROC statement with the LOAD-RESULT-INTO phrase | 3–21 |
| Example 3–14: | Multiple temp-tables in a prepared state | 3–26 |
| Example 3–15: | Temp-table code technique | 3–26 |
| Example 3–16: | Basic execution of a stored procedure using LOAD-RESULT-INTO functionality | 3–27 |
| Example 3–17: | Using an existing dynamic temp-table without the TEMP-TABLE-PREPARE ( ) method | 3–27 |
| Example 3–18: | Using an existing dynamic temp-table with the TEMP-TABLE-PREPARE ( ) method | 3–28 |
| Example 3–19: | Calling a stored procedure and using the EXTENT phrase | 3–28 |
| Example 3–20: | Calling an enhanced stored procedure with a single temp table and the send-sql-statement | 3–29 |
| Example 3–21: | Use of a PROC-STATUS phrase | 3–29 |
| Example 3–22: | Trapping errors within a procedure | 3–30 |
| Example 3–23: | Trapping errors using LOAD-RESULT-INTO | 3–30 |
| Example 3–24: | Expected ABL behavior—ROWID value of a temp-table buffer | 3–33 |
| Example 3–25: | ABL Query filling a ProDataSet temp-table | 3–34 |
| Example 3–26: | Using the LOAD-RESULT-INTO technique to populate the underlying Temp-Table of a ProDataSet | 3–35 |
| Example 3–27: | Use the LOAD-RESULT-INTO technique with BEFORE-FILL method to fill the temp-table(s) of a ProDataSet | 3–37 |
Preface

This Preface contains the following sections:

- Purpose
- Audience
- Organization
- Using this manual
- Before you begin
- Examples of syntax descriptions
- OpenEdge messages
- Third party acknowledgements
Purpose

This manual explains how to use the OpenEdge® DataServer for Microsoft SQL Server. 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 MS SQL Server database management systems, and provides tuning and troubleshooting information.

Note: In this manual, any reference to the term MSS or data source in the text, screen captures, or code examples presented refers to the MS SQL Server database.

Audience

This book is intended for programmers who want to develop OpenEdge applications that run with SQL Server databases. It assumes a fundamental knowledge of OpenEdge and MS SQL Server.

Organization

Chapter 1, “Introduction”

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

Chapter 2, “Initial Programming Considerations”

Discusses the differences between the OpenEdge database and MS SQL Server and the DataServer strategies for resolving these differences. Includes database design issues and application issues.

Chapter 3, “RDBMS Stored Procedure Details”

Defines relational database management system (RDBMS) stored procedures and describes how to create and implement them in an OpenEdge environment. It discusses various techniques to execute RDBMS stored procedures and Send SQL statements on the MS SQL server and load results sets directly into temp-tables. ProDataSet functionality, available through the use of temp-tables, is also briefly discussed.

Chapter 4, “Additional Features to Enhance DataServer Performance”

Presents various DataServer performance enhancement techniques you can use, including connection pooling.
Chapter 5, “Configuring the DataServer”

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

Chapter 6, “Connecting the DataServer”

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

Chapter 7, “The DataServer Tutorial”

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

Chapter 8, “Troubleshooting”

Describes common problems and how to work around them, including tuning your environment and resolving ODBC driver problems.

Appendix A, “Migration Issues”

Provides information on how to migrate from the DataServer for ODBC to the DataServer for MS SQL Server, and information on issues migrating from the Version 9.1 DataServer for MS SQL Server.

Appendix B, “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 C, “Data Type Details”

Presents a detailed, comprehensive list of the MS SQL Server data types and their associated OpenEdge-data type equivalencies.

Appendix D, “Using qt_debug to Analyze Performance,”

The qt_debug option of the DataServer (-Dsrv) startup parameter can be used as a supplement to the OpenEdge Enhanced Logger to provide you with the means to perform diagnostics.

Using this manual

Chapter 1, “Introduction” introduces you to the DataServer for MS SQL Server and discuss how OpenEdge and WebSpeed applications work with it to access data sources through the ODBC standard.

Subsequent chapters provide additional information about using the DataServer. If you are using the DataServer with WebSpeed and with applications written in Progress® SpeedScript®, all information regarding ABL applies to your application.
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.

Before you begin

Before attempting to use the MS SQL Server DataServer, first read the programming guidelines in Chapter 2, “Initial Programming Considerations.” Also, be sure to follow the step-by-step instructions in Chapter 5, “Configuring the DataServer,” and Chapter 6, “Connecting the DataServer,” for installing, configuring, and connecting to a data source.

See the “Documentation resources” section on page 1–19 for additional information.
Typographical conventions

This manual uses the following typographical conventions:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold</strong></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 <strong>simultaneous</strong> 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 <strong>sequential</strong> key sequence: you press and release the first key, then press another key. For example, ESCAPE H.</td>
</tr>
</tbody>
</table>

**Syntax:**

<table>
<thead>
<tr>
<th>Syntax Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fixed width</strong></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><strong>UPPERCASE fixed width</strong></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>]]&gt;</td>
<td>This icon (three arrows) introduces a multi-step procedure.</td>
</tr>
<tr>
<td>▼</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 the 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 the ABL. For example, a called external procedure must use braces when referencing arguments passed by a calling procedure.</td>
</tr>
<tr>
<td></td>
<td>A vertical bar indicates a choice.</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:

FOR EACH Customer:
  DISPLAY Name.
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**

```
{ &argument-name }
```

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

**Syntax**

```
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**

```
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**

```
MESSAGE { expression | SKIP [ ( n ) ] } ...
```

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

**Syntax**

```
{ include-file
  [ argument | &argument-name = "argument-value" ] ... }
```

**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**

```
WITH [ ACCUM max-length ] [ expression DOWN ]
  [ CENTERED ] [ n COLUMNS ] [ SIDE-LABELS ]
  [ 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

```
ASSIGN { [ FRAME frame ] { field [ = expression ] } |
[ WHEN expression ] } ... |
{ record [ EXCEPT field ... ] }
```

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 OpenEdge Procedure Editor, press the HELP key or F1.

On UNIX platforms, use the OpenEdge pro command to start a single-user mode character OpenEdge client session and view a brief description of a message by providing its number.

To use the pro command to obtain a message description by message number:

1. Start the Procedure Editor:

   ```
   OpenEdge-install-dir/dlc/bin/pro
   ```

2. Press F3 to access the menu bar, then choose Help→Messages.

3. Type the message number and press ENTER. Details about that message number appear.

4. Press F4 to close the message, press F3 to access the Procedure Editor menu, and choose File→Exit.

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_prod.s). 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 (decstrtod.c; strtodg.c); Copyright © 1998, 2000 by Lucent Technologies All Rights Reserved (decstrtof.c; strtord.c); Copyright © 1998 by Lucent Technologies All Rights Reserved (dmisc.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 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 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 code licensed from RSA Security, Inc. Some portions licensed from IBM are available at http://oss.software.ibm.com/icu4j/.

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 the 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.
Introduction

The OpenEdge® DataServer for Microsoft SQL Server allows you to develop and deploy ABL (Advanced Business Language) or WebSpeed® applications that access a Microsoft SQL Server database. These applications use ABL specifically designed to translate knowledge about business operations into software.

The DataServer for MS SQL Server supports Microsoft SQL Server Versions 7 and later. The DataServer itself is fully ODBC-2.0 compliant. It also complies partially with the ODBC 3.0 standard by supporting extensions for running stored procedures.

This chapter introduces you to the DataServer for MS SQL Server and discusses how OpenEdge and WebSpeed applications work with it to access data sources through the ODBC standard. This chapter contains the following sections:

- MS SQL Server DataServer overview
- DataServer components
- DataServer utilities
- DataServer demonstration databases
- DataServer configurations
- Software requirements
- Guidelines for using the DataServer
- Documentation resources
MS SQL Server DataServer overview

The OpenEdge DataServer for MS SQL Server allows you to access your MS SQL Server data source with the ABL and develop applications within the OpenEdge® Studio. The OpenEdge Studio is a set of tools that helps you 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 ABL business logic to access a foreign data source.

OpenEdge provides you to develop, deploy, manage, and integrate your application. The DataServer allows you to implement the OpenEdge features and ABL expansions in applications that run with the supported data sources.

Some of these tools and features are:

- **OpenEdge® Application Server** — Use to partition your application across multiple servers.

- **Progress Dynamics®** — Use to design and develop a repository-based application.

- **Data Dictionary** — Use to modify database schema; create indexes; and define database triggers, validation expressions, and help messages.

- **Data Administration tool** — Use to manage database data and perform DataServer tasks.

- **Database triggers** — Use 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 a MS SQL Server data source does not provide you with access to all ABL, WebSpeed, and database features. For details, see Chapter 2, “Initial Programming Considerations.”

The OpenEdge® DataServer for MS SQL Server is a fully featured DataServer with a wide range of deployment flexibility and scalability. Its features include the following:

- It is available for use by multiple users.

- It can be configured on remote servers.

- The ODBC driver for the data source is bundled with the product.

**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 the data source. The DataServer enables access by translating ABL and WebSpeed code into calls appropriate to the data source. The application can thus manipulate the data as though it were OpenEdge-based data.

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

Table 1–1: DataServer architecture components

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataServer for MS SQL Server (Windows platforms only)</td>
<td>An OpenEdge software module that allows you to use ABL or WebSpeed Speedscript.</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 in Windows by default as a Microsoft-supplied dynamic link library (DLL) module.</td>
</tr>
<tr>
<td>Schema holder</td>
<td>A repository for foreign data definitions.</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 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 MS SQL Server 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 utilities for MS SQL Server</td>
<td>A set of utilities that allows you to perform certain tasks related to the MS SQL Server DataServer. You access them from either the Data Administration or the Data Dictionary.</td>
</tr>
</tbody>
</table>
Figure 1–1 illustrates how the DataServer components are organized.

![Architecture for DataServer for MS SQL Server](image)

**Figure 1–1: Architecture for DataServer for MS SQL Server**

As shown in Figure 1–1, the components for the self-service DataServer for MS SQL Server are organized similarly to the client-server configuration except that the DataServer is embedded in the client rather than executing separately. With a client-server configuration, the components can be deployed so that the DataServer resides on the same machine as the client (loop back), on the server for efficiency and close proximity to the database, or distributed to a separate third tier.

The ODBC driver manager must reside on the same machine as the DataServer “server.” To ODBC, the “server” half of a client-server configuration, or the self-service DataServer appears as an ODBC client. 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 MS SQL Server database.

Some of the DataServer components are linked with elements of the standard OpenEdge architecture, as described in the “DataServer configurations” section on page 1–11.
DataServer for MS SQL Server 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 MS SQL Server database, the compiler translates ABL or SpeedScript statements into their SQL equivalents. The DataServer then issues the SQL statements to the 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 OpenEdge through the ODBC API.

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

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 MS SQL Server 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.

**Schema holder**

As noted in the previous section, a *schema holder* contains information about the MS SQL Server 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.

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

Before an OpenEdge client can access data in the MS SQL Server 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.
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 an empty data source.

However, the RDBMS stored procedures also supports `s` and ProDataSet functionality which does support returning data to the foreign data source. For information on using `RUN STORED-PROCEDURE`, see Chapter 3, “RDBMS Stored Procedure Details.”

Security

Using the DataServer for MS SQL Server involves following the security guidelines required by both the OpenEdge database and the MS SQL Server data source. By default, OpenEdge security allows unrestricted access to data sources, so at a 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
- Runtime 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*.

MS SQL Server database security

As noted previously, you should follow the security guidelines that your MS SQL Server data source has established for your applications. The MS SQL Server database might require that all users supply a valid login name and password to access it. Data source access security typically has four levels:

- **System administrator** — Grants or revokes permissions to other users to create or own a wide type of objects; for example, databases
- **Database owner** — Grants other users permission to access or modify a database or its objects
- **Database object owner** — Defines a user who can be the owner of objects in a database owned by another user
- **Public owner** — Allows access to public database objects by any users without restriction
Additional security considerations

Note the following additional security points:

- The DataServer for MS SQL Server will allow you to use either the Windows or MS SQL Server authentication model for your connection to the MS SQL Server data source. For more information on database security, see Microsoft’s SQL Server documentation.

- 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 5–15.
DataServer utilities

Progress Software Corporation provides a set of utilities that allows you to perform certain tasks related to the DataServer. There are utilities for:

- 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 MS SQL Server database, generating delta SQL files to update your MS SQL Server data source, and adjusting a pulled schema to match an existing OpenEdge database
- 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 the 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 demonstration data that you can use to experiment with the DataServer. Note that you do not install the demonstration databases as part of the OpenEdge Installation media. Instead, you create them by running the OpenEdge DB to MS SQL Server utility, which migrates the Sports database—objects and data—to your MS SQL Server data source. For instructions, see the “Demonstration databases for DataServer for MS SQL Server” section on page 7–2.
DataServer configurations

The DataServer for MS SQL Server 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 MS SQL Server 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 data source client software required 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 TCP/IP networking. A remote configuration exists within the framework of a distributed application, whereby DataServers are spawned on the host machine using the ProBroker executable or broker in the Progress Explorer administration framework.

The local DataServer configuration

Figure 1–4 shows the local DataServer configuration.

![Diagram](image)

**Figure 1–4: The local DataServer for MS SQL Server**

In the local DataServer configuration, all of the DataServer software components, the schema holder, and data source client software run on the same machine. Depending on the ODBC and data source client software implementation, the actual target database can be local or remote to the machine where the local OpenEdge DataServer for MS SQL Server executes.
With the MS SQL Server client software-supplied modules that manage networked database communications, a remote database configuration is possible. The DataServer cares only about the local MS SQL Server data source definition. Remote database access is transparent to the DataServer.

**The remote DataServer configuration**

Figure 1–5 shows the remote DataServer configuration.

![Figure 1–5: The remote DataServer for MS SQL Server](image)

In the remote configuration, the OpenEdge client and schema holder run on a client machine and the OpenEdge server components and any MS SQL Server client software run on a remote server machine. The actual target database can be local or remote to the machine where the OpenEdge DataServer for MS SQL Server executes. With the MS SQL Server client software supplied 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.

**Table 1–2** lists the supported configurations for setting up a remote DataServer. 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 client in Windows</td>
<td>None</td>
<td>The server, client, ODBC client, and data source are on the same machine.</td>
</tr>
<tr>
<td>OpenEdge client in Windows or on UNIX</td>
<td>OpenEdge</td>
<td>The client can reside in Windows or on UNIX. The server, ODBC driver, and data source are on a Windows server machine.</td>
</tr>
</tbody>
</table>
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 OpenEdge client runs on a client machine (either in Windows or on UNIX) and can access a remote Windows server. Multiple brokers can 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 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. 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 MS SQL Server 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 MS SQL Server DataServer’s broker-based technology. For example, the AdminService, the framework’s central element, enables supported products like the DataServer for MS SQL Server 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

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 MS SQL Server is running. Multiple NameServers and/or brokers can be running simultaneously on one server machine. The DataServer client connects to a broker for Progress Explorer either directly or through a controlling NameServer. (See the important caution that follows.) It 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. 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 MS SQL Server work within the framework, see OpenEdge Getting Started: Installation and Configuration.
Software requirements

The following list describes the software requirements for using the DataServer.

- OpenEdge-supplied software
  Your installation must have:
  - DataServer for MS SQL Server
  - WebSpeed Transaction Server Release 10 or later, or OpenEdge Release 10 or later

- Data-source server software
  Your installation must have:
  - Microsoft SQL Server Version 7.0 or higher

- Data source client software
  Your installation must have:
  - Client connectivity component software from your SQL Server installation.
    For information on exact client software compatibilities with Microsoft SQL Server Version 7 or later, see the Microsoft Web Site at http://www.microsoft.com.

- ODBC software
  Your installation might have:
  - An ODBC Driver for the version of Microsoft SQL Server you are using. (optional)
    For information on exact driver version compatibilities with the Microsoft SQL Server client and server data source software, see the Microsoft Web Site at http://www.microsoft.com.
  - Progress Software Corporation supplies the DataDirect ODBC driver, which is automatically installed when you install the DataServer for MS SQL Server. For additional information, see the DataDirect Web site at: http://www.datadirect.com/techres/progressdoc/index.ssp.
OpenEdge database management system 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 MS SQL Server databases.

Some functionality highlights

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

- ROWID function
- Arrays

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

If you create a MS SQL Server data source from an existing OpenEdge database with the OpenEdge DB to MS SQL Server migration utility and select the Create RECID Field option, you can use the FIND PREV/LAST statements in addition to taking advantage of OpenEdge-like cursor behavior.

How you use the DataServer depends on whether you plan to access information in a data source through an OpenEdge application or migrate an OpenEdge database to a MS SQL Server data source. The following sections summarize.

Using the DataServer for MS SQL Server for the first time

Before using the DataServer, you must:

1. Install the DataServer modules on the machines your configuration requires.

2. If you chose not to use the bundled ODBC driver, purchase and install the ODBC driver software on the machine that will execute the server component.

3. Verify connectivity between ODBC and the MS SQL Server database.

4. Create a local schema holder on the client or server machine, as appropriate. Schema holders cannot be transferred between different host machines.

Chapter 3, “RDBMS Stored Procedure Details,” provides information about where to install DataServer modules and how to create a schema holder.
Migrating an OpenEdge database to an MS SQL Server data source

You can migrate an OpenEdge database to an MS SQL Server data source.

To set up and use the DataServer:

1. Install the DataServer components on the machines that your configuration requires.

2. If you chose not to use the bundled ODBC driver, purchase and install the ODBC driver software on the machine that will execute the server component.

3. Verify connectivity between ODBC and the MS SQL Server database.

4. Run the OpenEdge DB to MS SQL Server migration utility.

See the “Migrating an OpenEdge database to MS SQL Server” section on page 7–20 for specific instructions.
Table 1–3 shows reference sites in this manual that explain different approaches to using the DataServer for MS SQL Server.

Table 1–3: How to use this manual

<table>
<thead>
<tr>
<th>If you are . . .</th>
<th>Refer to this information . . .</th>
</tr>
</thead>
</table>
| New to the DataServer for MS SQL Server | Chapter 2, “Initial Programming Considerations”  
Chapter 5, “Configuring the DataServer”  
Chapter 6, “Connecting the DataServer”  
Chapter 7, “The DataServer Tutorial” |
| Migrating an OpenEdge database to a MS SQL Server data source | The “Running the OpenEdge DB to MS SQL Server utility” section on page 7–20  
Chapter 2, “Initial Programming Considerations”  
Chapter 5, “Configuring the DataServer”  
Chapter 6, “Connecting the DataServer” |
| Upgrading from an earlier version of MS SQL Server | Appendix A, “Migration Issues”  
Chapter 5, “Configuring the DataServer” |

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

<table>
<thead>
<tr>
<th>Topic</th>
<th>OpenEdge manuals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting and setting up OpenEdge products</td>
<td><em>OpenEdge Getting Started: Installation and Configuration</em></td>
</tr>
<tr>
<td>Using the Data Dictionary</td>
<td><em>OpenEdge Getting Started: ABL Essentials</em></td>
</tr>
<tr>
<td>Defining security for an OpenEdge database</td>
<td><em>OpenEdge Data Management: Database Administration</em></td>
</tr>
</tbody>
</table>
| Writing applications in the ABL | *OpenEdge Development: ADM and SmartObjects*  
*OpenEdge Getting Started: ABL Essentials* |
| Connecting your data source with WebSpeed | *OpenEdge Getting Started: Installation and Configuration* |
| Writing WebSpeed applications | *OpenEdge Application Server: Developing WebSpeed Applications* |
Initial Programming Considerations

An application developed in the OpenEdge Studio can use the DataServer for MS SQL Server to access data from multiple sources transparently. With the DataServer, your applications can access information in both OpenEdge databases and MS SQL Server databases.

This chapter discusses fundamental considerations you need to know to plan your applications and design your databases. For example, you need to know the differences between OpenEdge databases and MS SQL Server databases, as well as the DataServer strategies for resolving these differences, which your application might need to accommodate. Specifically, it discusses the topics in the following sections:

- Database design issues
- Data types
- Record creation
- Data source record locking
- Transactions
- Error handling
- Cursors
- ABL issues
- RDBMS stored procedures
Follow these initial guidelines carefully when you develop your application to ensure that it can access OpenEdge databases and MS SQL Server databases transparently. Once you are comfortable with this information, refer to the stored procedure details presented in Chapter 3, “RDBMS Stored Procedure Details,” and the advanced guidelines in Chapter 4, “Additional Features to Enhance DataServer Performance.”

**Note:** The material in this chapter is also of interest to users who plan to migrate an OpenEdge database to a MS SQL Server 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 MS SQL Server utility” section on page 7–20.
Database design issues

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

OpenEdge database objects and MS SQL Server database objects

The OpenEdge database and the MS SQL Server database 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 MS SQL Server databases’s logical structure. Table 2–1 lists OpenEdge database objects and the MS SQL Server database equivalents.

Table 2–1: OpenEdge Database objects and MS SQL Server database objects

<table>
<thead>
<tr>
<th>OpenEdge</th>
<th>MS SQL Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table/File</td>
<td>Table</td>
</tr>
<tr>
<td>Field</td>
<td>Column</td>
</tr>
<tr>
<td>Record</td>
<td>Row</td>
</tr>
<tr>
<td>Index</td>
<td>Index</td>
</tr>
<tr>
<td>Unique index</td>
<td>Primary key</td>
</tr>
<tr>
<td>Nonunique index</td>
<td>Foreign key</td>
</tr>
<tr>
<td>Sequences</td>
<td>Native equivalent¹</td>
</tr>
<tr>
<td>Validation expression</td>
<td>Integrity constraint</td>
</tr>
<tr>
<td>Validation message</td>
<td>No equivalent</td>
</tr>
<tr>
<td>Initial value</td>
<td>Default</td>
</tr>
<tr>
<td>View</td>
<td>View.</td>
</tr>
<tr>
<td>No equivalent to stored procedures</td>
<td>Stored procedure.</td>
</tr>
<tr>
<td>Schema Trigger</td>
<td>Trigger</td>
</tr>
</tbody>
</table>

¹. The DataServer supports OpenEdge sequences through native stored procedures and triggers.
Initial Programming Considerations

Naming conventions

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

Table 2–2: Database naming restrictions

<table>
<thead>
<tr>
<th>Category</th>
<th>OpenEdge</th>
<th>MS SQL Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphanumeric characters</td>
<td>A–Z or a–z 0–9</td>
<td>All alphanumeric characters from the character set that you defined for your MS SQL Server</td>
</tr>
<tr>
<td>Special characters</td>
<td>Allows $, &amp;, #, %, –, and _1</td>
<td>Allows @, $, #, _, ¥, and £2</td>
</tr>
<tr>
<td>Initial character</td>
<td>Alphabetic only</td>
<td>Alphabetic only</td>
</tr>
<tr>
<td>Maximum length of object name</td>
<td>32 characters</td>
<td>24 characters3</td>
</tr>
<tr>
<td>Keywords</td>
<td>Not allowed4</td>
<td>Not allowed4</td>
</tr>
<tr>
<td>Unique table names</td>
<td>Required5</td>
<td>Required6</td>
</tr>
</tbody>
</table>

1. Although OpenEdge allows you to use the ampersand (&) in database-object names, the OpenEdge development tools use this character internally to denote preprocessor directives.

2. When the DataServer encounters characters not supported by OpenEdge in a sequence, table, index, or field name, it converts them to underscores (_) in the data source. 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.

3. Object names are limited to 32 characters. When you use the OpenEdge DB to MS SQL Server 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 DB to MS SQL Server utility, be sure to limit object names to 24 characters to accommodate this name resolution during conversion.

4. OpenEdge and MS SQL Server databases 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_”.

5. MS SQL Server qualifies 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 as follows: 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(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. Keep in mind that 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.

6. MS SQL Server allows you to qualify an object name by adding the database name, the owner’s name, or the table name (for columns).
**MS SQL Server DataServer data source limits**

A MS SQL Server data source has 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 restrictions might affect how OpenEdge applications run. See your Microsoft SQL Server documentation for full details.

**Code pages**

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

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

![Diagram](image)

**Figure 2–1: DataServer processes and code pages**

In the configuration shown in Figure 2–1, all components use the same code page. The OpenEdge iso8859–1 code page is a subset of the iso_1 code page used by the default installation of MS SQL Server. 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 a MS SQL Server data source accurately, you must specify the correct code page and collation 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 MS SQL Server 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 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 default code page setting in the schema holder is iso8859–1. You can specify a different code page for the schema holder at the following times:

- When you create the DataServer schema for the MS SQL Server 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.

**Note:** It is possible to change the code page at other times. However, because changing the code page does not affect the data already in the database, writing new data to your database using a different code page can corrupt the data in your database. You cannot use the PROUTIL utility to change the code page used by the DataServer.

Keep in mind that your MS SQL Server software configuration might have local requirements for defining the proper language interface between the ODBC drivers and the data source. See your Microsoft SQL Server 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 a MS SQL Server 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 and the default case insensitive sort ordering with a MS SQL Server 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 ~)
```

MS SQL Server 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 MS SQL Server collation.

Normally, the default OpenEdge collation sorts a tilde character above all alphanumeric characters. Therefore, in order for the above example to exhibit OpenEdge-like behavior and return alphanumeric records as well as records beginning with the tilde, the MS SQL Server 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)
```

MS SQL Server returns records with columns that begin 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 MS SQL Server from the OpenEdge client would require modifying the collation table associated with the OpenEdge code page and weighting it to match the MS SQL Server sort order.

For a complete discussion of how OpenEdge handles code-page issues, see OpenEdge Development: Internationalizing Applications.
Code page, collation, and case conflicts

How you specify code page, case sensitivity, and shadow column usage can lead to conflict between the attributes. It is best to establish the settings on these attributes prior to schema creation, and allow the schema pull to manage the integration into the schema image. Table 2–3 describes the behavior of the combinations.

Table 2–3: Case sensitivity interactions

<table>
<thead>
<tr>
<th>Code Page¹</th>
<th>Shadow Columns</th>
<th>Case Sensitive Field Attribute</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>Yes</td>
<td>Off</td>
<td>Shadow column provides case insensitivity</td>
</tr>
<tr>
<td>CS</td>
<td>No</td>
<td>Off</td>
<td>UPPER() function provides case insensitivity²</td>
</tr>
<tr>
<td>CS</td>
<td>Yes</td>
<td>On</td>
<td>Code page provides case sensitivity</td>
</tr>
<tr>
<td>CS</td>
<td>No</td>
<td>On</td>
<td>Code page provides case sensitivity</td>
</tr>
<tr>
<td>CI</td>
<td>Yes</td>
<td>Off³ (Is case-insensitive)</td>
<td>Shadow column provides case insensitivity⁴</td>
</tr>
<tr>
<td>CI</td>
<td>No</td>
<td>Off (Is case-insensitive)</td>
<td>Code page provides case insensitivity</td>
</tr>
<tr>
<td>CI</td>
<td>Yes</td>
<td>On</td>
<td>Code page provides case insensitivity⁵</td>
</tr>
<tr>
<td>CI</td>
<td>No</td>
<td>On</td>
<td>Code page provides case insensitivity⁶</td>
</tr>
</tbody>
</table>

1. The MS SQL Server code page is either Case Sensitive (CS) or Case Insensitive (CI).
2. Be aware that relying on the UPPER() function can have a negative performance impact.
3. This combination is not recommended. Don’t use shadow columns if the MS SQL Server code page can handle case-insensitivity instead.
4. This combination is not recommended. Use a code page to support case insensitivity by identifying the code page as case insensitive on a schema pull. Fields will be marked case insensitive on the pull.
5. Shadow columns are present, but play no role.
6. Case sensitivity cannot be supported by OpenEdge. The client will throw out any results not returned in case insensitive order.
Support for Unicode

Enabling Unicode support in the DataServer for MS SQL Server provides the following functionality:

- Completes the connection between UTF-8 support in the OpenEdge Advanced Business Language (ABL) client and UCS-2 support in MS SQL Server
- Allows Unicode data to be passed to and from MS SQL Server data sources
- Provides a migration path for Unicode character data type definitions in ABL to MS SQL Server, and vice versa

The result is that OpenEdge DataServer for MS SQL Server applications can be developed and deployed without significant language or country restrictions.

For general Unicode information, see *OpenEdge Development: Internationalizing Applications*.

Unicode support in the DataServer for MS SQL Server is enabled while migrating an OpenEdge database to MS SQL Server. For details see Chapter 7, “The DataServer Tutorial.”

Configuration requirements

In order to use Unicode in the OpenEdge DataServer for MS SQL Server, you must use Microsoft SQL Server 2005 or later, and the DataDirect Connect ODBC driver for MS SQL Server, Version 5.1 or later. Additionally, you must perform the following prerequisite tasks:

- **Prepare your Unicode Schema Holder** — If you create a new OpenEdge application to be Unicode-enabled for migration to an ORACLE DataServer, ensure that you create the database from a Unicode source. A copy of the empty OpenEdge UTF-8 database can be obtained from `$DLC/prolang/utf/empty`. If you use the Create DataServer Schema ORACLE DataServer utility to create a DataServer schema and you plan to set the schema holder’s code page to UTF-8, the schema holder must also be derived from a copy of the empty UTF-8 database located at `$DLC/prolang/utf/empty`.

  **Note:** The migration process derives a schema holder this way automatically if you set the code page of your migration to UTF-8.

The following command creates an empty OpenEdge database named “empty” that can be used to describe a Unicode-enabled foreign data source:

```
Prodb <OpenEdge utf-8 schema holder> $DLC/prolong/utf/empty
```

**Note:** The best way to ensure that the database path is correct for utilities that create a Unicode-enabled database is to set the `$DLCDB` environment variable to `$DLC/prolong/utf`. This ensures that databases will be constructed with Unicode enablement. If you do not set `$DLCDB`, you will need to reference the `$DLC/prolong/utf` directory explicitly whenever you want to create a Unicode-enabled database.
• **Use appropriate startup parameters** — To process Unicode data in your client application, set your cpinternal client code page to utf-8. If you are executing file I/O against your Unicode data, set your client’s cpstream code page to utf-8. You also may consider several other available client code page settings for Unicode. For more information, see *OpenEdge Development: Internationalizing Applications*.

• **Prepare OpenEdge databases for Unicode** — Your OpenEdge database does not need to be Unicode-enabled in order to be migrated to an ORACLE DataServer. The migration process will convert data from any ANSI code page to Unicode as part of the migration if you chose to move data during a migration and have specified a Unicode destination. If you have existing data in an OpenEdge database that you wish to convert to Unicode prior to migration, you can convert your non-Unicode data using one of two conversion utilities:
  
  – Data Administration dump and load utility
  – PROUTIL CONVCHAR character conversion utility

  For more information on these utilities, see *OpenEdge Development: Internationalizing Applications*.

**Data type support**

Unicode data and data types can be migrated to a MS SQL Server data source and pulled into a MS SQL Server DataServer schema holder. Table 2–4 defines the supported MS SQL Server data types and their maximum sizes.

**Table 2–4: Supported Unicode data types in MS SQL Server**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Maximum size</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCHAR</td>
<td>4000 characters</td>
</tr>
<tr>
<td>NVARCHAR</td>
<td>4000 characters</td>
</tr>
<tr>
<td>NVARCHAR(MAX)</td>
<td>$2^{30-1}$ characters</td>
</tr>
<tr>
<td>NTEXT</td>
<td>$2^{30-1}$ characters</td>
</tr>
</tbody>
</table>

MS SQL Server limits the size of CHAR and VARCHAR data types to 8000 bytes. When server data is represented by a single-byte ANSI code page, CHAR and VARCHAR data types can store 8000 characters. When SQL Server stores UCS-2 data in double-byte format, it restricts CHAR and VARCHAR data types to only 4000 characters. Therefore, when a non-Unicode OpenEdge database is migrated to MS SQL Server 2005 (or greater), the **Maximum Varchar Length** threshold is set to 8000 by default. Character data sized smaller than this threshold (which the user can configure lower) is converted to the VARCHAR native type. Character columns with widths greater than 8000 characters continue to be converted to the TEXT data type, MS SQL Server’s legacy CLOB equivalent data type. However, when a Unicode OpenEdge database is migrated to MS SQL Server 2005 (or greater), the **Maximum Varchar Length** threshold is set to 4000 by default and cannot be set any higher. Character columns with widths greater than 4000 characters are converted to the NVARCHAR(max) data type (MS SQL Server’s Unicode CLOB-equivalent data type).

The DataServer for MS SQL Server does not yet support CLOB-style native data types (TEXT, NTEXT, VARCHAR(max), NVARCHAR(max)). So, when CLOB-style server definitions in MS SQL Server are pulled back into an OpenEdge schema holder, their definitions are approximated to the mappings of an OpenEdge CHARACTER data type. This approximation results in data access
restrictions. While the server's CLOB-style definition may contain character data that exceeds 30,000 bytes, only approximately 30,000 bytes of that data can be retrieved and manipulated by the CHARACTER data type in ABL client. If the server data associated with a CLOB-style server definition is represented by UCS-2 character data in double-byte format, then the 30,000 /byte/ restriction on the CHAR data type also becomes an approximately 15,000 /character /restriction in the Unicode-enabled ABL client.

**Indexes and sorting**

You create and maintain all indexes from within the MS SQL Server data source using native tools, rather than with 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.

Give careful consideration to benefit and cost of creating indexes. Having indexes for frequently executed queries can greatly improve record retrieval performance. An abundance of unused or infrequently used indexes can have a negative impact on performance due to the overhead cost of maintaining the indexes.

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

1. They allow you to use the OF option in 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:

   ```ABL
   FOR EACH order OF customer:
   ```

2. They translate USE–INDEX to SQL ORDER BY for DataServer operations. A MS SQL Server data source uses the ORDER BY clause to assist in selecting the optimal index for the query. For example, if you define city-dept as a MS SQL Server data source primary key on the city and department fields, it is a unique index in the schema holder. In this case, the following OpenEdge statements are equivalent when accessing the data source:

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

   ```ABL
   FOR EACH employee BY city BY department:
   ```

**Note:** If you do not specify a USE–INDEX or BY clause, your query will return records in an unpredictable order. If your application requires a predictable order, use include a USE–INDEX or BY clause.
**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 MS SQL Server 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 MS SQL Server 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 MS SQL Server 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 a MS SQL Server 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 runtime errors. See Chapter 7, “The DataServer Tutorial,” for instructions on using the Data Dictionary to designate unique indexes.

MS SQL Server 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. For more information about views, see the “MS SQL Server data source views” section on page 2–13.

**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 OpenEdge **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, this enhancement will 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.
For example, migrating data to a MS SQL Server 2000 database will need to be limited to a wide key capacity of 900 bytes due to the 900-byte key restriction imposed by the foreign data source.

**Case sensitivity**

By default, an OpenEdge database is case insensitive. An MS SQL Server database is also case insensitive by default. Using case insensitivity in both OpenEdge and MS SQL Server enables seamless compatibility between the two, and provides the best performance, and least maintenance. However, you can set the attributes of a field to define it as either case sensitive or case insensitive. If you intend to use case sensitivity, consider the following:

- Pattern-matching literals in data source access statements retrieve case-sensitive data.
- The OpenEdge database considers the user ID and password submitted at connection time to be case sensitive.

If an indexed field is case insensitive, an OpenEdge database 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.

If you are using a case sensitive code page, the DataServer can make this feature compatible across OpenEdge and MS SQL Server data sources. To support case insensitivity with a case sensitive code page, an extra column, known as a shadow column, must be added to the data source immediately before the indexed column. This column is named _S#_column. See the “Migrating an OpenEdge database to MS SQL Server” section on page 7–20 for instructions on adding this column automatically with the OpenEdge DB to MS SQL Server utility.

**Note:** By default, sort order in MS SQL Server is not case sensitive.

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

Properly setting code page, collation, and case sensitivity values such that they compliment each other, will yield the best scenarios for data access. To avoid conflict between 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. *Table 2–3* describes the interaction between code page, collation, and case sensitivity.

**MS SQL Server data source views**

MS SQL Server data source schema objects include views. A *view* is a presentation of data in one or more tables. Views appear as tables, not as views, in the Data Dictionary’s table list for the schema holder. 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 OpenEdge statements. Furthermore, 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:

```
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. See the “Modifying field-level information” section on page 7–33 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 MS SQL Server 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(cust_num) from customer
```

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 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 MS SQL Server data sources. 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 MS SQL Server data source and the operation (INSERT, UPDATE, DELETE) executes.

Triggers for OpenEdge databases and MS SQL Server 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 MS SQL Server data sources are independent, they might affect each other based on the kind of transaction your application is executing. For example, ensure that triggers running in both OpenEdge and at the data source do not dead lock on the same record access.
OpenEdge sequences

A sequence generator is a database object that provides incremental values within any integer range. (You can specify any positive or negative increment.) During an OpenEdge DB to MS SQL Server migration, the DataServer internally simulates support for OpenEdge sequences using native MS SQL Server stored procedures and a sequence table. You cannot create sequences for MS SQL Server tables directly through the Data Dictionary.

The DataServer for MS SQL Server offers both a revised and a legacy version of the sequence generator. Each legacy sequence has one stored procedure and one table object associated with it whose names are prefixed with _SEQP_ and _SEQT_ respectively. The same objects in the new sequence generator are prefixed with SEQP_REV_ and SEQT_REV respectively and include a single common table object named SEQT_REV_SEQTMGR. When a sequence is migrated to MS SQL Server, the new sequence generator is used if the Use revised sequence generator option is checked in the migration dialog. If you remigrate with the same option turned off, the sequences will revert to the legacy model in the foreign database and the schema holder. Only one sequence generator is implemented for the DataServer at a given time. If you elect to use the revised sequence generator and the legacy version already exists, the legacy version is replaced, and visa versa. It is also possible to change the sequence version using the delta.sql utility by setting or unsetting the Use revised sequence generator option.

The sequence procedures all run in separate connections from your ABL transactions. In that connection, a different transaction is established to handle modifications to the sequence table rows. With all the sequence requests made against the legacy sequence generator, if a sequence is being actively modified when another user attempts to access the sequence record, the second user must wait for a lock. Under heavy use, it is possible for a lock timeout to occur on a sequence. To avoid any probability of sequence timeouts, sequence retry logic in the DataServer attempts to handle concurrency automatically. In the remote instance of a sequence lock timeout that is not resolved internally, the user is given the option to continue to wait and retry or to cancel the retries and abandon the sequence request. Setting a non-zero value for the --Dsrv switch PRGRS_NATIVE_LOCKWAIT may further reduce any possibility of an unmanaged sequence timeout. For more information on setting the PRGRS_NATIVE_LOCKWAIT value, refer to the “Handling lock timeouts” section on page 2–37 or “DataServer options” section on page 8–4.

The revised version of the sequence generator eliminates lock contention when using the commonly utilized NEXT-VALUE sequence function. Lock contention is also greatly minimized for the GET-VALUE function. Lock contention for SET-VALUE function is not reduced and may not be a better choice over the legacy sequence generator for this function. If your particular application is atypical in the sense that it uses the SET-VALUE function frequently as compared to your use of NEXT-VALUE and GET-VALUE, then the previous version of the sequence generator may provide an advantage.

**Note:** When using the revised sequence generator, your first next-value operation produces a value that is equal to the initial value specified for the sequence generator. By comparison, the first value produced for the legacy sequence generator is equal to the initial value plus the increment value.

The bit size of the legacy database sequences can be either 32-bit INTEGER or 64-bit INT64. The revised sequence generator was introduced after the DataServer had already transitioned to 64-bit values. Therefore all revised sequence generators are 64-bit. The following section highlights changes regarding 64-bit sequences.
Support for 64-bit sequences

The MS SQL Server DataServer supports the following changes for 64-bit sequences:

- An existing database sequence will remain defined as a 32-bit INTEGER and migrate as 32-bit, unless you explicitly choose to define a sequence as 64-bit INT64.

- The MS SQL Server DataServer can invoke a sequence function to process INT64 values through a standard implementation. That is, use GET, NEXT, and SET sequence functions to get or set an INT64 value.

- Starting with OpenEdge Release 10.1B, an OpenEdge database, by default, defines a Sequence field in the schema as an INT64 data type. Therefore, sequences migrated from a new OpenEdge database will automatically expand to support INT64 data types whereas previously they returned INTEGER values. You can override the INT64 return value using the INTEGER function. However, note that there is a potential for an overflow condition to occur if the target functionality expects an INTEGER value.

- When the MS SQL Server is the target database, and the sequence’s upper limit is defined as the Unknown value (?) in the source database, the migration utility will specify the sequence’s upper limit value as 922337203685775807. However, if support for large sequences is not enabled for an OpenEdge database, the sequence’s upper limit value is specified as 2147483647.

- An overflow condition can occur and subsequently an error message can appear on a client machine when a sequence generator is defined as a 64-bit value, but the value loaded into a client is OpenEdge Release 10.1A or earlier.
Data types

MS SQL Server data types differ from OpenEdge data types. However, each data type supported by the DataServer has at least one OpenEdge equivalent.

The DataServer translates MS SQL Server data types into OpenEdge equivalents and places the mapping into the schema holder. You can access this mapping information using the Data Dictionary. For example, the Data Dictionary lists the MS SQL Server datetime and smalldatetime data types as timestamp, which is the equivalent ODBC SQL data type and not the MS SQL Server timestamp data type. There can also be some restrictions in data source compatibility with OpenEdge.

You can also modify these definitions using the Data Dictionary. For example, the DataServer maps the MS SQL Server 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 runtime. Also, remember to specify a display format that is appropriate for the new data type. See the “Modifying a schema holder” section on page 7–32 for an explanation of how to use the Data Dictionary to change OpenEdge data types in the schema holder.

Table 2–5 lists these data types and their default OpenEdge equivalents. The data types in parentheses are alternative data types that you can specify in the schema holder for your MS SQL Server data source. See Chapter C, “Data Type Details” for more details about each MS SQL Server data type, its ODBC SQL equivalent, and OpenEdge data type equivalent.

Table 2–5: MS SQL Server data type equivalents in OpenEdge (1 of 3)

<table>
<thead>
<tr>
<th>MS SQL Server data type</th>
<th>OpenEdge data type equivalent¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>INTEGER (DECIMAL)</td>
</tr>
<tr>
<td></td>
<td>(LOGICAL)</td>
</tr>
<tr>
<td></td>
<td>(INT64)</td>
</tr>
<tr>
<td>bigint</td>
<td>INT64 (DECIMAL)</td>
</tr>
<tr>
<td></td>
<td>(INTEGER)</td>
</tr>
<tr>
<td>smallint</td>
<td>INTEGER (DECIMAL)</td>
</tr>
<tr>
<td></td>
<td>(LOGICAL)</td>
</tr>
<tr>
<td></td>
<td>(INT64)</td>
</tr>
<tr>
<td>tinyint</td>
<td>INTEGER (DECIMAL)</td>
</tr>
<tr>
<td></td>
<td>(LOGICAL)</td>
</tr>
<tr>
<td></td>
<td>(INT64)</td>
</tr>
<tr>
<td>decimal</td>
<td>DECIMAL (INTEGER)</td>
</tr>
<tr>
<td></td>
<td>(INT64)</td>
</tr>
<tr>
<td>numeric</td>
<td>DECIMAL (INTEGER)</td>
</tr>
<tr>
<td></td>
<td>(INT64)</td>
</tr>
</tbody>
</table>
## Table 2–5: MS SQL Server data type equivalents in OpenEdge (2 of 3)

<table>
<thead>
<tr>
<th>MS SQL Server data type</th>
<th>OpenEdge data type equivalent¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>DECIMAL (INTEGER)</td>
</tr>
<tr>
<td></td>
<td>(INT64)</td>
</tr>
<tr>
<td>double precision</td>
<td>DECIMAL (INTEGER)</td>
</tr>
<tr>
<td></td>
<td>(INT64)</td>
</tr>
<tr>
<td>real</td>
<td>DECIMAL (INTEGER)</td>
</tr>
<tr>
<td></td>
<td>(INT64)</td>
</tr>
<tr>
<td>char</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>nchar</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>varchar</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>nvarchar</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>text²</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>varchar(max)³</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>ntext²</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>nvarchar(max)</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>money</td>
<td>DECIMAL (INTEGER)</td>
</tr>
<tr>
<td>smallmoney</td>
<td>DECIMAL (INTEGER)</td>
</tr>
<tr>
<td>date⁴</td>
<td>DATE (DATETIME)</td>
</tr>
<tr>
<td></td>
<td>(DATETIME-TZ)</td>
</tr>
<tr>
<td></td>
<td>(CHARACTER)</td>
</tr>
<tr>
<td>time⁴</td>
<td>CHARACTER (DATETIME)</td>
</tr>
<tr>
<td>datenetime⁴</td>
<td>DATE (CHARACTER)</td>
</tr>
<tr>
<td>datenetime2⁴</td>
<td>DATETIME (DATE)</td>
</tr>
<tr>
<td></td>
<td>(DATETIME-TZ)</td>
</tr>
<tr>
<td></td>
<td>(CHARACTER)</td>
</tr>
<tr>
<td>smalldatetime</td>
<td>DATE (CHARACTER)</td>
</tr>
<tr>
<td>datenotimeoffset⁴</td>
<td>DATETIME-TZ (DATETIME)</td>
</tr>
<tr>
<td></td>
<td>(DATE)</td>
</tr>
<tr>
<td></td>
<td>(CHARACTER)</td>
</tr>
<tr>
<td>binary</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>varbinary</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>varbinary (max)</td>
<td>CHARACTER BLOB²</td>
</tr>
</tbody>
</table>
Working with unsupported data types

You can read data from tables containing unsupported data types by using two possible workarounds – first, by creating a view or, secondly, by using a stored procedure.

To use the first workaround, create a view in the SQL Server database that excludes the unsupported column so that only supported data types are available. For more information on views, see the “MS SQL Server data source views” section on page 2–13.

To use the second workaround, write your own stored procedure with the RUN STORED PROCEDURE statement or write SQL into a SEND-SQL-STATEMENT stored procedure call that expresses Data Manipulation Language (DML) adequate to handle the table object without the unsupported types. You can omit the unsupported data type in your DML statements such that they are not returned in the result set or are CAST to data types that are supported. For more information on stored procedures, see Chapter 3, “RDBMS Stored Procedure Details.”

1. The initial entry identifies the default data type. Data types in parentheses identify supported options.
2. Starting in MS SQL Server 2005, "image" and "text" LOB types on the server can be expressed as "varbinary(n)/varchar(max)" types respectively. While still supported, Progress recommends legacy LOB types be converted to "var(max)" types for better forward compatibility.
3. The default CHARACTER mapping for binary LOB server types can be changed to OpenEdge BLOB by selecting the "Default to OpenEdge LOB" checkbox during object selection on a schema pull. WARNING: CHARACTER and LOB columns are handled differently in OpenEdge. Code changes are required in your application if you to re-map a server column to an OpenEdge BLOB that had previously been mapped to a CHARACTER field.
4. Starting in MS SQL Server 2008 using native drivers, these data types can be pulled into an OpenEdge schema holder for MS SQL Server.
5. These non-updatable columns can be mapped into an MSS DataServer schema holder but cannot be written to on the server.
6. For more information on unsupported data types, see the “Working with unsupported data types” section on page 2–19.
Working with non-updatable data types

Because non-updatable columns can be read, they can also be dumped to a .d data file. But, these .d data files cannot be loaded with the load .d utility because of their non-updatable column status. The non-updatable values are applied by the server and therefore cannot be retained from the client. Use foreign database tools to dump and load tables with non-updatable columns that are controlled by the server. Alternatively, write a program to read data from your .d data file with non-updatable types into a temp table omitting the non-updatable columns using the EXCEPT phrase when you write them.

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

Starting with OpenEdge Release 10.1B, DataServer for MS SQL Server supports an existing or new 32-bit INTEGER data type defined in ABL. However, the internal bit size for an INTEGER data type and ROWID will automatically expand from a 32-bit field size to a 64-bit field size to accommodate larger values. 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. For more details about ROWIDs, see the “ROWID function” section on page 2–52. Also, refer to the following sources for additional language details about INT64 and ROWID:

- *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 OpenEdge database from INTEGER to INT64 and you had previously migrated your database to either a MS SQL Server or 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 OpenEdge database from INT64 to INTEGER and you had previously migrated your database to either a MS SQL Server or 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 run-time 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.

DataServer for MS SQL Server support for datetime data types

OpenEdge DataServer for MS SQL Server supports DATETIME data types in MS SQL Server up to SQL Server 2008 with full support for the ABL DATETIME-TZ data type.

Working with MS SQL Server and ABL datetime data types

Table 2–6 defines the MS SQL Server 2000, 2005 and 2008 data types and identifies compatible ABL data types.

Table 2–6: MS SQL Server 2000 and 2005 datetime data types

<table>
<thead>
<tr>
<th>MS SQL Server data type</th>
<th>Description</th>
<th>Compatible OpenEdge data types</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATETIME</td>
<td>Provides date and time data to an accuracy of one-thousandth of a second (equivalent to 3.33 milliseconds or 0.00333 seconds). Values are rounded to increments of .000, .003, or .007.</td>
<td>DATE¹, CHAR, DATETIME²,³ DATETIME-TZ³</td>
</tr>
<tr>
<td>SMALLDATETIME</td>
<td>Provides date and time data with accuracy to the minute.</td>
<td>DATE¹, CHAR, DATETIME²,⁴ DATETIME-TZ⁴</td>
</tr>
</tbody>
</table>

1. Default, compatible data type to which the native type can be converted.
2. Check the Default to OpenEdge DATETIME option from Pre-Selection Criteria For Schema Pull on the Update/Add Table Definitions dialog to get this data type instead of the default.
3. When converting OpenEdge Date/Time/TZ data types to the legacy MS SQL Server Date/Time data type, the same 3.33 millisecond accuracy that SQL Server uses is applied to OpenEdge values with millisecond precision. This means that .989, .990 and .991 round to .990, .992, .993 and .994 round to .993, .995, .996, .997 and .998 round to .997 and .999 rounds up to 1.000.
4. When converting OpenEdge Date/Time/TZ data types to the legacy MS SQL Server SmallDateTime data type, the same minute accuracy that SQL Server uses is applied to OpenEdge values with second or greater precision. This means that seconds values up to 29.998 seconds will round down to zero and values from 29.999 seconds and up will round up to the next minute.

Beginning in OpenEdge Release 10.2B, the DataServer for MS SQL Server supports mappings of MS SQL Server datetime data types DATETIME and SMALLDATETIME to OpenEdge data type DATETIME-TZ along with the compatible OpenEdge types listed in Table 2–5.
MS SQL Server 2008 offers the **DATETIMEOFFSET** data type to support time zones in a datetime column as well as three other datetime data types. Table 2–7 defines the MS SQL 2008 data types and identifies compatible ABL data types.

**Table 2–7: MS SQL Server 2008 datetime data types**

<table>
<thead>
<tr>
<th>MS SQL Server data type</th>
<th>Description</th>
<th>Compatible OpenEdge data types</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>Provides day of the year based on the Gregorian calendar.</td>
<td>DATE&lt;sup&gt;1&lt;/sup&gt; CHAR DATETIME&lt;br&gt;DATETIME&lt;sup&gt;1,2&lt;/sup&gt; DATETIME-TZ&lt;sup&gt;2&lt;/sup&gt;</td>
</tr>
<tr>
<td>TIME</td>
<td>Provides time of day based on 24-hour clock.</td>
<td>CHAR&lt;sup&gt;1&lt;/sup&gt; DATETIME&lt;sup&gt;2&lt;/sup&gt;</td>
</tr>
<tr>
<td>DATETIME&lt;sup&gt;2&lt;/sup&gt;</td>
<td>Provides variable precision of up to 100 nanoseconds.</td>
<td>DATE CHAR DATETIME&lt;br&gt;DATETIME&lt;sup&gt;1,2,3&lt;/sup&gt; DATETIME-TZ&lt;sup&gt;2,3&lt;/sup&gt;</td>
</tr>
<tr>
<td>DATETIMEOFFSET</td>
<td>Provides date and time data with time zone offset and variable precision of up to 100 nanoseconds.</td>
<td>DATE CHAR DATETIME&lt;br&gt;DATETIME&lt;sup&gt;2,3&lt;/sup&gt; DATETIME-TZ&lt;sup&gt;2,1,2,3&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

1. Default data type mapping for this MS SQL Server data type.
2. New MSS data types with a time component support fractional accuracy up to 100 nanoseconds. Hence, when we receive any time data from an OpenEdge datetime data type with a time component, the fractional portion of the time value greater than millisecond accuracy is dropped in the conversion. For example, a time value ‘12:00:00.1235678’ in MS SQL Server would yield a resultant value of ‘12:00:00.123’ in an OpenEdge DATETIME column. NOTE: The time component is truncated rather than rounded in order to be consistent with MS SQL Server which also truncates the value when converting between two datetime data types defined with inconsistent fractional accuracy.
3. ABL data type limits MSS data type accuracy to millisecond precision.

**Note:** Date time related data types of MS SQL Server 2008 will work only with the SQL Native Client 2008 (SNAC 10.0) driver and not with earlier native and non-native drivers, as earlier version drivers will not support 2008 date time data types.
The ABL DATETIME and DATETIME-TZ data types are compatible with datetime data types supported by MS SQL Server. Table 2–8 provides details on compatibility:

### Table 2–8: ABL datetime data types

<table>
<thead>
<tr>
<th>ABL Data Type</th>
<th>Definition</th>
<th>MS SQL Server compatible data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>ABL date</td>
<td>DATE(^1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SMALLDATETIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATETIME(^2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATETIME2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATETIMEOFFSET</td>
</tr>
<tr>
<td>DATETIME(^3)</td>
<td>The DATETIME data type consists of two parts, one an ABL date and one an ABL time. The unit of time is milliseconds from midnight.</td>
<td>DATE(^4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SMALLDATETIME(^4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATETIME(^2),(^4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATETIME(^2),(^4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATETIMEOFFSET(^1),(^4)</td>
</tr>
<tr>
<td>DATETIME-TZ(^5)</td>
<td>Variation of DATETIME data type with time zone offset.</td>
<td>DATE(^4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SMALLDATETIME(^4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATETIME(^2),(^4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATETIME(^2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATETIMEOFFSET(^1),(^4)</td>
</tr>
</tbody>
</table>

1. Default data type mapping on an OpenEdge migration to the MS SQL Server DataServer when Map to MSS Datetime Type check box is checked.
2. Default data type mapping on an OpenEdge migration to the MS SQL Server DataServer when the Map to MSS Datetime Type check box is unchecked. NOTE: You should only uncheck this box when your target database is MS SQL Server 2008 or later.
3. Compatible with MS SQL Servers 2000 and later.
4. Since the valid date range for OpenEdge date and datetime data types with a date component is wider than the range available to MS SQL Server data types with a date component, it is recommended that a validation expressions be applied to fields in the schema holder mapped to MS SQL Server datetime types with a narrower date range. This would ensure that the client first blocked out-of-range dates before they were invalidated by server operations.
5. Compatible with MS SQL Server 2008 and later.

### Performing data type conversions

Making changes between ABL and MS SQL Server data types will affect how the data is stored. The following tables describe the effects of changing one data type to another.
Table 2–9 provides details on converting ABL DATETIME data types and MS SQL Server data types:

<table>
<thead>
<tr>
<th>Source (OpenEdge)</th>
<th>Target (MSS)</th>
<th>Default Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>DATE</td>
<td>Straight copy of date portion.</td>
</tr>
<tr>
<td>DATE</td>
<td>DATETIME</td>
<td>Straight copy of date portion. Time set to midnight up to millisecond precision (00:00:00.000).</td>
</tr>
<tr>
<td>DATE</td>
<td>DATETIME2</td>
<td>Straight copy of date portion. Time set to midnight up to 1 minute of accuracy.</td>
</tr>
<tr>
<td>DATE</td>
<td>SMALLDATETIME</td>
<td>Straight copy of date portion. Time set to midnight up to millisecond precision (00:00:00.000). Time zone set to the OpenEdge client session time zone context.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>DATE</td>
<td>Straight copy of date portion of DATETIME value. Time and time zone portion are dropped.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>DATETIME2</td>
<td>Straight copy of the datetime portion up to millisecond precision.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>SMALLDATETIME</td>
<td>Straight copy of the datetime portion up to a minute of precision.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>TIME</td>
<td>Straight copy of time portion up to millisecond precision. Date portion is dropped.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>DATETIMEOFFSET</td>
<td>Straight copy of the datetime portion. Time zone set to the OpenEdge client time zone.</td>
</tr>
<tr>
<td>DATETIME-TZ</td>
<td>DATETIME</td>
<td>Straight copy of the datetime portion of DATETIME-TZ value (in client time-zone context) up to millisecond precision. Time zone portion is dropped.</td>
</tr>
<tr>
<td>DATETIME-TZ</td>
<td>DATETIME2</td>
<td>Straight copy of the datetime portion of DATETIME-TZ value (in client time-zone context) up to millisecond precision. Time zone portion is dropped.</td>
</tr>
<tr>
<td>DATETIME-TZ</td>
<td>SMALLDATETIME</td>
<td>Straight copy of the datetime portion of DATETIME-TZ value (in client time-zone context) up to 1 minute of precision. Second and millisecond portion is dropped. Time zone portion is dropped.</td>
</tr>
<tr>
<td>DATETIME-TZ</td>
<td>DATETIMEOFFSET</td>
<td>Straight copy with millisecond precision.</td>
</tr>
<tr>
<td>DATETIME-TZ</td>
<td>DATE</td>
<td>Straight copy of date portion of DATETIME-TZ value (in client time-zone context). Time and time zone portion are dropped.</td>
</tr>
</tbody>
</table>
Table 2–10 provides details on converting ABL DATETIME data types and MS SQL Server data types:

### Table 2–10: Converting ABL datetime types to MS SQL Server data types

<table>
<thead>
<tr>
<th>Source (MSS)</th>
<th>Target (OpenEdge)</th>
<th>Default Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>DATE</td>
<td>Straight copy of date portion.</td>
</tr>
<tr>
<td>SMALLDATETIME DATETIME DATETIME2</td>
<td>DATE</td>
<td>Straight copy of date portion. Time portion is dropped.</td>
</tr>
<tr>
<td>DATETIMEOFFSET</td>
<td>DATE</td>
<td>Receives MSS DATETIMEOFFSET (with time zone context from the database). DataServer converts to OE client time zone context. Date portion of converted timestamp is copied. Time and time zone portions are dropped.</td>
</tr>
<tr>
<td>DATE</td>
<td>DATETIME</td>
<td>Straight copy of the date portion into the DATETIME. Time portion is set to 00:00:00</td>
</tr>
<tr>
<td>DATETIME</td>
<td>DATETIME</td>
<td>Straight copy of datetime portion up to millisecond level accuracy.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>DATETIME</td>
<td>Straight copy of datetime portion up 1 minute of accuracy. Second and millisecond portion set to 0.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>DATETIME</td>
<td>Straight copy with millisecond accuracy.</td>
</tr>
<tr>
<td>DATETIMEOFFSET</td>
<td>DATETIME</td>
<td>Receives MSS DATETIMEOFFSET with stored time zone context. DataServer converts to OE client time zone context. Timestamp portion (in client time) is copied with millisecond accuracy. Time zone portion is dropped.</td>
</tr>
<tr>
<td>TIME</td>
<td>DATETIME</td>
<td>Straight copy of TIME portion up to millisecond accuracy. Date portion is set to TODAY based on client time zone context.¹</td>
</tr>
<tr>
<td>DATE</td>
<td>DATETIME-TZ</td>
<td>Straight copy of the date portion into the DATETIME-TZ. Time portion is set to 00:00:00. Time zone will be set to client time zone context.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>DATETIME-TZ</td>
<td>Straight copy of the date and time portions into the DATETIME-TZ up to millisecond of accuracy. Time zone will be set to client time zone context.</td>
</tr>
</tbody>
</table>
Enabling Datetime data types using the Data Administration tool

The OpenEdge Data Administration tool provides utilities for migrating an OpenEdge database to a MS SQL Server data source and for pulling definitions from an MS SQL Server data source into an OpenEdge database, referred to as a schema holder. In doing so, OpenEdge applications use the DataServer to access and update data stored in MS SQL Server data sources.

The Data Administration tool ensures that utilities can recognize and handle date time data types supported in starting in MS SQL Server 2008 and later. Those utilities include:

- Update/Add Table Definitions
- Verify Table Definitions
- Schema Migration > OpenEdge DB to MS SQL Server
- Adjust Schema
- Generate Delta.sql OpenEdge to MS SQL Server
- Dump as Create Table Statement

**Update/Add Table definitions utility**

The Update/Add Table Definitions utility allows you to pull definitions from a table in a foreign data source. The table may be new or already exist in the schema holder, in which case you can add definitions or update definitions that are outdated. By default, MS SQL Server DATETIME and SMALLDATETIME data types are mapped to the OpenEdge DATE data type. The utility allows you to change the mapping to OpenEdge DATETIME.

**Table 2–10:** Converting ABL datetime types to MS SQL Server data types

<table>
<thead>
<tr>
<th>Source (MSS)</th>
<th>Target (OpenEdge)</th>
<th>Default Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALLDATETIME</td>
<td>DATETIME-TZ</td>
<td>Straight copy of the date and time portions of the MSS SMALLDATETIME with 1 minute accuracy. Time zone will be set to client time zone context.</td>
</tr>
<tr>
<td>DATETIMEOFFSET</td>
<td>DATETIME-TZ</td>
<td>Straight copy with millisecond accuracy.</td>
</tr>
</tbody>
</table>

1. When a TIME data type in MS SQL Server is mapped to an OpenEdge Datetime data type, the date portion defaults to TODAY's date unless a valid date is specified for the -Dsrvv PRGRS_DATEPART_OVERRIDE switch at connection time. This switch allows the user to set a date for this conversion other than TODAY's date.

**Note:** Since time zone support is new in 10.2B, any DATETIMEOFFSET column pulled from an MSS SQL Server 2008 or later database is automatically mapped to DATETIME-TZ by default.
Data types

Schema Migration > OpenEdge DB to MS SQL Server

By default, ABL DATE and DATETIME data types were mapped to MS SQL Server DATETIME data types. However, the migration tool provides you with the option of selecting the following datetime data type mapping:

<table>
<thead>
<tr>
<th>OpenEdge Data Type</th>
<th>MSS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>DATE</td>
</tr>
<tr>
<td>DATETIME</td>
<td>DATETIME2</td>
</tr>
<tr>
<td>DATETIME-TZ</td>
<td>DATETIMEOFFSET</td>
</tr>
</tbody>
</table>

To override the default mapping and choose a mapping to the data types supported by MS SQL Server 2008, uncheck the Map to MSS ‘Datet ime’ Type option in the migration dialog:

Note: Do not uncheck the Map to MSS ‘Datet ime’ Type option if your migration target is not an MS SQL Server 2008 database or above. You will receive an error message if you try to connect to a version of MS SQL Server prior to 2008, or if you use a native driver lower than SNAC-10, or if you use a non-native driver to connect.

Verify Table Definitions and Adjust Schema utilities

The Verify Table Definitions utility enables you to verify that the definitions in the schema holder match the definition of objects in the foreign database. The Adjust Schema utility compares the connected OpenEdge database to the schema image in the connected schema holder and updates the OpenEdge attributes necessary for your ABL code to run against both data sources. Both utilities are updated to work with datetime data types.
Generate delta.sql OpenEdge to MS SQL Server utility

The Generate delta.sql utility allows you to migrate schema changes from an OpenEdge database to an MS SQL Server database. The utility reads a delta.df file that has been created using the incremental dump procedure and creates a new file containing SQL DDL for updating the .df file which can then be used to complete the migration process.

By default, the Map to MSS Datetime Type option is checked. Uncheck the box to map MS SQL Server 2008 data types to ABL DATE, DATETIME, and DATETIME-TZ data types:

Dump as Create Table Statement

The Dump as Create Table Statement utility allows you to select one or all tables of the working database and generate scripts for MS SQL Server that contain statements equivalent to those originally used to define selected tables in the foreign data source. By default, the Map to MSS Datetime Type option is selected. Uncheck the option to generate columns with datetime data types available in MS SQL Server 2008.
Default and Special Datetime Default Values

When a DATETIME or a DATETIME-TZ value is defined to a table in the DataServer schema without a default value, its initial value is automatically set to “?” (the Unknown value). An option for setting the initial value is to use the ABL NOW function. It initializes both date and time parts of the current date and time and the time zone portion for DATETIME-TZ columns. Using NOW for initialization sets the date, time, and time zone based on SESSION:TIME-SOURCE.

Using Datetime Data Types with Stored Procedures

Beginning in OpenEdge Release 10.2B, DATETIME-TZ is included as a parameter type that can be passed to and received from stored procedures.

The DATETIME-TZ data types are definable in both static and dynamic temp tables. Therefore, along with parameterization, SENDSQL and stored procedure result sets can convert DATETIME-TZ data types when using LOAD-RESULT-INTO.

If query results derived from a stored procedure are defined by foreign data types that require special mapping in order to retain their value or a portion of their value, they should be CAST to a type that can handle them. For instance, a TIME data type in a stored procedure result set should be CAST to a datetime data type that can receive the time component since OpenEdge does not support a TIME data type directly. The CAST should be done as part of the result set before it is received into a SQL View or LOAD-RESULT-INTO temp tables. Similarly, an MS SQL Server datetime data type with time precision greater than OpenEdge should be CAST to a native datetime data type with the same or less precision than OpenEdge before it is returned in the result set so it can be properly received. Or, you could CAST a more precise value to a VARCHAR column in advance and preserve the precision in the alphanumeric form of an OpenEdge CHARACTER field.

Datetime index components

Indexing of DATETIME and DATETIME-TZ columns is supported in schema holders. Using DATETIME-TZ fields in an index is particularly useful since the date and time value is stored in UTC and therefore will display results in absolute time.

Using datetime data types in a WHERE clause

Mixing DATE, DATETIME, DATETIME-TZ in a WHERE clause or in other comparisons will cause compile time errors and are not valid comparisons.
Support for OpenEdge ABL BLOB data type

The DataServer provides support for the OpenEdge ABL BLOB (Binary Large Object) data type by mapping to equivalent data types in the MS SQL Server. This support is limited to data sources versions of MS SQL Server 2005 and later. The ability to map to an OpenEdge BLOB data type enables you to:

- Migrate to an OpenEdge database with BLOB data type to an MS SQL Server database as VARBINARY (MAX) and to pull data back as a BLOB data type into the schema holder. For more information on enabling the BLOB data type during a schema migration, see the “Migrating an OpenEdge database to MS SQL Server” section on page 7–20.

- Pull MS SQL Server VARBINARY(MAX), IMAGE, or VARBINARY(MAX) FILESTREAM as OpenEdge BLOB data type into the schema holder.

- Read and write data to and from an MS SQL Server database that is BLOB-enabled by using ABL ASSIGN and COPY-LOB operations.

**Note:** In a COPY-LOB operation, the entire BLOB object must be copied, therefore the STARTING AT, OVERLAY AT, TRIM, APPEND and FOR phrases are unsupported.

For more information, on using the ASSIGN and COPY-LOB statements, see OpenEdge Development: ABL Reference.

ABL-to-MS SQL Server data type mapping

The following tables depict the mapping between ABL BLOB and equivalent MS SQL Server data types.

Table 2–11 depicts the mapping between the ABL BLOB data type and its MS SQL Server equivalent during a schema migration.

**Table 2–11: BLOB data type in schema migration**

<table>
<thead>
<tr>
<th>OpenEdge Data Type</th>
<th>MSS Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MS SQL Server 2005</td>
</tr>
<tr>
<td>BLOB</td>
<td>VARBINARY(MAX)</td>
</tr>
</tbody>
</table>

The Map to MSS Datetime Type option is on by default, which means that the default behavior is to use the behavior from the previous release, mapping to the DATETIME data type, and not the new data types.
Table 2–12 depicts mapping between the ABL BLOB data type and its MS SQL Server equivalent during a schema pull.

Table 2–12:  BLOB data type in schema pull

<table>
<thead>
<tr>
<th>MS SQL Server Data Type</th>
<th>OpenEdge Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSS 2005</td>
<td>MSS 2008</td>
</tr>
<tr>
<td>VARBINARY(MAX)</td>
<td>VARBINARY(MAX)</td>
</tr>
<tr>
<td>IMAGE</td>
<td>IMAGE</td>
</tr>
<tr>
<td>VARBINARY(MAX)</td>
<td>VARBINARY(MAX)</td>
</tr>
</tbody>
</table>

User-defined data types

MS SQL Server allows you to define your own data types that map to native MS SQL Server data types. When the DataServer reads the schema information for a user-defined data type, it reads the MS SQL Server 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. The DataServer interprets specially named data source columns of the same data type as an OpenEdge field with the same number of array elements. You name the data source columns column_name##1, column_name##2, and so forth, to correspond to an OpenEdge array named column_name. The DataServer creates a single field definition in the schema holder for the field extents. See the “Migrating an OpenEdge database to MS SQL Server” section on page 7–20 for instructions on adding these columns automatically with the OpenEdge DB to MS SQL Server utility.

Unknown value (?)

The DataServer supports 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 null-capable, the following statement fails at run time:

FIND customer WHERE customer.custnum NE ?

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

```
FIND FIRST customer.
customer.address2 = ?.
```

### Zero-length character strings

When you use the Unknown value (?) in a **WHERE** clause with the DataServer, the Unknown value (?) satisfies only the equals (=) or not equal (<> or <>) 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 EQ " ".
FIND FIRST customer WHERE customer.address2 EQ "".
```

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 " ":
```
Record creation

Record creation is handled differently for OpenEdge databases and MS SQL Server 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 a MS SQL Server 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 MS SQL Server 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 a MS SQL Server data source since the record lock will be held from the time of the write until the end of a transaction.

While both OpenEdge and MS SQL Server 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. A MS SQL Server data source drops all record locks at the end of a transaction. See the “Transaction scoping and buffer management” section on page 2–43 for more information.

The following ABL statements will force a physical database write to a MS SQL Server 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 affect 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 MS SQL Server 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 MS SQL Server data source. OpenEdge can assign a RECID or ROWID without writing a record. For a MS SQL Server 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, a MS SQL Server 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, a MS SQL Server 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:

```
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:

```
DEFINE BUFFER xcust FOR customer.
CREATE customer.
  custnum = 111.
FIND xcust WHERE xcust.custnum EQ 111.
DISPLAY xcust.
```

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:

```
DEFINE BUFFER xcust FOR customer.
CREATE customer.
  custnum = 111.
VALIDATE customer. /* or RELEASE customer. */
FIND xcust WHERE xcust.custnum EQ 111.
DISPLAY xcust.
```
The VALIDATE or RELEASE statement causes the DataServer to write the customer 111 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 a MS SQL Server 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:

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

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

```plaintext
FIND customer WHERE customer.custnum EQ 12.
DO TRANSACTION:
    ASSIGN customer.address = "14 Oak Park".
    VALIDATE customer.
    FIND customer WHERE customer.address EQ "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, MS SQL Server handles all of its own locking issues. ABL locking rules are modified when you access information from a MS SQL Server data source. As a result, the OpenEdge phrases NO–LOCK and SHARE–LOCK have isolation-level dependencies. The EXCLUSIVE–LOCK behaves the same in MS SQL Server as in an OpenEdge database.

Table 2–13 provides data source specific comparisons.

<table>
<thead>
<tr>
<th>OpenEdge</th>
<th>Data source</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO–LOCK</td>
<td>Supports the NO–LOCK option in a manner consistent with the OpenEdge database when transaction isolation level is set to read uncommitted.</td>
</tr>
<tr>
<td>SHARE–LOCK</td>
<td>Supports shared locks at the table, page, and record level. However, the scope and duration of the OpenEdge database vs. MS SQL Server shared locks can differ depending on how data source cursors behave at a transaction boundary and how isolation levels are set. The repeatable read isolation level emulates the OpenEdge database SHARE–LOCK behavior most closely. For more information, see your MS SQL Server documentation.</td>
</tr>
<tr>
<td>EXCLUSIVE–LOCK</td>
<td>Supports the EXCLUSIVE–LOCK option in a manner consistent with the OpenEdge database using any available isolation level. However, the MS SQL Server optimizer might produce locks at either the table, page, or the record level.</td>
</tr>
</tbody>
</table>

The DataDirect drivers provide four transaction isolation levels in the following order from least to most restrictive: read uncommitted, read committed, repeatable read, and serializable. In a multi-user configuration, you can isolate users from each other in your data source by setting the isolation level. 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 Microsoft documentation and the MS SQL Server documentation for more information.

**Note:** MS SQL Server might use page-level or table-level locking rather than record-level locking, if its optimizer determines this is the best choice. 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 MS SQL Server documentation for details.

Table 2–14 shows the possible –Dsrv TXN_ISOLATION, n values with the respective meaning.

<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

The default isolation level for the MS SQL Server DataServer is read uncommitted. At this level, a SHARE-LOCK and NO-LOCK are identical from the perspective of the DataServer and the MS SQL Server data source. The higher isolation levels will determine what kind of share locks will take effect. In MS SQL Server, a repeatable read and serializable isolation level are synonymous.

The MS SQL Server DataServer ignores ABL SHARE-LOCK option when used in ABL statements. Instead, share locks are governed by the data source and the available ODBC isolation levels. If you wish to change the share lock behavior, you may be able to do so by changing the isolation level at connection time using the -Dsrv parameter.

When you read records with an ABL statement, regardless of whether you include the SHARE-LOCK option, the MS SQL Server data source typically performs as follows:

- It puts some form of share 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 an ABL statement.

- After the data source reads the record, it releases the share 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 you hold a record with a share lock, other users can usually access that record and apply a share lock, but this is dependent on the isolation level they have selected. Refer to the transaction and locking references in the Microsoft documentation that addresses ODBC programming or data source reference manuals for more information.

Exclusive locks

When you update, delete, or create a record, MS SQL Server puts an exclusive lock on the record; however, the data source does not apply the exclusive lock to a record until all share 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 OpenEdge 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.

Handling lock timeouts

The default behavior for handling a lock timeout condition in the OpenEdge DataServers is to return control immediately to the OpenEdge client. Therefore, the lock wait timeout at the data source is set to zero at the beginning of a client session when using the OpenEdge DataServer for MS SQL Server. This is desirable behavior for clients that want immediate control over lock handling. The client application can choose to handle the lock timeout directly using the NO-WAIT and NO-ERROR keywords. Then, when a record cannot be accessed because it is locked by another user, the application can test for the server timeout condition by testing for TRUE returned from the LOCKED function. The application consumes the timeout condition in this case and is free to perform whatever action is deemed necessary.
If the client application does not specify **NO-WAIT** , then the application automatically loops back to the server in an internal wait mode and retries record access. It continues to do so until the Lock Timeout period set on the client (–lkwtmo parameter specified in seconds) is exceeded. If this wait period is exceeded without being able to successfully access the server resource, the process times out, the wait is canceled and the client raises a stop condition.

When **NO-WAIT** is unspecified, the client consistently returns to the server to retry access to the locked resource. If **NO-ERROR** is also unspecified, then during the Lock Timeout period (-lkwtmo) a resource wait dialog box continues to be displayed to the user. It allows the user to select **cancel** from the dialog to end the wait period. If the user does not cancel and the –lkwtmo period has not been exceeded, the client performs constant retries and multiple round trips to the server. This constant re-cycling, especially during a period of high resource contention, can be normalized by setting a small timeout period on the server in which to handle lock conditions before returning timeouts to the client application. The server wait period is set through the `PRGRS_NATIVE_LOCKWAIT –Dsrv` connection parameter. The disadvantage to setting this parameter to a non-zero value is that the client application is blocked for the timeout period set on the server. This may produce some amount of server-bound latency that should be considered when setting the number of milliseconds for this option. However, if the server is able to complete the resource request in the server timeout period, the resource is returned to the client immediately and the application unblocks. Therefore, the advantage of setting a non-zero server timeout is that the server is given the opportunity to resolve record access without further round trips from the client repeatedly request the same resource. A nonzero value may be especially useful during periods of high contention and may increase the overall efficiency of the DataServer application. Progress recommends a nominal but non-zero setting for the number of milliseconds in most cases. Evaluate your average contention for resources in setting this value for your own application.

The `PRGRS_NATIVE_LOCKWAIT –Dsrv` option permits an application to set a maximum time threshold that the server will block the application waiting to fulfill a resource request that is locked. When the server waits for the resource longer than the `PRGRS_NATIVE_LOCKWAIT` number of milliseconds, control is returned to the client application which then handles the lock condition as described earlier. As the `PRGRS_NATIVE_LOCKWAIT` time is increased, the number of retries from the client within the –lkwtmo period is decreased (assuming **NO-WAIT** is unspecified).

The `PRGRS_NATIVE_LOCKWAIT` setting will affect all transactions for all connections to the foreign data source for a given application session. This includes read-only connections, stored-procedure connections, and transactions on the sequences connection.

**Note:** Setting the `PRGRS_NATIVE_LOCKWAIT` is equivalent to setting the “LOCK_TIMEOUT” value natively in MS SQL Server.
Additional record locking details

In MS SQL Server, if transactions are scoped within other transactions, nested or embedded, all commit/rollback operations on the nested transactions are ignored by SQL server. Not until the transaction nesting levels are reduced to zero are all the nested levels actually committed or rolled back. This is a restriction of the data source with ramifications illustrated by the following code:

```plaintext
DO TRANSACTION:
  RUN STORED-PROC SP.
  CLOSE STORED-PROC SP.
  UPDATE record 2.
  ASSIGN f1d2 = X.
END.
```

If the stored procedure SP has a commit or rollback statement within its TSQL, the commit or rollback isn’t actually executed until the END statement in ABL, when the transaction level returns to zero. If record 2 is also updated by SP, then record 2 will be locked out to an ABL transaction that’s also trying to update it. SP will continue to hold record 2 and lock out an ABL update even if SP explicitly commits the update inside the stored procedure. To avoid this problem, it is necessary to recode the example as follows:

```plaintext
DO TRANSACTION:
  RUN STORED-PROC SP.
  CLOSE STORED-PROC SP.
END.

DO TRANSACTION:
  UPDATE Record 2.
  ASSIGN f1d2 = X.
END.
```

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 OpenEdge and MS SQL Server data source locks differently:

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

Note the following points:

- When you access an OpenEdge database with this procedure, the customer record is share-locked when the first transaction ends.
- When you access a MS SQL Server data source with the DataServer, the customer record is released when the first transaction ends.

This example illustrates how OpenEdge and MS SQL Server data source share locks differ in scope and duration:

```
FIND customer WHERE customer.custnum EQ 10 SHARE-LOCK NO-WAIT NO-ERROR.
IF AVAILABLE customer THEN DO:
  DISPLAY customer.
  PROMPT-FOR customer.
  tx:
  DO TRANSACTION ON ERROR UNDO tx, RETRY tx:
    FIND customer WHERE customer.custnum EQ 10
      EXCLUSIVE-LOCK NO-WAIT NO-ERROR.
    IF LOCKED customer THEN DO:
      MESSAGE "customer locked - retrying".
      UNDO tx, RETRY tx.
    END.
    ELSE
      ASSIGN customer.
    END.
  END.
END.
```

In this example, the first record is only share-locked within the MS SQL Server 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, if the isolation level is read uncommitted or read committed, the record might be updated by another client before the second FIND statement executes. This could cause the record that the second FIND statement fetches to be different from the record fetched by the first FIND statement. This procedure might cause updated information.
to be lost because the procedure applies updates based on the first find of the record, and these updates will overwrite the values from the second find of the record.

Using the DataServer to access a MS SQL Server database 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 a MS SQL Server data source:

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

The record is share-locked when it is fetched. The DataServer upgrades the share 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.

You might have to wait to access a record under 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, OpenEdge 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 MS SQL Server 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 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 the table, page, or record that they want either has an exclusive lock on it, or one of the users needs to put an exclusive lock on it. Neither table, page, or record will give up its lock until the other table, page, or record is available. When a MS SQL Server 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.
- The ABL displays a message that the transaction was killed.
- The system responds as if you had chosen Cancel.

For details on how OpenEdge database locks work, see *OpenEdge Getting Started: ABL Essentials*. See ODBC and MS SQL Server documentation for more information about locks in MS SQL Server.
The NO–WAIT option

The NO–WAIT option works for DataServer applications in the same way that it works for OpenEdge applications: the DataServer uses a time-out mechanism. If MS SQL Server does not return a record, the DataServer considers the record to be locked. It then cancels the request to the MS SQL Server 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, a MS SQL Server data source handles its own transaction roll back and recovery operations. However, the OpenEdge transaction scoping rules apply: a transaction ends when the code exits the outermost block that performs an update. With the DataServer, ABL performs as follows:

- Sends a `COMMIT` to the data source, when a transaction that updates a MS SQL Server data source ends successfully.
- Sends a `ROLLBACK` to the data source if you interrupt the transaction.

See *OpenEdge Getting Started: ABL Essentials* for details on how ABL 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:

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

DO TRANSACTION: /* Transaction 1 */
    FIND st_buf1 WHERE st_buf1.state EQ "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 EQ "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.
Executing the above 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 an error is generated.

When this type of error occurs, ABL run time engine tries to resolve it by working back through the procedure, looking at each block header until the closest block with the error-handling property is found, 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, duplicate-key errors can not be detected until the end of a transaction block. Therefore, if an error occurs in a subtransaction, it is not detected until the end of the entire transaction block and default error handling must be performed for the entire transaction block.

The following example illustrates OpenEdge database and DataServer error handling:

```
rep-blk:
REPEAT:
PROMPT-FOR customer.custnum. /* User input */
FIND customer USING custnum NO-ERROR.
IF AVAILABLE customer THEN
   UPDATE customer.custnum customer.name customer.state. /* User input */
do-blk:
DO ON ERROR UNDO do-blk, RETRY do-blk:
   FIND state WHERE st.state EQ 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>Region</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</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>

Suppose that the user enters an existing state (for example, NH) while ABL is processing the DO block. When this duplicate-key entry occurs for an OpenEdge database, ABL returns control to the DO block, displays a message that the record exists, and rep-prompts the user for a state abbreviation.

However, with the DataServer, if a duplicate key entry occurs in the DO block, ABL returns control to the REPEAT block rather than the DO block. As a result, the procedure rep-prompts the user for a customer number after the inner transaction completes.
If you use NO-ERROR to do your own error handling, you must account for the fact that a MS SQL Server data source creates or updates a record later than an OpenEdge database does. For example, the following code does not trap data source errors, because the requests to perform the operations have not yet been sent to the data source:

```
CREATE customer NO-ERROR.
ASSIGN customer.custnum = 45 NO-ERROR.
ASSIGN customer.name = "Smith" NO-ERROR.
```

The VALIDATE statement causes the DataServer to send requests to your MS SQL Server data source, so incorporate it into your error-handling technique, as in the following example:

```
DEFINE VAR jx AS INTEGER.

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 jx = 1 TO ERROR-STATUS:NUM-MESSAGES:
      MESSAGE "error" ERROR-STATUS:GET-NUMBER(j)
      ERROR-STATUS:GET-MESSAGE (jx).
    END.
    UNDO, LEAVE.
  END.

  ASSIGN customer.name = "Smith" NO-ERROR.
  VALIDATE customer.
  IF ERROR-STATUS:ERROR THEN . . .
END.
```

This code returns data-source errors after the VALIDATE statement.

Another difference in behavior occurs when two users simultaneously attempt to create records with duplicate OpenEdge keys. An OpenEdge database 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 RELEASE or VALIDATE statement when you check for the key’s uniqueness.
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 now 64-bit, promoting potential conflicts when data is exchanged between OpenEdge Release 10.1A and earlier OpenEdge clients, and 10.1B MS SQL Server 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 MS SQL Server will detect an overflow condition and an error message appears on the client machine.
Cursors

A cursor is like a pointer that points to consecutive records in a table. The 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 MS SQL Server 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 a MS SQL Server 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 a MS SQL Server 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:

```sql
DEFINE QUERY x FOR customer SCROLLING.
OPEN QUERY x FOR EACH customer MAX-ROWS 15.
GET LAST x.
DISPLAY customer.custnum.
REPEAT:
    GET PREV x.
    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 MS SQL Server, the first record returned has custnum 15, and the repeat loop displays values in descending order from there. This difference in results is 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.

Considering block cursors and firehose cursors

Block cursors are the default behavior replacing look-ahead cursors for NO-LOCK queries that are processed on the server side. A NO-LOCK query is one that executes with the NO-LOCK lock condition attached to its ABL statement.

Block cursors are the preferred choice to process NO-LOCK queries on the server side. However, server-side cursor processing is not the primary choice overall. Firehose cursors, providing optimum performance for NO-LOCK queries, surpass the benefits of block cursors because they are client-based. For an in-depth presentation of firehose cursors, see the “Firehose and Fast Forward-Only Cursors” section on page 4–5.

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 -Dsrv 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 -Dsrv 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 your 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:

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

In this example, the QUERY-TUNING CACHE-SIZE value overrides the -Dsrv 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 lookahead cursors when PRGRS_BLOCK_CURS is enabled and a server-side cursor is needed. The switch -Dsrv PRGRS_BLOCK_CURS,0 will turn off block cursors in MS SQL Server if you want to turn off the feature for this connection. 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 lookahead cursor is mutually exclusive. In certain circumstances, block cursors will downgrade to lookahead. These are the criteria:

- When -Dsrv 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 2 times the maximum row size. At least 2 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.
Initial Programming Considerations

ABL issues

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

ROWID function

The ROWID data type provides a unique record identifier that is compatible across OpenEdge and MS SQL Server databases. Applications that use the ROWID function behave in the same way whether they access records in an OpenEdge databases or in a MS SQL Server data source. The ROWID function is more consistent than RECID across data sources, since RECID cannot guarantee uniqueness. Progress Software Corporation recommends that you replace the RECID function with ROWID in existing applications.

The DataServer supports the ROWID function for MS SQL Server data source tables that have a unique index. The DataServer utilities use an index that meets this criterion to provide values for the ROWID function. If you build your schema holder using the OpenEdge DB to MS SQL Server utility and you select the Create RECID field option, it is automatically designated as the ROWID index. However, if you do not create the RECID field, you can select a unique index to support ROWID. See the “Defining the ROWID” section on page 7–35 for instructions.

Note: Starting with OpenEdge Release 10.1B, the default external representation of ROWID enables the MS SQL Server DataServer to use a 64-bit integer value for ROWID. Using an integer value to define ROWID is not required, but rather an option to efficiently represent ROWID as a numeric binary value. The MS SQL Server DataServer supports the INT64 data type to emulate ROWID expansion support in this use.

The ROWID value in a MS SQL Server data source differs from the ROWID value in an OpenEdge database if you use the unique index method in the following ways:

- In an OpenEdge database, 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.

- 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 OpenEdge ROWID function fails if you search for an Unknown value (?), because an OpenEdge 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) EQ ?.
  ```
• 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 MS SQL Server data sources:

• Do not try to get a record’s ROWID value before the user assigns values to the unique keys of the record.

• 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 recreated.

• ROWID values are stable for a session, but you cannot rely on them to be the same across sessions.

**Note:** Reposition functions such as REPOSITION-BACKWARDS and REPOSITION-TO-ROW typically use ROWID to identify records. Functions of this type require integer expressions, which can be either INTEGER or INT64.

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 MS SQL Server data source tables that have a unique 4-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 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.

When the Create RECID Field option is selected, the OpenEdge DB to MS SQL Server migration utility creates an indexed column with unique values for each row called PROGRESS_RECID. Starting with OpenEdge Release 10.1B, the field is defined as bigint and in Release 10.1A or earlier, the field is defined as integer. You can also add this column to tables manually if you are using an existing MS SQL Server database or if you ported an OpenEdge database without the Create RECID Field option selected.

If the PROGRESS_RECID field does not exist in the table, the DataServer utility automatically designates the index that meets the unique key criteria. For a complete description of the RECID function, see its reference entry in *OpenEdge Development: ABL Reference*. 
RECID values can be stored in an INTEGER data type. The internal bit size for the INTEGER data type is not expanded from 32-bit to a 64-bit field size. If the RECID value stored in the INTEGER data type does exceed 32 bits however, an overflow condition occurs. See the “Overflow checking” section on page 2–47.

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 2–52 for more information.

**DEFINE BROWSE statement**

The **DEFINE BROWSE** statement relies on a unique record identifier for forward and backward scrolling. If your MS SQL Server data source table does not support the OpenEdge ROWID function (through either a PROGRESS_RECID column or an indexed column with unique values, defined as bigint in OpenEdge Release 10.1B and later or as integer in Release 10.A or earlier), you can write code that explicitly requests the default OpenEdge browse scrolling behavior, similar to this:

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

OPEN QUERY q FOR EACH customer NO-LOCK.
ENABLE upd b WITH FRAME x.
ON CHOOSE OF upd DO:
    iRow = CURRENT-RESULT-ROW("q").
    GET PREV q.
    GET NEXT q EXCLUSIVE-LOCK.
    IF CURRENT-RESULT-ROW("q") EQ 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 a MS SQL Server data source:

```
DEFINE QUERY myquery FOR customer FIELDS (custnum name) SCROLLING.
OPEN QUERY myquery FOR EACH customer NO-LOCK WHERE custnum LT 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:

```abl
FOR EACH customer FIELDS (cust_num 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 postalcode field to process the following query:

```ABL
FOR EACH customer FIELDS (name) WHERE customer.postalcode EQ 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 text or image fields should be explicitly excluded from your field list if possible because MS SQL Server does 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:

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

```abl
FOR EACH customer FIELDS (custnum name)
  WHERE customer.name EQ "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 section provides 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 the MS SQL Server data source, the cursor behaves as if the failed `FIND` had never occurred. To support these statements, a table must include support for the OpenEdge `ROWID` function (through either a `PROGRESS_RECID` column or an indexed `NUMBER` column with unique values). See the “`ROWID function`” section on page 2–52 for more information.

For example, the procedure `find.p` accesses OpenEdge and MS SQL Server databases using the same `FIND` and `FIND PREV` statements in each case, as shown:

```abl
/* This code accesses an OpenEdge database. */
FIND demo.customer WHERE demo.customer.custnum EQ 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 EQ 3.
FIND PREV mssdemo.customer.
DISPLAY mssdemo.customer.custnum
  LABEL "MSS DATA SOURCE RESULT" WITH COL 29.
```
When you run `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):

<table>
<thead>
<tr>
<th>OPENEDGE DATABASE RESULT</th>
<th>MSS DATA SOURCE RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

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

**Compiling OpenEdge procedures**

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

To compile procedures that access a MS SQL Server data source, start up an OpenEdge session and connect to the schema holder using the schema holder’s logical database name, then use the `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 “COMPILE Statement” reference entry in *OpenEdge Development: ABL Reference*.

**Note:** You do not have to connect to the MS SQL Server database to compile a procedure. The schema holder contains all the information that the compiler requires.

**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 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 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 sports will not run with a database named mysports.
- Code that you compile for MS SQL Server will not run on an OpenEdge database.

See *OpenEdge Deployment: Managing ABL Applications* for more details on r-code portability.
The size of r-code grows significantly when you compile procedures against a MS SQL Server data source as compared to compiling against an OpenEdge database. The r-code for a DataServer application contains as text 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 a MS SQL Server data source than when returning data from an OpenEdge database. The data source might not sort data like the OpenEdge database does, so the order in which records return might differ. In addition, the server-side result set is limited to a single record to improve performance.

**Features and language differences**

The DataServer supplies you with complete ABL functionality when accessing MS SQL Server data sources. Many 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 MS SQL Server data source does not support. For the DataServer for MS SQL Server, DBRESTRICTIONS can return SETUSERID and COUNT–OF. In certain circumstances, it can return additional values. For example, it can return READ–ONLY if you connect to the schema holder in read-only mode. See the “DBRESTRICTIONS function” reference entry in OpenEdge Development: ABL Reference for information on syntax.

Table 2–15 summarizes ABL differences between OpenEdge databases and MS SQL Server data sources.

**Table 2–15: ABL differences between OpenEdge Databases and MS SQL Server data sources**

<table>
<thead>
<tr>
<th>OpenEdge feature</th>
<th>MS SQL Server data source difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTAINS operator</td>
<td>This operator relates to word indexing, which the DataServer does not support.</td>
</tr>
<tr>
<td>COUNT–OF function</td>
<td>The DataServer does not support this function.</td>
</tr>
<tr>
<td>CREATE statement</td>
<td>Records that you create after opening a cursor might be invisible to that cursor.</td>
</tr>
<tr>
<td>FIND statements</td>
<td>To reduce the number of records included in the results set, qualify your FIND statements and queries with a WHERE clause.</td>
</tr>
<tr>
<td>FOR EACH statements OPEN QUERY statement</td>
<td></td>
</tr>
<tr>
<td>MATCHES function</td>
<td>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.</td>
</tr>
<tr>
<td>BEGINS function</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** It is theoretically possible to do this with an OpenEdge database, but using this kind of criteria results in poor performance.
An OpenEdge EXCLUSIVE-LOCK is emulated in the DataServer. An OpenEdge NO-LOCK can be emulated in the MS SQL Server database 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 2–36 for more information.

### Table 2–15: ABL differences between OpenEdge Databases and MS SQL Server data sources

<table>
<thead>
<tr>
<th>OpenEdge feature</th>
<th>MS SQL Server data source difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO-LOCK option¹</td>
<td>An OpenEdge EXCLUSIVE-LOCK is emulated in the DataServer. An OpenEdge NO-LOCK can be emulated in the MS SQL Server database when the isolation level is set to read-uncommitted.</td>
</tr>
<tr>
<td>SHARE-LOCK option</td>
<td>SHARE-LOCK behavior is data source and isolation-level dependent. See the “Data source record locking” section on page 2–36 for more information.</td>
</tr>
<tr>
<td>EXCLUSIVE-LOCK option</td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td>MS SQL Server NULL = Unknown value (?)</td>
</tr>
<tr>
<td></td>
<td>OpenEdge empty string (&quot;&quot;) = a one space string in MS SQL Server.</td>
</tr>
<tr>
<td>Record creation</td>
<td>A record is created at the end of a record’s scope and not when the required index information becomes available.</td>
</tr>
<tr>
<td>SESSION:TIME-SOURCE handle</td>
<td>This system handle returns the MS SQL Server’s server time information.</td>
</tr>
<tr>
<td>SETUSERID function</td>
<td>You cannot use this function to change the login name and password.</td>
</tr>
</tbody>
</table>

¹. For more information, see the “Data source record locking” section on page 2–36.
RDBMS stored procedures

Relational database management system (RDBMS) stored procedures can be an efficient, productive approach to obtain data from a foreign data source. In general, using stored procedures can help reduce network traffic, promote better performance, and improve response time because the server-side processing of the stored procedure can be accomplished without communicating with the client; once this processing is completed, the server returns only the data results to the requesting client.

See Chapter 3, “RDBMS Stored Procedure Details,” for a complete discussion of various RDBMS stored procedure and send-sql- statement techniques and enhancements you can use to run against the MS SQL Server DataServer.
This chapter defines relational database management system (RDBMS) stored procedures and describes how to create and implement them in the OpenEdge environment. It discusses various RDBMS stored procedure and send-sql-statement techniques you can employ. It also details how to execute an RDBMS stored procedure or send-sql-statement against a MS SQL Server and load its result sets directly into temp-tables.

This chapter contains the following sections:

- Overview
- RDBMS stored procedure basics
- Run Stored-Procedure details
- Data output and retrieval options
- Interfacing with RDBMS stored procedures
- Handling errors
- ROWID Support
Overview

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:** This chapter uses the phrase RDBMS stored procedure and stored procedure interchangeably.

Enhancements to stored procedure techniques also allow you to extend your code in new ways and effectively employ more ABL features and functionality. For example, you can retrieve data from a foreign data source through a stored procedure and load this result set into a temp-table. Also, this technique is especially useful if you elect to further manage and manipulate your result set using ProDataSet capabilities.

Stored procedures can allow you to improve your dataserver’s performance because they utilize the foreign data source’s native capabilities. Stored procedures can be an efficient, productive approach to obtain data from a foreign data source or execute native business rules and procedures. In general, using stored procedures can help reduce network traffic, promote better performance, and improve response time because the server-side processing of the stored procedure can be accomplished without sustained communication with the client; once this processing is completed, the server returns the data results to the requesting client.

The first task to effectively employ any stored procedure technique or approach is to determine what information you want to receive or process you want to run. Once you know your data requirements, you can proceed to define the stored procedure details.

**Note:** A complete discussion of creating and using native stored procedures is beyond the scope of this chapter. For this type of information, see your Microsoft SQL Server documentation.

**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 the precompiled procedure, assuming it is still cached in the database. This makes access to the database quicker and more efficient than when you access it with new queries each time.

The DataServer allows you to use ABL to run native MS SQL Server stored procedures. All pre-defined stored procedures initiated on behalf of the MS SQL Server are executed from within ABL, using the `RUN STORED-PROCEDURE` statement. You define specific ABL language elements to the `RUN STORED-PROCEDURE` statement to match the profile or signature of your native stored procedure. You can also provide additional ABL statements subsequent to the `RUN STORED-PROCEDURE` statement to handle result sets from stored procedures.
RDBMS stored procedure basics

In the OpenEdge environment, you can think of a stored procedure definition as having two basic, interrelated parts:

- **Execution controls to run and close a store procedure** — Comprises the information needed to execute a stored procedure request against the MS S data source. At a minimum, all stored procedures discussed in this guide are assessable using the `RUN STORED-PROCEDURE` statement.

- **Language elements that enable access to specific data results** — Qualifies the retrieved data, or *result sets*, that the stored procedure’s execution returns. Various keywords, phrases, statements, and syntax elements support different retrieval options for stored procedure output. This part of the stored procedure implementation reflects your analysis of your data needs; based on this analysis, you determine the additional syntax elements you need to define the output elements and data results you want retrieved.

Table 3–1 identifies and briefly introduces the elements that comprise a stored procedure definition; each of these elements is also more fully discussed later in this chapter.

### Table 3–1: Stored procedure language elements

<table>
<thead>
<tr>
<th>ABL language element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN STORED-PROCEDURE statement</td>
<td>Executes a stored procedure</td>
</tr>
<tr>
<td>PROC-HANDLE phrase</td>
<td>Allows you to specify a handle to identify a stored procedure</td>
</tr>
<tr>
<td>PROC-STATUS phrase</td>
<td>Reads the return value</td>
</tr>
<tr>
<td>LOAD-RESULT-INTO phrase</td>
<td>Allows data from a result set that is returned for a foreign data source to be put into one or more temp-tables</td>
</tr>
<tr>
<td>PARAM phrase</td>
<td>Identifies run-time parameters to be passed to and/or from the stored procedure</td>
</tr>
<tr>
<td>CLOSE STORED-PROCEDURE statement</td>
<td>Enables the values to be retrieved from the output parameters that you defined for the stored procedure, finalizes result sets data processing, and tells OpenEdge that the stored procedure has ended</td>
</tr>
</tbody>
</table>

**Note:** You can substitute the abbreviations RUN STORED-PROC and CLOSE STORED-PROC for the full names RUN STORED-PROCEDURE and CLOSE STORED-PROCEDURE, respectively. The remainder of this guide generally uses the abbreviated form.

See the “Run Stored-Procedure details” section on page 3–6 for more details about the reference entries presented in Table 3–1.
As previously noted in Table 3–1, you can pass data types in the RUN STORED-PROCEDURE statement using the PARAM phrase. Table 3–2 lists issues that occur when you pass certain data types as parameters.

Table 3–2: Argument data types for stored procedures

<table>
<thead>
<tr>
<th>OpenEdge</th>
<th>MS SQL Server data source</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECIMAL</td>
<td>The DataServer converts each of these data types in the schema image to the equivalent default OpenEdge data type as follows:</td>
</tr>
<tr>
<td>INTEGER</td>
<td>• DECIMAL=DECIMAL(default), or Float</td>
</tr>
<tr>
<td>INT64</td>
<td>• INTEGER=INTEGER</td>
</tr>
<tr>
<td>INT64</td>
<td>• INT64=BIGINT</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 the VARCHAR parameter exceeds this limit, it causes an 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 DATEPART() or DATEDIFF() in the Transact-SQL of your native stored procedure to isolate parts of the date structure for which you might want to test.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>You can specify a DATETIME data type in a temp table used to receive results from a stored procedure using the LOAD-RESULT-INTO phrase.</td>
</tr>
</tbody>
</table>

Note these stored procedure points:

- Input and output parameters are displayed as fields.

- Stored procedures called from within OpenEdge applications cannot return Boolean values to LOGICAL data types.

- If you are running several stored procedures, run them serially and process all the results from one stored procedure and close the procedure before you run a second one. By default, the DataServer allows one active request for running a stored procedure. It is not necessary to specify the PROC-HANDLE phrase when procedures are run serially.

When you run stored procedures concurrently, the DataServer uses one connection to the data source per procedure. If different stored procedures attempt to update the same record from a single client’s requests, the connections could block each other or a deadlock might occur.

Notes: You must define a PROC-HANDLE phrase for each stored procedure phrase that is simultaneously active. This technique provides a CLOSE STORED-PROC statement that can identify the targeted open procedure and close it.

In contrast, since a stored procedure executed with the LOAD-RESULT-INTO phrase implicitly closes the procedure once the execution ends and the data retrieved is placed into temp tables, it essentially runs serially and has no use for a PROC-HANDLE phrase.

- When you create or update your schema image, the stored procedures appear in the list of accessible objects along with tables, view, and sequences. OpenEdge allows you to run the stored procedures that you create in data sources using the procedure definitions in your
schema image. See your Microsoft SQL Server documentation for complete information about creating and using stored procedures.

- If the ABL that executes a stored procedure is already within a transaction block, the stored procedure becomes an extension of that transaction and will not commit to the database until the ABL transaction is completed. However, because the stored procedure does not execute as part of ABL client process, it cannot be rolled back by ABL.

- The DataServer cannot roll back sub-transactions in the stored-procedure context since it has no control over what the stored procedure executes.

The following section expands on the use of the RUN STORED-PROC statement.
Run Stored-Procedure details

This section provides:

- A complete, detailed reference for all the syntax expressions and options you can use to define the **RUN STORED-PROCEDURE** and the associated language elements that comprise a stored procedure definition.

- Syntax to use the **RUN STORED-PROCEDURE** statement with and without the **LOAD-RESULT-INTO** phrase.

- Detailed information about using the **RUN STORED-PROCEDURE** statement with the send-sql option. Examples to use the **RUN STORED-PROCEDURE** statement with and without the **LOAD-RESULT-INTO** phrase are provided.

Complete syntax reference for Run Stored-Procedure

The following syntax shows the **RUN STORED-PROC** statement and all options that can be used with the statement:

**Syntax**

```plaintext
RUN STORED-PROCEDURE procedure-name
[[[LOAD-RESULT-INTO <handle> [<int> = PROC-STATUS]] |
 [ <int> = PROC-HANDLE ]]
[ NO-ERROR ]
[[[INPUT | OUTPUT | INPUT OUTPUT] [PARAM parameter-name =] expression, ...]
[[INPUT | OUTPUT | INPUT OUTPUT] [PARAM parameter-name =] expression ]]
```

However, keep in mind that not all options shown can be used simultaneously.

**Note:** In the **RUN STORED-PROCEDURE** syntax, note that the `<handle>` is either a pure handle or a handle that points to a temp-table. For more information about the use of temp-table handle with the **LOAD-RESULT-INTO** phrase, see the “**LOAD-RESULT-INTO phrase**” section on page 3–7.

**Syntax**

```plaintext
CLOSE STORED-PROCEDURE procedure-name
[ integer-field = PROC-STATUS ]
[ WHERE PROC-HANDLE = integer-field ]
```

The **CLOSE STORED-PROCEDURE** is not used when the **LOAD-RESULT-INTO** keyword is used with the **RUN-STORED-PROCEDURE** statement. See the “Closed Stored-Procedure statement” section on page 3–9.

For valid combinations and usage of these syntax elements see the:

- “Run Stored-Proc statement execution without the **LOAD-RESULT-INTO** phrase” section on page 3–10
Run Stored-Procedure statement

The RUN STORED-PROC statement runs a RDBMS stored procedure or allows you to send Transact-SQL to a MS SQL Server based data source using an OpenEdge DataServer. It contains a procedure-name which is either the:

- Name of the RDBMS stored procedure that you want to run
- OpenEdge built-in procedure name, send-sql-statement, to send Transact-SQL to a MS SQL Server based data source

Note: If your ABL procedure is in a transaction block when a RUN STORED-PROCEDURE statement is executed, the RDBMS stored procedure runs within the same RDBMS transaction, even if your stored procedure declares its own transaction levels.

LOAD-RESULT-INTO phrase

The LOAD-RESULT-INTO function loads the result sets into a temp-table which is represented by a handle variable. Note that handle can also be defined as extent to enable you to pass more than one temp-table handle in those instances where SQL statement(s) are defined to return more than one result set.

When used with the RUN STORED-PROC statement or stored procedure to load result sets into temp-tables, this function carries an implicit CLOSE-STORED PROCEDURE statement.

Note: The compiler issues an error at runtime if the variable of type handle with the LOAD-RESULT-INTO function does not point to a temp-table.

For additional details about using the LOAD-RESULT-INTO phrase with the temp-table handle, see the “Loading result sets into temp-tables” section on page 3–21.

When used with the LOAD-RESULT-INTO phrase, the temp-table handle identifies the temp-table to which the result set will be loaded.

You can specify an array of one or more temp-table handle elements to retrieve stored procedure result sets and have the DataServer load the result set data directly into the associated temp-table(s). This approach allows you to have direct ABL access to the fields defined in the temp-table.
The following types of temp-tables can support result sets:

- **Static** — A temp-table whose schema is defined at compile time.
- **Dynamic** — A temp-table whose schema is defined at run time. There are two types of dynamic temp-tables: dynamic-prepared and dynamic-unprepared.

For additional details about using the `LOAD-RESULT-INTO` phrase with the temp-table handle, see the “Loading result sets into temp-tables” section on page 3–21.

**PROC-STATUS phrase**

The `PROC-STATUS` phase returns the return status from a MS SQL Server stored procedure. The *return status* is an integer value that typically indicates whether a stored procedure succeeded or failed; if a stored procedure fails, a code indicates why it failed. See your Microsoft SQL Server documentation for descriptions of the possible values for the return status.

**PROC-HANDLE phrase**

The `PROC-HANDLE` phrase allows you to specify a handle to act as a unique identifier for a MS SQL DataServer stored procedure. For example, the `PROC-HANDLE` assigns a value to the specified integer field or variable *(integer-field)* that uniquely identifies the stored procedure that is returning results from MS SQL Server.

Note these additional points about the `PROC-HANDLE`:

- 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 OpenEdge application.

**NO-ERROR option**

The NO-ERROR option specifies that any ERROR condition that the `RUN STORED-PROCEDURE` statement produces is 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.

**Note:** This option must appear before any runtime parameter list.

**PARAM phrase**

The Param phrase identifies a run-time parameter to be passed to the stored procedure. A *parameter* has the following syntax:

**Syntax**

```plaintext
[C ( [INPUT | OUTPUT | INPUT-OUTPUT ] [ PARAM parameter-name = ] expression, ... ]
[ 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.

**Note:** When you run send-sql-statement for a MS SQL-based data source, it passes 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.

### Closed Stored-Procedure statement

For a RDBMS stored procedure, a PROC-STATUS received value indicates that the procedure has completed execution and makes available the return code value and any output parameters. For a send-sql-statement stored procedure, CLOSE STORED-PROCEDURE closes the SQL cursor used by the procedure, as shown:

#### Syntax

```
CLOSE STORED-PROCEDURE procedure-name
  [ 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.

Consider these points concerning the CLOSED STORED-PROCEDURE statement:

- 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.
Run Stored-Proc statement execution without the LOAD-RESULT-INTO phrase

To implement most data retrieval options associated with a stored procedure, (excluding those that involve loading retrieved data into temp-tables) you must define the RUN STORED-PROC statement and explicitly end the stored procedure using the CLOSE STORED-PROC statement.

You must define the RUN STORED-PROC and explicitly complete access to the stored procedure OUTPUT with the CLOSE STORED-PROC syntax expressions for a stored procedure implementation that retrieves the following types of data from the foreign data source:

- proc-text-buffer
- user-defined views

For more information about these data retrieval options, see the “Data output and retrieval options” section on page 3–13.

**Note:** You can also implicitly or explicitly close a stored procedure to retrieve these types of data: return code or output parameters. For more information about each of these data retrieval options, see the “Data output and retrieval options” section on page 3–13.

The syntax details presented in this section describe how to run and close a stored procedure in the MS SQL Server data source.

This is the syntax for the RUN STORED-PROC statement to execute an RDBMS stored procedure **without** the LOAD-RESULT-INTO phrase:

**Syntax**

```
RUN STORED-PROC procedure [<int> = PROC-HANDLE]
[ NO-ERROR ]
[ INPUT | OUTPUT | INPUT OUTPUT ]
[ PARAM parameter-name = ] expression, ...
[ INPUT | OUTPUT | INPUT OUTPUT ]
[ PARAM parameter-name = ] expression )
```

This type of syntax expression requires the explicit CLOSE STORED-PROC statement. This is the partial syntax for the CLOSE STORED-PROC statement:

**Syntax**

```
CLOSE STORED-PROC procedure
[ integer-field = PROC-STATUS ]
[ WHERE PROC-HANDLE = integer-field ]
```
Run Stored-Proc statement execution *with the LOAD-RESULT-INTO phrase*

You can also execute a stored procedure implementation to obtain result sets and load the results into temp-tables. You must define result set(s) target as part of the RUN STORED-PROC statement. You do not explicitly close a STORED-PROC defined this way as you do with buffer methods of retrieving result sets; the LOAD-RESULT-INTO function achieves this goal implicitly.

This is the syntax for the RUN STORED-PROC statement implementation when loading result sets to temp-tables:

**Syntax**

```
RUN STORED-PROC  procedure-name
[[ [LOAD-RESULT-INTO <handle>  [<int> = PROC-STATUS]]
 [NO-ERROR]]
 [INPUT | OUTPUT | INPUT OUTPUT] [PARAM parameter-name = expression, ...]
 [INPUT | OUTPUT | INPUT OUTPUT] [PARAM parameter-name = expression)]
```

The RUN STORED-PROC statement sets up the execution of the stored-procedure, retrieves the result set(s), and loads it into the temp-table(s) provided you use the LOAD-RESULT-INTO function. The CLOSE STORED-PROC statement is automatically applied when the result set(s) is loaded into temp-tables because using temp-tables must consume all result sets of the procedure.

Run Stored-Proc statement execution using the send-sql-statement option

ABL also allows you to use stored-procedure syntax to send SQL statements and their native language extensions 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. This option gives you access to Transact-SQL, providing you access to business logic for MS SQL Server. For example, you can issue Data Definition Language (DDL) statements from within OpenEdge procedures. You can send multiple SQL statements by concatenating them and passing the string as a single parameter to the send-sql-statement option.

Like the option to define a RUN STORED-PROC statement either with or without the LOAD-RESULT-INTO options, you can define a send-sql-statement with or without the LOAD-RESULT-INTO option. The following section presents and briefly describes a sample use of each technique.

**Without the Load-Result-Into option**

You use the RUN STORED-PROC statement with the send-sql-statement option and pass the Transact-SQL statements as a parameter. The syntax of the statement must be valid Transact-SQL syntax. Example 3–1 shows how this code passes a SELECT statement as a parameter.

**Note:** The SQL statement(s) passed to the send-sql-statement stored procedure is not limited to the 32,000 bytes of a OpenEdge CHARACTER field. To pass SQL of greater size, store the SQL text in a LONGCHAR field and pass that as a parameter to the send-sql-statement stored procedure.
Example 3–1: Passing a SELECT statement as a parameter

Example 3–1 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 OpenEdge procedure. You can read the results into the proc-text-buffer defined by OpenEdge as shown in the example above. Alternatively, you can define your own buffer from within your data source that can accept other data types as well as the CHARACTER data type.

With the Load-Result-Into option

Example 3–2 shows how to use the send-sql-statement with the LOAD-RESULT-INTO option. It also shows that the PROC-STATUS phrase must be defined as part of the RUN STORED-PROC statement because of the implicit CLOSE STORED-PROC that is associated with the LOAD-RESULT-INTO phrase.

Example 3–2: Using the send-sql-statement with the LOAD-RESULT-INTO option

Also note in Example 3–2 that the PROC-STATUS phrase does not need an associated PROC-HANDLE phrase to close the associated procedure because it is retrieved using the RUN STORED-PROC statement; although the PROC-HANDLE is typically used after the execution of the RUN STORED-PROC statement, it is not needed in this context because of the implicit procedure close.

| DEFINE VARIABLE handle1 AS INTEGER NO-UNDO. |
| RUN STORED-PROC send-sql-statement handle1 = PROC-HANDLE |
| ("SELECT name, address, city, state, postal_code |
| FROM customer WHERE credit_limit >= 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. |

| DEFINE VARIABLE res AS INTEGER NO-UNDO. |
| DEFINE VARIABLE tthndl AS HANDLE NO-UNDO. |
| CREATE TEMP-TABLE tthndl. |
| RUN STORED-PROCEDURE send-sql-statement LOAD-RESULT-INTO tthndl |
| res = PROC-STATUS ("SELECT * FROM customer"). |
| IF res THEN |
| DISPLAY "succeeded". |
| END. |
| DISPLAY stat. |
Data output and retrieval options

This section briefly highlights the data retrieval options that you can initiate through the execution of a stored procedure. See the “Interfacing with RDBMS stored procedures” section on page 3–15 for more information and examples of each option.

Return codes

This value might be a success code or a value returned by the stored procedure (as defined by the data source).

Values of output parameters defined when creating a procedure

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; your selected INPUT and OUTPUT parameter options must match the corresponding signature of the stored procedure at the data source.

Results retrieved from a database

This section identifies and briefly defines three coding techniques that ABL supports to retrieve data source results. In general, you can think of these techniques as handling complex query results for which each technique has its own purpose and benefits.

Using the OpenEdge-supplied proc-text-buffer for row results

This technique:

- Allows you to access result sets available through a pseudo table for which each row is a long character string
- Requires you to parse each long string to access specific fields

The Proc-text-buffer is part of the schema holder, making this result set option independent of the foreign data source.

Defining a special view on the MS SQL Server data source to use as a buffer for row results

This technique allows you to:

- Define views within the foreign data source
- Use the result set buffer configuration of the view to access the data returned by the stored procedure

Defining of a special view involves some administration on the foreign data source and therefore is considered a result set option that is dependent on the foreign data source.
Loading results into a temp-table

This technique allows you to:

- Specify one or more temp-table handle elements to retrieve stored procedure result sets and have the DataServer load the result set data directly into the associated temp-tables
- Have direct access to the fields defined in the temp-table

The temp-table approach maintains some separation between the data and the foreign data source which allows you to process the result sets quite independent of its database origin. Also, the result set definition is not contained within the schema holder.

Unless you pass unprepared dynamic temp-table handles into which the result sets are to be loaded, you still need to ensure that the temp-table record definition matches that of the stored procedures result set(s).

Passing empty temp-tables with the LOAD-RESULT-INTO phrase generates temp-table definitions dynamically based on a default mapping of data types. This method provides a data retrieval method completely independent of the foreign data source.

The LOAD-RESULT-INTO method represents a highly efficient means to transfer stored procedure result sets into temp-tables and provides significant performance gains over other techniques used to accomplish this same goal.

**Note:** The proc-text-buffer and the special native views techniques use the basic RUN STORED-PROC statement and are backward compatible to earlier versions of ABL. The LOAD-RESULT-INTO phrase provides a new technique to populate temp-tables directly in ABL.
Interfacing with RDBMS stored procedures

As previously mentioned, the RUN STORED-PROCEDURE statement is the initial statement required for the execution of all stored procedures you define and initiate through OpenEdge. It can run an RDBMS stored procedure bound by the data source, or allow you to send SQL to a MS SQL Server data source using an OpenEdge DataServer. The specific, additional keywords and elements you must include in a stored procedure or define for a send-sql-statement depend on the stored procedure characteristics including its signature, data results, and methods chosen to retrieve output to an ABL procedure.

This section identifies and describes the syntax elements, keywords, and other requirements associated with obtaining stored procedure output from a stored procedure data request.

Example 3–3 shows a stored procedure with multiple result sets, output parameters, and a return value.

```sql
/* pcust stored procedure*/
CREATE PROCEDURE pcust (@num INT, @orders INT OUT, @states INT OUT) AS
BEGIN
SELECT customer.cust_num, customer.name, order_num FROM customer, order_
WHERE customer.cust_num = order_.cust_num AND customer.cust_num > @num
SELECT @orders = @@rowcount
SELECT cust_num, state.state FROM customer, state WHERE
    customer.state = state.state AND customer.cust_num > @num
SELECT @states = @@rowcount
END
RETURN 0
```

Example 3–3: A stored procedure with multiple result sets, output parameters, and a single return value

This Transact-SQL code creates the stored procedure pcust and defines three parameters: num, which is an input parameter (the default parameter type), 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 associated with a record buffer defined by the procedure definition in the OpenEdge schema image; 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, it is stored as 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.

Retrieving return codes

You can create a stored procedure that provides return code information. For example, it might indicate whether the stored procedure was successful or whether it encountered an error condition.

Example 3–4 shows how 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.
Example 3–4: Return status example

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.

Example 3–5, an ABL procedure, uses the second option for passing parameters—it passes them by name with the PARAM option.

Example 3–5: Passing parameters by name using the PARAM option

Retrieving result sets using proc-text-buffer

The proc-text-buffer technique offers one approach to access results returned from a data source. The following 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 reference entry that discusses the DEFINE BUFFER statement in *OpenEdge Development: ABL Reference*.
Technique to use proc-text-buffer

Example 3–6 shows the results of the first ABL stored procedure, pcust, which is Example 3–3 as presented earlier in this section.

```abl
/* pcust stored procedure */
DEFINE BUFFER cust-state-join-buffer FOR proc-text-buffer.
RUN STORED-PROC pcust (20, output 0, output 0).
FOR EACH cust-state-join-buffer:
  DISPLAY cust-state-join-buffer.
END.
CLOSE STORED-PROC pcust.
DISPLAY pcust.orders pcust.states.
```

Example 3–6: pcust stored procedure

The OpenEdge-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.

Example 3–7 illustrates returning database results into the proc-text-buffer and converting the results to the INTEGER data type.

```abl
DEFINE VARIABLE handle1 AS INTEGER NO-UNDO.
DEFINE VARIABLE iMax as INTEGER NO-UNDO.
RUN STORED-PROC send-sql-statement handle1 = PROC-HANDLE
  ("SELECT max (cust_num) 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.
```

Example 3–7: Returning database results into the proc-text-buffer and results conversion

The DataServer passes the SQL statement directly to the MS SQL data source. The ABL does not process it, so errors occur only at run time and not when you compile a procedure.

Note: The ABL QUOTER function can be useful for quoting values or handling embedded quotes when building SQL statements that will be passed into stored procedures with the send-sql-statement option.

Assessing result sets obtained from the proc-text-buffer technique

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, and converts it to the character equivalent. Another benefit of the proc-text-buffer is that it can be used to hold 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 results set record form one result set at a time.
However, a disadvantage in using the `proc-text-buffer` technique is that it is much more difficult to manipulate the data after you receive it as it requires parsing the data. 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.

## Defining a view to use as a buffer

As an alternative to using the OpenEdge buffer `proc-text-buffer` definition, you can define a view in the data source that can serve as a buffer allowing you to retrieve database results in their original data types. However, keep in mind that using views creates a database dependency beyond the stored procedure itself.

### Technique to define a view to use as a buffer

While a stored procedure can include multiple SQL statements, a buffer that you define contains the format of only a single results set. You need to define multiple views to accommodate multiple results sets.

### To define a buffer:

1. Define a view in the MS SQL Server data source with the following characteristics:
   - The naming convention `_BUFFER_buffername`
   - 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 a SQL utility to define the following view in the data source:

   ```sql
   CREATE VIEW _BUFFER_custlist AS SELECT customer.cust_num, 
   customer.name FROM customer WHERE 1 = 0
   GO
   ```

   Notice that these views are defined to ensure that they never return any results. This helps to indicate that the purpose of the view is its buffer content and not its SQL capabilities. 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 OpenEdge can use. (See the “Updating a schema holder” section on page 7–11 for instructions on using this utility.)

### Assessing result sets obtained by defining a view as buffer technique

The buffer in the previous procedure 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 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_helptext to view the stored procedure from a MS SQL Server, or view procedures in the system tables.

The examples in this section do not use the supplied proc-text-buffer buffer. Instead, they show how to define formatted buffers by creating views in the data source, using the following syntax:

**Syntax**

```
CREATE VIEW _BUFFER_buffer-name
```

Example 3–8 and Example 3–9 show the views created in your MS SQL Server 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.cust_num, customer.name, order_num FROM customer, order_ WHERE 1 = 0
GO
```

**Example 3–8: First view created in your MS SQL Server data source**

```
CREATE VIEW _BUFFER_pcust_states AS SELECT cust_num, state.state FROM customer, state WHERE 1 = 0
GO
```

**Example 3–9: Second view created in your MS SQL Server data source**

Example 3–10 shows ABL procedure results of the previous stored procedure pcust as it is written into the new buffers pcust_orders and pcust_states.

```
/* Results of the first two example code examples - 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.
```

**Example 3–10: Result set of pcust - typed buffers**

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 OpenEdge-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 ABL 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.
Example 3–11 accesses the stored procedure pcust twice; procedure handles (through the PROC–HANDLE function) identify the different results from your data source.

```/* 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.
```

**Example 3–11: Procedure handles**

The results look the same as in the first two examples. However, because you are running a stored procedure twice, ABL uses the procedure handles to identify the different instances. If you have more than one stored procedure open simultaneously in your application, you must explicitly define procedure handles for each.

Example 3–12 shows how to use standard OpenEdge syntax to join the results from a stored procedure with other tables in the database.

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

**Example 3–12: Joining stored procedure results with other database tables**

Example 3–12 joins the order information returned from the stored procedure with the orderline information in the same database.
Loading result sets into temp-tables

Enhancements implemented through changes to the RUN STORED-PROC statement allow you to retrieve result sets from a foreign data source and load each result set, for which a temp-table handle is defined, into its own temp-table. The LOAD_RESULT-INTO function enables data retrieved to be loaded into temp-tables where the data can then be manipulated, employing all characteristics inherent to temp-tables. The capability to load result sets into temp-tables is not limited by the parsing requirements associated with the proc-text-buffer nor the database dependencies associated with views.

Temp-tables can provide data management capabilities associated with ABL directly to the result sets of a stored procedure, but completely independent of the foreign data source from which it was populated and/or derived. Temporary tables are effectively database tables in which OpenEdge stores data temporarily. Because temp-tables have the same support features that actual OpenEdge databases use, you can take advantage of almost all the OpenEdge database features that do not require data persistence and multi-user access. For example, you can define indexes for fields in the temp-table. For more information about temp-tables, see OpenEdge Getting Started: ABL Essentials.

Example 3–13 shows how to use the RUN STORED-PROC statement with the LOAD-RESULT-INTO phrase with a single dynamic temp-table. It highlights the coding techniques discussed in the “Run Stored-Proc statement execution using the send-sql-statement option” section on page 3–11 and it introduces the dynamic temp-table topic further discussed in this section.

```abl
DEFINE VARIABLE bhCustomer AS HANDLE NO-UNDO.
DEFINE VARIABLE hBrowse AS HANDLE NO-UNDO.
DEFINE VARIABLE hQuery AS HANDLE NO-UNDO.
DEFINE VARIABLE ttCustomer AS HANDLE NO-UNDO.

DEFINE FRAME BrowseFrame WITH SIZE 80 BY 10.

CREATE TEMP-TABLE ttCustomer.

RUN STORED-PROC send-sql-statement LOAD-RESULT-INTO ttCustomer
("SELECT * from CUSTOMER").

bhCustomer = ttCustomer:DEFAULT-BUFFER-HANDLE.
CREATE QUERY hQuery.
hQuery:SET-BUFFERS (bhCustomer).
DISPLAY ttCustomer:Name.
hQuery:QUERY-PREPARE("FOR EACH" + ttCustomer:Name).
hQuery:QUERY-OPEN.

CREATE BROWSE hBrowse ASSIGN
  ROW  = 1
  COL  = 1
  WIDTH = 79
  DOWN = 10
  FRAME = FRAME BrowseFrame:HANDLE
  QUERY = hQuery
  SENSITIVE = TRUE
  SEPARATORS = TRUE
  ROW-MARKERS = FALSE
  VISIBLE = TRUE.

hBrowse:ADD-COLUMNS-FROM(bhCustomer).
ENABLE ALL WITH FRAME BrowseFrame.
WAIT-FOR CLOSE OF CURRENT-WINDOW.
```

Example 3–13: RUN STORED-PROC statement with the LOAD-RESULT-INTO phrase
Getting started

If you are using static temp-tables, you must define the temp-table layout in your program to accommodate a specific stored procedure result set before attempting to populate these tables. Once this prerequisite task is done, however, temp-tables can also be automatically populated, offering a potential performance gain in most instances.

Unlike the proc-text-buffer technique, you do not have to parse the strings from the proc-text-buffer pseudo table buffer where each row is a character string. Similarly, you do not need to perform any administration to maintain views in the foreign data source or their definitions in the schema holder. For more details about planning your temp-table layout, see the “Creating a temp-table layout plan” section on page 3–23.

Employing additional enhancements

The temp-table technique offers even greater programming benefits as it extends both send-sql-statements and stored procedures through the result processing techniques previously described in this chapter. For example, by mapping the PROGRESS_RECID to the ROWID field in temp-tables, you can easily support KEY definitions required by the DataSet (commonly referred to as a ProDataSets) to ensure that your data mappings between the foreign data source and the temp-table are accurate. Accurate data mappings are essential for sending data back from the temp-table to the data source. Non-ROWID key definitions can also be described with a unique key.

ProDataSets functionality is based on one or more temp-tables that share and extend basic temp-table functionality. For more information about ROWID field, see the “ROWID Support” section on page 3–32. For in depth discussion of temp-tables and more information about ProDataSets, see OpenEdge Development: ProDataSets.
Table 3–3 highlights additional language elements you can use with the stored procedure and the send-sql-statement language to use ROWID.

Table 3–3: Returning result sets and loading the data into temp-tables

<table>
<thead>
<tr>
<th>ABL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN STORED-PROCEDURE statement</td>
<td>Executes the stored procedure or send-sql-statement and tells OpenEdge that the stored procedure has ended.</td>
</tr>
<tr>
<td>LOAD-RESULT-INTO phrase</td>
<td>Allows data from a result set that is returned for a foreign data source either through a stored procedure or a send-sql-statement to be put into one or more temp-tables. Static, unprepared dynamic, and prepared dynamic temp-tables are supported. Pass as many temp-tables as result set instances are returned by SQL statements in your Send-SQL or RUN STORED-PROC definition.</td>
</tr>
<tr>
<td>PROC-STATUS phrase</td>
<td>Reads the return value (optional).</td>
</tr>
</tbody>
</table>

Note: When using SQL statement(s) through a send-sql-statement or stored procedure to load result sets into temp-tables, RUN STORED-PROC carries an implicit RUN CLOSE-PROC statement. (The stored procedure’s output parameters are available after the RUN STORED-PROC executes and closes the procedure.)

Note: To optimize enhanced stored procedure calls, it is recommended that you use firehose cursor functionality. Use of firehose cursors provided through the connection pooling functionality supports the fastest way to read and return a result set. For details about connection pooling and firehose cursors, see the “Connection pooling” section on page 4–2.

Creating a temp-table layout plan

When using *static* or *prepared dynamic temp-tables*, you must define the temp-table layout in your application program to accommodate specific result sets before you attempt to populate the temp-tables with data. If a SQL statement retrieves more than one results set, you must define multiple temp-tables to be able to retrieve all the data. Therefore, the success of this approach depends to a large extent on your:

- Understanding of the specific data your foreign data source is providing you through a given stored procedure
- Ability to correctly define temp-tables

Table 3–3
The following types of temp-tables can support results sets:

- **Static** — A temp-table whose schema is defined at compile time.
- **Dynamic** — A temp-table whose schema is defined at run time. There are two types of dynamic temp-tables: dynamic-prepared and dynamic-unprepared.

Keep in mind that you can pass handles of temp-tables that contain a mixed array. A mixed array is one in which some of the temp-table handle elements can be static while others can be dynamic. Also, note that a stored procedure supports the use of an INT64 data type in static and dynamic temp tables when the LOAD-RESULT-INTO phrase processes the procedure’s result set on the RUN-STORED-PROC statement.

**Using a temp-table handle with an unprepared dynamic temp-table**

When a temp-table handle points to an unprepared dynamic temp-table, the MS SQL Server DataServer defines the temp-table schema in the form of the result sets record structure which is passed back to the DataServer from the foreign data source. The data types defined for the temp-table schema are determined based on the default data type mapping that exists between the SQL data type and its equivalent OpenEdge default data type. Once the temp-table schema is dynamically established by the DataServer, the result set begins to populate it.

Recognize that there is the possibility of a small performance price to be paid when you build dynamic temp-tables. However, considering the database independence that this technique affords over building static temp-tables, you might consider the price of dynamically built temp-tables to be a small, reasonable one.

*Table 3–4 identifies the temp-table options for which you can plan and the requirements you must fulfill for each option.*

**Table 3–4: Options to plan the temp-table layout for result sets**

<table>
<thead>
<tr>
<th>To return a result set to this type of temp-table . . .</th>
<th>Then the layout definition is . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Static</strong></td>
<td>Defined by you; you must base the layout on the expected fields to be returned and each of these fields’ data types so that the first field defined in the temp-table corresponds to the first column of the result set. This column matching and data type matching must be repeated successfully for each temp-table and its corresponding result set.</td>
</tr>
<tr>
<td><strong>Dynamic-prepared state</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Dynamic - unprepared state</strong></td>
<td>Not defined by you; the schema of the temp-table is based on the result-set schema and a mapping of default OpenEdge data types for each SQL type. For more details, see the “Details about a dynamic temp-table in an unprepared state” section on page 3–25.</td>
</tr>
</tbody>
</table>

**Note:** Once the data is loaded into the temp-table, any updates made to the records in the temp-table are not propagated back to the foreign database. Result sets are available through temp-tables for the purpose of obtaining a snapshot of the data. For example, you can use this technique to populate a browser from a temp-table. You must re-read the record using the proper lock mechanism to actually update the record.
Details about a dynamic temp-table in an unprepared state

A dynamic temp-table is considered to be in an unprepared state after the first definitional method is called until the temp-table is prepared. If a clear dynamic temp-table handle is passed, the DataServer populates the temp-table schema based on the result-set schema and prepares the temp-table. A clear dynamic temp-table is a table that is in an unprepared state where definitional methods have not yet been called. The DataServer then executes the temp-table handle: ADD-NEW-FIELD (field name, data type) internally for each one of the columns of the result set. If a column in the result set from the foreign schema does not have a name (for example, an array element field or a SQL-derived field), the DataServer assigns a name to it based on the column position on the row.

For example, if you run the following statement, then the temp-table contains these columns:

- Column 1
- Cust_num
- Name
- Column 4

```sql
/* */
SELECT "mytag," cust_num, name, (salary * 0.10) FROM <anytablename>
```

The data type associated with each column follows the mapping rules that exist between OpenEdge and the foreign data source’s data types. For more information about data types and default mapping, see Chapter 2, “Initial Programming Considerations.”

**Note:** Since a stored procedure can return multiple result sets, the DataServer prepares the temp-table as “result<n>” where <n> is the result-set sequence number that corresponds to its element position in the temp table handle array, starting with 1. Therefore, if there are 3 result sets and 3 clear dynamic temp-tables are passed, the temp-tables are called result1, result2, and result3.

Note the following error conditions as they specifically apply to a dynamic temp-table:

- If an error occurs during the schema population or during the prepare of a temp-table, the DataServer raises an error condition.

- The dynamic temp-table must be either already prepared or clear with no defined fields in it. If fields have already been added to a dynamic temp-table before the RUN STORED-PROC statement is executed and the temp-table is not prepared, the DataServer raises an error condition due to the invalid state of the temp-table.
Details about a dynamic temp-table in a prepared state

Example 3–14 shows multiple dynamic temp-tables in a prepared state. The temp tables, tt1 and tt2, are prepared in the code just before the RUN STORED-PROC statement is called and the temp-tables are passed. The temp-table definition is considered prepared because of the explicit inclusion of each field name and associated data type.

Example 3–14: Multiple temp-tables in a prepared state

In the previous dynamic temp-table example, note that the temp-table definitions must match the result set that the send-sql-statement returns. For example, the number of fields and the data type of each field in the result must have a corresponding field defined in the temp-table that will receive the result set. No mapping exists between the result set and the temp-table. Therefore, the first field defined in the temp-table corresponds to the first column of the result set and so forth.

Additional temp-table examples

This section presents more examples that show various techniques to code temp-tables.

Example 3–15 follows this initial explanation; the stored procedure myProc is comprised of two fields—one is an integer and the other is a character, and a result set.

Example 3–15: Temp-table code technique
Example 3–16 shows the basics of executing a call to a stored procedure using the 
LOAD-RESULT-INTO functionality. Note that in this example, PROC-STATUS function is not 
specified. Therefore, there is no possibility of a return value.

Example 3–16: Basic execution of a stored procedure using 
LOAD-RESULT-INTO functionality

Example 3–17 shows the basics of using an existing dynamic temp-table without the 
TEMP-TABLE-PREPARE() method. In this instance, send-sql-statement is used rather than a 
predefined stored proc. In contrast, the third example code that appears later in this section 
shows the same approach, but explicitly defines the existing dynamic temp-table with the 
TEMP-TABLE-PREPARE() method.

Example 3–17: Using an existing dynamic temp-table without the 
TEMP-TABLE-PREPARE() method
Example 3–18 shows the basics of using an existing dynamic temp-table with the `TEMP-TABLE-PREPARE()` method.

Example 3–18: Using an existing dynamic temp-table with the `TEMP-TABLE-PREPARE()` method

Note the following points as they relate to Example 3–18:

- As a prerequisite for creating the code shown, the developer would need to define the schema for the table.

- Once the temp-table schema begins preparation from the clear state, the temp-table must be defined to the exact specifications for the result sets as generated by the `RUN STORED-PROC` statement or `send-sql-statement`. Otherwise, the `RUN STORED-PROC` will end in failure. Also, note the `TEMP-TABLE-PREPARE` must be called at the completion of the temp-table definition associated with the stored procedure results.

Example 3–19 shows passing the handles of empty dynamic temp-tables to enable a MS SQL Server DataServer to create schema based on the result-set schema.

Example 3–19: Calling a stored procedure and using the `EXTENT` phrase to pass handles of empty temp-tables to enable the MS SQL DataServer to create schema holder definitions based on each temp-table's result-set schema.

```sql
/* Calling a stored procedure, using an existing temp-table with temp table prepare*/
DEFINE VARIABLE ttHndl AS HANDLE NO-UNDO.
CREATE TEMP-TABLE tt1Hndl.
ttHndl:ADD-NEW-FIELD("custNum", "integer").
ttHndl:ADD-NEW-FIELD("name", "character").
ttHndl:TEMP-TABLE-PREPARE("ordX").
RUN STORED-PROC send-sql-statement LOAD-RESULT-INTO ttHndl
("select custNum, name from myDB.customer").

Example 3–19: Calling a stored procedure and using the `EXTENT` phrase

/*Calling a stored procedure, and using the EXTENT phrase to pass handles of empty temp-tables to enable the MS SQL DataServer to create schema holder definitions based on each temp-table's result-set schema. */
DEFINE VARIABLE ttCustomer AS HANDLE NO-UNDO.
DEFINE VARIABLE ttHandle AS HANDLE NO-UNDO EXTENT 2.
DEFINE VARIABLE ttOrder AS HANDLE NO-UNDO.
CREATE TEMP-TABLE ttCustomer.
CREATE TEMP-TABLE ttOrder.
RUN STORED-PROC send-sql-statement LOAD-RESULT-INTO ttHandle
("Select * from customer; select * from order").
```
Example 3–20 shows how to use the enhanced stored procedure syntax with a single static temp-table and the send-sql-statement.

```/* Calling an enhance stored procedure with a single temp table and the send-sql-statement */
DEFINE VARIABLE ttHandle AS HANDLE NO-UNDO.
DEFINE TEMP-TABLE ttHandle
    FIELD f1 AS INTEGER
    FIELD f2 AS CHARACTER.

ttHandle = TEMP-TABLE ttHandle:HANDLE.
RUN STORED-PROC send-sql-statement LOAD-RESULT-INTO ttHandle
    ("Select cust_num,name from customer").
```

**Example 3–20: Calling an enhanced stored procedure with a single temp table and the send-sql-statement**

Example 3–21 shows the use of a PROC-STATUS phrase. The PROC-STATUS phrase must be defined as part of the RUN-STORED-PROC statement because of the implicit CLOSE STORED-PROC that is associated with the LOAD-RESULT-INTO phrase.

```/* Sixth example - Example of the implicit close stored-proc and use of LOAD-RESULT-INTO */
DEFINE VARIABLE iStat AS INTEGER NO-UNDO.
DEFINE VARIABLE ttHandle AS HANDLE NO-UNDO.

CREATE TEMP-TABLE ttHandle.
RUN STORED-PROCEDURE pcust LOAD-RESULT-INTO ttHandle
    iStat = PROC-STATUS (20, output 0, output 0).

DISPLAY iStat.
```

**Example 3–21: Use of a PROC-STATUS phrase**

In Example 3–21, note that the PROC-STATUS phrase does not need a PROC-HANDLE phrase because it is retrieved using the RUN STORED-PROC statement and not after this statement’s execution as it typically is used.
Handling errors

The RUN STORED-PROC statement supports the NO-ERROR option. Example 3–22 shows how to trap errors within a procedure.

```sql
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(j)
    ERROR-STATUS:GET-MESSAGE(j).
  END.
END.
CLOSE STORED-PROC send-sql-statement.
```

Example 3–22: Trapping errors within a procedure

Trapping errors when using Load-Result-Into

By properly positioning the NO-ERROR in the RUN statement, error information can also be retrieved from attempted SQL execution at the data source when using LOAD-RESULT-INTO. Example 3–23 shows how errors are trapped after LOAD-RESULT-INTO stored procedure execution.

```sql
DEFINE VARIABLE hTables AS HANDLE EXTENT 1 NO-UNDO.
RUN STORED-PROC send-sql-statement LOAD-RESULT-INTO hTables NO-ERROR
("Invalid SQL Statement").
IF ERROR-STATUS:ERROR THEN
  MESSAGE ERROR-STATUS:GET-MESSAGE(1) VIEW-AS ALERT-BOX.
ELSE
  MESSAGE "SQL call was successful." VIEW-AS ALERT-BOX.
```

Example 3–23: Trapping errors using LOAD-RESULT-INTO

Error messages related to using the Load-Result-Into phrase

Note the following situations in which error conditions related to the use of LOAD-RESULT-INTO are reported:

- The compiler issues an error if you try to specify PROC-HANDLE and LOAD-RESULT-INTO on the same RUN STORED-PROC statement.
- The compiler issues an error if you use PROC-STATUS without specifying the LOAD-RESULT-INTO function.
• If you try to execute the `CLOSE STORED-PROC` statement after the `RUN STORED-PROC` statement for which you have defined the `LOAD-RESULT-INTO` function, the following error message appears:

  - **Error “No active stored procedures. (2101)”** — This error message occurs when no handle is specified in the statement.

  - **Error “No active procedure for specified proc-handle. (2475)”** — This error message occurs when a handle is specified in the statement.

  The system generates either of the previous error messages as it assumes a second `CLOSE STORED-PROC` statement has been incorrectly executed in the current implementation.

  For more information about the enhanced store procedure functions and statements, see Table 3–3.
ROWID Support

This section presents details about ROWID function, focusing on:

- Understanding the ROWID implementation
- Using ROWID with RUN STORED-PROCEDURE and LOAD-RESULT-INTO
- Additional ProDataSet support

Understanding the ROWID implementation

The OpenEdge ROWID function returns the unique internal identifier of the database record currently associated with the record buffer you name. This internal identifier has the data type ROWID, which is supported for OpenEdge and all other DataSever databases.

ROWID characteristics

The ROWID value is not stored directly in the MS SQL Server data source, but it is represented by a unique index key in your database. If you migrated your database to a MS SQL Server and elected to use the Create RECID Field option during the migration, a unique 8-byte integer column named PROGRESS_RECID and a corresponding seed column named PROGRESS_RECID_IDENT will have been automatically generated in your database tables along with a corresponding Index containing the PROGRESS_RECID value as its key.

**Note:** While the PROGRESS_RECID_IDENT column is not used to resolve the ROWID value, it is important to recognize the presence of this column for purposes of the following discussion. (This point is especially significant because its physical presence is unknown to the schema holder definition that represents this table to the client.)

Databases that are only pulled from the native environment or are migrated without the Create RECID Field option must choose a unique index key from each table to represent the PROGRESS_RECID to support the RECID and ROWID functions, and forward and backward scrolling within their ABL sessions. ABL will internally map the selected unique key to the Progress RECID and ROWID functions.

**Note:** Reposition functions such as REPOSITION-BACKWARDS and REPOSITION-TO-ROW typically use ROWID to identify records. And functions of this type require integer expressions which, beginning with OpenEdge Release 10.1B, can be either INTEGER or INT64.
It is important to note that the unique index key used to derive the Progress ROWID must be a single component index for it to map to the ROWID of a temp-table as described in the following discussion. For more details, see the ROWID function discussion in Chapter 2, “Initial Programming Considerations.”

Starting in OpenEdge Release 10.1B, the default external representation of ROWID enables the MS SQL Server DataServer to use a 64-bit integer value for ROWID.

Also, before discussing ROWID as it relates to the RUN STORED-PROC statement, it is important to understand an important property of the OpenEdge ROWID. The “ROWID—Standard ABL behavior” section on page 3–33 presents this information.

**ROWID—Standard ABL behavior**

The ROWID value of a temp-table buffer will be different than the ROWID value of a record BUFFER even if the underlying data is identical. This difference exists because the ROWID function relies on the record buffer name. Example 3–24 shows the standard, expected ABL behavior.

```abl
DEFINE VARIABLE rid-1 AS ROWID NO-UNDO.
DEFINE VARIABLE rid-2 AS ROWID NO-UNDO.
DEFINE VARIABLE ttHandle AS HANDLE NO-UNDO EXTENT 1.

  FIELD tRecid AS INTEGER
  FIELD tRECID_ident AS INTEGER.

  ttHandle[1] = TEMP-TABLE ttCustomer:HANDLE.

FIND FIRST Customer WHERE Customer.CustNum = 1 NO-LOCK.
  rid-1 = ROWID(Customer).

RUN STORED-PROC send-sql-statement
  LOAD-RESULT-INTO ttHandle ("select * from customer where custnum = 1").
  rid-2 = ROWID(ttCustomer).

IF rid-1 NE rid-2 THEN
  MESSAGE "The same record but different ROWID's".
```

**Example 3–24: Expected ABL behavior—ROWID value of a temp-table buffer**

The following sections illustrate the differences between the ROWID value of a temp-table buffer and the ROWID value of a record BUFFER so that you can best understand, program for, and leverage the use of the ROWID function with the LOAD-RESULT-INTO clause of the RUN STORED-PROCEDURE command.
Using ROWID with RUN STORED-PROCEDURE and LOAD-RESULT-INTO

In the previous section, Example 3–24 assumes you migrated your database to MS SQL Server using the Create RECID Field option, as discussed in the “ROWID characteristics” section on page 3–32. Example 3-24 and all subsequent examples in this section show ROWID as being represented by the 8-byte integer value of the PROGRESS_RECID column as opposed to some other unique single-component index designated in your database to be the PROGRESS_RECID.

Note: If you used a different single-component index to load the ROWID of a temp-table, you would need to map the column accordingly, just as the example maps PROGRESS_RECID.

The RUN STORED-PROC command has no native awareness that the MS SQL Server Database table is being queried for the result set(s) it generates. Therefore, to allow DataServer technology to convert the stored PROGRESS_RECID value into a native OpenEdge ROWID value, the physical name of the target database table needs to be known. To achieve this bond, the temp-table that the stored procedure will populate must be associated with an OpenEdge ProDataSet object.

Example 3–25 shows an ABL query filling the temp tables of a ProDataSet. It will be used as the baseline code which will be referenced throughout the remainder of this section.

```
DEFINE VARIABLE phDataSet AS HANDLE NO-UNDO.
  FIELD tRecid AS INTEGER
  FIELD tRECID_ident AS INTEGER.
DEFINE DATASET dsCustomer FOR ttCustomer.
DEFINE QUERY qCustomer FOR Customer.
phDataSet=DATASET dsCustomer:HANDLE.
DEFINE DATA-SOURCE srcCustomer FOR QUERY qCustomer.
BUFFER ttCustomer:HANDLE:ATTACH-DATA-SOURCE
  (DATA-SOURCE srcCustomer:HANDLE,?,?,?).
QUERY qCustomer:QUERY-PREPARE ("FOR EACH Customer").
DATASET dsCustomer:FILL().
FOR EACH ttCustomer:
  DISPLAY ttCustomer.name ttCustomer.tRecid.
END.
```

Example 3–25: ABL Query filling a ProDataSet temp-table
If the table uses computed-column PROGRESS_RECID option, then ttCustomer temp-table definition should be:

```ABL
DEFINE TEMP-TABLE ttCustomer LIKE Sports.Customer
FIELD tRecid AS ROWID /* Must be changed to ROWID type */
FIELD tRECID_ident AS INT64
FIELD tRECID_alt as INT64.
```

Example 3–26 combines code from Example 3–24 and Example 3–25 by applying the results of the RUN STORED-PROC [LOAD-RESULT-INTO] technique, rather than an ABL query, to fill the TEMP-TABLE associated with a ProDataSet.

Example 3–26: Using the LOAD-RESULT-INTO technique to populate the underlying Temp-Table of a ProDataSet

Keep the following key points in mind regarding Example 3–26:

- The TEMP-TABLE field that is mapped to the PROGRESS_RECID column should be changed from its standard definition of INTEGER to ROWID.

In Example 3–26, the result column location where PROGRESS_RECID is being returned has been named tRecid in the temp-table. The PROGRESS_RECID_IDENT has been renamed tRECID_ident. This renaming occurs in Example 3-26 because of the following line:

```ABL
DEFINE TEMP-TABLE ttCustomer LIKE Sports.Customer
FIELD tRecid AS ROWID /* Must be changed to ROWID type */
FIELD tRECID_ident AS INT64.
```
If the table uses computed-column `PROGRESS_RECID` option, then ttCustomer temp-table definition should be:

```
DEFINE TEMP-TABLE ttCustomer LIKE Sports.Customer
FIELD tRecid AS ROWID /* Must be changed to ROWID type */
FIELD tRECID_ident AS INT64
FIELD tRECID_alt as INT64.
```

- The TEMP-TABLE must be defined to the ProDataSet. The following line excerpted from Example 3–26 shows this definition:

```
DEFINE DATASET dsCustomer FOR ttCustomer.
```

- The technique, demonstrated in Example 3–26, does not change the default behavior of the ROWID function, but provides a mechanism for the ROWID value to be stored along side its corresponding result rows; therefore, using the ROWID value to access database rows is unconventional with respect to the normal, or more typical, association between ROWID and a database table row. The following code demonstrates this difference.

Default use of ROWID function on a record buffer, as excerpted from Example 3–26:

```
FIND FIRST Customer WHERE Customer.CustNum = 1 NO-LOCK.
    rid-1 = ROWID(Customer).
```

In contrast, the following code excerpt from Example 3–26, demonstrates an alternative use of the ROWID value with a temp-table:

```
FIND FIRST Customer WHERE Customer.CustNum = 1 NO-LOCK.
    rid-2 = ttCustomer.tRecid.
```

### Additional ProDataSet support

As previously demonstrated in Example 3–26, the `LOAD-RESULT-INTO` functionality provides a very fast and efficient way to populate the temp-table(s) of a ProDataSet Object.

Example 3–27, a two-part example, builds on the techniques demonstrated in Example 3–26, but shows the ProDataSet object BEFORE-FILL procedure can be modified to provide a single ProDataSet data access object that can be used against native OpenEdge or against MS SQL Server and other DataServer data sources.
Example 3–27: Use the LOAD-RESULT-INTO technique with BEFORE-FILL method to fill the temp-table(s) of a ProDataSet
Additional Features to Enhance DataServer Performance

There are a variety of additional DataServer-related features you can employ to enhance your DataServer’s performance, as described in the following sections:

- Enhancements overview
- Connection pooling
- OpenEdge query types
- Query tuning
- Caching records
- Join by SQLDB
- Writing queries for performance
- Modifying the run-time schema check
- Replacing FIND FIRST for performance
Enhancements overview

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 MS SQL Server database. For example, you might write a query that gives you consistent results across databases or one that takes advantage of MS SQL Server’s 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 document the QUERY–TUNING phrase.
- For information on tuning queries at compile time and run time, see the “Query tuning with connection and startup parameters” section on page 6–16.

Connection pooling

The DataServer for Microsoft SQL Server is enhanced with the ability to form a connection pool. A connection pool is a set of database connections that are available for an application to use and reuse without having to be reestablished. Connection pooling significantly improves the cursor management associated with no-lock queries, particularly multi-table joins. Creating and tearing down connections can be resource intensive. Using a pooled connection to keep existing connections alive results in significant performance gains because the DataServer avoids the overhead of making a connection for each request. ABL applications that open multiple no-lock queries and handle their results simultaneously experience the best cumulative performance gains from connection pooling.

Main components

Connection pooling for the DataServer for Microsoft SQL Server is a combination of ODBC connection pooling and DataServer connection management. These connection components can be used as follows:

- Individually, ODBC connection pooling or DataServer connection management provides the foundation required for firehose cursors, but enabling both provides the best performance. For more information on firehose cursors, see the “Firehose and Fast Forward-Only Cursors” section on page 4–5.

Without a connection pool, firehose cursors would block an application until a full result set is retrieved. Because of this, when connection pooling is disabled, firehose cursors are also disabled. By maintaining multiple connections and one cursor per connection, read-only requests only block the connection on which they retrieve results, freeing ABL applications to continue processing data on the other connections.

- ODBC connection pooling and DataServer connection management provide the highest performance improvements when enabled together, but they can also be enabled independent of one another.
For installations where the number of ODBC connections is limited, you might decide to enable only the DataServer connection management. For deployments in which memory constraints are a concern, enabling only the ODBC connection pool provides on-demand connections, with optimized resource efficiency. If both an ODBC connection pool and managed connections coexist in the same DataServer process, the managed connections will come from the ODBC connection pool. The managed connections are distinguished from the connection pool connections in that they are never released back to the pool for the life of an ABL session.

**Considerations when using ODBC connection pooling and DataServer connection management**

When both ODBC connection pooling and DataServer connection management are enabled, set the number of managed connections to a value that covers the required number of connections for a typical application. Reuse of the managed connections takes precedence over ODBC connection pool connections. The ODBC connection pool is then an overflow handler for connections required beyond the capacity of the managed connections. This configuration ensures that there is little risk of having a negative impact on performance by downgrading to a Fast Forward-Only server-side cursor in the mainline transaction-oriented connection. For more information on monitoring the performance of your connection pool, see the “Monitoring cursor and connection use” section on page 4–6.

**ODBC Connection Pooling**

ODBC connection pooling is managed by the ODBC driver manager, external to the DataServer. The size of the pool grows and contracts dynamically based on demand and time-out intervals. The upper limit to the pool size is limited only by memory and other server resource constraints. The DataServer retrieves connections out of the externally managed pool. ODBC connection pooling is maintained with the CONNECTION_POOLING option to the -Dsrv connection parameter. ODBC connection pooling is enabled by default.

Disable ODBC connection pooling with the following syntax:

**Syntax**

```
-Dsrv CONNECTION_POOLING,0
```

**Note:** To take advantage of ODBC connection pooling in the DataServer, you must rebuild your schema holder. However, if your schema holder was built before OpenEdge Version 10.0B, contact Technical Support for information about how to proceed.
DataServer connection management

When an ABL session is established, the DataServer for Microsoft SQL Server opens the number of read-only connections specified in the PRGRS_CACHE_CONN value. The DataServer maintains these connections for the duration of an ABL session, reusing them for subsequent queries.

Enabling connection management

DataServer connection management is enabled and tuned with the PRGRS_CACHE_CONN option to the -Dsrv connection parameter. Connection management is enabled by default, and the default number of managed connections is 5. Change the number of connections using the following syntax:

**Syntax**

```
-Dsrv PRGRS_CACHE_CONN, n
```

Where \( n \) is the number of managed connections.

Disabling connection management

DataServer connection management is disabled by setting the number of managed connections to zero, as shown:

```
-Dsrv PRGRS_CACHE_CONN, 0
```

Connections exclusively for stored procedures

When DataServer connection management is enabled, connections for stored procedures are allocated from the managed connections. If your stored procedures modify the connection environment or attributes, allocate a set of managed connections exclusively for stored procedures to prevent the modified connections from being reallocated for general use.

Allocate managed connections exclusively for stored procedures with the following syntax:

**Syntax**

```
-Dsrv PRGRS_SP_CACHE_CONN, n
```

Where \( n \) is the number of managed connections allocated to stored procedures. Set \( n \) to a value large enough to handle the largest number of stored procedures you can run simultaneously. If you exceed \( n \), the connection will be allocated from the ODBC connection pool, if enabled, making the modified connection available for general reuse.

**Note:** This switch is off by default, and is only needed for applications that modify the connection environment or attributes within stored procedures.

Impact on commits in stored procedures

Running a stored procedure in a separate connection changes the timing of changes being committed to the data source. When a stored procedure is run in a separate connection, changes not explicitly committed during the execution of the stored procedure are committed at the time the procedure handle is closed and the connection is recycled.
Firehose and Fast Forward-Only Cursors

Firehose cursors deliver a streamlined, unmanaged, client-side cursor-processing mechanism for handling result sets from a DataServer query. When connection pooling is enabled, firehose cursors are the default mechanism for handling read-only results. If a firehose cursor is denied to an application, the DataServer first attempts to substitute a Fast Forward-Only (FFO) server-side cursor with Auto-Fetch and Auto-Close attributes in its place. If the query cannot be handled by a FFO cursor, the cursor is further downgraded.

Firehose cursors

Firehose cursors are identified in Microsoft SQL Server as the default result set. A default result set is generated when the statement attributes of a cursor are left unchanged from their standard MS SQL defaults. The default result set allows rows from a query result to be pulled without locks in forward-only sequence into a client-side cache. The default result set is referred to as a firehose cursor because it can “flood” the client with results. It is unencumbered by the cursor management necessary with server-side cursors.

The following DataServer operations are eligible for the firehose cursor implementation:

- All NO-LOCK queries.
- All SHARE-LOCK queries with transaction isolation level set to read-uncommitted.
- Internal no-lock queries that populate the key cache for transaction-oriented operations.
- All stored procedure result sets.
- All send-sql-statement result sets.
- Queries written with the QUERY-TUNING(SEPARATE-CONNECTION) keyword. When connection pooling is enabled, the QUERY-TUNING(SEPARATE-CONNECTION) is redundant.

Note: Prepared statements associated with firehose cursors are now cached on a statement cache that is associated with the managed connection. Statement reuse may decrease based on the recycling of managed connections. To completely disable the prepared statement cache, use the following connection switch: “-Dsrv PRGRS_PREPCACHE,0”. For information on monitoring the statement cache reuse, see the “Monitoring cursor and connection use” section on page 4–6.

Firehose exclusions

Certain statements and fields automatically exclude a query from consideration for a firehose cursor. Two examples are:

- FOR FIRST and FOR LAST statements. Because these statements return at most one row, they are optimized with the Transact-SQL keyword TOP, which renders them ineligible for a firehose cursor.
- Queries that include text or image fields that translate to SQL LONGVARCHAR and LONGVARBINARY data types. If possible, use field lists to exclude these fields from your result set.
Fast Forward-Only cursors

In the event that a firehose cursor cannot be used, the DataServer attempts to use a Fast Forward-Only (FFO) cursor with Auto-Fetch and Auto-Close attributes. FFO cursors are the server-side equivalent of firehose cursors. FFO cursors have special optimization characteristics that distinguish them from other server-side cursors. They require a minimum of server-side resources and are capable of minimizing round trips to the server. FFO cursors are an extension to the ODBC specification and are unique to ODBC drivers that conform to Microsoft SQL Server driver requirements. The Auto-Fetch attribute directs the server to return the initial block of results in the same network message that provided the SQL request to be executed by the server. The Auto-close attribute directs the server to automatically close the cursor on the same round trip in which the last query result is received by the client.

Note: Result sets that include text or image columns cause an implicit conversion from an FFO to a dynamic cursor type. These are columns that translate through ODBC to SQL LONGVARCHAR and LONGVARBINARY data types.

Monitoring cursor and connection use

Monitor the use of your managed connections to tune the number of connections you allocate. If you regularly exceed your allocation, consider increasing the number of managed connections. If you never use your total allocation, consider decreasing the number of managed connections.

You can monitor connections either through using OpenEdge logging or by enabling logging using the “-Dsrv qt_debug,cursor” switch.

Note: The OpenEdge logging infrastructure offers more extensive reporting capabilities than qt_debug. For details on enhanced logging, see

Monitoring connections with qt_debug

The DataServer log contains messages indicating the status of connections. At startup, the number of managed connections initialized is written to the log file. If connections are rejected, this is also logged. If a connection from the ODBC connection pool cannot be reused, the ODBC driver issues the message DEAD Connection which is written to the log file.

At the end of a session, the DataServer writes summary information about cursor use. The summary contains the following information:

- **Needed connections** — The number of connections actually used during a session
- **Peak connections** — The maximum number of connections used simultaneously during a session

When debug logging is enabled with the “-Dsrv qt_debug,cursor” switch, a summary of connection activity is written to the log file. This summary contains:

- Number of connections (defaults to 5)
- Number of peak connections (defaults to 5)
- Highest connection value
Enhancements overview

- Lowest connection value
- Number of connections pulled out of pool
- Number of connections placed in the pool
- Total sum of active connections

The following excerpt from a dataserv.log file shows the summary of the managed connection activity written when the "-Dsrv qt_debug,cursor" switch is used, plus the standard messages:

```
16:32:09 : Standard secondary connection:
16:32:09 : Connection chain 0xbc6e64: num_conns 1
16:32:09 : Connection chain 0xbc6e64: peak_conns 1
16:32:09 : Connection chain 0xbc6e64: high_conns 1
16:32:09 : Connection chain 0xbc6e64: low_conns 0
16:32:09 : Connection chain 0xbc6e64: num_off_conns 1
16:32:09 : Connection chain 0xbc6e64: num_on_conns 1
16:32:09 : Connection chain 0xbc6e64: sum_active 1
16:32:09 : Connection chain 0xbc6e64: Needed Connections: 1, (Needed 4 less than requested)
16:32:09 : Connection chain 0xbc6e64: Peak Connections: 1, (Peaked at 4 less than requested)
16:32:09 : Connect: 0x1012398    Stmt: 0xb95f80    Handle: 0x1011d98    Crc: 1613 Property: Statement dropped
16:32:09 : Statement cache (-Dsrv PRGRS_PREPCACHE setting) reuse ratio is 0.
```

For more information on using the qt_debug option to the -Dsrv switch, see Appendix B, “Server Related Command Line Utilities and Startup Parameters.”

Cursor downgrades

Cursor downgrades are monitored by the DataServer, according to the type of cursors in use:

- For firehose cursors, the DataServer counts the number of times a cursor is downgraded, and writes the count to the log file at the end of the session.
  
  Consider increasing the number of DataServer managed connections if firehose cursors are frequently downgraded.

- For downgrades of Fast Forward-Only cursors, a message is written to the log file when the downgrade occurs.

Statement cache

The reuse of the statement cache is calculated as a ratio. Firehose cursors decrease the probability of a statement being reused. A reuse ratio of 3 indicates that statements are reused an average of 3 times. Reuse ratios of 3 or higher indicate good utilization of the statement cache. Reuse ratios less than 1 indicate poor utilization of the statement cache. If your ratio is less than 1, consider disabling the statement cache with "-Dsrv PRGRS_PREPCACHE,0". For more information on the PRGRS_PREPCACHE option to the -Dsrv switch, see Appendix B, “Server Related Command Line Utilities and Startup Parameters.”
OpenEdge query types

The DataServer provides several ways to submit an OpenEdge query to a MS SQL Server data source:

- **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 `SELECT` statement. When you use this statement in an OpenEdge 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 RDBMS stored procedures. If you want to use specialized query syntax supported only by Transact-SQL extensions, you can use `RUN–STORED–PROC send–sql–statement` to send the syntax to MS SQL Server. You might use a stored procedure to include `BEGINS` as a search criterion; this SQL query can result in better performance. For more information, see Chapter 3, “RDBMS Stored Procedure Details.”

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 Transact-SQL extensions are processed by the server.
Query tuning

How you structure a query determines how efficiently you access a database. Using your MS SQL Server 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 OpenEdge QUERY–TUNING phrase.

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

- FOR EACH

  **Syntax**

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

- OPEN QUERY

  **Syntax**

  ```plaintext
  OPEN QUERY query QUERY-TUNING(query-tuning-option
  query-tuning-option...)
  ```

- DO PRESELECT

  **Syntax**

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

- REPEAT PRESELECT

  **Syntax**

  ```plaintext
  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 4–1 describes the query-tuning options.

<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>
<tr>
<td>CACHE–SIZE <em>integer</em></td>
<td>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. This value will override a cache size specified with -Dsrv QT_CACHE_SIZE. Minimum: The DataServer always caches at least one record. Maximum: None. Default: 30000.</td>
</tr>
<tr>
<td>DEBUG EXTENDED</td>
<td>Specifies whether the DataServer should print to the dataserv.lg file the debugging information that it generates for a query. Specify DEBUG SQL to print only the SQL that the DataServer executes against the ODBC data source. Specify DEBUG EXTENDED to print additional information, such as cursor statistics. Specify DEBUG option to override the NO–DEBUG default. Default: NO–DEBUG.</td>
</tr>
<tr>
<td>DEBUG SQL</td>
<td></td>
</tr>
<tr>
<td>NO–DEBUG</td>
<td></td>
</tr>
<tr>
<td>JOIN–BY–SQLDB</td>
<td>Specifies whether the DataServer allows its data source to perform a join (this usually improves performance). JOIN–BY–SQLDB implies SEPARATE–CONNECTION queries that include joins. Default: 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 an OpenEdge session. This parameter does not override the explicit use of JOIN–BY–SQLDB in the QUERY–TUNING phrase.</td>
</tr>
<tr>
<td>NO–JOIN–BY–SQLDB</td>
<td></td>
</tr>
</tbody>
</table>
Table 4–1: Query-tuning options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOOKAHEAD</td>
<td>Specifies whether the DataServer uses lookahead or standard cursors. Lookahead cursors fetch as many records as fit in the allocated cache. 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 an OpenEdge database because changes made to the records in the cache might not be immediately visible. Specify NO-LOOKAHEAD for behavior that is consistent with OpenEdge. Default: LOOKAHEAD, when statements use NO-LOCK or when statements use SHARE-LOCK with TXN_ISOLATION level set to 1 (read uncommitted.)</td>
</tr>
<tr>
<td>NO-LOOKAHEAD</td>
<td></td>
</tr>
<tr>
<td>SEPARATE-CONNECTION</td>
<td>Specifies whether each cursor should use a separate database connection. Executing cursors in separate connections might improve performance because the DataServer does not have to restart the cursors and sort the results. Do not specify SEPARATE-CONNECTION if you require behavior that is consistent with OpenEdge. Default: NO-SEPARATE-CONNECTION except in certain cases. For details, see the “Managing connections to an MS SQL Server database” section on page 6–29.</td>
</tr>
<tr>
<td>NO-SEPARATE-CONNECTION</td>
<td></td>
</tr>
<tr>
<td>NO-QUERY-ORDER-ADDED</td>
<td>Specifies that OpenEdge should not choose an index in the absence of a USE-INDEX or BY clause in the query request. OpenEdge may otherwise select an index if it is needed to provide ABL language compatibility. <strong>Note:</strong> If you elect to use this option to omit index selection on the query, you may see better performance using the optimizer’s sort selections. However, compatibility with OpenEdge forward/backward scrolling and reposition capability may be lost. Only use this option when compatibility is not required and can be overlooked for the sake of better performance.</td>
</tr>
<tr>
<td>NO-QUERY-UNIQUE-ADDED</td>
<td>Specifies that OpenEdge should omit the record identifier from the end of the query’s generated ORDER BY clause when trying to obtain record uniqueness from a selected non-unique index. A sort order that is modified to derive uniqueness may produce a query that can’t find a useful index to perform sorting thus impacting query performance. <strong>Note:</strong> If you elect to use this option, the query may find an index match to provide better performance. However, turning off uniqueness in a query where scrolling is required may result in behavior that is incompatible with the OpenEdge ABL. Only use this option when compatibility is not required and can be overlooked for the sake of better performance.</td>
</tr>
</tbody>
</table>
Additional Features to Enhance DataServer Performance

All but two of the QUERY–TUNING options take effect at both compile time and run time. 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. It includes a join, JOIN–BY–SQLDB, that the DataServer instructs the MS SQL Server data source to perform by default, as shown:

```
FOR EACH customer, EACH order OF customer WHERE order.ordernum GT 20
   BY customer.custnum QUERY–TUNING (JOIN–BY–SQLDB)
```

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 that it executes (the DEBUG EXTENDED option)

When the DataServer constructs queries for a MS SQL Server 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.

---

### Table 4–1: Query-tuning options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIREHOSE–CURSOR</td>
<td>Specifies at the query level that the firehose cursor type should be considered to satisfy the query when the NO–LOCK phrase is used.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This query-level option overrides the connection-level -Dsrv options, QT_FIREHOSE and QT_NO_FIREHOSE that determine if firehose cursors should be considered for the DataServer connection.</td>
</tr>
<tr>
<td>NO–FIREHOSE–CURSOR</td>
<td>Specifies at the query level that the firehose cursor type should not be considered to satisfy the query when the NO–LOCK phrase is used.</td>
</tr>
<tr>
<td></td>
<td><strong>Notes:</strong> This query-level option overrides the connection-level -Dsrv options, QT_FIREHOSE and QT_NO_FIREHOSE that determine if firehose cursors should be considered for the DataServer connection.</td>
</tr>
<tr>
<td></td>
<td>By default, firehose cursors are available to satisfy NO–LOCK queries during a DataServer session. It is generally recommended this default use be retained and overridden by QT_NO_FIREHOSE on an individual query basis in the event that slow query performance is observed on a very large result set.</td>
</tr>
</tbody>
</table>

---

4–12
Caching records

The DataServer caches results sets from a MS SQL Server data source to enhance performance when using block cursors. It caches as much data as fits in its allocated cache size. When using firehose and/or lookahead calls, the allocations are made to fit the size of the result set.

Depending on what kind of cursor a query is using, the DataServer caches row identifiers or records:

- **Standard cursors** — The DataServer caches row identifiers (PROGRESS_RECID column or other unique index) for the results set. If you use the PROGRESS_RECID, each identifier requires 4 bytes of cache, therefore, a results set of 100 records requires 400 bytes of cache. If you do not use the PROGRESS_RECID field, the cache size might be greater if the selected unique index has a greater length than this field.

- **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 MS SQL Server 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 (4 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 when lookahead cursors are used. 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 GT 20
QUERY–TUNING(CACHE–SIZE 20 DEBUG EXTENDED):
```
Join by SQLDB

For queries that include joins issued in FOR EACH and OPEN QUERY statements, the DataServer evaluates the queries and in some cases instructs the MS SQL Server data source to perform the joins. A join performed by the data source, called a join by SQLDB, can improve performance; however, you receive the associated query results in an order consistent with the data source, not with the OpenEdge database. To get results that are consistent with the OpenEdge database, 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 MS SQL Server data source can perform it and estimates whether doing so improves performance. To determine whether a join by SQLDB 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 SQLDB will not occur for this query:

  ```sql
  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:

  ```sql
  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:

  ```sql
  FOR EACH customer, EACH order
  WHERE customer.custnum = order.custnum:
  ```

The DataServer also performs a join by SQLDB for the following query:
However, for the following query, 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 GT 100:
```

By default, the DataServer instructs a MS SQL Server 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. The query-level setting overrides the session-level setting.

Table 4–2 describes how these controls interact and affect the behavior.

### Table 4–2: Controlling join by SQLDB 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 MS SQL Server data source performs the join if possible.</td>
</tr>
<tr>
<td>JOIN–BY–SQLDB</td>
<td>None</td>
<td>The MS SQL Server 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 MS SQL Server data source performs the join if possible and the join contains the selection criteria described previously.</td>
</tr>
</tbody>
</table>

A join by SQLDB 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 MS SQL Server 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 FIRST.

  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 MS SQL Server 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 MS SQL Server 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 an OpenEdge SQL statement. If you use an OpenEdge SQL statement with a cursor, declare the cursor read-only.

- When you test for the existence of a record, use the CAN–FIND FIRST function, which does not retrieve the record if the DataServer passes the entire WHERE clause to the MS SQL Server data source for processing. However, avoid nesting CAN–FIND functions.

- Avoid using the RECID function. Instead, use the ROWID function.
Modifying the run-time schema check

At runtime, the DataServer for MS SQL Server performs a schema check to match a table definition in the data source against the schema image in the schema holder. It performs this task once for each table utilized in a OpenEdge procedure. In order to retrieve table attributes without having to access data, the DataServer executes the following SQL statement against the data source:

```
SELECT * FROM table-name WHERE 1 = 2.
```

Skip schema verification

When r-code runs (each time a table, view, or buffer is opened), the DataServer checks the data definitions of the MS SQL Server 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 data. 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 MSS
-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 schema definitions and the data definitions. If there are discrepancies and the DataServer continues to process queries, inserts, and deletions, your data can 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 QUERIES 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 not reliably available beyond the end of the FOR FIRST loop unless a LEAVE is specified. Similar performance advantages can be gained when retrieving last record as well.
Configuring the DataServer

Configuring the DataServer for MS SQL Server involves starting executables for several processes. This chapter provides step-by-step instructions for initially setting up the DataServer, as described in the following sections:

- DataServer components
- Configuring an ODBC driver and registering the data source
- Configuring a local DataServer
- Configuring a 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 1–16.
DataServer components

The DataServer for MS SQL Server 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 MS SQL Server database as an ODBC data source. For details, see the “Registering your data source” section on page 5–4.
- Determine which components you need on which platforms and then set up the appropriate executables on those platforms.

Table 5–1 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>
<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>
</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 5–5. If you are building a remote DataServer configuration host, see the “Configuring a remote DataServer” section on page 5–6.

Table 5–1: 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>Remote Windows</td>
<td>On the <strong>client machine</strong>, use the default OpenEdge client executable (<strong>prowin32.exe</strong>). On the <strong>host machine</strong>, to make a connection to the default DataServer executable (<strong>_mssrv.exe</strong>), use the default broker executable (<strong>_probrkr.exe</strong>) or configure a broker using the Progress Explorer.</td>
</tr>
<tr>
<td>UNIX</td>
<td>Remote Windows</td>
<td>On the <strong>client machine</strong>, use the default OpenEdge client executable (<strong>_progres</strong>). On the <strong>host machine</strong>, to make a connection to the default DataServer executable (<strong>_mssrv.exe</strong>), use the default broker executable (<strong>_probrkr.exe</strong>) or configure a broker using the Progress Explorer.</td>
</tr>
</tbody>
</table>
Configuring an ODBC driver and registering the data source

Before you can configure the DataServer you must:

- Install your ODBC drivers and the client connectivity component from MS SQL Server
- Set up your data sources

When you install the DataServer for MS SQL Server, the DataDirect ODBC driver for MS SQL Server is bundled in and installed automatically. If you prefer to use a different driver, you must obtain the drivers from the vendor and install them separately.

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.

For DataDirect ODBC Driver information, see the http://www.datadirect.com/techres/progressdoc/index.ssp.

For any alternate driver, refer to the configuration information provided by your ODBC driver vendor.

Registering your data source

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

To register a data source:

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 MS SQL Server database.
4. Specify a name for your 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.)

To configure a local DataServer in Windows:

1. Install the ODBC software and the client software 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 a schema holder for your MS SQL Server database and connect using the client executable. See the “Creating a schema holder” section on page 5–15 for instructions.
Configuring a 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 (_mssqlsrv.exe) for the client process.

- **DataServer** — The DataServer (_mssqlsrv.exe) on the host machine accesses the MS SQL Server database and communicates with the client process.

**Note:** The OpenEdge MS SQL Server DataServer component is an ODBC client with respect to the MS SQL Server database configuration even though from the OpenEdge point-of-view it resides on the server machine. As a result, your MS SQL Server client software must reside on the machine from which the DataServer and the broker processes will execute. In the remote DataServer configuration, the OpenEdge client component requires no special software or configuration; it only requires a standalone OpenEdge Windows or UNIX client. There are no OpenEdge restrictions regarding the location of the actual MS SQL Server database.

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 configure the DataServer using the Progress Explorer tool. See the “Configuring with the OpenEdge Explorer and Progress Explorer” section on page 5–6 for details.

Configuring with the OpenEdge Explorer and Progress Explorer

In Windows, you can use the Progress Explorer tool to configure the DataServer for MS SQL Server.

**Note:** This section primarily focuses on how to access the Progress Explorer to configure the DataServer for MS SQL Server in Windows. It briefly presents a few examples about how to connect the DataServer. However, for more complete connection instructions, see the “Starting and stopping a broker process from the OpenEdge Explorer and Progress Explorer and connecting a client” section on page 6–3. For information about the Progress Explorer, see the Progress Explorer online help.

To access the Progress Explorer in Windows:

1. Make sure that the AdminService is running.

2. Start the Progress Explorer. Choose **Start**→ **All Programs**→ **OpenEdge**→ **Progress Explorer Tool**.

3. Connect to localhost.

4. From the Progress Explorer’s left pane, select the **MSS DataServer** folder and double-click. The list of existing DataServer brokers for MS SQL Server 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 MS SQL Server installation provides one predefined DataServer Broker (mssbroker1) and one predefined NameServer (NS1). (The NameServer is located in the NameServer folder.) 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.

Proceed to Step a to use the DataServer for MS SQL Server broker instance with a controlling NameServer (as provided by a default Data Service). Otherwise, proceed to Step b to use the broker instances or without the NameServer (using the -DirectConnect parameter option):

a. To use the controlling NameServer to manage your broker instances, use the Data Service (This is the default and recommended option. For syntax and examples, see the “Connecting to a MS SQL Server DataServer broker using the NameServer” section on page 5–7.) As a result, the DataServer client’s initial connection is to the NameServer.

b. To connect the DataServer directly to the broker instance, do one of the following:

   - Set the -DataService value to none in the connection parameters of your schema holder.
     
     If you plan to always use a -DataService value of none, remove the controlling NameServer from your broker instance definition.

   - Add the -DirectConnect parameter to override your -DataService parameter.

See 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 connecting the DataServer to the NameServer and the broker.

**Connecting to a MS SQL Server DataServer broker using the NameServer**

This section presents an example of how to connect to a MS SQL Server DataServer using the NameServer.

To connect to a MS SQL Server DataServer using the NameServer:

1. Create a schema holder.
2. Set your MS SQL Server instance username (-U) and password (-P) connection parameters for the schema holder.

Progress Software Corporation recommends that you use the -U and -P convention to establish the username and password.

3. Set up your connection, using parameters to run the schema holder. This step shows examples from which you can model your connection:

   a. The following example shows how to set up the connection in Windows in a single-user mode:

   ```
   prowin32 f:\wrk\holder -1 -RO -db MSS_DB_DSN -ld MSS_DB_INS -dt MSS -H MSS_DSRV_Host -S 5162 -DataService mssbroker1 -Dsrv SVUB,1
   ```

   Note that although the schema holder is defined as a single-user mode, access to the MS SQL Server instance is in multi-user mode.

   b. The following example shows how to set up the connection in Windows in a multi-user mode:

   ```
   prowin32 holder -H MSS_DSRV_Host -S Holder_Service -db MSS_DB_DSN -ld MSS_DB_INS -dt MSS -H MSS_DSRV_Host -S 5162 -DataService mssbroker1 -Dsrv SVUB,1
   ```

   c. The following example shows how to set up the connection on UNIX:

   ```
   mpro holder -H MSS_DSRV_Host -S Holder_Service -db MSS_DB_DSN -ld MSS_DB_INS -dt MSS -H MSS_DSRV_Host -S 5162 -DataService mssbroker1 -Dsrv SVUB,1
   ```

The following list identifies the values used in the examples shown in Step 3:

- **MSS_DB_DSN**: Identifies the ODBC Data Source name.
- **MSS_DB_INS**: Identifies the MS SQL Server instance name.
- **MSS_Dsrv_host**: Identifies the name of the MSS DataServer broker host.
- **5162**: Identifies the NS1 NameServer default port where the MSS broker is registered.
- **Holder_Service**: Identifies the service name used to server the database holder; however, it can also be a port number.
For details about the database connection parameters noted in each example in Step 3 of this procedure, see the “Connecting a schema holder at startup” section on page 6–9.

**Caution: Do not** simultaneously run some DataServers for MS SQL Server 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 (PSC) 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, PSC 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 MS SQL Server database. See the “Creating a schema holder” section on page 5–15 for instructions.

**Configuring from the command line**

You can configure a remote DataServer broker process from the command line. You can use the _probrkr.exe_ executable or use the _mssman.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 5–4 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 _mssman.bat_ 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 your environment variables in the same environment (DOS shell) from which you plan to run the DataServer.
Table 5–2 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 dataserv.lg.</td>
</tr>
<tr>
<td>MSSSRV</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 mssman, 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 mssman.bat, 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>
<tr>
<td>UNIQUE_DSLOG</td>
<td>Used to generate unique log file name on the server with dataserv_&lt;pid&gt;.lg, where pid is the process-id of server. When set, the designated DataServerlog log file name becomes a root name. Can be disabled by unsettening the environment variable.</td>
</tr>
</tbody>
</table>

Once you have completely set up your environment, you can build the schema holder for your MS SQL Server database and connect using the client executable. See the “Creating a schema holder” section on page 5–15 for instructions.

**The ubroker.properties file**

When using the Progress Explorer administration framework, you configure the DataServer for MS SQL Server by editing the $OpenEdge-install-dir\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 MS SQL Server, you provide information that enables the host to start a broker that spawns the appropriate DataServer process (_mssrv.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. 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 5–3 describes the sections in the `ubroker.properties` file that apply to the DataServer for MS SQL Server. The file configures a default NameServer named `NameServer.NS1` and a default broker named `mssbroker1`, which you can use either without editing 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>
<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></td>
<td>NameServer.NS1</td>
<td>A sample property entity of a NameServer instance. It defines property settings for this NameServer instance.</td>
</tr>
<tr>
<td></td>
<td>Environment.NS1</td>
<td>A sample environment entity of a NameServer instance. It defines environment variable settings for this NameServer instance.</td>
</tr>
<tr>
<td>DataServer</td>
<td>UBroker.MS</td>
<td>The parent entity of DataServer for MS SQL Server brokers. It defines default property settings for all of these broker instances.</td>
</tr>
</tbody>
</table>
Configuring the DataServer

**Table 5–3: DataServer for MS SQL Server sections of the ubroker.properties file**

<table>
<thead>
<tr>
<th>Product</th>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
</table>
| DataServer  | UBroker.MS.mssbroker1       | A sample property entity of a DataServer for MS SQL Server broker instance. It defines default property settings for the broker instance named mssbroker1. Note that although many of the settings in this section can work in your environment, some of them are for demonstration purposes only. You must specify the appropriate settings for the following:  
  - `srvrExecFile`: Specify the pathname of the DataServer executable that the broker runs.  
  - `@{Startup}\DLC\bin\_mssrv.exe`, the default, which runs the default DataServer from your OpenEdge install path.  
  - `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.  
  - `srvrDSLogFile`: To specify the name of dataserver log file. If not specified default name as dataserv.lg used.  
  - `srvrmInport`: To specify maximum port number for server.  
  - `srvrmaxport`: To specify minimum port number for server. |
| DataServer  | Environment.mssbroker1      | A sample environment entity of a DataServer for MS SQL Server broker instance. It defines environment variable settings that apply to the broker instance named mssbroker1. 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 a remote DataServer” section on page 5–6 for the environment variable settings required to configure the DataServer for MS SQL Server. |
The following example illustrates the DataServer sections of the `ubroker.properties` file:

```properties
# Default properties for broker instances serving MSS DataServers
#
[UBroker.MS]
srvrExecFile="@{Startup\DLC}\bin\_msssrv.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=4446
defaultService=0
initialSrvrInstance=0
minSrvrInstance=0
maxSrvrInstance=256
brkrLoggingLevel=3
description=MSS DataServer Broker

# Sample MSS DataServer Broker definition
#
[UBroker.MS.mssbroker1]
srvrExecFile="@{Startup\DLC}\bin\_msssrv.exe"
srvrStartupParam=-svub -S X -N TCP -U X -P X -hs 0 -s 40
srvrLogFile=@{WorkPath}\mssbroker1.server.log
brokerLogFile=@{WorkPath}\mssbroker1.broker.log
portNumber=4446
defaultService=1
appservice NamenList=mssbroker1
controllingNameServer=NS1
environment=mssbroker1
uuid=172.18.103.53:32:e031b1e7bc:-7d29
description=A sample MSS DataServer Broker

# Environment for MSS Dataserver Broker: mssbroker1
#
[Environment.mssbroker1]
DSLOGDIR=@{WorkPath}
```

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

The `ubroker.properties` file is read on startup of the AdminService process. For changes in any used environment variables to take effect, the AdminService 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 MS SQL Server, you can validate the file by using the `mssconfig.bat` utility or the `mergeprop` utility. For more information about the `MSSCONFIG` utility, see Appendix B, “Server Related Command Line Utilities and Startup Parameters.” For details about the `mergeprop` utility, see *OpenEdge Getting Started: Installation and Configuration.*
Configuring multiple brokers

You can configure multiple brokers by adding more UBroker.MS.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.MS (parent entity) section, but can override them.

If you want to access multiple MS SQL Server databases 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 MS SQL Server 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 OpenEdge Explorer and Progress Explorer and connecting a client” section on page 6–3 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 MS SQL Server 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 (PSC) 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, PSC recommends that you reintroduce the NameServer into your configuration.
Creating a schema holder

A schema holder contains a schema image of the MS SQL Server database 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 OpenEdge 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 server permissions needed 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.

Permission requirements

When you use the DataServer to create a schema image in a schema holder, it accesses the associated MS SQL Server database. 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 privileges to use the SELECT statement on certain data source objects to perform certain tasks. In this chapter, these privileges are referred to as SELECT privileges.
Permissions for connecting a schema holder

When you connect to a schema holder, you can also choose to connect to your MS SQL Server database. To connect to the data source, you must provide a valid login name and password combination. Use the User ID (-u) parameter to provide the user ID and the Password (-P) parameter to provide the password. Figure 5–1 shows an example of the User ID and Password dialog box.

![User ID and Password dialog box](image)

Figure 5–1: User ID and Password dialog box

Note that value xxxxxx that appears in Figure 5–1 is only a placeholder for the actual MS SQL Server database name to which you intend to connect.

Application-specific permissions

In addition to the MS SQL Server database 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 OpenEdge application that queries but does not update the employee table in the 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 select, 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 tables that the application accesses
- The type of access required to those tables

Preparing to create the schema holder

This section addresses the steps required to create the schema holder.

To prepare to create the schema holder:

1. Verify that your MS SQL Server database is accessible and that you can connect to it. Transact-SQL and the MS SQL Server Query Analyzer can serve as a test for connectivity.
2. Verify that you have installed the ODBC drivers.
3. Once you have configured your data source (DSN), make sure that you can establish a connection independent of using the DataServer. From your DSN configuration, select to test the connection to MS SQL Server using the ODBC interface.
4. Start the DataServer as described in either the “Starting a local DataServer” section on page 6–2 or the “Starting a remote DataServer” section on page 6–3.

5. Open the Data Administration or the character Data Dictionary.

Creating a new empty OpenEdge database

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

If you create a new OpenEdge application to be Unicode-enabled, ensure that you create the database from a Unicode source. A copy of the empty OpenEdge UTF-8 database can be obtained from $DLC/prolang/utf/empty. If you choose UTF-8 as your schema image code page, your schema holder’s code page must also be UTF-8. If you create a new OpenEdge application to be Unicode-enabled, ensure that you create the database from a Unicode source. A copy of the empty OpenEdge UTF-8 database can be obtained from $DLC/prolang/utf/empty. If you choose UTF-8 as your schema image code page, your schema holder’s code page must also be UTF-8.

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

1. Start the Data Administration.

2. Select the Database→Create option. The Create Database dialog box appears:

3. Type the schema-holder name (for example, mssholder) in the New Physical Database Name field.

4. Select the An EMPTY Database option.

5. Choose OK. The following dialog box appears. By default, the name of the newly created data source appears in the Physical Name field, as shown:
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. Choose OK to connect the empty OpenEdge database and return to the Data Administration main window.

Creating a schema holder

This section addresses the steps needed to create a schema holder.

To create the schema holder:

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

2. In the Logical Database Name field, type the name that you will use to connect to your 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 MS SQL Server database 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. If you choose UTF-8 as your schema image code page, your schema holder’s code page must also be UTF-8.
Table 5–4 lists the most common MS SQL Server database code pages and the equivalent OpenEdge names.

### Table 5–4: MS SQL Server and OpenEdge code pages

<table>
<thead>
<tr>
<th>MS SQL Server 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 “Code pages” section on page 2–5 for a discussion of collation issues to consider.

5. In the **Case Insensitive** field, the default value **yes** indicates that MS SQL Server’s case insensitivity feature is in use. To change this value, type **no**.

6. Type the connection parameters in the **Connection Parameters** field.

   See Chapter 6, “Connecting the DataServer,” for a description of the required and optional connection parameters.

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

8. Choose **OK**. The utility prompts you for your data source user ID and password. If they are required by the MS SQL Server data source and you did not provide the user ID and password in the **Connection Parameters** field (see Step 6), enter a valid data-source user ID and password combination now. For more information, see the “Permission requirements” section on page 5–15.

9. Choose **OK**. When the DataServer connects to the MS SQL Server database, it reads information about data source objects. The following dialog box appears:
You can select tables based on the object name, owner/library information, and qualifier. For example, you can specify A* in the **Object Name** field to list all the tables whose names begin with A or a.

**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.)

Check **Default to OpenEdge DATETIME** to automatically map MS SQL Server data types to the associated OpenEdge data type. If you have modified your client application to handle LOB data types, check **Default to OpenEdge LOB** to map the OpenEdge LOB data type to MS SQL Server `VARBINARY (MAX)`, `IMAGE` and `FILESTREAM` data types. For more information on mapping OpenEdge and MS SQL Server data types, see the “Support for OpenEdge ABL BLOB data type” section on page 2–30.

10. Choose **OK**. OpenEdge displays a list of the data source objects that you can include in the schema holder, as shown:

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.

11. 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 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.

12. Choose **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 table selected, the DataServer attempts to select an index to support the OpenEdge ROWID. If an appropriate index does not exist, the DataServer issues the warning, **Please check errors, 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 7–35 for instructions. For additional information, see the “Indexes and sorting” section on page 2–11.
Maintaining a schema holder

The Data Administration provides a set of DataServer utilities that you can use to maintain a schema holder. Chapter 7, “The DataServer Tutorial,” describes these utilities.

**Note:** Keep in mind that you must have `SELECT` privileges on certain data source objects to update a schema holder. See the “Permission requirements” section on page 5–15 for details.

If you make changes to a MS SQL Server database, 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 a MS SQL Server database. 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 MS SQL Server database.
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 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 the MS SQL Server 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 with either the MS SQL Server driver or an OpenEdge branded ODBC Wire Protocol driver for MS SQL Server. 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 see 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 brokерыervice -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 a remote DataServer” section on page 5–6.

### 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 as you did in Step 4 of the “Starting a broker” section on page 5–23, type the following example command line 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 the MS SQL Server DataServer broker

The activity described in this section concludes setting up 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.

Type 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 MSS -ld logical-datasrc-name -H hostname -S brokерыervice -N tcp -U userID -P password
```
Typical configuration for a remote client to connect to a remote DataServer

Type the following command line entries for a remote GUI client machine:

```
prowin32 schemaholdername -H hostname -S databaseservice -N tcp -d dsn_name -dt MSS -l 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 the following methods:

- Progress Explorer tool (in Windows platforms only)
- Mergeprop utilities (command-line alternative to the OpenEdge Explorer and Progress Explorer tool)
- Progress Explorer Command line utilities (MSSMAN and MSSCONFIG)
- Command line statements (_probrkr.exe)

This chapter describes how to connect the DataServer using these methods, as detailed in the following sections:

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

This section describes how to start a local DataServer for MS SQL Server from the startup command.

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 the client:

1. Make sure that your ODBC drivers are installed and configured properly.
2. Start your MS SQL Server database.
3. Set any environment variables required for your configuration as described in the relevant section of Chapter 5, “Configuring the DataServer” (for example, D$LOGDIR). You must set them in the environment from which you are starting the DataServer.

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

4. Enter the following command:

```
prowin32 schema-holder-name -db datasrc-name -dt MSS -ld mss-logical-db-name -U userID -P password
```

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

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

This section describes how to start and stop a remote DataServer for MS SQL Server. Starting a remote DataServer involves starting the processes that make up the remote DataServer.

To start a remote DataServer:

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

2. Start an OpenEdge client process on a UNIX machine or in a PC running Windows that connects to the schema holder and the MS SQL Server database.

Starting and stopping a broker process from the OpenEdge Explorer and Progress Explorer and connecting a client

You can use the Progress Explorer to administer the server module (_mssrv.exe) of the DataServer in Windows.

Performing from 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.

Performing 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 MS SQL Server database in a standard ProBroker configuration. In addition:

- 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 only, 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 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 ProBroker 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 mssholder -db mssdemo -dt MSS -U bob -P bobpass -H host1 -S mssport -DataService mssbroker1 -Dsrv SVUB,1.
Starting a remote DataServer

Starting and stopping a broker process using MSSMAN

You can use the MSSMAN utility to start a DataServer broker on a Windows host. Prior to using MSSMAN, verify that the AdminService is running.

**To start and shut down the DataServer:**

1. To start the DataServer broker, enter this command at the system prompt on the machine where the broker will run:

   ```
   mssman -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:

   ```
   mssman -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:

   ```
   mssman -name broker-name -stop
   ```

   You can stop a broker on a remote machine by adding the `-host` and `-user` options.

For more information about the MSSMAN utility and a description of all the command options, see Appendix B, “Server Related Command Line Utilities and Startup Parameters.”
Starting and stopping a broker process from the command line

This section addresses how you can start and stop a broker process from the command line.

To start and stop a broker process from the command line:

1. Set the environment variable `MSSRV` to the name of the executable (including the path) of the DataServer for MS SQL Server. 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 `\OpenEdge-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 in Windows or on a UNIX machine. See the “Starting the Windows client process” section on page 6–6 or the “Starting the UNIX client process” section on page 6–7 for instructions.

Starting the Windows client process

Start the OpenEdge client process on your Windows machine by running the `prowin32.exe` 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

3. The connection parameters required by the remote DataServer configuration
For example, a command line for an OpenEdge client process that you use to access an MS SQL Server database might look like this:

```
prowin32 mssholder -RO -db demo -dt MSS -ld demo -H host1 -S oserviceA -U bob -P bobpass
```

Create a program icon for each unique set of connection parameters. See the “Connecting a schema holder at startup” section on page 6–9 for command-line information and more examples.

### Starting the UNIX client process

After starting the remote broker, you start the OpenEdge client process on a 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 *mssholder*, and connects the MS SQL Server demo database using the *mssdemo* DSN with the user *bob* whose password is *bobpass*:

```
pro mssholder -RO -db demo -dt MSS -ld mssdemo -H host1 -S oserviceA -U bob -P bobpass
```

See the “Connecting a schema holder at startup” section on page 6–9 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 MS SQL Server database 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, proceed to connect the schema holder and then the 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 MS SQL Server database validates against the list of users in the syslogins table. The request typically comes from a nontrusted 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 MS SQL Server database 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 MS SQL Server database performs no additional validation.

- **Mixed** — Accepts requests from trusted or nontrusted connections:
  - 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 MS SQL Server database accepts the connection.
  - If the connection is nontrusted, the OpenEdge client must provide the user ID and password.

Progress Software Corporation recommends the following guidelines for working with a MS SQL Server database and Windows security:

- Configure a MS SQL Server database 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 can then select your MS SQL Server database as your working database and connect to it. You cannot use the Utilities → Edit OpenEdge Auto–Connect option to connect to a MS SQL Server 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 MS SQL Server database and related parameters.

This command connects a schema holder named mssholder and MS SQL Server database named mssdemo:

```
CONNECT mssholder -1 -db mssdemo -dt MSS -Dsrv qt_debug,EXTENDED
```

- Use connection parameters when you start OpenEdge. You can include these parameters on the program item command line or in a parameter file that you specify when you start OpenEdge. A parameter file is portable and easy to maintain. For information on creating a parameter file, see OpenEdge Deployment: Startup Command and Parameter Reference.

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 OpenEdge CONNECT statement. If the proper connect information is stored in the schema holder, referencing a table in the MS SQL Server database will cause the DataServer to issue the connect statement automatically. Any combination of connection techniques works, as long as you first connect to the schema holder and then the MS SQL Server 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 MS SQL Server database 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 MS SQL Server database connection after the -db parameter.

The following section explains how to connect both a schema holder and a MS SQL Server database 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 a MS SQL Server database at startup. These parameters control how your system connects to a database. If the DataServer runs in a remote configuration, your startup command or parameter file always includes parameters that control networking options.
Table 6–1 describes the database connection parameters that you use when you connect to a schema holder and a MS SQL Server database through the DataServer.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Database Type MSS</strong> (<strong>-dt MSS</strong>)</td>
<td>Optional</td>
<td>Specifies that the type of the target data source is MS SQL Server. If not specified, the server will resolve the database type.</td>
</tr>
<tr>
<td><strong>Physical Database Name</strong> (<strong>-db</strong>)</td>
<td>Required</td>
<td>Indicates the name by which OpenEdge recognizes the MS SQL Server database to which you want to connect. This name must match the name that you used when you registered the data source as a MS SQL Server database.</td>
</tr>
<tr>
<td><strong>Logical Database Name</strong> (<strong>-ld</strong>)</td>
<td>Optional</td>
<td>Specifies the logical name of the MS SQL Server database. 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 MS SQL Server demo database as mydemo. In this case, the physical name is demo, and the logical name is mydemo. If not specified, the logical database name is assumed to be the same as the physical name. An error will be generated if the physical name and logical name do not match when this parameter is omitted.</td>
</tr>
<tr>
<td><strong>Host Name</strong> (<strong>-H</strong>)</td>
<td>Required for remote DataServer</td>
<td>Indicates the name of the Windows host machine in the network.</td>
</tr>
<tr>
<td><strong>Service Name</strong> (<strong>-S</strong>)</td>
<td>Required for remote DataServer</td>
<td>Indicates the name of the service you are calling. If you use NameServer with Progress Explorer, specify the service name or port number on the host machine where the NameServer resides. If you are using ProBroker or the Progress Explorer without a NameServer, specify the service name or port number on the host machine where the broker resides.</td>
</tr>
<tr>
<td><strong>User ID</strong> (<strong>-U</strong>)</td>
<td>Required</td>
<td>Supplies the login name that the DataServer uses to log into the MS SQL Server database.</td>
</tr>
<tr>
<td><strong>Explorer Redirection</strong> (<strong>-Dsrv SVUB,1</strong>)</td>
<td>Required for OpenEdge Explorer and Progress Explorer connections</td>
<td>Redirects connection logic to the OpenEdge Explorer and Progress Explorer instead of ProBroker. This parameter must be set if the -DataService connection parameter is to also be used for connectivity. For more information see the “Starting and stopping a broker process from the OpenEdge Explorer and Progress Explorer and connecting a client” section on page 6–3. Also refer to the “Using MS SQL Server and DataServer options” section on page 8–6.</td>
</tr>
<tr>
<td><strong>Data Service</strong> (<strong>-DataService</strong>)</td>
<td>Required for Progress Explorer connections</td>
<td>Specifies the data service the NameServer uses. For more information, see the “Starting and stopping a broker process from the OpenEdge Explorer and Progress Explorer and connecting a client” section on page 6–3.</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.

### Table 6–1: DataServer connection parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct Connect (<code>-DirectConnect</code>)</td>
<td>Optional</td>
<td>Instructs the client to bypass the controlling NameServer and connect directly to the MS SQL Server DataServer Broker. If not specified, the connection is made through the NameServer.</td>
</tr>
<tr>
<td>Password (<code>-P</code>)</td>
<td>Required</td>
<td>Supplies the password that the DataServer uses to log into the MS SQL Server database. Different login name and password combinations allow for different levels of user privileges.</td>
</tr>
<tr>
<td>Single–User Mode (<code>-1</code>)</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. If not specified, <code>-RO</code> or remote connection parameters are required.</td>
</tr>
<tr>
<td>Read-Only (<code>-RO</code>)</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. If not specified, the database is opened for update as well as read access.</td>
</tr>
<tr>
<td>Local Cache (<code>-cache</code>)</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. If not specified, no cache file is used.</td>
</tr>
<tr>
<td>DataServer (<code>-Dsrv</code>)</td>
<td>See note</td>
<td>Specifies options with which you control your ODBC Driver and DataServer environment. See the “DataServer options” section on page 8–4 and the “Using MS SQL Server and DataServer options” section on page 8–6 for more information. <strong>Note:</strong> When you specify a list of <code>-Dsrv</code> parameters, be sure not to include any spaces anywhere in this list.</td>
</tr>
<tr>
<td>Server Join (<code>-nojoinbysqldb</code>)</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 the query of an OpenEdge database. Use <code>-nojoinbysqldb</code> at startup time. If not specified, the server will perform joins, but results may not be consistent with identical joins run against an OpenEdge database.</td>
</tr>
</tbody>
</table>
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

Use the following general command syntax to start OpenEdge:

- In single-user mode
- In a local DataServer configuration
- With a local schema holder connected
- With a MS SQL Server database connected

You can type these commands on the command line of a program item property box.

The following examples start OpenEdge in a local DataServer configuration:

- The schema holder’s physical name is `mssholder` and it is read-only.
- The physical data source name and the MS SQL Server database name is `sports`.
- The data source type is `MSS` (Microsoft SQL Server).
- The logical database name is `mysport`.
- The user ID is `bob`.
- The password is `bobpass`.
- Assorted `-Dsrv` options are specified.

Using a remote DataServer configuration

The examples in the first part of this section show how to use specific command syntax to start OpenEdge in the following modes:

- In single-user mode
- In a remote-DataServer configuration
- With a local schema holder connected
- With a MS SQL Server database connected
Note the following command syntax used in a Windows client:

**Syntax**

```
prowin32 schema-holder-name -1 -db datasrc-name -dt MSS
   -ld logical-datasrc-name -H hostname -S service-name
   -U userID -P password -Dsrv qt_debug,EXTENDED
```

Note the following command used on a UNIX client:

**Syntax**

```
pro schema-holder-name -db datasrc-name -dt MSS
   -ld logical-datasrc-name -H hostname -S service-name
   -U userID -P password -Dsrv qt_debug,EXTENDED
```

The examples in the second part of this section allow you to start OpenEdge in a remote DataServer configuration:

- The schema holder’s physical name is `mssholder` and it is read-only.
- The data-source name is `sports`.
- The data source type is `MSS` (Microsoft SQL Server).
- The logical data source name is `mydemo`.
- The host name is `host1`.
- The service name is `mssport`.
- The user ID is `bob`.
- The password is `bobpass`.

Note the following command used in a Windows client:

```
prowin32 mssholder -RO -db sports -dt MSS -ld mydemo -H host1
   -S mssport -U bob -P bobpass -Dsrv qt_debug,EXTENDED
```

Note the following command used on a UNIX client:

```
pro mssholder -RO -db sports -dt MSS -ld mydemo -H host1
   -S mssport -U bob -P bobpass -Dsrv qt_debug,EXTENDED
```
Unsupported connection parameters

You cannot use the following OpenEdge connection parameters when connecting to a MS SQL Server database through the DataServer. If you do specify them, OpenEdge and WebSpeed ignore them:

- Blocks in Database Buffers (-B)
- Before–image Filename (-g)
- Lock-table Entries (-L)
- Number of Users (-n)
- Buffered I/O—UNIX only (-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 MS SQL Server passes the connection string specified by the PRGRS_CONNECT option directly through to the ODBC driver without modification. For more information and a complete list of keywords, refer to the Microsoft ODBC programming reference documentation. The 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 (;). There can be no spaces within connection-string. If any of the name/value pairs does include a space, substitute the value &^ for the space.

Value pairs within the connect string are delimited by a semicolon (;).

Note: PRGRS_CO 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:

```plaintext
CONNECT datasrc-name -1d logical-datasrc-name -dt MSS
-Dsrv
PRGRS_CO,DSN=datasrc-name;server=servername;UID=name;PWD=password;
```

For `datasrc-name`, supply the name of the MS SQL Server database. 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 a MS SQL Server database 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:

```plaintext
CONNECT datasrc-name -1d logical-database-name -dt MSS
-Dsrv PRGRS_CO,data-&^source&^name;.
```

- To connect to the MS SQL Server database using the ODBC driver as a guide, specify an empty PRGRS_CONNECT, which tells the ODBC driver to handle the entire connection process interactively. For example:

```plaintext
CONNECT datasrc-name -1d logical-name -dt MSS
-Dsrv PRGRS_CO,;.
```

To simplify your PRGRS_CONNECT string, you can specify a File DSN. When you configure access to your MS SQL Server database, specify a File DSN and then reference that file in your connect string. For example:

```plaintext
CONNECT datasrc-name -1d logical-name -dt MSS
-Dsrv PRGRS_CO,FILEDSN=dsn-filename;.
```

**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:

**Syntax**

-Dsrv BINDING, n;

Table 6–2 describes the BINDING options that you can specify with the -Dsrv parameter.

**Table 6–2: Connection BINDING options**

<table>
<thead>
<tr>
<th>BINDING value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Indicates that binding is not enabled at all.</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 is the default value.</td>
</tr>
</tbody>
</table>

**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 4–9 for more information.

You override query-tuning defaults with the DataServer (-Dsrv) connection parameter when you connect to a MS SQL Server database. The syntax is:

**Syntax**

CONNECT data-source-name -dt MSS
-Dsrv query-tuning-option1,value1
-Dsrv query-tuning-option2,value2.

An alternate syntax for the -Dsrv parameter is as follows:

**Syntax**

CONNECT data-source-name -dt MSS -Dsrv query-tuning-option1,value1,query-tuning-option2,value2.
Connecting a schema holder

Table 6–3 describes the query-tuning options that you can specify with the -Dsrv parameter.

**Table 6–3: Connection query-tuning options**

<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 generates for the query to the dataserv.lg file. The default is qt_no_debug, to supply no debugging information. To override the default, specify qt_debug.option as follows:</td>
</tr>
<tr>
<td>qt_debug,SQL</td>
<td>• Specify qt_debug.SQL to record the SQL sent to the ODBC driver in the dataserv.lg file. Note that this SQL contains place holders for values which will be bound to the statement when sent to the data source</td>
</tr>
<tr>
<td>qt_debug,EXTENDED</td>
<td>• Specify qt_debug.EXTENDED to print information such as cursor statistics in addition to the SQL statements executed by the DataServer</td>
</tr>
<tr>
<td>qt_debug,CURSOR</td>
<td>• Specify qt_debug.CURSOR to print information about the cursors that the DataServer uses for internal calls and for opening queries</td>
</tr>
<tr>
<td>qt_debug,PERFORMANCE</td>
<td>• Specify qt_debug.PERFORMANCE to print information on the amount of time that certain operations take</td>
</tr>
<tr>
<td>qt_debug,CALL_SUMMARY</td>
<td>• Specify qt_debug,CALL_SUMMARY to print information on cursors and timing</td>
</tr>
<tr>
<td>qt_debug,VERBOSE</td>
<td>• Specify qt_debug,VERBOSE to print all of the information gathered by the other qt_debug options</td>
</tr>
<tr>
<td></td>
<td>For more detailed descriptions of these options, see the Appendix D, “Using qt_debug to Analyze Performance.” For information on Enhanced Logger options that replace the qt_debug options, see “Analyzing performance with the Enhanced Logger” section on page 6–18.</td>
</tr>
<tr>
<td>qt_lookahead</td>
<td>Specifies whether the DataServer uses lookahead or standard cursors. To generate efficient queries, qt_lookahead is the default in the following cases:</td>
</tr>
<tr>
<td>qt_no_lookahead</td>
<td>• Statements that use NO-LOCK</td>
</tr>
<tr>
<td></td>
<td>• Statements that use SHARE-LOCK with transaction isolation level set to read uncommitted</td>
</tr>
<tr>
<td></td>
<td>Specify qt_no_lookahead for query behavior that is consistent with an OpenEdge database.</td>
</tr>
<tr>
<td>qt_separate_connection</td>
<td>Specifies whether each cursor should use a separate connection to the MS SQL Server database. The default is qt_no_separate_connection, which provides behavior that is consistent with an OpenEdge database. Specify qt_separate_connection to use a separate connection. Executing cursors in separate connections can improve performance because the DataServer does not have to restart the cursors.</td>
</tr>
<tr>
<td>qt_no_separate_connection</td>
<td></td>
</tr>
<tr>
<td>qt_cache_size, integer</td>
<td>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.</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: 10,000 when block cursors are enabled (the default). 30,000 when block cursors are disabled.</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 MSS (Microsoft SQL Server) for each cursor and writes an extended report on the SQL statements it executes, as shown:

```
CONNECT holder -db infdb -dt MSS -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 queries run against an OpenEdge database. See Chapter 2, “Initial Programming Considerations,” for more information.

### Analyzing performance with the Enhanced Logger

The Enhanced Logger infrastructure provides a means for either standard reporting of run-time activity or logging diagnostic data for troubleshooting problems based on a set of logging characteristics. The DataServer can be enabled to write information to a log file. You control the type and amount of information logged using a combination of log entry types and logging levels. Log entry types are categories of information written to the log file. Logging levels control the amount of logging information for any given log entry type.

The legacy `qt_debug` logging infrastructure only logged server information but the Enhanced Logger framework for DataServers offers two logging contexts:

- Server context for Server-specific log entries
- Client context, which shares logging information with OpenEdge clients and AppServer agents.

### LOG-MANAGER system handle attributes and methods

The LOG-MANAGER system handle controls logging settings for the OpenEdge client and DataServer client contexts. The LOG-MANAGER attributes and methods allow you to programmatically control the logging levels and log entry types during the execution of your program.

Table 6–4 lists the function of each LOG-MANAGER system handle attribute.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Access</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTRY-TYPES-LIST</td>
<td>Readable</td>
<td>Returns a list of all valid entry types.</td>
</tr>
<tr>
<td>LOG-ENTRY-TYPES</td>
<td>Readable/Writable</td>
<td>Returns a list of one or more types of log entries.</td>
</tr>
<tr>
<td>LOGFILE-NAME</td>
<td>Readable/Writable</td>
<td>Returns the name of the log file.</td>
</tr>
<tr>
<td>LOGGING-LEVEL</td>
<td>Readable/Writable</td>
<td>Returns the level at which log entries are written.</td>
</tr>
</tbody>
</table>

Table 6–4: LOG-MANAGER system handle attributes (1 of 2)
Table 6–5 lists the function of each LOG-MANAGER system handle method.

### Table 6–5: LOG-MANAGER system handle methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR-LOG( )</td>
<td>Clears all messages existing in the current client log file and leaves the file open for writing</td>
</tr>
<tr>
<td>WRITE-MESSAGE( )</td>
<td>Writes user messages to the current log file</td>
</tr>
<tr>
<td>CLOSE-LOG( )</td>
<td>Stops an interactive or batch OpenEdge client from writing messages to the current client log file and closes the log file</td>
</tr>
</tbody>
</table>

**Note:** The CLOSE-LOG( ) method is valid only for interactive and batch clients. WebSpeed agents and AppServer servers write a message to the server log file indicating that it is invalid to use the CLOSE-LOG( ) method to close a WebSpeed or AppServer server log file. In this case, the method returns FALSE.

For more information on the LOG-MANAGER system handle or its attributes and methods, see the LOG-MANAGER system handle reference entry in *OpenEdge Development: ABL Reference.*

### DSLOG-MANAGER system handle attributes

The DSLOG–MANAGER system handle is similar to the LOG–MANAGER system handle. Is is used to control the logging settings for the server context of the OpenEdge DataServer logging infrastructure, while the LOG–MANAGER controls logging settings for OpenEdge Client and DataServer client context. Although both provide attributes and methods for enabling, disabling, and changing logging capabilities, the DSLOG–MANAGER applies only to DataServer components.

The DB–CONTEXT attribute must be set before any other DSLOG–MANAGER attribute can be accessed.

Table 6–6 lists the DSLOG–MANAGER system handle attributes.
### Table 6–6: DSLOG-MANAGER system handle attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Access</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB-CONTEXT¹</td>
<td>Readable/Writable</td>
<td>Specifies the logical database name of a valid and connected data source for a supported db-type. It set the system handle to a specific server context to enable access to other DSLOG-MANAGER attributes and methods. You can also set it to the alias of a connected data source.</td>
</tr>
<tr>
<td>ENTRY-TYPES-LIST</td>
<td>Readable</td>
<td>Returns a list of all valid entry types.</td>
</tr>
<tr>
<td>LOG-ENTRY-TYPES</td>
<td>Readable/Writable</td>
<td>Returns a list of one or more types of log entries. The default value returned by the LOG-ENTRY-TYPES attribute are the logentrytypes specified in the -Dsrv parameter when the foreign data source was connected.</td>
</tr>
<tr>
<td>LOGFILE-NAME²</td>
<td>Readable</td>
<td>Returns the name of the log file. By default, the DSLOG-MANAGER uses the dataserv.lg log file. You can change the file name by using the -dslog startup parameter from the OpenEdge client or the srvrDSLLogFile UBROKER property from the DataServer broker instance. The server context cannot work without a log file opened so it cannot be changed at runtime. This would risk having no log file opened in the event that the new log file name failed to open.</td>
</tr>
<tr>
<td>LOGGING-LEVEL</td>
<td>Readable/Writable</td>
<td>Returns the level at which log entries are written. The default value returned by the LOGGING-LEVEL attribute is the logginglevel specified in the -Dsrv parameter when the foreign data source was connected.</td>
</tr>
<tr>
<td>LOG-THRESHOLD³</td>
<td>Readable</td>
<td>Returns the file size threshold of log files. When the current log file becomes equal to or greater than the specified size, OpenEdge renames and saves the log file and creates a new log file.</td>
</tr>
<tr>
<td>NUM-LOG-FILES⁴</td>
<td>Readable</td>
<td>Returns the number of rolled over log files to keep on disk at any one time, for OpenEdge session, including the current log file.</td>
</tr>
<tr>
<td>HANDLE</td>
<td>Readable</td>
<td>Returns a handle.</td>
</tr>
<tr>
<td>INSTANTIATING-PROCEDURE</td>
<td>Readable</td>
<td>Returns a procedure handle for the first, or instantiating, procedure.</td>
</tr>
<tr>
<td>TYPE</td>
<td>Readable</td>
<td>Returns the type of handle.</td>
</tr>
</tbody>
</table>

1. Each connection to a DataServer is its own entity. The DB-CONTEXT attribute enables you to provide different logging settings for each data source connection. The DB-CONTEXT attribute must be set to the logical database name or alias which is currently connected, prior to setting any other attributes. The attribute defaults to the unknown value. If DB-CONTEXT does not contain the value of a valid logical database name, any attempt to access other attributes of DSLOG-MANAGER fails.

2. The value returned by the LOGFILE-NAME attribute is the default name dataserv.lg or; the name specified by the -dslog client startup parameter for self-service clients; the name specified by the -dslog startup parameter when ProBroker is started; the name specified by the srvrDSLLogFile attribute in the ubroker.properties file when a Unified Broker instance is started for DataServer connections.
Connecting a schema holder

By default, the DSLOG-MANAGER uses the dataserv.1g log file. You can change the file name by using the -dslog startup parameter from the OpenEdge client or the srvrDSLogFile UBROKER property from the DataServer broker instance.

The following example tests for an active connection before setting the DSLOG-MANAGER to that connected DataServer context and getting a list of log entry types and setting a specific message inside the application at run-time.

```
IF CONNECTED("mymss") THEN DO:
  DSLOG-MANAGER:DB-CONTEXT = 'mymss'.
  MESSAGE DS-LOGMANAGER:LOG-ENTRY-TYPES.
  DSLOG-MANAGER:WRITE-MESSAGE ("Some message after connection").
END.
```

For more information on the DSLOG-MANAGER system handle or its attributes, see the LOG-MANAGER system handle reference entry in OpenEdge Development: ABL Reference.

Table 6–7 lists the function of each LOG-MANAGER system handle method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR-LOG( )</td>
<td>Clears all messages existing in the current client log file and leaves the file open for writing</td>
</tr>
<tr>
<td>WRITE-MESSAGE( )</td>
<td>Writes user messages to the current log file</td>
</tr>
</tbody>
</table>

**Note:** The close-LOG ( ) method is not supported as the log file must be kept open while working with DataServers.

Log entry types

Log entry types represent categories of information you can write to a log file. The amount of detail written to the log file is determined by the logging level.

Table 6–8 lists the client DataServer log entry types.

<table>
<thead>
<tr>
<th>Log entry type</th>
<th>Description</th>
</tr>
</thead>
</table>
| DS.QryInfo     | Logs messages about client processing for SQL queries. By default, setting the DS.QryInfo log entry type in the client context will automatically set the SQL log entry type in the server context.
Table 6–8: Client DataServer Log entry types

<table>
<thead>
<tr>
<th>Log entry type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS.Cursor</td>
<td>Logs client-side cursor details. By default, setting the DS.Cursor log entry type in the client context will automatically set the Cursor log entry type in the server context.</td>
</tr>
<tr>
<td>4GLTrans</td>
<td>Logs client side transaction details. Level 4 of the entry type writes client context information to the log file specific to DataServers. Output from the 4GLTrans log entry type can supplement output of the TRANS log entry type in the server context.</td>
</tr>
</tbody>
</table>

For more detail on log entry types of the DataServer client context and other log entry types specific to the client context, refer to OpenEdge Development: Debugging and Troubleshooting.

Table 6–9 lists the server DataServer log entry types.

Table 6–9: Server Log entry types

<table>
<thead>
<tr>
<th>Log entry type</th>
<th>Description</th>
</tr>
</thead>
</table>
| SQL            | Logs information about SQL queries. Similar to the legacy qt_debug, SQL option.  
- Level 2 (Basic) — Logs details of SQL query preparation and execution.  
- Level 3 (Verbose) — Adds bound values when prepared and/or execute SQL utilize bind parameters. |
| Cursor         | Logs information about cursor details. Similar to the legacy qt_debug, Cursor option.  
- Level 2 (Basic) — Logs basic cursor details such as table number, client and server cursor identifiers. |
| Trans          | Logs information about transaction details.  
- Level 2 (Basic) — Logs messages when a SQL transaction begins or ends on the server or when a transaction is undone against the MS SQL Server database.  
- Level 3 (Verbose) — Adds messages for the beginning and ending of sub-transactions and when it is undone.  
- Level 4 (Extended) — Logs details about records locked in a transaction including the type of lock, lock upgrades, etc. |
Table 6–10 identifies the logging levels and entry types for client and server contexts.

Table 6–10: Setting logging levels and log entry types

<table>
<thead>
<tr>
<th>Operation Mode</th>
<th>Logging Option</th>
<th>OE Client</th>
<th>MSS DataServer Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup / connection</td>
<td>Logging level</td>
<td>-logginglevel, parameter</td>
<td>-Dsrv logginglevel, parameter</td>
</tr>
<tr>
<td></td>
<td>Log entry types</td>
<td>-logentrytypes, parameter</td>
<td>-Dsrv logentrytypes, parameter</td>
</tr>
<tr>
<td>ubroker.properties</td>
<td>Logging level</td>
<td>-logginglevel, parameter</td>
<td>-Dsrv logginglevel</td>
</tr>
<tr>
<td></td>
<td>Log entry types</td>
<td>-logentrytypes, parameter</td>
<td>-Dsrv logentrytype</td>
</tr>
<tr>
<td>Runtime setting</td>
<td>Logging level</td>
<td>LOG-MANAGER: LOGGING-LEVEL attribute</td>
<td>DSLOG-MANAGER: LOGGING-LEVEL attribute</td>
</tr>
<tr>
<td></td>
<td>Logging level</td>
<td>LOG-MANAGER: LOG-ENTRY-TYPES attribute</td>
<td>DSLOG-MANAGER: LOG-ENTRY-TYPES attribute</td>
</tr>
</tbody>
</table>

1. logginglevel and logentrytypes specified in the ubroker properties file are for AppServer agents in the client context only.
2. To specify logginglevel and logentrytypes at startup for the DataServer’s server context, continue to use the -Dsrv Startup/connection options. Logginglevel and Logentrytypes specified in the ubroker properties file for a DataServer broker instance are ignored. Specifying the -Dsrv options within the srvrStartupParam of a DataServer broker instance is also ignored.

The following code example shows how logging levels and log entry types are set using the client startup switch:

```
-logentrytypes DS.Cursor:3
-logginglevel 3
```

In the following example, logging levels and the log entry type are set using the LOG-MANAGER system handle:

```
LOG-MANAGER:LOG-ENTRY-TYPES = "DS.Cursor:3"
```
In the next example, the logging level and log entry types are defined using the -Dsrv switch.

```
-Dsrv logentrytypes,SQL:3
-Dsrv logentrytypes,SQL,logginglevel 3
```

The following example demonstrates the use of a DSLOG-MANAGER system handle with a DataServer DB-CONTEXT attribute.

```
DS-LOG-MANAGER:LOG-ENTRY-TYPES = "SQL:3"
```

**Logging levels**

You can choose one of five logging levels, as shown in Table 6-11.

<table>
<thead>
<tr>
<th>Logging Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>Errors</td>
</tr>
<tr>
<td>2</td>
<td>Basic</td>
</tr>
<tr>
<td>3</td>
<td>Verbose</td>
</tr>
<tr>
<td>4</td>
<td>Extended</td>
</tr>
</tbody>
</table>

**Setting the DataServer logging levels and log entry types**

Client context log settings can be set using:

- Start-up parameters at client startup
- LOG-MANAGER system handles in client applications

Server context settings can be set using:

- The -Dsrv switch at client startup or during a database connection.
- DSLOG-MANAGER system handle in client applications

Setting all other DataServer log settings (clientlog, dslog, numlogfiles, logthreshold and logappend)

- Start-up parameters at client startup. These settings are shared with the server context when DataServers are connected self-service.

**Note:** Of these, -dslog is not shared but simply passed onto the server.

- Ubroker.properties specified for agents at AppServer startup.
Note: AppServer client context does not include the ability to forward a DataServer log file name onto the server context.

- LOG-MANAGER system handles in client applications

Note: Client context does not include the ability to forward a DataServer log file name onto the server context.

Server context settings can be set using:

- ProBroker startup parameters (-dslog, -numlogfiles, -logthreshold, -logappend)
- Unified-Broker DataServer instance attributes (srvrDSLogFile, srvrNumLogFiles, srvrLogThreshold, srvrLogAppend)

Note: DSLOG-MANAGER system handle can be used in client applications to return the above settings but cannot modify them.

For a details on LOG-MANAGER and DSLOG-MANAGER startup parameters, refer to the LOG-MANAGER and DSLOG-MANAGER entries in OpenEdge Development: ABL Reference and OpenEdge Development: Debugging and Troubleshooting.

Server Context Log File Naming

When no log file name is specified for the server context, its default name continues to be dataserv.lg in the working directory, just as it was using the legacy qt_debug logging options.

OpenEdge client startup and ProBroker startup both allow the server context log file to be specified with the -dslog parameter. For a DataServer broker instance specified to the Unified Broker, the srvrDSLogFile attribute in the ubroker.properties file specifies the server context log file name.

If the server context log file name is specified without any path, it is placed in the working directory. If it is specified with a relative path name, it searches for that relative path under the absolute path of the working directory unless the $DSLOGDIR environment variable is set. When $DSLOGDIR is set, it overrides the working directory as the root path for placing the server log file. The relative path name of the specified log name is combined with this absolute path to place the server context file. If you specify a full absolute path for the server context log file name, it is used exclusively to set the log file path.

Creating a unique log file per connected client

By default the log file started in the server context is shared by all client/server connections started under a specific broker.

You can optionally set another environment variable at startup called $UNIQUE_DSLOG if you wish to have a unique log file name for each client session.

When the $UNIQUE_DSLOG environment variable is set on, the log file name starts with ds_ followed by the log file name specified for the server context followed by the server's process id number. The file extension continues to be .lg. The full name syntax is as follows:
Enabling ABL-to-SQL Correlation in Logging

The Enhanced Logger allows ABL-to-SQL correlation so you can compare ABL database requests with SQL directives that are executed on the server. In the DataServer

In the DataServer log file, ABL-to-SQL correlation includes information from the client context, the ABL procedure name and line number, along with the server context SQL that was generated based on the ABL code and run against the foreign data source. The correlation is enabled using the `-Dsrv PRGRS_LOG_ABL,n` switch where `n` ranges from 0 to 2, as demonstrated in the following example:

```
-Dsrv Logentrytypes,SQL,PRGRS_LOG_ABL,1
```

Table 6–12 describes the PRGRS_LOG_ABL connect option values:

<table>
<thead>
<tr>
<th>Option Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disables the ABL to SQL correlation feature. No correlation information will be logged.</td>
</tr>
<tr>
<td>1</td>
<td>Enables the ABL to SQL correlation feature for SQL-related log entry types. Every SQL request derived from the log entry type output is preceded by ABL filename and line number which can be correlated to a compiled listings file.</td>
</tr>
<tr>
<td>2</td>
<td>Enables a superset of value 1 which, along with correlation information, includes visual delimiters marking the start and end of each ABL file module.</td>
</tr>
</tbody>
</table>

**Note:** The SQL log entry type is specifically designed to output SQL-related logging information and therefore would be used most commonly in conjunction with the `PRGRS_LOG_ABL -Dsrv` switch. However, the `PRGRS_LOG_ABL` switch can be enabled to acquire module and line number information with any log entry type that writes SQL request information to the log.

Local schema caching

By using a local file to store schema definitions, you can access them more quickly. Running DataServer applications with a local schema cache can result 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 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 a MS SQL Server database whose data source name is sqlbdb with the schema sqlhold and tells OpenEdge to use the cache file sqlcache:

```
CONNECT sqlhold -RO -cache sqlcache -db sqlbdb -dt MSS -ld sqldemo -U bob -P bobpass -Dsrv qt_debug,EXTENDED.
```

If you make any changes to a schema holder, you must create a new cache file for it. For more information, see the “SAVE CACHE Statement” reference entry in *OpenEdge Development: ABL Reference*. 
### Connection failures and OpenEdge responses

Table 6–13 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 <code>CONNECT</code> statement</td>
<td>The system aborts the remainder of the <code>CONNECT</code> 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 <code>NO-ERROR</code> option with the <code>CONNECT</code> statement to trap run-time errors. If you use the <code>NO-ERROR</code> option and it fails, you see the same failure behavior as you do with an unsuccessful <code>CONNECT</code> 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 connection 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 MS SQL Server database with a different logical name</td>
<td>The system responds with a run-time error condition but you can continue. You can use the <code>NO-ERROR</code> option to suppress the error.</td>
</tr>
<tr>
<td>During an attempt to connect a connected MS SQL Server database with the same logical name</td>
<td>The system responds with a warning but you can continue. You can use the <code>NO-ERROR</code> option to suppress the warning.</td>
</tr>
<tr>
<td>During an attempt to connect an unconnected MS SQL Server database whose logical name is already in use by a connected MS SQL Server database</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

Here are some reasons why a connection attempt to an MS SQL Server database might fail:

- The schema holder is not connected.

- The OpenEdge or MS SQL Server required environment variables are not set correctly when using the DataServer and a broker. For environment-variable information, see Chapter 5, “Configuring the DataServer.”

- The data source is not registered properly for ODBC client connectivity. See your Microsoft SQL Server documentation for information on configuring a user, system, or file DSN to properly connect to your data source.

- 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 Query Analyzer.

- 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.

- 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 8, “Troubleshooting.”

Managing connections to an MS SQL Server database

Typically, the DataServer maintains one connection to an MS SQL Server 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 the capabilities of the ODBC driver being used, the following cases may be candidates for using additional connections to accommodate additional cursors:

- Running multiple stored procedures

- Running a stored procedure and a send-sql-statement simultaneously

- 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 (SEPARATE–CONNECTION)` option 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.
END.
```

### 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.

### Naming the log file

The DataServer log file name, the number of log files, the log threshold, and the log append
switch settings are derived from their respective client startup switches:`–dslog`, `–numlogfiles`,
`–logthreshold` and `–logappend`. For example:

- **In self-service mode**, `dslog`, `numlogfiles`, `logthreshold` and `logappend` startup switches
  apply to both client and DataServer log files
- **In client-server mode**, the `dslog`, `numlogfiles`, `logthreshold`, and `logappend` settings
  are derived from the broker startup
- **In ProBroker**, these are the `–dslog`, `–numlogfiles`, `–logthreshold` and `–logappend`
  startup parameters from the `_probrkr` command line
- **In the Unified Broker**, these are `srvrDSLogFile`, `srvrNumLogFiles`, `srvrLogThreshold`
  and `srvrLogAppend` ubroker properties

All the above attributes are static in nature. They are set at startup and cannot be modified once
established. They are read-only attributes when requested from the DSLOG–MANAGER run-time
system handle.

Use the `–dslog` client startup parameter to specify the DataServer log file name. If `–dslog` is not
specified, then the DataServer chooses the default log file name of `dataserv.lg`. 
Setting the log file location

You can specify the absolute path of the log file name with –dslog startup switch to set the location of the log file. The absolute path is obtained from the $DSLOGDIR environment variable. So for example, setting $DSLOGDIR="C:\OE\Wrk\logs" and setting –dslog to mydatasrv.lg, yields the target server log file name of C:\OE\Wrk\logs\mydatasrv.lg. If $DSLOGDIR is not set and just the log file name is specified, then the log file is created in the directory where the OpenEdge session (client or agent) is started.

The path for the log file name is derived in the following order of preference:

1. Usage precedence is for –dslog with an absolute path name.
2. On specifying –dslog with relative path then path name, the –dslog specified is relative to the $DSLOGDIR path.
3. If –dslog is a relative path and no value has been specified for $DSLOGDIR, then the log file will located in the working directory where the DataServer server component is started. This can be:
   a. The current directory where the OpenEdge client is started, if using an OpenEdge client with a self-service DataServer.
   b. The working directory of your AppServer broker instance when using a self-service DataServer in the Unified Broker in the Explorer Framework.
   c. The working directory of your DataServer broker instance when using a client-server DataServer in the Unified Broker in the Explorer Framework.
   d. The current directory where you started your traditional broker using a client-server DataServer with ProBroker.

If neither –dslog nor $DSLOGDIR are specified, the default log file name dataserv.lg is used with the same directory path order of precedence as previously mentioned.

Handling log file creation errors

If creation of the DataServer log file fails for any reason, an error is reported and the DataServer continues to execute. If an error occurs, the DataServer will derive its log file name as ds_<pid>.lg where pid is the process id of the server process. The DataServer continues to write log messages to ds_<pid>.lg log file. For example, if you don’t have write privilege on the $DSLOGDIR directory, the ds_<pid>.lg file is created in the directory where the client or broker agent is started. This behavior is optionally available by default by setting environment variable $UNIQUE_DSLOG at startup.

Restrictions on dynamic changes to log file

When the client and server share the same log file in self-service mode, the LOG-MANAGER system handle is available to allow the log file to be managed. However, when self-service DataServers don’t share the same log file with the client, or if the DataServer is run in client-server mode, the server log file is likely to exist among other servers writing to the same log simultaneously. For this reason, dynamic changes to the log file, such as renaming or closing the log file, are restricted for all server log file contexts. The one exception is the CLEAR-LOG(_)) function which is allowed only if no other process than the one initiating the CLEAR-LOG(_)) is writing to the log file simultaneously.
Using the $UNIQUE_DSLOG environment variable

You can use the $UNIQUE_DSLOG environment variable to isolate the log output of a given server instance and prevent the sharing of log files resources. When the $UNIQUE_DSLOG variable is set, the designated DataServer log file name becomes a root name. Specifically, it becomes the prefix of a log file name that is followed by the process id <pid> of the running server process – which is then suffixed with the familiar .lg extension if the server name is dataserv. This feature can be disabled by unsetting the UNIQUELOG environment variable.

With a unique name, the server executable can manage its own log file copy. However, the DSLOG-MANAGER system handle continues to disallow changes to the log file name or closing of the file at run-time. This restriction ensures that the DataServer always has an open log file to which it can write information. The legacy qt_debug logging mechanism relies on this being the case.

For information on managing log settings, see the “Analyzing performance with the Enhanced Logger” section on page 6–18.

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.
This chapter presents step-by-step instructions for tasks associated with the DataServer. Some of these exercises relate to maintaining the schema holder. After providing an overview of the DataServer for MS SQL Server demonstration databases, the tutorial describes how to perform the tasks outlined in the following sections:

- Demonstration databases for DataServer for MS SQL Server
- Preparing to create demonstration databases
- DataServer utilities
- Creating a schema holder
- Updating a schema holder
- Verifying 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 MS SQL Server
- Adjust schema utility
- Delta df to MS SQL Server Incremental Schema Migration utility
- Modifying a schema holder
- Adding extended ABL support
Demonstration databases for DataServer for MS SQL Server

The demonstration databases for the DataServer for MS SQL Server 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 MS SQL Server utility, which migrates the sports database to your data source and then creates a schema holder. (See the “Preparing to create demonstration databases” section on page 7–3 for detailed instructions.) After you do this, you are ready to run the tutorial exercises or your own applications against the data source.

Before you create a demonstration database, be sure that your system meets the following prerequisites:

- Sufficient privileges to create a database, add users, and create tables
- Sufficient disk space for your database
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 MS SQL Server Utilities menu.

The following lists identifies the basic preliminary steps you must complete to create demonstration databases:

- Install and start your data source. This specific step depends on your unique environment. As necessary, refer to your data source documentation.
- Install your DataServer drivers.
- Install the OpenEdge client. As necessary, refer to the OpenEdge client-related documentation.

To create the demonstration database and schema holder using the OpenEdge DB to MS SQL Server 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 5–4.
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 5–5 and the “Configuring a remote DataServer” section on page 5–6.
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 select the specific product you want to use to begin your OpenEdge session or type the following command line in Windows:

   ```
   install-path\d1c\bin\prowin32
   ```

   **Note:** Starting a local OpenEdge session also automatically starts the local DataServer.

7. Access the Data Administration. Create a copy of the sports database and connect to it.
8. Select DataServer ➔ MS SQL Server Utilities ➔ Schema Migration Tools ➔ OpenEdge DB to MS SQL Server to start the OpenEdge DB to MS SQL Server utility.
9. Specify OpenEdge DB to MS SQL Server parameter values as noted in Table 7–1.
Table 7–1: OpenEdge DB to MS SQL Server utility

<table>
<thead>
<tr>
<th>Interface element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original OpenEdge Database</td>
<td>Accept the default value; it is the name of the connected data source. For example, if you copied the sports database to create a demonstration database, the default value <code>sports</code> displays.</td>
</tr>
<tr>
<td>Connect parameters for OpenEdge</td>
<td>Accept the default value, which specifies the current working database.</td>
</tr>
<tr>
<td>Name of Schema holder Database</td>
<td>Type the name you want to use for your schema holder. For example, type <code>myholder</code>.</td>
</tr>
<tr>
<td>Logical Database Name</td>
<td>Type name that OpenEdge will use to identify the MS SQL Server database.</td>
</tr>
<tr>
<td>ODBC Data Source Name</td>
<td>Type the name you used when registering the data source. For example, if you accepted the ODBC Data Source Name suggested earlier in the “Preparing to create demonstration databases” section on page 7–3, type <code>mysports</code>. Otherwise, type the ODBC Data Source name you previously set up.</td>
</tr>
<tr>
<td>Username</td>
<td>Type the user ID for MS SQL Server database.</td>
</tr>
<tr>
<td>User’s Password</td>
<td>Type the password of the user.</td>
</tr>
<tr>
<td>Connect parameters</td>
<td>See Chapter 6, “Connecting the DataServer,” for connection parameters.</td>
</tr>
<tr>
<td>Maximum Varchar Length</td>
<td>Enter the maximum size of the VARCHAR field. Any size greater will be text. Max size is 8000.</td>
</tr>
<tr>
<td>Codepage</td>
<td>Accept the default code page value.</td>
</tr>
<tr>
<td>Collation</td>
<td>Accept the default value.</td>
</tr>
<tr>
<td>Insensitive</td>
<td>Accept the default value.</td>
</tr>
<tr>
<td>Create RECID Field</td>
<td>Select this check box to obtain the cursor scrolling capability.</td>
</tr>
<tr>
<td>Load SQL</td>
<td>Accept the default check mark in this check box to specify that the generated SQL script will be loaded into the data source.</td>
</tr>
<tr>
<td>Move Data</td>
<td>Select this check box to dump and load data from OpenEdge to MS SQL Server.</td>
</tr>
<tr>
<td>Create Shadow Columns</td>
<td>MS SQL Server is case insensitive by default, and this box is unmodifiable. When SQL Server is configured with a case sensitive code page, this box is available for selection. Selecting it provides case insensitivity compatible with the behavior of an OpenEdge database.</td>
</tr>
</tbody>
</table>
Preparing to create demonstration databases

10. Running the utility creates and connects a schema holder and the MS SQL Server database. It operates as follows:

   a. SQL script is generated.
   
   b. SQL that creates the schema is sent to the MS SQL Server data manager.
   
   c. The schema is pulled back to the schema holder.
   
   d. The schema holder and the OpenEdge database are compared and all information needed by OpenEdge is applied to the schema holder.
   
   e. Data is loaded if selected.
   
   f. The schema holder is disconnected.
   
   g. A message is displayed that tells the user which startup procedure to use to connect. Additionally, the entered connect information is also stored in the db record which the DataServer can use to automatically connect.

---

Table 7–1: OpenEdge DB to MS SQL Server utility (2 of 2)

<table>
<thead>
<tr>
<th>Interface element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include Defaults</td>
<td>Check if you want the OpenEdge initial value used as the default in the SQL script.</td>
</tr>
<tr>
<td>For field widths 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>• 4GL Format — Compiles with the current default width specified. (default)</td>
</tr>
<tr>
<td></td>
<td>If you select the 4GL 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>Note: You cannot use the Expand x(8) to 30 setting with the Width option.</td>
</tr>
<tr>
<td>For Create RECID use:</td>
<td>Select either Trigger or Computed Column. Trigger is selected by default. Computed Column is preferred as it offers greater reliability, but is only available for MS SQL Server 2005 and later.</td>
</tr>
</tbody>
</table>

---

Note: For a complete description for running while in batch mode, see the “Running the OpenEdge DB to MS SQL Server utility” section on page 7–20.
DataServer utilities

OpenEdge supports a number of utilities that allow you to create and maintain a MS SQL DataServer.

To see all available DataServer utilities, choose DataServer → MS SQL Server Utilities. Table 7–2 describes the DataServer utilities.

Table 7–2: DataServer utilities

<table>
<thead>
<tr>
<th>Utility</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create DataServer Schema...</td>
<td>Creates a schema image in the schema holder for the MS SQL Server database</td>
</tr>
<tr>
<td>Update/Add Table Definitions...</td>
<td>Updates the schema holder to reflect any changes that you make to 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 MS SQL Server database</td>
</tr>
<tr>
<td>Change DataServer Schema Code Page...</td>
<td>Changes the code page in the schema holder associated with the MS SQL Server database</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 an OpenEdge database to a MS SQL Server database, for incrementally migrating a schema to MS SQL Server, and adjusting the schema image</td>
</tr>
</tbody>
</table>
When you access a DataServer utility (as you will do in the tutorials that follow this section), the following dialog box might appear before the utility opens:

![User ID and Password dialog box]

To accept the user ID and password combination that you supplied in the **User ID and Password** dialog box, choose **OK**. 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 "Permission requirements" section on page 5–15 for information on privileges.

Note that the value **mssports** that appears in the previous **User ID and Password** dialog box identifies the MS SQL Server database to which a connection will be attempted. The actual database name that appears will depend on the specific MS SQL Server database to which you are trying to connect.
Creating a schema holder

To create the schema holder:

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

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

2. In the Logical Database Name field, type the name that you will use to connect to your 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 MS SQL Server database 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. If you choose UTF-8 as your schema image code page, your schema holder’s code page must also be UTF-8.

Table 7–3 lists the most common MS SQL Server database code pages and the equivalent OpenEdge names.

<table>
<thead>
<tr>
<th>MS SQL Server Code page</th>
<th>OpenEdge equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>iso_1</td>
<td>iso8859–1 (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 “Code pages” section on page 2–5 for a discussion of collation issues to consider.

5. In the **Case Insensitive** field, the default value *yes* indicates that MS SQL Server’s case insensitivity feature is in use. To change this value, type *no*.

6. Type the connection parameters in the **Connection Parameters** field.

   See Chapter 6, “Connecting the DataServer,” for a description of the required and optional connection parameters.

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

8. Choose **OK**. The utility prompts you for your data source user ID and password. If they are required by the MS SQL Server data source and you did not provide them in the **Connection Parameters** field (see Step 6), enter a data-source user ID and password combination that has SELECT privileges for the system objects listed in the “Permission requirements” section on page 5–15 and read access to other database objects that the schema holder will include.

9. Choose **OK**. When the DataServer connects to the MS SQL Server database, it reads information about data source objects. The following dialog box appears:

![Pre-Selection Criteria For Schema Pull](image)

You can select tables based on the object name, owner information, and qualifier. For example, you can specify A* in the **Object Name** field to list all the tables whose names begin with A or a.

**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.)

Check **Default to OpenEdge DATETIME** to automatically map MS SQL Server data types to the associated OpenEdge data type. If you have modified your client application to handle LOB data types, check **Default to OpenEdge LOB** to map the OpenEdge LOB data type to MS SQL Server VARBINARY (MAX), IMAGE and FILESTREAM data types. For more information on mapping OpenEdge and MS SQL Server data types, see “Support for OpenEdge ABL BLOB data type” section on page 2–30.
10. Choose OK. OpenEdge displays a list of the data source objects that you can include in the schema holder, as shown:

![List of MSS Objects](image)

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.

11. 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 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.

12. Choose 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 table, the DataServer attempts to select an index to support the OpenEdge ROWID. If an appropriate index does not exist, the DataServer issues the warning, **Please check errors, 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 7–35 for instructions. For additional information, see the “Indexes and sorting” section on page 2–11.
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 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 a data source.

The Update/Add Table Definitions utility allows you to:

- Add object definitions from the data source to a schema holder. Use this option if you add a new table, stored procedure, 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. For example, an integer data type that is pulled from a MS SQL Server data source can be mapped to any of the following OpenEdge data types: INTEGER, DECIMAL, LOGICAL, or INT64. See the “Data types” section on page 2–17 for more information about support for MS SQL Server data types and their equivalent OpenEdge data type mapping options.

To update a schema holder from the Data Administration tool:

1. Access the **Data Administration**, if you are not already there, and select DataServer→MS SQL Server Utilities→Update/Add Table Definitions. The following dialog box appears:

2. Type preselection criteria values into the fields and select data type options as required. 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. For more information on data type options,

**Note:** Consider whether you will choose to perform a schema pull that consists only of wild cards. Doing so might degrade your database’s performance as the activity will include system catalog files that are not typically included in a user database.
Check **Default to OpenEdge DATETIME** to automatically map MS SQL Server data types to the associated OpenEdge data type. If you have modified your client application to handle LOB data types, check **Default to OpenEdge LOB** to map the OpenEdge LOB data type to MS SQL Server `VARBINARY (MAX)`, `IMAGE` and `FILESTREAM` data types. For more information on mapping OpenEdge and MS SQL Server data types, see the “Support for OpenEdge ABL BLOB data type” section on page 2–30.

3. Choose **OK**. A dialog box like the following example lists the objects and table information that you have preselected:

4. Select the objects that you want to update, then choose **OK**. When the update completes, you return to the **Data Administration** main window.

When the update completes, you are reminded 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 data source and then update the definition, you do not have to re-enter 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 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 Table Definition utility reads the definitions in the 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 7–4 lists the differences that the Verify Table Definition 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>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</td>
<td>Unique</td>
<td>Retained1</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>
<tr>
<td>Field</td>
<td>Case sensitivity</td>
<td>Retained</td>
</tr>
</tbody>
</table>
To verify a table from the Data Administration tool:

1. Access the Data Administration, if you are not already there, and select DataServer → MS SQL Server Utilities → Verify Table Definition. The following dialog box appears:

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.

3. By default, the utility verifies objects in the schema holder that match objects in the data source. To check whether there are objects in the data source that are not represented in the schema holder, deselect the check box Verify only objects that currently exist in the schema holder.

{| Database object | Difference | Category |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<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>

1. When you update an index, it is identified as unique if it was defined as unique in the data source or the schema holder.

2. If the corresponding information in the 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.
4. There is a 32K limit on the contents of the verify report. Select **Output differences to file** if you anticipate a large volume of differences, or wish to save the report. The file that is produced will be named, `<dbname>+.vfy` and will be written to the working directory.

5. Choose **OK**. A dialog box like the following example lists the objects and table information that you preselected:

![Example dialog box](image)

6. Select the objects that you want to update, then choose **OK**.

7. To select tables by matching a pattern, choose **Select Some**. The following dialog box appears:

![Select objects by pattern match](image)

Note that you can also choose **Deselect Some** to deselect objects, using the identical matching a pattern options to deselect objects so they are not included in the schema holder.
8. Type the pattern that you want to match, then choose **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 appears:

![Select MSS Objects dialog box]

9. When reading the text of the report, **SH** indicates the value in the schema image; **NS** indicated the value in the MS SQL Server database. For example:

![Schema-Verify - Detected Differences]

10. Choose **Close** to return to the **Schema Verify** dialog box.

11. 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.

12. Choose **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 a MS SQL Server database’s connection information or logical name in the associated schema.

To change connection information for a schema holder from the Data Administration tool:

1. Access the Data Administration, if you are not already there, and select DataServer→MSS SQL Server Utilities→Edit Connection Information. The following dialog box appears:

2. Make changes to the Connection Parameters fields as required. When you are done, choose 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 6, “Connecting the DataServer,” and OpenEdge Deployment: Startup Command and Parameter Reference.

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 the driver name. You will receive only a warning if you use a different driver, but the schema holder configuration may no longer match the characteristics of the driver and could 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; you can add or correct the code page information at a later date. However, if you have been writing 8-bit character data to your data source with the DataServer and then you change the code page, the data is unaffected by the 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 from the Data Administration tool:

1. Access the Data Administration, if you are not already there, and select DataServer → MSS SQL Server 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 following dialog box appears:

3. Either accept the current value or type the data source code page that the data source supports. If you choose UTF-8 as your schema image code page, your schema holder’s code page must also be UTF-8.

   See OpenEdge Development: Internationalizing Applications for a complete list of code pages that OpenEdge supports. If you are using an unsupported code page, you are allowed to create your own conversion tables.

4. Choose OK to change the code page and return to the Data Administration main window.

   If you were connected to the schema holder and the 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 a data source.

To delete a schema holder from the Data Administration tool:

1. Access the Data Administration and select DataServer → MS SQL Server Utilities → Delete DataServer Schema. A dialog box appears, prompting you to verify the deletion.

2. Choose Yes to verify your selection. After the schema holder is deleted, a confirmation message is displayed.

3. Choose OK to return to the Data Administration main window.
Migrating an OpenEdge database to MS SQL Server

The DataServer for MS SQL Server supports the OpenEdge DB to MS SQL Server utility that allows you to migrate an OpenEdge database to a foreign data source through the use of the MS SQL Server DataServer. While the DataServer makes a 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 data source.

The OpenEdge DB to MS SQL Server utility performs the following tasks:

- Creates objects in the target data source
- Creates the schema holder and schema image
- Optionally populates the data source by dumping and loading the data from the OpenEdge database

The MS SQL Server data source that you create with this utility is a basis for an application database. Before deploying your new data source, you might want to make manual adjustments to take advantage of additional MS SQL Server features that are not supported by the migration utility.

The OpenEdge DB to MS SQL Server utility requires a local OpenEdge database.

Preparing a data source for the utility

The OpenEdge DB to MS SQL Server migration utility translates definitions for OpenEdge objects into an SQL DDL script that loads into 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.

When you use the OpenEdge DB to MS SQL Server utility to make a MS SQL Server database compatible, it converts arrays by using one column in the MS SQL Server database 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.

Running the OpenEdge DB to MS SQL Server utility

The OpenEdge DB to MS SQL Server utility runs in Windows with a DataServer accessing MS SQL Server through an ODBC driver.

You can run the utility interactively from Windows or Unix, or in batch mode.

To run the OpenEdge DB to MS SQL Server utility interactively:

1. Create a target MS SQL Server. You must use an empty target data source when you run the OpenEdge DB to MS SQL Server utility.
2. Configure your ODBC driver to connect to your new 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 the Data Administration, choose DataServer → MS SQL Server Utilities → Schema Migration Tools → OpenEdge DB to MS SQL Server. The following screen appears:

![OpenEdge DB to MS SQL Server Conversion](image)

Table 7–5 describes the information for which you are prompted to enter.

<table>
<thead>
<tr>
<th>Interface element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original OpenEdge Database</td>
<td>Accept the default value. If you are using suggested OpenEdge database name, the default is sports.</td>
</tr>
<tr>
<td>Connect parameters for OpenEdge</td>
<td>Accept the default value, which specifies the current working database.</td>
</tr>
<tr>
<td>Name of Schema holder Database</td>
<td>Type myholder, or another schema holder name of your choice.</td>
</tr>
<tr>
<td>Logical Database Name</td>
<td>Type the name that OpenEdge will use to identify the MS SQL Server database.</td>
</tr>
<tr>
<td>ODBC Data Source Name</td>
<td>Type the ODBC Data Source Name you used when registering the data source.</td>
</tr>
<tr>
<td>Username</td>
<td>Type the user ID for MS SQL Server.</td>
</tr>
<tr>
<td>User’s Password</td>
<td>Type the password of the user.</td>
</tr>
</tbody>
</table>

Table 7–5: OpenEdge-to-MS SQL Server Conversion utility (1 of 3)
<table>
<thead>
<tr>
<th>Interface element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect parameters</td>
<td>See Chapter 6, “Connecting the DataServer,” for connection parameters.</td>
</tr>
<tr>
<td>Maximum Varchar Length</td>
<td>By default, the maximum size of the VARCHAR data type is 8000. The TEXT data type is greater than 8000. If you enter a value that is less than 8000, the TEXT data type will be used when a character field display format or width is greater than the value you have entered. <strong>NOTE:</strong> When <strong>Use Unicode Types</strong> is selected, NVARCHAR maximum length is 4000 by default. For UTF-8, because characters can use up to four bytes, only 2000 characters are guaranteed to fit.</td>
</tr>
<tr>
<td>Codepage</td>
<td>Must be the OpenEdge name for the code page that the data source uses. Use UTF-8 if Unicode support is desired.</td>
</tr>
<tr>
<td>Collation</td>
<td>Accept the default value.</td>
</tr>
<tr>
<td>Insensitive</td>
<td>Accept the default value to retain case insensitivity.</td>
</tr>
<tr>
<td>Create RECID Field</td>
<td>If this parameter is available for your data source type, check this check box to obtain the Extended ABL capability.</td>
</tr>
<tr>
<td>Load SQL</td>
<td>Leave this box checked to have the generated SQL script loaded into the data source.</td>
</tr>
<tr>
<td>Move Data</td>
<td>Select this check box to dump and load data from OpenEdge to MS SQL Server. When <strong>Use Unicode Types</strong> is selected, the data is translated as it is moved through the use of a conversion file.</td>
</tr>
<tr>
<td>Create Shadow Columns</td>
<td>MS SQL Server is case insensitive by default and this box is unmodifiable. When SQL Server is configured with a case sensitive code page, this box is available for selection. Selecting it provides case insensitivity compatible with the behavior of an OpenEdge database.</td>
</tr>
<tr>
<td>Include Defaults</td>
<td>Check if you want the OpenEdge initial value used as the default in the SQL script.</td>
</tr>
<tr>
<td>Use Unicode Types</td>
<td>Changes default code page to UTF-8 and directs the schema migration to convert all character data types in OpenEdge database to Unicode data types.</td>
</tr>
<tr>
<td>Expand width (utf-8)</td>
<td>Doubles the size of character fields based on specified field width criteria.</td>
</tr>
<tr>
<td>Use revised sequence generator</td>
<td>Select to enable revised generator and to prevent lock timeouts. Leave unchecked to retain older sequence generator, particularly if SET-VALUE sequence statement is used extensively in your program.</td>
</tr>
</tbody>
</table>
To perform a complete migration of your OpenEdge database to a target data source, you must enter information in all appropriate fields and select all appropriate check boxes. 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 logical name for your target database. For example, if you specified "sports" as the logical data source name, the utility creates the `csports.p` startup procedure.

To run the OpenEdge-to-MS SQL Server Conversion utility in batch mode:

1. Create a target MS SQL Server database. You must use an empty target data source when you run the OpenEdge DB to MS SQL Server 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 7–6.

<table>
<thead>
<tr>
<th>Interface element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>For field widths 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 <code>_width</code> field in the <code>_field</code> record.</td>
</tr>
<tr>
<td></td>
<td>• 4GL Format — Compiles with the current default width specified. (default)</td>
</tr>
<tr>
<td></td>
<td>If you select the 4GL 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>
<tr>
<td>For Create RECID use:</td>
<td>Select either Trigger or Computed Column. Trigger is selected by default. Computed Column is preferred as it offers greater reliability, but is only available for MS SQL Server 2005 and later.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRODBNAME</td>
<td>Specify the source OpenEdge database name.</td>
</tr>
<tr>
<td>PROCONPARMS</td>
<td>Specify parameters for the connection to the source OpenEdge database.</td>
</tr>
</tbody>
</table>
Table 7–6:  **OpenEdge-to-MS SQL Server Conversion utility batch parameters**

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHDBNAME</td>
<td>Specify the new schema-holder name.</td>
</tr>
<tr>
<td>MSSDBNAME</td>
<td>Specify ODBC data source name.</td>
</tr>
<tr>
<td>MSSPDBNAME</td>
<td>Specify logical database name.</td>
</tr>
<tr>
<td>MSSUSERNAME</td>
<td>Specify the user name for the target data source.</td>
</tr>
<tr>
<td>MSSPASSWORD</td>
<td>Specify the password of the user for the target data source.</td>
</tr>
<tr>
<td>MSSCONPARMS</td>
<td>Specify additional connection parameters for the schema holder.</td>
</tr>
<tr>
<td>MSSCODEPAGE</td>
<td>Specify 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 unset and use the Change DataServer Schema Code Page utility to add the code page information for the schema holder later.</td>
</tr>
<tr>
<td>MSSCOLLNAME</td>
<td>Collation name.</td>
</tr>
<tr>
<td>MSSCASESEN</td>
<td>Yes or No for Code Page case sensitive.</td>
</tr>
<tr>
<td>COMPATIBLE</td>
<td>Create arrays and RECID. Set to 2 for using a computed column and 1 or Yes for using a trigger.</td>
</tr>
<tr>
<td>MAPSMSSDATETIME</td>
<td>Specify if new date and time data types should be used (set on only for MS SQL Server 2008 or above).</td>
</tr>
<tr>
<td>VARLENGTH</td>
<td>Total number of characters a VARCHAR can have. Any field whose format is greater that will be defined as a text field. Maximum is 8000.</td>
</tr>
<tr>
<td>SQLWIDTH</td>
<td>If present, use the OpenEdge file _WIDTH field size instead of calculating length by using the display format.</td>
</tr>
<tr>
<td>LOADSQL</td>
<td>Allows you to specify whether you want the utility to create the schema in your empty MS SQL Server database. Specify YES to enable this behavior.</td>
</tr>
<tr>
<td>MOVEDATA</td>
<td>Allows you to specify whether to populate the database. Specify YES to dump and load data or NO to not populate the database. For example, you might specify NO if your database is large, and you want to dump and load data at a more convenient time. The default is NO.</td>
</tr>
<tr>
<td>UNICODETYPES</td>
<td>Set to YES to use Unicode types, and NO otherwise. The default is NO.</td>
</tr>
<tr>
<td>UNICODE_EXPAND</td>
<td>Set to YES to double the length of fields on conversion, and NO otherwise. The default is NO.</td>
</tr>
</tbody>
</table>
4. Enter the following commands to set and export environment variables at the system prompt:

```
PRODBNAME=db-name; export PRODBNAME
PROCONPARMS=-1 -i
SHDBNAME=schema-holder-name; export SHDBNAME
pro -b -p product/mss/protomss.p
```

Adjust schema utility

The Adjust Schema utility allows you to compare your MS SQL Server 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. Select the utility from the Data Admin Tool → DataServer → MS SQL Server Utilities → Schema Migration Tools → Adjust Schema. The following screen appears:

   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. Select OK.

   All of the objects that are compared will be displayed on screen as they are processed.
Delta df to MS SQL Server Incremental Schema Migration utility

The Incremental Schema Migration utility allows you to migrate schema changes from OpenEdge to a MS SQL Server database. For example, in the process of developing an application in ABL that you will migrate to MS SQL Server, you might want to make and test schema changes in the OpenEdge database that you want reflected in the MS SQL Server database. The utility reads an OpenEdge delta.df file that has been created using the standard incremental dump procedure, and creates a delta.sql file that contains the SQL DDL for making the changes and a delta.df file. You can then load the delta.df file into the schema holder. You can then apply the delta.sql file to the MS SQL Server database to complete the migration process.

**Note:** You do not make schema changes directly in the schema holder, which must remain synchronized with the MS SQL Server database. The utility uses the schema holder to determine what the MS SQL Server definitions are.

To run the Incremental Schema Migration utility:

1. From the Data Admin main menu, choose DataServers → MS SQL Server Utilities → Schema Migration Tools → Generate Delta.sql OpenEdge to MSS. The following dialog box appears:
2. Provide the information described in Table 7–7.

<table>
<thead>
<tr>
<th>Interface element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delta DF File</td>
<td>The name of the delta.df file that was created when you ran the incremental dump routine against two OpenEdge databases. You can browse for the file name by choosing the Files button.</td>
</tr>
<tr>
<td>Schema Holder Database</td>
<td>The name of the schema holder. The schema information must match the schema of the MS SQL Server database to be updated. Progress Software Corporation recommends that you are connected to the schema holder.</td>
</tr>
<tr>
<td>Connect parameters for Schema</td>
<td>Specify these parameters to connect to the schema holder if you are not already connected.</td>
</tr>
<tr>
<td>Logical name for MSS Database</td>
<td>Specify the MS SQL Server database logical name, that is, the name by which you refer to the MS SQL Server database in your application.</td>
</tr>
<tr>
<td>MSS Object Owner Name</td>
<td>Specify the name of the MSS database object owner.</td>
</tr>
<tr>
<td>Maximum Varchar Length</td>
<td>By default, the maximum size of the VARCHAR data type is 8000. The text data type is greater than 8000. If you enter a value that is less than 8000, the text data type will be used when a character field display format or width is greater than the value you have entered.</td>
</tr>
<tr>
<td>Create RECID field</td>
<td>Leave this check box checked to obtain the cursor scrolling capability.</td>
</tr>
<tr>
<td>Include Default</td>
<td>Select this check box if you want the OpenEdge initial value used as the default SQL script.</td>
</tr>
<tr>
<td>Create Shadow Columns</td>
<td>MS SQL Server is case insensitive by default, and this box is unmodifiable. When SQL Server is configured with a case sensitive code page, this box is available for selection. Selecting it provides case insensitivity compatible with the behavior of an OpenEdge database.</td>
</tr>
</tbody>
</table>
### Table 7–7: Generate Delta.sql OpenEdge to MSS utility (2 of 3)

<table>
<thead>
<tr>
<th>Interface element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create schema holder delta df</td>
<td>Select this check box if you want the utility to generate a .df file that includes the incremental schema information. You can then load this .df file into the schema holder. By default, this check box is checked.</td>
</tr>
<tr>
<td>Use Unicode Types</td>
<td>Maps character fields to Unicode data types. Only applies if schema image’s code page is UTF-8.</td>
</tr>
<tr>
<td>Use revised sequence generator</td>
<td>Select to enable revised generator and to prevent lock timeouts. Leave unchecked to retain older sequence generator, particularly if SET-VALUE sequence statement is used extensively in your program.</td>
</tr>
<tr>
<td>Map to MSS Datetime Type</td>
<td>Checked by default. Uncheck box to map to datetime data types with time zone components. For details, see “DataServer for MS SQL Server support for datetime data types” section on page 2–21.</td>
</tr>
<tr>
<td>Expand width (utf-8)</td>
<td>Doubles the size of character fields based on specified field width criteria.</td>
</tr>
</tbody>
</table>
3. Choose OK. The utility generates a `delta.sql` file and, optionally, a `delta.df` file.

When this step is performed for an MS SQL Server DataServer, and the incremental update procedure includes changing an existing field from `INTEGER` to `INT64` in the schema holder, the DataServer supports the update process through the `ALTER TABLE ALTER COLUMN` statement. If a column is part of an index, the incremental schema migration utility generates a script to drop any indexes that contain the column, change the column’s data type to `BIGINT`, and then re-adds the indexes. During this process, the utility uses the index definitions defined in the schema holder.

4. After running the utility, you must apply the SQL that the utility generates to the MS SQL Server database and load the new `delta.df` file into the original schema holder so that it is synchronized with the modified MS SQL Server database.

Note that if you updated an existing field from `INTEGER` to `INT64` as described in Step 3 of this procedure, you also must confirm that none of the data is lost.

The Incremental Schema Migration utility generates SQL that will create objects in the MS SQL Server database that are compatible with OpenEdge. Table 7–8 shows that the utility creates the same objects as the OpenEdge DB to MS SQL Server migration utility.

### Table 7–7: Generate Delta.sql OpenEdge to MSS utility

<table>
<thead>
<tr>
<th>Interface element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>For field widths use:</strong></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>- <strong>Width</strong> — Uses the value of the <code>_width</code> field in the _field record.</td>
</tr>
<tr>
<td></td>
<td>- <strong>4GL Format</strong> — Use the value of the <code>_format</code> field in the _field record. (default)</td>
</tr>
<tr>
<td></td>
<td>If you select the <strong>4GL Format</strong> option, you have an additional setting to define:</td>
</tr>
<tr>
<td></td>
<td>- <strong>Expand x(8) to 30</strong> — This setting is on by default to indicate that the format for the character fields defined as <code>x(8)</code> will be created as 30 characters.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> You cannot use the <strong>Expand x(8) to 30</strong> setting with the <strong>Width</strong> option.</td>
</tr>
<tr>
<td><strong>For Create RECID use:</strong></td>
<td>Select either <strong>Trigger</strong> or <strong>Computed Column. Trigger</strong> is selected by default. <strong>Computed Column</strong> is preferred as it offers greater reliability, but is only available for MS SQL Server 2005 and later.</td>
</tr>
</tbody>
</table>
The utility ensures that the migrated objects have names that are unique to the MS SQL Server database. If you have given the object a name that is not unique, it drops characters from the end of the name and appends numbers until it creates a unique name.

Since MS SQL Server requires that index names be unique to the database, the utility appends the table name to the indexed column name to create a unique name.

**Table 7–8: MS SQL Server equivalents of OpenEdge objects**

<table>
<thead>
<tr>
<th>OpenEdge object</th>
<th>MS SQL Server equivalents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>One column for each extent of the OpenEdge array.</td>
</tr>
<tr>
<td>Table</td>
<td>For new tables, if the Create RECID Field option was checked, a PROGRESS_RECID column and associated triggers are created. This indexed column provides a unique key on the MS SQL Server table.</td>
</tr>
</tbody>
</table>

**Updating the MS SQL Server database**

You should review, and modify if necessary, the `delta.sql` file that the utility generates before applying it. You can apply the `delta.sql` file to the MS SQL Server database through SQL-based tools.

After applying the `delta.sql` file to the MS SQL Server database, you must update the original schema holder by loading the new `delta.df` file into the original schema holder so that the original schema holder reflects the modifications you made to the MS SQL Server database.
Modifying a schema holder

You can begin using the DataServer as soon as you load your 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 can 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. Choose the Table Properties button. The following dialog box appears:

   ![Table Properties dialog box]

4. Choose 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. Choose OK to return to the Table Properties dialog box.
6. Choose 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 it is not already showing. Choose Fields. The following Fields list appears:

2. Select a table from the Tables list.

3. Select a field from the Fields list.

4. Choose the Field Properties button. The following dialog box appears:

You can enter information at the field level, such as a validation expression or a validation message.
The Data Dictionary displays the standard MS SQL Server 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 the data. For example, choose the MS SQL Server \texttt{decimal} data type, which in turn defaults to the OpenEdge \texttt{INTEGER} data type mapping. However, you can change the \texttt{decimal} mapping to either the OpenEdge \texttt{INTEGER} or \texttt{INT64} data type instead. (The previous \textbf{Field Properties} dialog box shows the \texttt{DECIMAL} data type with the \texttt{INT64} data type option selected.) For more information about optional settings for data types, see the “Data types” section on page 2–17 and Appendix C, “Data Type Details.”

- For \texttt{CHARACTER} fields that are not indexed, you can change the case sensitivity.

\textbf{Note:} You cannot create fields or add mandatory or extent properties to them.

5. Choose \textbf{DataServer} to view the field name and position as stored in the data source. A dialog box similar to the following appears:

![MSS Specific Fields dialog box]

\textbf{Note:} You cannot change data source information using the Data Dictionary. For example, the \texttt{total_paid} field is named \texttt{total_paid} in the database.

6. Choose \textbf{OK}.

7. When you are done making changes, choose \textbf{OK} to return to the \textbf{Data Dictionary} main window.

\textbf{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 a 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. To create a PROGRESS_RECID column, see the “Using triggers to enable PROGRESS_RECID support of the ROWID function” section on page 7–37.

- 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 criteria:
  - The index must be defined as unique.
  - The index must be defined as mandatory, or at least treated as such by your application.

The index selection process proceeds, according to the following order:

1. If you select an index, then you must select a qualified index.

2. If you do not select an index, the MS SQL Server DataServer selects a qualified index based on a single-component integer.
   
   However, if the DataServer determines that more than one qualified index meets the additional criterion as a single-component integer, the DataServer selects the first index that fulfills the complete criteria.

3. If the MS SQL Server DataServer cannot find a qualified index based on a single-component integer, it seeks a qualified index that is multi-component and/or non-integer.
   
   However, if the DataServer determines that more than one qualified index meets the additional criterion as multi-component and/or non-integer, the DataServer selects the first index that fulfills the complete criteria.

4. If the MS SQL Server DataServer cannot find a qualified index that is a multi-component and/or non-integer, the task cannot be completed and ROWID support is not provided. The table will not be able to uniquely identify any records. As a result, implementing FIND statements and performing queries could lead to inaccuracies in locating records.

Note that the indexes in this class are not mandatory; therefore, it is essential that you enforce the column supporting ROWID as mandatory at least through code if not through definitions.

The DataServer allocates a buffer to hold the key information for the selected RECID index. The buffer is 245 bytes by default. If the index you select for the RECID is a multi-component index, this may be inadequate. The size of the buffer can be modified with the -Dsrv ZPRGRS_RECID_BUF_SIZE, n option. See the “RECID Buffer size—the ZPRGRS_RECID_BUF_SIZE,n Option” section on page 8–10 for more information on setting this option.

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.
To select an index to support the ROWID function, use the following procedure in the Data Dictionary with the schema holder connected (you do not have to connect to the MS SQL Server database).

To select an index to support the ROWID function:

1. Choose Tables mode.
2. Select the table whose ROWID you want to modify.
3. Choose Table Properties.
4. Choose DataServer. The following dialog box appears:

5. Double-click an index name listed in the Name field to display detailed information about its attributes. The following dialog box appears:

6. Choose OK to return to the ROWID Choices dialog box.
7. Select the index you want to use to support ROWID.
8. Choose OK to return to the Table Properties dialog box.
Adding extended ABL support

The DataServer provides support for arrays and the RECID/ROWID function when certain objects exist in a data source. You can modify your data source to enable support for these extended features. You can also add support when you create new tables.

The DataServer maintains the columns that you add to your data source tables to support the RECID/ROWID function. If non-OpenEdge applications are updating those data source tables, the applications must recognize the additional columns and update them appropriately. You can enable tables either through the use of triggers or through the use of computed columns. The use of computed columns is the preferred method as it offers greater efficiency and reliability, however it is only available when using MS SQL Server 2005 or later versions.

The following sections describe how to modify an existing data source so that it supports arrays, and OpenEdge record identifiers. See the “ROWID function” section on page 2–52 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 data source. To have access to this functionality, you must make changes to your data source table.

To modify a table 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 and size. For example, if you want the schema holder to include an array named MONTH with 12 elements, the 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.

Using triggers to enable PROGRESS_RECID support of the ROWID function

If you want to use the ROWID function with a data source, you must select an index to associate with this function. Adding a new integer column named PROGRESS_RECID and defining it as a single-component index is the preferred method. This method can only be applied to a table that does not currently have an identity column defined. If your table already has an identity column, you can support ROWID by selecting an existing index. See the “Defining the ROWID” section on page 7–35 for instructions on selecting an existing index. The following procedure describes how to create the PROGRESS_RECID column to support ROWID. To use the ROWID function with a MS SQL Server database, you must make changes to your data source table.

Note: Starting in OpenEdge Release 10.1B, the default PROGRESS_RECID recognizes an INT64 value for a MS SQL Server DataServer. The unique key value in the PROGRESS_RECID
will be derived by default from a 64-bit value. (For prior releases, the value was derived, by default, from the 32-bit INTEGER.)

To modify a table to support the ROWID function:

1. Perform these tasks:
   a. Add a column of the integer data type named PROGRESS_RECID. The new column must be able to contain null, as shown:

   ```sql
   alter table table
   add PROGRESS_RECID integer null
   ```

   b. Add a column with identity characteristics named PROGRESS_RECID_IDENT_. The new column must have the integer data type, as shown:

   ```sql
   alter table table
   add PROGRESS_RECID_IDENT_ integer identity
   ```

2. Create a trigger to maintain the PROGRESS_RECID column:

   ```sql
   create trigger _TI_table ON table for insert as
   RAISERROR ('PSC-init',0,1)
   SET XACT_ABORT ON
   SET LOCK_TIMEOUT -1
   if ( select PROGRESS_RECID from inserted) is NULL
   begin
   update t set PROGRESS_RECID = i.IDENTITYCOL
   from customer t  JOIN inserted i ON
   t.PROGRESS_RECID_IDENT_ = i.PROGRESS_RECID_IDENT_ 
   select convert (bigint, @@identity)
   end
   SET XACT_ABORT OFF
   ```

3. Create a non-unique index for the PROGRESS_RECID and a unique index for the PROGRESS_RECID_IDENT, respectively:

   ```sql
   create index <table>##progress_recid on <table> (PROGRESS_RECID)
   ```

   ```sql
   create unique index <table>##progress_recid_ident_on <table>
   (PROGRESS_RECID_IDENT_)
   ```

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, delete and recreate it.
Using computed columns to enable PROGRESS_RECID support of the ROWID function

Use the following procedures to create PROGRESS_RECID to support the ROWID function. Note that computed column technology is available only in MS SQL Server 2005 and later versions.

To modify a table to support the ROWID function using a computed column:

1. Add new columns PROGRESS_RECID, PROGRESS_RECID_IDENT_ and PROGRESS_RECID_ALT_, as shown:

   ```
   ALTER TABLE table ADD PROGRESS_RECID AS
   CASE WHEN PROGRESS_RECID_ALT_ is NULL
       THEN PROGRESS_RECID_IDENT_
       ELSE PROGRESS_RECID_ALT_
   END PERSISTED NOT NULL,
   PROGRESS_RECID_IDENT_ bigint identity,
   PROGRESS_RECID_ALT_ bigint null default NULL,
   CONSTRAINT table#_table_name progress_recid UNIQUE(PROGRESS_RECID)
   ```

2. If you have already created your schema holder, delete and recreate it.

Migrating RECID-trigger mechanism to RECID computed column mechanism

Use the following procedure to convert a table that supports the ROWID function from one that is trigger-enabled to the more reliable computed column-enabled. Computed column enablement is available only through MS SQL Server 2005 and later.

Note: Prior to performing the following steps, ensure you have backed up the tables with which you will be working.

To adopt a table to support the ROWID function using computed column from trigger mechanism:

1. Drop the trigger on INSERT for PROGRESS_RECID column as:

   ```
   DROP TRIGGER _TI_tbl-name
   GO
   ```

2. Store the current identity value on the table as follows:

   ```
   select ident_current('<schemaname.tbl-name>') as old_identity_value
   go
   ```

3. Drop the indexes on PROGRESS_RECID, PROGRESS_REID_IDENT_ columns as:
4. Drop the other indexes which include PROGRESS_RECID column as:

```
drop index <table>#_#progress_recid on <table>
drop index <table>#_#progress_recid_ident_on <table>
```

5. Rename the PROGRESS_RECID column to PROGRESS_RECID_bkp as:

```
EXEC sp_rename '<tbl-name.PROGRESS_RECID', 'PROGRESS_RECID_BKP'
    ,'COLUMN'
```

6. Drop the PROGRESS_RECID_IDENT_ column as:

```
alter table <tbl-name> drop column PROGRESS_RECID_IDENT_
```

7. Add new columns for computed column mechanism as:

```
ALTER TABLE <tbl-name>
ADD PROGRESS_RECID AS
    CASE WHEN PROGRESS_RECID_ALT_ is null
        THEN PROGRESS_RECID_IDENT_
        ELSE PROGRESS_RECID_ALT_
        END PERSISTED not null,
    PROGRESS_RECID_IDENT_ bigint identity,
    PROGRESS_RECID_ALT_ bigint null default NULL,
CONSTRAINT <tbl-name>#_#progress_recid UNIQUE(PROGRESS_RECID)
GO
```

8. Update the PROGRESS_RECID_ALT_ column with PROGRESS_RECID_BKP as:

```
update <tbl-name> set PROGRESS_RECID_ALT_ = PROGRESS_RECID_BKP
go
```

9. Reseed the altered table with previously stored identity value (in step-2) as:

```
DBCC CHECKIDENT ('<tbl-name>', RESEED, <old-identity-value>)
go
```

10. Drop the PROGRESS_RECID_BKP column as:

```
alter table <tbl-name> drop column PROGRESS_RECID_BKP
go
```

11. Re-create the dropped indexes during Step 4 as:
12. If you have already created your schema holder, delete and recreate it.

Note: It is recommended that you back-up database tables before following the steps to adopt a new computed column mechanism.
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

For information on troubleshooting DataServer connections, see the “Connection failures and OpenEdge responses” section on page 6–28, and the “Accessing the DataServer log” section on page 6–30.
Tuning your environment with the –Dsrv startup parameter

The DataServer (–Dsrv) startup parameter allows you to use special DataServer options to tune your environment. You pass these options as arguments to –Dsrv when connecting to a MS SQL Server database.

There are two versions of the syntax, as follows:

Syntax

```
CONNECT data-source-name -ld logical-name -dt MSS
    -Dsrv arg1,va11 -Dsrv arg2,va12 -Dsrv arg3,va13...
```

Syntax

```
CONNECT data-source-name -ld logical-name -dt MSS
    -Dsrv arg1,va1,va12,va13,va14...
```

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 create 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 -U x -P y
    -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 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 8–1 lists the OpenEdge-supplied ODBC startup options and the corresponding startup options defined in the ODBC interface.

Table 8–1: ODBC options

<table>
<thead>
<tr>
<th>OpenEdge-supplied option</th>
<th>ODBC-defined option¹</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>MSS_PRESERVE_CURS</td>
<td>SQL_MSS_PRESERVE_CURS²</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>

¹ Refer to an ODBC application developer’s guide for information on the ODBC-defined options.
² Cursor preservation allows server-side cursors to be preserved beyond the transaction boundary. See the “Preserve cursors—the -Dsrv MSS_PRESERVE_CURS,1 option” section on page 8–10 for information and warnings.

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 `-Dsrv` 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 8–2 lists and describes these options.

#### Table 8–2: DataServer options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRGRS_ALWAYS_INDEX</td>
<td>Specifies if the DataServer should always order result sets. The value is either 1 for YES or 0 for NO. If this option is set off, then results that do not explicitly describe a desired ordering and/or that do not require ordering will be returned unordered. (The default setting is 0.) Note: The driver and data source need to have the capability to preserve cursors beyond a transaction boundary in order for the PRGRS_ALWAYS_INDEX option to be available.</td>
</tr>
<tr>
<td>PRGRS_BLOCK_CURS</td>
<td>Specified if the DataServer should use block cursors for NO-LOCK queries. The value is either 1 for YES or 0 for NO. This option is set to 1 by default.</td>
</tr>
<tr>
<td>PRGRS_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_DATEPART_OVERRIDE</td>
<td>Specifies the date component used to override the current date when an MSS Time column is converted to an OpenEdge DATETIME data type.</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. This parameter applies to all nonlookahead cursors.</td>
</tr>
<tr>
<td>PRGRS_LOCK_ERRORS</td>
<td>Specifies that when an application gets the stated error message, standard ABL 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>PRGS_LOGGINGLEVEL</td>
<td>Specifies the level at which log entries are written. For more information, see the “Analyzing performance with the Enhanced Logger” section on page 6–18.</td>
</tr>
</tbody>
</table>
Tuning your environment with the –Dsrv startup parameter

Table 8–2: DataServer options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRGS_LOGENTRYTYPES</td>
<td>Specifies one or more types of log entries. For more information, see the “Analyzing performance with the Enhanced Logger” section on page 6–18.</td>
</tr>
<tr>
<td>PRGRS_MAX_BLOCKSIZE</td>
<td>Specifies the size in bytes of the total allowable memory for all block cursors. By default, this value is set to ODBC_DEF_MAXBLOCKSZ, or 1MB.</td>
</tr>
<tr>
<td>PRGRS_MINBUF</td>
<td>Specifies the minimum size of the buffer used to get data from the DataServer. Use this keyword only with drivers that fail to fetch large records. The default value is 500 bytes. If you specify a value smaller than the default, OpenEdge uses the default value. The optimal setting for PRGRS_MINBUF is the size of the largest record data size plus 500 bytes. This can prevent run-time record expansion during the retrieval of query results. <strong>Note:</strong> Do not use this option when the -Dsrv BINDING switch is set to 3. With the binding set to 3, the size of the data is known, and this switch will cause the allocation of unneeded additional memory.</td>
</tr>
<tr>
<td>PRGRS_NATIVE_LOCKWAIT</td>
<td>Specifies the server-based wait period in which the server will block wait for access to a locked resource before returning to a MSS DataServer client application. The default value is zero, meaning that control is returned to the client immediately upon receiving a lock condition from the server. <strong>Note:</strong> A number of milliseconds is recommended at startup for transactional applications that anticipate a higher contention for server resources. The optimal value is application-specific.</td>
</tr>
<tr>
<td>PRGRS_NO_INDEX</td>
<td>Suppresses index creation at schema import time. If you specify a value of 1 for -Dsrv PRGRS_NO_INDEX, you must create the indexes manually in the schema holder.</td>
</tr>
<tr>
<td>PRGRS_PREPCACHE</td>
<td>Specifies the maximum number of prepared statements to be kept in cache. The default value is 20 statements.</td>
</tr>
<tr>
<td>PRGRS_TABLE_BLOCKSIZE</td>
<td>Specifies the total amount of memory allowed for block cursor use per table. The default value is ODBC_DEF_TABBLOCKSZ, 65,000 bytes.</td>
</tr>
</tbody>
</table>
The following example of the `-Dsrv` startup parameter sets the number of keys in the nonlookahead scrolling buffer to 100:

```
-Dsrv PRGRS_IDBUF,100
```

See the “Using MS SQL Server and DataServer options” section on page 8–6 for information on when and how to use these options.

### Using MS SQL Server and DataServer options

This section provides information on using various ODBC and DataServer options.

#### Transaction Management: the AUTOCOMMIT option

To avoid using the MS SQL Server transaction manager, specify `-Dsrv AUTOCOMMIT,1`. This option exercises the OpenEdge client’s local before-image mechanism to manage transactions. This 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.

**Note:** OpenEdge does not guarantee crash recovery when it uses the local before-image mechanism to emulate transactions.
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 DataServer for Microsoft SQL Server, 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 MS SQL Server. Opening a separate connection to run a query or stored procedure can provide better performance, although this is not guaranteed. Having too many open connections can 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, the session will need to wait for an existing connection to complete 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. This option 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 a 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.
Troubleshooting

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-login-pass;UIDDBMS=dblogin;
PWDDBMS=dblogin-pass
```

For more information and syntax examples, see the “Special connection issues” section on page 6–14.

**Key-buffer size—the PRGRS_IDBUF option**

The `PRGRS_IDBUF` option sets the size of the keys buffer. Generally, a default of 25 keys is sufficient. If the ODBC driver being used to access the MS SQL Server database has preserved cursors enabled across a transaction boundary, the keys buffer is used with all non-lookahead cursors. If the driver does have preserved cursors enabled, the `PRGRS_IDBUF` value and the keys buffer are unused.

**Locking error messages—the PRGRS_LOCK_ERRORS option**

DataServer for MS SQL Server identifies and handles conditions and errors. However, 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 an OpenEdge database lock problem; that is, the DataServer waits and retries, rather than halting the application:

```
CONNECT data-source-name -ld logical-name -dt mss
-Dsrv PRGRS_LOCK_ERRORS, error-number1, error-number2.
```

**Large rows—the PRGRS_MINBUF option**

Some data rows can be very large; for example, in a MS SQL Server database, 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.

The optimal setting for `PRGRS_MINBUF` is the size of the largest record data size plus 500 bytes. This can prevent run-time record expansion during the retrieval of query results.

**Notes:** Do not use this option when the `-Dsrv BINDING` switch is set to 3. With the binding set to 3, the size of the data is known, and this switch will cause the allocation of unneeded additional memory.

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.
Tuning your environment with the –Dsrv startup parameter

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:

- The MS SQL Server drivers are capable of re-using prepared statements and should do so whenever possible. However, 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 stored procedure. However, you can process results form 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, up to the maximum number of connections specified by the PRGRS_CACHE_CONN option.

Caution: When procedures run in separate connections of the same DataServer session, the scope of their respective transactions is isolated from one another. If one active procedure attempts to update the same record used by another active procedure in the same session, a lock timeout or even a deadlock could occur.

Cursor characteristics—the PRGRS_STABLE_CURS option

Enabling this switch indicates to the DataServer that it should assume that all cursors are stable. Normally, the ODBC driver and MS SQL Server determines 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.

Progress Software Corporation does not recommend bypassing normal operation under most circumstances, but under very limited circumstances, you can improve performance by overriding the derived cursor setting by setting the PRGRS_STABLE_CURS option to 1 (ON). Your application must comply to one of the following restrictions to safely enable this option:

- All ABL run in your DataServer session is without transactions.
- ABL run in your DataServer session has transactions, but all ABL queries and resultant data sets are fully processed on one side of any existing transaction boundary.

Note: This is a session-level switch, which means that all ABL run in the session must comply with the listed restrictions.
Troubleshooting

Wait time for asynchronous connections—the PRGRS_WAIT_DELAY option

The PRGRS_WAIT_DELAY switch allows you to determine the number of seconds you will allow the DataServer to delay further execution while waiting for the ODBC driver to respond to a request that may initiate a lock on a database resource. The delay is initiated when the NO-WAIT option is used in your ABL and the DataServer is awaiting a response to a database request made through the ODBC driver to MS SQL Server. The PRGRS_WAIT_DELAY switch 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.

Preserve cursors—the -Dsrv MSS_PRESERVE_CURS,1 option

The -Dsrv MSS_PRESERVE_CURS,1 startup option allows server side cursors in Microsoft SQL Server to be preserved beyond a transaction boundary. Under many circumstances, preserved cursors can result in a significant performance improvement. Enabling this feature can result in enhanced performance, but performance gains are not guaranteed and in limited circumstances performance degrades. For this reason, this option is disabled by default. To test if preserve cursors benefit your application, enable this option by toggling the switch on with -Dsrv MSS_PRESERVE_CURS,1.

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 Progress RECID/ROWID 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. The range limits for nnn are inclusive of values between 44 to 1024.

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
- + ...
SVUB,x—the Unified Broker Server Option

When \( x=0 \) (or when this parameter is unspecified) and the DataServer is connected client/server (i.e., the -S and/or -H parameters are specific), the client attempts to connect to a DataServer server through the traditional ProBroker brokering environment.

When \( x=1 \) and the DataServer is connected client/server (i.e., the -S and/or -H parameters are specific), the client’s broker connection is redirected away from the traditional ProBroker environment. Instead the client adopts the protocol used by the OpenEdge Explorer framework to make the broker connection. You must ensure that you have started a DataServer broker instance in the OpenEdge Explorer (or via the ubroker.properties files) before attempting to connect a client with the \(-Dsrv\ SVUB,1\) parameter. There are additional configuration parameters to also consider for connectivity in the Explorer framework. See Table 6–1 for DataServer connection parameters and see the “Starting and stopping a broker process from the OpenEdge Explorer and Progress Explorer and connecting a client” section on page 6–3 for additional connectivity information.

Logging options

When connecting to a client with the -Dsrv switch, you can specify logging options. Use logginglevel to define the granularity of your log reports, and use logentrytypes to specify the types of entries to be included in the log. For more information on logging, see the “Analyzing performance with the Enhanced Logger” section on page 6–18.

The following example demonstrates the use of the logginglevel parameter:

```
CONNECT data-source-name -ld logical-name -dt mss
-Dsrv logginglevel,3.
```

This next example demonstrates the use of the logentrytypes parameter:

```
CONNECT data-source-name -ld logical-name -dt mss
-Dsrv logentrytypes,SQL:3,Cursor:4,Trans.
```

In the above example, the Trans log entry type inherits the default logging level of 2 unless the logginglevel -Dsrv switch is also specified as a startup parameter - in which case that logging level would be set generically for all log entry types: SQL, Cursor and Trans.

Using the block cursor switches

The DataServer provides one switch to enable or disable block cursors, and two switches to regulate memory consumption by block cursors. Block cursors also interact with the query tuning cache size allocations.

Block cursors—the -Dsrv PRGRS_BLOCK_CURS,1 option

Block cursors are ON (1) by default. To disable block cursors for the entire connection, set this option to OFF (0). The connection-level setting can be overridden for a specific query with the query-tuning option: QUERY-TUNING(NO-LOOKAHEAD).
**Block cursor allocation by table—the `-Dsrv PRGRS_TABLE_BLOCKSIZE, nnn` option**

The `-Dsrv PRGRS_TABLE_BLOCKSIZE, nnn` option where `nnn` is the size in bytes, indicates the total allowable memory available to cursors of a given table. This value puts a cap on the amount of memory all the open cursors against a given table can consume. It does not determine an amount to be allocated. This upper limit on memory allocation ensures that memory is not overutilized.

`ODBC_DEF_TABBLOCKSZ` is the default value for `PRGRS_TABLE_BLOCKSIZE`. It is currently set at 65,000 bytes. At this rate approximately 18 tables open and utilizing all available cursor space for block cursors would utilize just 1 MB of memory. If record sizes are large or the typical row count of your result sets is large and you have plenty of available memory and your open table count is not too large, you might consider increasing this size. If you are using served OpenEdge clients, keep in mind that the memory requirements of the DataServer server executable on the server machine are compounded by the number of users attached to the database through the same broker.

**Total Block cursor allocation—the `-Dsrv PRGRS_MAX_BLOCKSIZE, nnn` option**

The `-Dsrv PRGRS_MAX_BLOCKSIZE, nnn` option where `nnn` is the size in bytes, indicates the total allowable memory available to all cursors established as block cursors. This value limits the total memory available to all block cursors irrespective of the table to which it is allocated. `ODBC_DEF_MAXBLOCKSZ` sets the default size which is currently 1MB. The overall memory allocated to block cursors by the DataServer connection cannot exceed the value set for `PRGRS_MAX_BLOCKSIZE`. Any block cursor that will place memory consumption over this threshold will be rejected as a block cursor and instead be established as a lookahead cursor.

**Impact of block cursors on cache size—the `-Dsrv QT_CACHE_SIZE, nnn` option**

The `-Dsrv QT_CACHE_SIZE, nnn` where `nnn` is the size in bytes of the cache to be allocated for a cursor’s result set. This is a connection level default that can be overridden at the query level. If the connection level cache size is not overridden at the query level, then the query cache size times the number of open cursors cannot exceed the maximum block areas for the table set by the `-Dsrv PRGRS_TABLE_BLOCKSIZE` switch at any given time. The accumulation of query cache sizes from each block cursor that has been allocated cannot exceed the total maximum block areas available to block cursors as set by the `-Dsrv PRGRS_MAX_BLOCKSIZE` switch. If either block cursor limit is reached, cursors will downgrade to lookahead cursoring. There is no minimum for this value, however if two or more records cannot be returned to the established block, a lookahead cursor is used. The query tuning cache size should be set higher than the maximum record size times two to prevent cursor downgrades.

`ODBC_DEF_BLKCACHESZ` is the default value for `QT_CACHE_SIZE` when block cursors are enabled. This value is currently set to 10,000 bytes.

`ODBC_DEF_LHDCACHESZ` is the default value of `QT_CACHE_SIZE` when lookahead cursors are enabled. It is currently set at 30,000 bytes.

The value of `QT_CACHE_SIZE` represents an upper limit for the row space available to the lookahead cache, not the amount of space a lookahead cache will actually use. This highlights a key distinction between block and lookahead cursors. The “cache size” for block cursors is preallocated before results are retrieved so this value represents an actual allocated amount. For lookahead cursors, memory is accrued as rows are read back from the result set and added to the cache. So the “cache size” specifies an upper limit on the number of cached rows allowable, not an exact amount of space that will be allocated as is the case with block cursors.
Adjusting values

With the default size, approximately 6.5 open cursors and result sets per table can be established at a given time before the maximum block memory area (PRGRS_TABLE_BLOCKSIZE) is exceeded. At 10,000 bytes, 104 new block allocations can be established before the maximum block memory (PRGRS_MAX_MBLOCKSIZE) is exceeded. If record sizes are large or the typical row counts for your results sets are large, you can consider increasing one or both of these sizes. Conversely, if record sizes are small or the typical row counts for your results are small, you can consider decreasing theses sizes. Consider setting these tuning parameters based on your typical requirements and then override them for specific queries that are the exceptions using the query tuning option.

ODBC driver problems

The ODBC drivers use the Microsoft 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.
This appendix discusses migration issues. It explains how to migrate from the DataServer for ODBC to the DataServer for MS SQL Server, and also discusses a trigger issue with upgrades from Version 9.1 of the MS SQL Server DataServer. Specifically, this appendix documents the topics outlined in the following sections:

- Creating a schema holder
- Modifying your application
- Modifying your insert trigger
Creating a schema holder

The MS SQL Server Utilities are located under the DataServer menu option in the Data Administration. Progress Software Corporation recommends that you use the Create DataServer Schema option to pull in all the objects that your application needs. This will create the schema information as a database type of MS SQL Server instead of ODBC. After completing the schema pull, the schema needs to be adjusted to conform to OpenEdge behavior.

To adjust the schema after completing the pull:

1. Connect to the original OpenEdge database.

2. Select the DataServer → MSSQL Server Utilities → Schema Migration Tools → Adjust Schema option to fix the OpenEdge attributes needed for your application to run. You can specify the entire schema holder or specific objects.
Modifying your application

Note the following considerations and plan, as needed:

- The DataServer for MS SQL Server supports the same functionality as the DataServer for ODBC. Therefore, you do not have to change your application. However, if you used any reference to DBTYPE as ODBC, you will have to change these to MS SQL Server.

- 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. To review additional considerations about determining when to use INT64 versus INTEGER, see 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.
Modifying your insert trigger

Progress Version 9.1D modified the insert trigger that it places on a table in the foreign database when the PROGRESS RECID support is selected. The modification reduces the level of contention between multiple users performing inserts simultaneously. The reduction in contention can produce performance improvement.

If your database was created using the OpenEdge DB to MS SQL Server utility and PROGRESS RECID support is enabled, Progress Software Corporation recommends that you re-create your database by dumping your data, executing the OpenEdge DB to MS SQL Server migration again, and then reloading your data. If this is not possible, then you can manually change your existing MS SQL Server database by doing the following on each table that has been defined with PROGRESS RECID support from your MS SQL Server Migration:

1. Drop the unique index created over the PROGRESS_RECID field. There should be one index for each table created during the initial migration. The syntax is:

   Syntax
   
   ```sql
   DROP index <database>.<owner>.<tablename>##progress_recid.
   ```

2. Drop the old insert trigger. The name typically looks like _TI_<tablename>. The syntax is:

   Syntax
   
   ```sql
   DROP trigger <database>.<owner>_TI_<tablename>
   ```

3. Add the new trigger. The syntax is:

   Syntax
   
   ```sql
   create trigger _TI_<tablename> ON <tablename> for insert as
   RAISERROR ('PSC-init',0,1)
   SET XACT_ABORT ON
   SET LOCK_TIMEOUT -1
   if ( select PROGRESS_RECID from inserted) is NULL
      begin
      update t set PROGRESS_RECID = i.IDENTITYCOL
      from customer t JOIN inserted i ON
      t.PROGRESS_RECID_IDENT_ = i.PROGRESS_RECID_IDENT_
      select convert (bigint, @@identity)
      end
   SET XACT_ABORT OFF
   RAISERROR ('PSC-end',0,1)
   RAISERROR ('PSC-end',0,1)
   ```
Sample code for the `db.owner.Customer` table is:

```
SET XACT_ABORT ON
SET LOCK_TIMEOUT -1
if ( select PROGRESS_RECID from inserted) is NULL
begin
    update t set PROGRESS_RECID = i.IDENTITYCOL
    from customer t JOIN inserted i ON
    t.PROGRESS_RECID_IDENT_ = i.PROGRESS_RECID_IDENT_
    select convert (bigint, @@identity)
end
SET XACT_ABORT OFF
RAISERROR ('PSC-end',0,1)
```

4. Create a new nonunique index for each table naming the new index the same name as was dropped earlier in Step 1. Make sure it is not unique. The syntax is:

**Syntax**

```
CREATE INDEX <database>.<owner>.<table>#_#progress_recid ON <tablename>(PROGRESS_RECID)
```

A second index must be created for the `PROGRESS_RECID_IDENT` field. This index must be created as unique and named `<tablename>#__progress_recid_ident`. The syntax is:

**Syntax**

```
CREATE UNIQUE INDEX <tablename>#__progress_recid_ident_ ON <tablename>(PROGRESS_RECID_IDENT_)
```

An example of the two indices that must be created for the `customer` table is as follows:

```
CREATE INDEX customer#_#progress_recid ON CUSTOMER (PROGRESS_RECID)
CREATE UNIQUE INDEX customer#_#progress_recid_ident_ ON CUSTOMER(PROGRESS_RECID_IDENT_)
```
Server Related Command Line Utilities and Startup Parameters

This appendix describes the following utilities and parameters that you use to configure, manage, start, and stop the DataServer host and client, as outlined in the following sections:

- Progress Explorer command line utilities for the DataServer
- Non-Progress Explorer command line utilities for the DataServer

See *OpenEdge Getting Started: Installation and Configuration* for additional information about the utilities and their role and relationship to other system administration facilities. See *OpenEdge Deployment: Startup Command and Parameter Reference* for additional information about syntax and usage.
Progress Explorer command line utilities for the DataServer

This section describes the utilities you use to configure, manage, start, and stop a DataServer. It discusses the purpose, syntax, and primary parameters for each operating system. The utilities are presented in alphabetical order. The utilities are:

- MSSCONFIG utility
- MSSMAN utility
- NSCONFIG utility
- NSMAN utility

MSSCONFIG utility

Use the MSSCONFIG utility to help you debug existing DataServer for MS SQL Server configurations defined in a properties file, such as the `ubroker.properties` file. This utility displays the property settings associated with a DataServer for MS SQL Server configuration, and checks that the syntax and values are valid.

The MSSCONFIG utility runs locally, on the machine on which the AdminService 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>mssconfig</td>
</tr>
<tr>
<td></td>
<td>[ [ [ -name DataServer-name ] [ -propfile path-to-properties-file ] [ -validate ] ] [ -help ] ]</td>
</tr>
</tbody>
</table>

Parameters

- `-name DataServer-name`

  Specifies which existing configuration to examine. The name must match the name of an existing DataServer for MS SQL Server configuration defined in the specified properties file. If you do not specify a DataServer by name, the MSSCONFIG utility analyzes all DataServer for MS SQL Server 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`. 
-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 MSSCONFIG, 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 DataServer for MS SQL Server configurations in ubroker.properties can 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.MS]</td>
<td>Defines default property settings for all instances of DataServers for MS SQL Server.</td>
</tr>
<tr>
<td>[UBroker.MS.product-instance-name]</td>
<td>Defines property settings for this instance of a DataServer for MS SQL Server. The ubroker.properties file can 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.MS], and then again by its child [UBroker.MS.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 can 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 MSSCONFIG to validate the changes and make sure there are no syntax errors or conflicts.

**MSSMAN utility**

Use the MSSMAN utility to control the operation of a configured DataServer for MS SQL Server. 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><code>mssman</code></td>
</tr>
</tbody>
</table>

```plaintext

{name DataServer-name

{ -kill
| -start
| -stop
| -query
}

[ -host host-name -user user-name
| -user user-name
]

[ -port port-number ]

| -help
}
```

**Parameters**

- `name DataServer-name`

  Specifies the name of a broker. This parameter is required.

- `-kill`

  Stops and removes the DataServer broker from memory, no matter what it is doing.

- `-start`

  Starts the DataServer broker.

- `-stop`

  Tells the DataServer broker to stop itself.

- `-query`

  Queries the DataServer brokers and servers for status.
-host  *host-name*

Specifies the name of the machine where the AdminService 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 AdminService is running. If a port number is not specified, it defaults to 20931.

-help

Displays command line help.

**Note**

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 AdminService.

- A user name as an explicit local user name, in which the user account is defined on the same machine that runs the AdminService 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 AdminService is running.
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 AdminService 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
  - `$DLC/properties/ubroker.properties` on UNIX

- `-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.

The NameServer configurations in `ubroker.properties` can 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>[NameServer]</td>
<td>Defines default property settings for all instances of a NameServer.</td>
</tr>
<tr>
<td>[NameServer.product-instance-name]</td>
<td>Defines property settings for this instance of a NameServer. The <code>ubroker.properties</code> file can 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 can 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 NSCONFIG 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. Unlike the OpenEdge Explorer and Progress Explorer, the NSMAN utility does not support a means to view log files or delete configured NameServer instances.

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>-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>}</td>
</tr>
</tbody>
</table>

Parameters

- **-name name-server**
  
  Specifies the name of the NameServer. This parameter is required.

- **-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 AdminService is running. If a host name is not specified, it defaults to the local host name.
Progress Explorer command line utilities for the DataServer

-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 AdminService 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 AdminService.

  - A user name as an explicit local user name, in which the user account is defined on the same machine that runs the AdminService 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 AdminService is running.
Non-Progress Explorer command line utilities for the DataServer

This section describes the command line utilities you use to start and stop a DataServer. It discusses the purpose, syntax, and primary parameters for each operating system. The utilities are presented in alphabetical order. The utilities are:

- PROBRKR command
- PROSHUT command
- DataServer startup parameters
**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 DataServer.

### Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>_probrkr.exe -S service-name [ -H host-name ]</td>
</tr>
</tbody>
</table>

### Parameters

- **service-name**
  
  Specifies the name of the broker process service on the host machine.

- **host-name**
  
  Specifies the name of the machine where the DataServer broker is installed. The default value is the local host.

### Notes

- See *OpenEdge Deployment: Startup Command and Parameter Reference* for more details on the Server Name (-S), 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 machine where 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 MSSSRV to the name of the executable (including the path) of the DataServer for MS SQL Server. Be sure to set this variable on the host machine. Also, in the same environment, make sure you have set all MS SQL Server environment variables required to connect to the data source. See Chapter 5, “Configuring the DataServer,” for examples of required variables.

- To set the Message Buffer Size (-Mm) startup parameter for the broker to a value different from the default buffer size of 1024, you must set the PROSTARTUP environment variable equal to the path and name of the .pf file that contains the -Mm parameter and value that you are passing before you start the broker. This approach ensures that your value is recognized and therefore used by the broker when it starts. Only Windows clients must use this technique.

- Start the broker on a server that is locally connected to the disk containing the data source.
PROSHUT command

Shuts down the OpenEdge database server and individual OpenEdge processes. Before you shut down the server, have all application users quit their OpenEdge sessions. If necessary, you can disconnect users by using the PROSHUT command’s Disconnect a User or Unconditional Shutdown parameters.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>proshut { db-name</td>
</tr>
</tbody>
</table>

Parameters

- `db-name`

  Specify the name of the schema holder to shut down its server, if it was started in multi-user mode.

- `-Gw`

  For DataServers, specifies that the DataServer broker is to be shut down.

- `-S service-name`

  Specifies the database server or broker process. You must specify the service name if you issue the shutdown command from a machine other than the host.

Note

For a complete listing of PROSHUT parameters and their functions, see *OpenEdge Data Management: Database Administration.*
## DataServer startup parameters

Table B–1 lists the parameters that you use with the command line utilities to start a DataServer for MS SQL Server.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataServer Logging</td>
<td>-dslog filename</td>
</tr>
<tr>
<td>DataServer</td>
<td>-Dsrv keyword,value</td>
</tr>
<tr>
<td>DataServer minimum port number</td>
<td>-dsminport port-num</td>
</tr>
<tr>
<td>DataServer maximum port number</td>
<td>-dsmaxport port-num</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>
Data Type Details

This appendix supplements the data type information presented in the “Data types” section on page 2–17. Specifically, this appendix presents each supported MS SQL Server data type, its OpenEdge data type equivalences, and any notes related to using each data type, as described in this section:

• Data type conversion details
Data type conversion details

Table C–1 identifies each supported DataServer for MS SQL Server data type, its SQL-ODBC equivalent data type, and its supported OpenEdge equivalent data types. In the OpenEdge equivalent data types column, the initial data type identified is the default. The data types in parentheses are alternative data types that you can specify in the schema holder for your MS SQL Server data source.

Note that the SQL-ODBC data types are presented as information only; you do not need to know nor use the SQL-ODBC data types to work with the DataServer for MS SQL Server.

<table>
<thead>
<tr>
<th>MS SQL Server data type</th>
<th>SQL-ODBC(^1) data type</th>
<th>OpenEdge-equivalent data types</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>SQL_INTEGER</td>
<td>INTEGER(^2) (DECIMAL)(^3) (INT64)(^3)</td>
</tr>
<tr>
<td>bigint</td>
<td>SQL_BIGINT</td>
<td>INT64(^4)</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) (INT64)(^3)</td>
</tr>
<tr>
<td>numeric(^5)</td>
<td>SQL_DECIMAL</td>
<td>DECIMAL (INTEGER) (INT64)(^3)</td>
</tr>
<tr>
<td>float(^6)</td>
<td>SQL_FLOAT</td>
<td>DECIMAL (INTEGER) (INT64)(^3)</td>
</tr>
<tr>
<td>double precision</td>
<td>SQL_DOUBLE</td>
<td>DECIMAL (INTEGER) (INT64)(^3)</td>
</tr>
<tr>
<td>real</td>
<td>SQL_REAL</td>
<td>DECIMAL (INTEGER) (INT64)(^3)</td>
</tr>
<tr>
<td>char(^7)</td>
<td>SQL_CHAR</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>nchar</td>
<td>SQL_NCHAR</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>varchar(^7)</td>
<td>SQL_VARCHAR</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>nvarchar</td>
<td>SQL_NVARCHAR</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>text(^9,8,10)</td>
<td>SQL_LONGVARCHAR</td>
<td>CHARACTER(^12)</td>
</tr>
<tr>
<td>varchar(max)(^11)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### MS SQL Server data type details

<table>
<thead>
<tr>
<th>MS SQL Server data type</th>
<th>SQL-ODBC data type</th>
<th>OpenEdge-equivalent data types</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntext9</td>
<td>SQL_NLONGVARCHAR</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>nvarchar(max)</td>
<td></td>
<td></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>CHARACTER13 (DATE14)</td>
</tr>
<tr>
<td>smalldatetime</td>
<td>SQL_TIMESTAMP</td>
<td>CHARACTER13 (DATE14)</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>CHARACTER12</td>
</tr>
<tr>
<td>bit</td>
<td>SQL_BIT</td>
<td>LOGICAL</td>
</tr>
<tr>
<td>timestamp15</td>
<td>SQL_VARBINARY</td>
<td>Unsupported</td>
</tr>
<tr>
<td>identity16</td>
<td>NA</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

1. The SQL-ODBC data types demonstrate the mapping of native MS SQL Server data types to the ODBC standard.
2. Existing or new 32-bit INTEGER data types have a maximum internal field size of 64-bits. The internal field expansion supports larger values defined as INTEGER data types.
3. Numeric expressions supported include DECIMAL, INTEGER, and INT64.
4. The OpenEdge INT64 data type enables the MS SQL Server DataServer BIGINT data type to be successfully pulled into a schema holder.
5. The DataServer truncates values in MS SQL Server 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.
6. Do not use the FLOAT or REAL data types in joins, in primary keys, or with the equality operator.
7. When you define a binary or char column to allow nulls, MSS stores the data type definitions as varbinary and varchar respectively. This does not affect how the DataServer maps the MSS data types to OpenEdge data types.
8. When migrating an OpenEdge database to SQL Server, character fields whose format is less than x(8000) are defined as VARCHAR by default. Larger character fields are defined as TEXT. If there are several large text fields in one record that each individually fall under the size limits to be defined as VARCHAR, but in sum exceed the maximum record size for SQL Server, it will be necessary to define the fields as TEXT to successfully create the records.
9. Text/ntext types changed to varchar(max)/nvarchar(max) in MS SQL Server 2005.
10. The amount of data that can be accessed in a field defined to be TEXT by an OpenEdge application is limited by the maximum size of a record that can be passed to the client. The maximum is 32K. The performance option BINDING cannot be used with a TEXT data type. To work around that limitation, construct field lists to exclude the retrieval of the TEXT data. Starting in MS SQL Server 2005, non-Unicode CLOB types are mapped to varchar(max) instead of text and Unicode CLOB types are mapped to nvarchar(max).
11. Starting in MS SQL Server 2005, CLOB types are mapped to varchar(max) instead of text.
12. Although the MS SQL Server text and image fields can hold up to 2MB, OpenEdge retrieves only up to 32,000 bytes. If you are using an OpenEdge Format phrase, there might be additional limits on the size of text and image fields. See *OpenEdge Development: ABL Reference*. You can use the DataServer (-Dsrv MAX_LENGTH) startup parameter to limit the amount of returned text data.
13. By default, the initial value of a MS SQL Server datetime or smalldatetime column is the Unknown value (?). The default initial values for binary and varbinary are also the Unknown value (?). The MS SQL Server datetime and smalldatetime data types contain both date and time information. The DataServer maps these to the OpenEdge DATE data type; however, you can change the DATE data type to CHARACTER 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 CHARACTER, specify a character format, such as x(26).
14. When mapping of the MS SQL Server `datetime` or `smalldatetime` data types is to the OpenEdge `DATE` data type, OpenEdge truncates the time portion of the date.

15. 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 MS SQL Server table that contains a timestamp column.

16. Identity columns are limited by SQL Server to one per table. Creating an identity column to define `PROGRESS_RECID` is the preferred method of supporting the `ROWID` function. See the “`ROWID function`” section on page 2–52 for more information. If your SQL Server table has an existing identity column, turn off the mandatory flag, and set its initial value to the Unknown value (?) in your schema holder to prevent the DataServer from attempting to overwrite the identity value generated by SQL Server.
Using qt_debug to Analyze Performance

The qt_debug option of the DataServer (-Dsrv) startup parameter can be used as a supplement to the OpenEdge Enhanced Logger to provide you with the means to perform diagnostics. Information in this appendix describes options used to enable logging with the qt_debug option. For information on using the Enhanced Logger as an alternative to analyzing application activity and performance, see “Analyzing performance with the Enhanced Logger” section on page 6–18 and OpenEdge Development: Debugging and Troubleshooting.
Using qt_debug to Analyze 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.1g 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 MS SQL Server database. Table D–1 lists the diagnostic capabilities of qt_debug.

### Table D–1: 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.</td>
</tr>
<tr>
<td>qt_debug,EXTENDED</td>
<td>Prints the SQL statements that the DataServer executes 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>
<tr>
<td>qt_debug,PERFORMANCE</td>
<td>Prints information on the amount of time that certain operations take. These statistics are available only for some platforms. Note that any time differences between what the DataServer and data source report might be due to network performance issues rather than to DataServer or data source behavior.</td>
</tr>
<tr>
<td>qt_debug,CALL_SUMMARY</td>
<td>Prints information on cursors and timing. This information is supplied in summary form as an OpenEdge data (.d) file. Contact Progress Software Corporation for assistance with this file. <strong>Note:</strong> This option is not available as a QUERY–TUNING phrase option.</td>
</tr>
<tr>
<td>qt_debug,VERBOSE</td>
<td>Prints all of the information gathered by the other qt_debug options.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This option generates a very large log file. Be sure to clear your log file before using this option to test a procedure.</td>
</tr>
</tbody>
</table>

**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 to run:

```
CONNECT msscdb -Dsrv qt_cache_size,32000,qt_debug,PERFORMANCE.
```
Index

Numbers
4GLTrans log entry type 6–22
64-bit sequences
   support for 2–16

A
ABL differences
   OpenEdge databases and MS SQL Server
data sources 2–58
ABL transaction logging
   log entry types 6–22
Aggregates
   in views 2–14
Arrays 2–31
   in joins 4–14
   manually creating support 7–37
   supporting in data source 7–37
Auto-commit 8–6
Auto-connect
   failure response 6–28

B
BEGINS function 2–58
Binding 6–15

Brokers
   DataServer B–11
   starting from MS-DOS 5–9, 6–6
   Unified Broker 5–23
Buffers
   allocating dynamically 8–8
   defining for stored procedures 3–18
BY option
   FOR EACH statement 2–11
   joins 4–14

C
Cache files
   changes to schema holders 6–27
   creating 6–11
   local schema cache 6–26
Caching
   schema 1–6
Case-insensitive indexes
   supporting in data source 7–37
Case-sensitive indexes 2–13
Character sets 2–5
   changing in a schema holder 7–18
   character sets
      defined 2–5
Clear dynamic temp-table
   defined 3–25
CLOSED STORED-PROCEDURE statement
  complete syntax reference 3–6
  description 3–3, 3–9
  retrieving parameter values 3–16

Code pages 2–5
  changing in a schema holder 7–18
  collation table 2–6
  conversion table 2–6
  setting in schema holder 5–18, 7–8

COMPILE statement 2–57

Configuration examples
  local DataServer 6–12
  remote DataServer 6–12

CONNECT statement
  failure response 6–28
  NO-ERROR option 6–28

Connecting schema holders 6–9
  at startup 6–9

Connection BINDING Options 6–16

Connection pool
  defined 4–2

Connection pooling 4–2
  considerations 4–3
  disabling 4–4
  enabling 4–4
  impact on commits 4–4
  ODBC 4–3

Connection to data source
  blank space in data source name 6–15
  DataServer opens multiple connections 6–29
  failure response due to existing connection 6–28
  failure response due to logical name 6–28
  failures with SQL Server 6–28
  guidelines 6–8
  interactive connection 6–15
  non-standard connections 6–15
  security on NT 6–8
  troubleshooting 6–29, 8–7

Connection-string
  defined 6–14

CONTAINS operator 2–58

COUNT-OF function 2–58

CREATE statement 2–34, 2–58

Creating a temp-table layout plan 3–23
  dynamic temp-table in an prepared state 3–26
  dynamic temp-table in an unprepared state 3–25
  options 3–24

Creating OpenEdge databases 5–17

Cursor
  defined 2–48

Cursors 2–48
  cursor downgrades 4–7
  monitoring cursor and connection use 4–6
  multiple 6–29
  statement cache 4–7

D

Data Dictionary
  changing data types 7–34
  connection failure response 6–28
  modifying schema holders 7–32

Data output and retrieval options
  results retrieved from a database 3–13
  defining a special view on the MSS data source 3–13
  OpenEdge-supplied proc-text-buffer 3–13
  return codes 3–13
  values of output parameters you define when creating a procedure 3–13

data source
  defined 2–3

Data sources 1–3
  adding objects to schema holder 5–20, 7–10
  code page 5–18, 7–8
  data types 2–17
  database objects 2–3
  keys 2–11
  locking behavior 2–36
  logical name 5–18, 7–8
  login name 2–59, 5–16
  naming conventions 2–4
  null value 2–31
  password 2–59, 5–16
  permissions 5–15
  physical name 5–19, 7–9
  registering 5–4
  security levels 1–7
  triggers 2–14
  views 2–13
  zero-length character strings 2–32
Data types 2–17, C–1
changing 7–34
conversion details C–2
display format 7–34
overflow checking 2–47
stored procedures 3–4
   INT64 in static and dynamic temp tables 3–24
   support for INT64 3–4
   supported by DataServer 2–17
   user-defined 2–31

Database conversion
   OpenEdge-to-MS SQL Server utility 7–20

Database Type (-dt) startup parameter 6–10

Databases
   OpenEdge objects 2–3
   SQL Server data-source objects 2–3
   triggers 2–14

DataServer
   architecture 1–4
   architecture components 1–3
   broker
      on host machine 5–6
   compiling applications 2–57
   components 5–2
   configurations 1–11
   configuring for distributed applications using probrkr 1–13
   configuring with Progress Explorer on NT 5–6
   connection failure 6–28
   creating schema holders 5–18, 7–8
   diagnostic options D–2
   error handling 2–45
   internal logic 1–6
   local configuration 1–11
   log file 5–10, 6–30
   MS SQL Server utilities 1–3
   on host machine 5–6
   options 8–4
   record creation 2–33
   remote configuration 1–12
   scrolling 2–48
   security considerations 1–7
   starting
      in Progress Explorer 6–3
      UNIX client 6–7
      Windows client 6–6
   utilities 1–9

DataServer (-Dsrv) startup parameter 6–11
   alternate syntax 6–16
   DataServer options 8–16
   ODBC options 8–3
   PRGRS_CONNECT option 6–14
   qt_debug option 6–17
   qt_lookahead option 6–17
   qt_no debug option 6–17
   qt_no_lookahead option 6–17
   qt_no_separate_connection option 6–17
   qt_separate_connection option 6–17
   query-tuning 6–16
   stored procedures 3–4
   tuning your environment 8–2

DataServer for SQL Server 1–3

DataServers
   starting a broker B–11

DBE DataServer
   startup parameters 7–21

DBRESTRICTIONS function 2–58

Deadlock 2–41
   defined 2–41
   stored procedures 3–4

Default result set
   defined 4–5

DEFINE BROWSE statement 2–54

DEFINE BUFFER statement 3–16

Defining a view to use as a buffer 3–18

Deleting a view to use as a buffer 3–18

Deleting schema holders 7–19

Demonstration databases 1–10, 7–2
   requirements for use 7–2

Diagnostic options
   qt_debug D–2

Display formats 7–34

Distributed DataServer
   configured using probrkr 1–13

Documentation
   suggested reading list 1–19

DSLOGDIR environment variable 5–10, 6–30

Dummy indexes
   for non-indexed data sources 2–12

Dynamic temp-table 3–8

Dynamic temp-table error conditions 3–25
E
Enterprise DataServer for SQL Server 1–2, 5–2
Environment variables
   DSLOGDIR 5–10
   MSSSRV 5–10
   PROBRKR 5–10
Error handling 2–45
Error messages
   connection failures 6–28
   locking problems 8–8
Error messages related to using the LOAD-RESULT-INTO phrase 3–30
Exclusive locks 2–37, 2–39
EXCLUSIVE-LOCK option 2–59
   in joins 4–14
   record locking 2–36
expression
   defined 3–9
Extended 4GL
   adding support to data source 7–37
   supported features 1–17
F
Fast Forward-Only cursors 4–6
field extends
   defined 2–31
Field extents 2–31
Field formats 7–34
Field lists 2–54
Field properties
   modifying 7–33
Field size
   retrieval limits C–3
FIELDS option 2–54
FIND statement 2–11, 2–56, 2–58
   LAST option 2–56
   PREV option 2–56
Firehose cursors 4–5
FOR statement 2–58
   EACH option 2–11, 2–58
Foreign keys 2–11
H
Handling errors 3–30
   Error messages related to using the LOAD-RESULT-INTO phrase 3–30
Holding locks 2–37
Host Name (-H) startup parameter 6–10
I
Incremental schema changes
   support for changing INTEGER to INT64 using ALTER TABLE ALTER COLUMN statement 7–30
Indexes 2–11
   case-sensitive 2–13
   dummy 2–12
   for stored procedure results 2–14
   for views 2–14
   problems importing 8–8
   selecting for views 2–12
   simulated 2–12
   support for large key entries 2–12
   unique 2–12
   FIND statements 2–12
   for browsing 2–12
INPUT option
   stored procedures 3–9
INPUT-OUTPUT option
   retrieving parameter values 3–16
   stored procedures 3–9
INT64 data type
   MS SQL Server data type equivalents C–2
Internal Code Page (-cpinternal) startup parameter 2–6
   OpenEdge-to-MS SQL Server utility 7–21
Isolation levels 2–37
J
Join by SQLDB
   defined 4–14
   option 4–14
JOIN-BY-SQLDB option 4–15
Joins 2–13
   controlling by SQLDB behavior 4–15
K

Key buffer
troubleshooting 8–8

Keys
foreign 2–11
primary 2–11

L

Loading result sets into temp-tables 3–21
creating a temp-table layout plan 3–21
Using a temp-table handle with an
unprepared dynamic temp-table 3–24
LOAD-RESULT-INTO phrase 3–3
description 3–7

Local DataServer 1–11
starting 6–2

LocalCache (-cache) startup parameter 6–11

Locks
deadlock 2–41
page-level 2–37
record locking 2–36
stored procedures 3–4
table-level 2–37
upgrading 2–37

Log entry types
4GLTrans 6–22

Log files 5–10
accessing 6–30

Logical database name 5–18, 7–8
changing 7–17

Logical Database Name (-ld) startup parameter 6–10

Login names 5–16, 6–10, 6–29
SETUSERID function 2–59

LOG-MANAGER handle
attributes and methods 6–18

M

Maintaining schema holders 7–3
recompiling code 5–22

MATCHES function 2–58

mergeprop utility 1–14

Microsoft SQL Server
data types 2–17

Mixed array
defined 3–24

Modifying schema holders 7–32

MSSCONFIG Utility B–2

MSSMAN Utility B–4
DataServer processes on UNIX 6–5

MSSSRV environment variable 5–10

N

Naming conventions 2–4
keywords in object names 2–4
non-unique table names 2–4

NO-ERROR option 3–30
description 3–8

NO-JOIN-BY-SQLDB option 4–14

NO-LOCK option 2–59
record locking 2–36

Non-unique table names 2–4

NO-WAIT option 2–4

NSCONFIG B–6

NSMAN B–8

NULL search condition 2–59

Null value
in query 2–31
ROWID 2–53

O

ODBC
Application Programming Interface
(API) 1–6
compliance 1–1
data source 1–3
driver 1–3
interactive connections 6–15
NT 5–4
troubleshooting 8–13
unstable DLLs 8–13
Windows 5–4
Driver Manager 1–3
options 8–3

ODBC connection pooling 4–3

OF option
FIND statement 2–11
FOR EACH statement 2–11

OPEN QUERY statement 2–58
OpenEdge
- data types 2–17
- database objects 2–3
- database security levels 1–7
- locking behavior 2–36
- naming conventions 2–4
- triggers 2–14
- Unknown value (?) 2–31
- zero-length character strings 2–32

OpenEdge sequences
- 64-bit sequences 2–16

OpenEdge-to-MS SQL Server utility 7–25
- migrating OpenEdge database to SQL Server data source 7–20
- parameters 7–3, 7–13, 7–21
- running in batch mode 7–21
- running interactively 7–20

OUTPUT option
- retrieving parameter values 3–16
- stored procedures 3–9

Overflow checking
- defined 2–47
- potential conflicts 2–47

P

PARAM option
- retrieving parameter values 3–16
- stored procedures 3–9

PARAM phrase
- description 3–3, 3–8

Param phrase
- defined 3–8

Parameter (.pf) files
- connection requirements for DataServer 6–12

Parameters
- abbreviating names 8–2

Passing ODBC-specific information
- PRGRS_CONNECT option 6–14

Password (-P) startup parameter 6–11

Passwords 5–16, 6–29
- SETUSERID function 2–59

Performance
- caching records 4–13
- joins 4–14
- modifying run-time schema check 4–17
- OpenEdge query tips 4–8
- query tuning 4–9
- writing queries 4–16

Permissions
- data source security 1–7
- establishing 5–15
- requirements 5–15

Physical database name 5–19, 7–9

Physical Database Name (-db) startup parameter 6–10

PRGRS_CONNECT option
- DataServer (-Dsrv) startup parameter 8–7

PRGRS_IDBUF option
- DataServer (-Dsrv) startup parameter 8–8

PRGRS_LOCK_ERROR option
- DataServer (-Dsrv) startup parameter 8–8

PRGRS_MNBUF option
- DataServer (-Dsrv) startup option 8–8

PRGRS_PREPCACHE option
- DataServer (-Dsrv) option 8–9

Primary keys 2–11

pro executable
- UNIX example 6–7

PROBRKR command B–11

PROBRKR environment variable 5–10

Procedures
- compiling 2–57

Processing considerations
- 32-bit and 64-bit data types 2–20
- caution in changing INTEGER data types 2–21
- requiring additional action 2–20
- bigint data type and pulling a table 2–20
- data definition changes 2–20
- r-code references 2–20

PROC-HANDLE function 3–20

PROC-HANDLE phrase
- description 3–3, 3–8

PROC-STATUS function 3–15

PROC-STATUS phrase
- description 3–3, 3–8

ProDataSet support 3–36

Progress Explorer
- configuring the DataServer on NT 5–6
- starting DataServer on NT 6–3
- stopping DataServer on UNIX 6–5
Progress Explorer Framework 5–23
and the mergeprop utility 1–14
and the Progress Explorer tool 1–14
configuring DataServer applications using 1–14
using the mergeprop utility 5–23
using the Progress Explorer tool 5–23

Progress Explorer tool
definition 1–14
PROGRESS_RECID
and INT64 data type 7–37
PROSHUT command B–12

Q
qt_debug
CALL_SUMMARY option
DataServer (-Dsrv) startup parameter 6–17
CURSOR option
DataServer (-Dsrv) startup parameter 6–17
EXTENDED option
DataServer (-Dsrv) startup parameter 6–17
PERFORMANCE option
DataServer (-Dsrv) startup parameter 6–17
SQL option
DataServer (-Dsrv) startup parameter 6–17
VERBOSE option
DataServer (-Dsrv) startup parameter 6–17
qt_debug option
CALL SUMMARY option D–2
CURSOR option D–2
EXTENDED option D–2
PERFORMANCE option D–2
SQL option D–2
VERBOSE option D–2
qt_lookahead option
DataServer (-Dsrv) startup parameter 6–17
qt_no_debug option D–2
DataServer (-Dsrv) startup parameter 6–17
qt_no_lookahead option
DataServer (-Dsrv) startup parameter 6–17
qt_no_separate_connection option
DataServer (-Dsrv) startup parameter 6–17
managing data sources 6–30
qt_separate_connection option
DataServer (-Dsrv) startup parameter 6–17
managing data sources 6–30
Query tuning
precedence of methods 6–16
startup parameters 6–16, 6–18
Query-tuning options 4–10
ARRAY-MESSAGE 4–10
CACHE-SIZE 4–10
DEBUG SQL 4–10
DEBUG-EXTENDED 4–10
JOIN-BY-SQLDB 4–10
LOOKAHEAD 4–11
NO-ARRAY-MESSAGE 4–10
NO-DEBUG 4–10
NO-JOIN-BY-SQLDB 4–10
NO-LOOKAHEAD 4–11
NO-SEPARATE-CONNECTION 4–11
SEPARATE-CONNECTION 4–11
QUERY-TUNING phrase 4–9

R
R-code 2–57
size 2–58
RDBMS stored procedure basics 3–3
Read-only (-RO) startup parameter 6–9,
6–11
Read-only schema holders 6–11
RECID function 2–53
replacing with ROWID 2–53
Recompiling code
schema changes 5–22
Record caching
lookahead cursors 4–13
standard cursors 4–13
Record creation 2–33, 2–59
Record locking
data sources 2–36
exclusive locks 2–37, 2–39
EXCLUSIVE-LOCK option 2–36
NO-LOCK option 2–36
shared locks 2–37
SHARE-LOCK option 2–36
Recovery
on data sources 2–43
Register
defined 1–3
Registering data sources 5–4

RELEASE statement 2–35

Remote databases
starting a DataServer broker B–11

Remote DataServer 1–12
starting 6–3

Requirements
software 1–16

result 3–3

Result set
definition 3–3

Retrieving output parameter values 3–16

Retrieving result sets using proc-text-buffer 3–16

Retrieving return codes 3–15

Roll back
on data sources 2–43

ROWID characteristics 3–32

ROWID function 2–52
defining supporting index 7–35
listing objects not supported 5–21, 7–10
manually creating support 7–37
reposition functions, INTEGER data type, and INT64 data type 2–53
supporting in data source 7–37

ROWID support 3–32
additional ProDataSet support 3–36
ROWID characteristics 3–32
using ROWID with RUN STORED-PROCEDURE and LOAD-RESULT-INTO 3–32

Rows
troubleshooting large rows 8–8

RUN STORED-PROCEDURE statement
complete syntax reference 3–6
description 3–3
retrieving parameter values 3–16
send-sql-statement option
with LOAD-RESULT-INTO option 3–12
without LOAD-RESULT-INTO option 3–11
using the send-sql statement option 3–11
with LOAD-RESULT-INTO phrase 3–11
with LOAD-RESULT-INTO phrase syntax 3–11

without LOAD-RESULT-INTO phrase syntax 3–10

Run Stored-Procedure statement
details 3–7
handling errors 3–30
without LOAD-RESULT-INTO phrase 3–10

S

-S shutdown parameter
PROSHUT command B–12

SAVE CACHE COMPLETE statement 6–11, 6–27, D–2

Schema
defined 1–6
loading process 1–6

Schema caching
defined 1–6

Schema holder
before you begin 5–15
defined 1–6

Schema holders 1–3, 1–6
changing logical name 7–17
connecting 6–9
connection failure 6–28
connection permissions 5–16
creating 5–15
deleting 7–19
maintaining 5–22, 7–3
modifying 7–32
setting code page 5–18, 7–8
system administration permissions 7–7
updating 5–22, 7–11
verifying 7–13
where best to build 5–15

Schema images
security 5–15

Scrolling
backward 2–48
forward 2–48

Security
guideline considerations 1–7

send-sql-statement option
RUN-STORED-PROCEDURE statement 3–11

Sequence generator 2–15
defined 2–15

Server Join (-nojoinbysqldb) startup parameter 4–10, 4–14, 6–18
Index

Service Name (-S) shutdown parameter
PROSHUT B–12

Service Name (-S) startup parameter 6–10

SETUSERID function 2–59

Shared locks 2–37

SHARE-LOCK option 2–59
record locking 2–36

Shutdown commands B–12

Simulated indexes
for nonindexed data sources 2–12

Single-user Mode (-1) startup parameter
6–11

Skip schema verification 4–17

smalldatetime data type
on Microsoft SQL Server 2–17, C–3

Software requirements 1–16

SQL ORDER BY option
OpenEdge USE-INDEX option 2–11

SQL Server
Administrator 1–3
creating schema holders 5–18, 7–8

SQL statement cache
troubleshooting 8–9

Starting a DataServer broker B–11

Startup parameters
Database Type (-dt) 6–10
DataServer (-Dsrv) 6–11, 6–16
Host Name (-H) 6–10
Internal Code Page (-cpinternal) 2–5
LocalCache (-cache) 6–11
Logical Database Name (-ld) 6–10
Password (-P) 6–11
Physical Database Name (-db) 6–10
Read-only (-RO) 6–9, 6–11
Server Join (-nojoinbysqldb) 6–11, 6–18
Service Name (-S) 6–10
Single-user Mode (-1) 6–11
unsupported 6–14
User ID (-U) 6–10

Static temp-table 3–8

Stored procedure
defined 3–2
implementing and running points 3–4
language elements 3–3

Stored procedures
and dataserver performance
improvement 3–2

basics 3–3
browsing results 2–12
data types 3–4
defining buffers 3–18
INPUT option 3–9
INPUT-OUTPUT option 3–9
joining database results 3–20
OUTPUT option 3–9
PARAM option 3–9
returning status values 3–9
unique indexes for results 2–12

Stream Code Page (-cpstream) startup parameter 7–21

T

Table properties
modifying 7–32
ROWID 7–36

Table validation 7–32

Temp-table
and INT64 data type 3–24
dynamic 3–8
loading results into using the
LOAD-RESULT-INTO phrase 3–14
static 3–8
that support result sets 3–8

Terminology
differences between data sources 2–3

Transaction scoping
OpenEdge 2–43

Transactions 2–43

Triggers 2–14
defined 2–14

Troubleshooting 8–1

U

ubroker.properties file 5–10
hierarchy B–3, B–7

Unicode
support for 2–9

Unique indexes 2–12
stored procedures results 2–12

Unknown value (?)
in query 2–31
ROWID 2–52

Unsupported 4GL statements 2–58

Updating schema holders 7–11
Index

USE-INDEX option
   FOR EACH statement 2–11

User ID 5–16
   SETUSERID function 2–59
   startup parameter (-U) 6–10

User-defined data types 2–31

USING phrase
   joins 4–14

V

VALIDATE statement 2–35

Validation
   expressions 7–32, 7–34
   messages 7–32

Verifying schema holders 7–13

View
   defined 2–13

Views 2–13
   aggregates 2–14
   browsing 2–14

W

Word index
   CONTAINS operator 2–58

Z

Zero-length character strings 2–32