A Fast Path to AS/400 Client/Server
Using AS/400 OLE DB Support

Bob Maatta, Mehboob Alam, Geert Maertens, Ray McRoberts, Craig Pelkie, Pankaj Sharma
A Fast Path to AS/400 Client/Server
Using AS/400 OLE DB Support

July 1998
Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix D, “Special Notices” on page 403.

First Edition (July 1998)

This edition applies to Version 3, Release 1, Modification 3 of Client Access for Windows 95/NT for use with the OS/400 Version 3, Release 1, Modification 0 operating system.

Note

This book is based on a pre-GA version of a product and may not apply when the product becomes generally available. We recommend that you consult the product documentation or follow-on versions of this redbook for more current information.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JLU Building 107-2
3605 Highway 52N
Rochester, Minnesota 55901-7829

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1998. All rights reserved
Note to U.S Government Users - Documentation related to restricted rights - Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.
Figures ................................................................. ix

Tables ..................................................................... xiii

Preface .................................................................... xv
The Team That Wrote This Redbook ............................... xv
Comments Welcome ...................................................... xvi

Chapter 1.  OLE DB and ActiveX Overview ...................... 1
  1.1  The Quick Summary ............................................. 1
  1.2  Past Practices for Windows to the AS/400 System ............ 2
      1.2.1  Developing Windows to AS/400 applications with ODBC .... 2
      1.2.2  Client Access APIs ........................................... 4
      1.2.3  Third-Party Products ......................................... 8
  1.3  OLE DB – The Foundation of AS/400 OLE DB Support .......... 8
      1.3.1  What OLE DB Does ........................................... 9
      1.3.2  What is COM? .................................................. 10
      1.3.3  Back to OLE DB ............................................... 13
  1.4  The ActiveX Data Object Components .......................... 13
      1.4.1  What Does ActiveX Mean? ................................. 13
      1.4.2  The ADO Object Model ..................................... 14
  1.5  Working with ADO objects ....................................... 18
      1.5.1  Dependencies among ADO Objects ....................... 18
      1.5.2  Connection Object ............................................ 20
      1.5.3  Command Object ............................................. 21
      1.5.4  Recordset Object ............................................. 26
      1.5.5  Fields Collection, Field Object ............................ 30
      1.5.6  Errors Collection, Error Object ............................ 32
      1.5.7  Parameters Collection and Parameter Object ............... 35
      1.5.8  Properties Collection, Property Object .................... 38

Chapter 2.  AS/400 OLE DB Support ................................. 39
  2.1  Introduction ....................................................... 39
  2.1.1  Client Support ................................................ 39
  2.1.2  Host Support .................................................. 40
  2.1.3  One Common Interface ....................................... 42
  2.1.4  Final Product Distribution ................................... 42
  2.2  OLE DB Provider ................................................ 42
      2.2.1  Registering OLE DB Provider DLL .......................... 43
      2.2.2  Naming Conventions ......................................... 43
      2.2.3  Supported Properties and Methods ........................ 44
      2.2.4  Error Handling ................................................. 44
  2.3  ADO Object Reference .......................................... 45
  2.4  Client Access OLE DB Functions ............................... 46
      2.4.1  SQL Statements ............................................... 47
      2.4.2  Opening a Connection ....................................... 48
      2.4.3  Stored Procedures Calls .................................... 52
      2.4.4  Record-Level Access ........................................ 63
      2.4.5  AS/400 Data Queues ....................................... 75
      2.4.6  Remote Commands and Distributed Program Calls ......... 80
Chapter 3. Overview of the Order Entry Application .............................................. 87
3.1 Overview of the Order Entry Application ......................................................... 87
    3.1.1 The ABC Company .................................................................................... 87
    3.1.2 The ABC Company Database ...................................................................... 87
    3.1.3 A Customer Transaction ........................................................................... 88
    3.1.4 Application Flow ....................................................................................... 88
    3.1.5 Customer Transaction Flow ....................................................................... 89
    3.1.6 Database Table Structure ......................................................................... 96
    3.1.7 Order Entry Application Database Layout ................................................ 97
    3.1.8 Database Terminology .............................................................................. 101

Chapter 4. Application Development Using AS/400 OLE DB Support .................... 103
4.1 Migrating the RPG Program to Use AS/400 OLE DB Support ............................ 104
4.2 Working with the Program .............................................................................. 107
    4.2.1 First Time Processing ............................................................................. 110
    4.2.2 AS/400 System Name .............................................................................. 111
    4.2.3 Effect of Changing Options ....................................................................... 112
    4.2.4 Procedure Trace Option ........................................................................... 112
    4.2.5 Changing Options .................................................................................... 113
    4.2.6 Use of the Registry ................................................................................. 113
    4.2.7 Error Report ........................................................................................... 114
4.3 Setting References to ADO in Visual Basic ...................................................... 115
4.4 Preparing AS/400 Program Objects .................................................................... 118
4.5 Overview of the Visual Basic Application ....................................................... 119
4.6 Detailed Description of Forms Used in the Project ............................................ 120
    4.6.1 frmCustomer — Customer Details ........................................................ 120
    4.6.2 frmCustomerSearch — Customer Search ............................................... 122
    4.6.3 frmErrors — ADO Error Report .............................................................. 123
    4.6.4 frmItem — Item Details .......................................................................... 124
    4.6.5 frmItemSearch — Item Search ............................................................... 125
    4.6.6 frmOptions — Options for ADOTest ...................................................... 127
    4.6.7 frmOrder — Order Entry Using ADO ...................................................... 128
4.7 Review of ADO Code Used in the Order Entry Using ADO Application ............ 130
    4.7.1 ADO Code in modMain.bas .................................................................... 130
    4.7.2 ADO Code in Class CADO ....................................................................... 131
    4.7.3 ADO Code in Class CCustomer ................................................................ 132
    4.7.4 ADO Code in Class CCustomers ............................................................. 142
    4.7.5 ADO Code in Class CDistrict .................................................................. 146
    4.7.6 ADO Code in Class CErrors ................................................................... 146
    4.7.7 ADO Code in Classes CItem and CItems ................................................. 148
    4.7.8 ADO Code in Class CLinItems .................................................................. 148
    4.7.9 ADO Code in Class COrder .................................................................... 153
    4.7.10 ADO Code in Class COrderHeader ....................................................... 158
    4.7.11 ADO Code in Class CStock .................................................................... 158
    4.7.12 Final Notes about this Sample ............................................................... 161

Chapter 5. PowerBuilder Using OLE/DB ................................................................. 163
5.1 Introduction to Application Examples ............................................................... 163
5.2 An SQL Application Example .......................................................................... 164
    5.2.1 Opening a Connection ............................................................................ 167
    5.2.2 Get One Part-SQL ................................................................................... 167
    5.2.3 Get All Parts - SQL ............................................................................... 169
    5.2.4 Error Handling ....................................................................................... 171
<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4.9</td>
</tr>
<tr>
<td>6.5</td>
</tr>
<tr>
<td>6.5.1</td>
</tr>
<tr>
<td>6.5.2</td>
</tr>
<tr>
<td>6.5.3</td>
</tr>
<tr>
<td>6.5.4</td>
</tr>
<tr>
<td>6.5.5</td>
</tr>
<tr>
<td>6.6</td>
</tr>
<tr>
<td>6.6.1</td>
</tr>
<tr>
<td>6.6.2</td>
</tr>
<tr>
<td>6.7</td>
</tr>
<tr>
<td>6.7.1</td>
</tr>
<tr>
<td>6.7.2</td>
</tr>
<tr>
<td>6.7.3</td>
</tr>
<tr>
<td>6.7.4</td>
</tr>
<tr>
<td>7.1</td>
</tr>
<tr>
<td>7.2</td>
</tr>
<tr>
<td>7.2.1</td>
</tr>
<tr>
<td>7.3</td>
</tr>
<tr>
<td>7.4</td>
</tr>
<tr>
<td>7.4.1</td>
</tr>
<tr>
<td>7.4.2</td>
</tr>
<tr>
<td>7.4.3</td>
</tr>
<tr>
<td>7.4.4</td>
</tr>
<tr>
<td>7.5</td>
</tr>
<tr>
<td>7.5.1</td>
</tr>
<tr>
<td>7.5.2</td>
</tr>
<tr>
<td>7.5.3</td>
</tr>
<tr>
<td>7.5.4</td>
</tr>
<tr>
<td>7.6</td>
</tr>
<tr>
<td>7.6.1</td>
</tr>
<tr>
<td>7.6.2</td>
</tr>
<tr>
<td>7.7</td>
</tr>
<tr>
<td>7.7.1</td>
</tr>
<tr>
<td>7.7.2</td>
</tr>
<tr>
<td>7.8</td>
</tr>
</tbody>
</table>

Chapter 7. Lotus Notes with OLE DB

| 7.1     | Introduction to Application Examples |
| 7.2     | An SQL Example |
| 7.2.1   | Error Handling |
| 7.3     | A Stored Procedure Example |
| 7.4     | A Record Level Access Example |
| 7.4.1   | Opening the File for Record-Level Access |
| 7.4.2   | Navigating the Recordset |
| 7.4.3   | Updating the Table |
| 7.4.4   | Finding a Record |
| 7.5     | A Data Queue Application Example |
| 7.5.1   | Data Queue Server Program Background |
| 7.5.2   | Opening the Data Queue |
| 7.5.3   | Get All Parts - Data Queue |
| 7.5.4   | Closing a Data Queue |
| 7.6     | Remote Command Execution Example |
| 7.6.1   | Executing a Remote Command |
| 7.6.2   | Restrictions |
| 7.7     | A Distributed Program Call Example |
| 7.7.1   | Get a Part Using a Program Call |
| 7.7.2   | Restrictions |
| 7.8     | A Sample Notes Application |

Chapter 8. Microsoft Visual FoxPro Using OLE/DB

| 8.1     | Introduction to Application Examples |
| 8.2     | An SQL Application Example |
| 8.2.1   | Opening a Connection |
| 8.2.2   | Get One Part-SQL |
| 8.2.3   | Get All Parts - SQL |
| 8.2.4   | Error Handling |

Chapter 9. End-User Tools

| 9.1     | Lotus 1-2-3 |
| 9.2     | Microsoft Excel |
| 9.3     | Microsoft Access 97 |

Chapter 10. Internet-Based Applications

<p>| 10.1    | Client Side Scripting |</p>
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.2 Server Side Scripting: an ASP example</td>
<td>307</td>
</tr>
<tr>
<td>10.3 Using a Custom Business Object</td>
<td>311</td>
</tr>
<tr>
<td>10.3.1 Creating the Web Page</td>
<td>312</td>
</tr>
<tr>
<td>10.3.2 Creating the Business Object</td>
<td>318</td>
</tr>
<tr>
<td>10.3.3 Installing the Business Object</td>
<td>320</td>
</tr>
<tr>
<td>10.3.4 Marking the Business Object as Safe</td>
<td>321</td>
</tr>
<tr>
<td>10.3.5 Advantages and Disadvantages</td>
<td>323</td>
</tr>
<tr>
<td>Chapter 11. Three-Tier Approach</td>
<td>325</td>
</tr>
<tr>
<td>11.1 Microsoft OLE DB Provider for AS/400 and VSAM</td>
<td>325</td>
</tr>
<tr>
<td>11.1.1 Overview</td>
<td>325</td>
</tr>
<tr>
<td>11.1.2 A Comparison with the Client Access OLE DB Provider</td>
<td>326</td>
</tr>
<tr>
<td>11.1.3 Software Prerequisites</td>
<td>327</td>
</tr>
<tr>
<td>11.1.4 Configuration Issues</td>
<td>328</td>
</tr>
<tr>
<td>11.1.5 Developing with the Microsoft OLE DB Provider</td>
<td>331</td>
</tr>
<tr>
<td>11.1.6 Comparing Performance with the Client Access OLE DB Provider</td>
<td>335</td>
</tr>
<tr>
<td>11.1.7 Summary</td>
<td>336</td>
</tr>
<tr>
<td>11.2 Client Access OLE DB Provider</td>
<td>337</td>
</tr>
<tr>
<td>11.2.1 Overview</td>
<td>337</td>
</tr>
<tr>
<td>11.2.2 Configuration Issues</td>
<td>337</td>
</tr>
<tr>
<td>11.2.3 Programming in a Three-Tier Environment</td>
<td>339</td>
</tr>
<tr>
<td>11.2.4 Changing an Existing Application</td>
<td>340</td>
</tr>
<tr>
<td>11.2.5 Summary</td>
<td>350</td>
</tr>
<tr>
<td>Chapter 12. Security Considerations</td>
<td>351</td>
</tr>
<tr>
<td>12.1 Overview</td>
<td>351</td>
</tr>
<tr>
<td>12.1.1 Resource Centric Security</td>
<td>351</td>
</tr>
<tr>
<td>12.1.2 Access Centric Security</td>
<td>353</td>
</tr>
<tr>
<td>Chapter 13. Performance Measurements and Comparisons</td>
<td>361</td>
</tr>
<tr>
<td>13.1 Measurement Environment</td>
<td>361</td>
</tr>
<tr>
<td>13.2 Benchmark Results (Speed Application)</td>
<td>361</td>
</tr>
<tr>
<td>13.2.1 Benchmark Overview</td>
<td>362</td>
</tr>
<tr>
<td>13.2.2 Application Overview</td>
<td>362</td>
</tr>
<tr>
<td>13.2.3 Summary of Transaction Database Operations</td>
<td>363</td>
</tr>
<tr>
<td>13.2.4 Speed Benchmark Screens</td>
<td>364</td>
</tr>
<tr>
<td>13.2.5 Benchmark Response Time Comparison of IBM OLE DB</td>
<td>367</td>
</tr>
<tr>
<td>13.2.6 Benchmark Response Time by Functional Components</td>
<td>368</td>
</tr>
<tr>
<td>13.3 Function Timings</td>
<td>368</td>
</tr>
<tr>
<td>13.4 Tips and Techniques</td>
<td>369</td>
</tr>
<tr>
<td>13.5 Summary - Usage and Trade-Offs</td>
<td>370</td>
</tr>
<tr>
<td>13.5.1 OLE DB Record Level (Using DDM Server) Versus OLE DB SQL</td>
<td>370</td>
</tr>
<tr>
<td>13.5.2 Performance Scalability</td>
<td>370</td>
</tr>
<tr>
<td>13.5.3 When and Why You Should Use OLE DB Versus Other Methods</td>
<td>370</td>
</tr>
<tr>
<td>Appendix A. Supporting Documentation</td>
<td>373</td>
</tr>
<tr>
<td>A.1 Downloading the Samples</td>
<td>373</td>
</tr>
<tr>
<td>A.2 Other Documentation and Resources</td>
<td>373</td>
</tr>
<tr>
<td>Appendix B. Listings for AS/400 Programs</td>
<td>375</td>
</tr>
<tr>
<td>B.1 PARTS/PF</td>
<td>375</td>
</tr>
<tr>
<td>B.2 SPROC2/SQLRPGLE</td>
<td>375</td>
</tr>
<tr>
<td>B.3 DPCXRPG/RPGLE</td>
<td>377</td>
</tr>
<tr>
<td>B.4 DQXRPG/RPGLE</td>
<td>378</td>
</tr>
</tbody>
</table>
**Appendix C. Data Types, ADO Support and Error Messages.**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Types</td>
<td>381</td>
</tr>
<tr>
<td>Table Data Types</td>
<td>381</td>
</tr>
<tr>
<td>SQL Data Types</td>
<td>382</td>
</tr>
<tr>
<td>Data Queue Data Types</td>
<td>383</td>
</tr>
<tr>
<td>Program Data Types</td>
<td>383</td>
</tr>
<tr>
<td>AS/400 Data Type Descriptions</td>
<td>384</td>
</tr>
<tr>
<td>SQL Data Type Descriptions</td>
<td>385</td>
</tr>
<tr>
<td>ADO Data Type Descriptions</td>
<td>386</td>
</tr>
<tr>
<td>OLE DB Data Type Descriptions</td>
<td>386</td>
</tr>
<tr>
<td>ADO Object Summary</td>
<td>388</td>
</tr>
<tr>
<td>Connection Object Support Summary</td>
<td>388</td>
</tr>
<tr>
<td>Error Object, Errors Collection Support Summary</td>
<td>389</td>
</tr>
<tr>
<td>Command Object Support Summary</td>
<td>390</td>
</tr>
<tr>
<td>Parameter Object, Parameters Collection Support Summary</td>
<td>391</td>
</tr>
<tr>
<td>Recordset Object Support Summary</td>
<td>393</td>
</tr>
<tr>
<td>Field Object, Fields Collection Support Summary</td>
<td>397</td>
</tr>
<tr>
<td>Fields Collection</td>
<td>398</td>
</tr>
<tr>
<td>Property Object, Properties Collection Support Summary</td>
<td>398</td>
</tr>
<tr>
<td>Messages Sent from the AS/400 OLE DB Provider</td>
<td>400</td>
</tr>
<tr>
<td>Connection Messages</td>
<td>400</td>
</tr>
<tr>
<td>Table Messages</td>
<td>400</td>
</tr>
<tr>
<td>Data Queue Messages</td>
<td>401</td>
</tr>
<tr>
<td>Command Messages</td>
<td>401</td>
</tr>
<tr>
<td>Program Messages</td>
<td>401</td>
</tr>
<tr>
<td>SQL Messages</td>
<td>401</td>
</tr>
<tr>
<td>Conversion Messages</td>
<td>402</td>
</tr>
</tbody>
</table>

**Appendix D. Special Notices**

**Appendix E. Related Publications**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>International Technical Support Organization Publications</td>
<td>405</td>
</tr>
<tr>
<td>Redbooks on CD-ROMs</td>
<td>405</td>
</tr>
<tr>
<td>Other Publications</td>
<td>405</td>
</tr>
</tbody>
</table>

**How to Get ITSO Redbooks**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>How IBM Employees Can Get ITSO Redbooks</td>
<td>407</td>
</tr>
<tr>
<td>How Customers Can Get ITSO Redbooks</td>
<td>408</td>
</tr>
<tr>
<td>IBM Redbook Order Form</td>
<td>409</td>
</tr>
</tbody>
</table>

**List of Abbreviations**

**Index**

**ITSO Redbook Evaluation**
Figures

1. The ADO Object Model .................................................... 15
2. Connection, Command, Recordset and Field Objects Properties Collection .... 15
3. Objects Used with ADO ...................................................... 18
4. cwbztrc Command Options ................................................. 44
5. IBMDA400.trc ............................................................... 45
6. Sample ADO SQL Application ............................................. 47
7. ADO SQL Statement VB Application .................................... 48
8. ADO - SQL VB Application - Get All Parts ............................ 49
9. ADO - SQL VB Application - Get a Part ............................... 50
10. ADO Stored Procedure Sample .......................................... 53
11. ADO -Stored Procedure VB Application ............................... 54
12. SDK - VB Wizard .......................................................... 55
13. Specify the AS/400 Program Option .................................... 56
14. Specify the Stored Procedure Name and Return Type .................... 56
15. Specify the Parameter Type .............................................. 57
16. Wizard - Stored Procedure - Create a Link ............................ 58
17. VB Wizard - Stored Procedure - Specify Name ....................... 58
18. VB Wizard - Stored Procedure - Specify Default Parameter ............. 59
19. VB Wizard - Stored Procedure - Return Type ........................ 59
20. VB Wizard - Stored Procedure - Specify Control Type ............... 60
21. VB Wizard - Stored Procedure - Specify Data Control Title .......... 60
22. ADO - Stored Procedure VB Application - Get All Parts ............. 62
23. ADO - Stored Procedure VB Application - Get a Part ................ 63
24. ADO- DDM Record Level Access Sample ................................ 64
25. ADO - Record Level Access VB Application .......................... 65
26. ADO - Tables Record Level Access VB Application - Get a Part ....... 68
27. VB Wizard - Tables - Add New Link ................................... 72
28. VB Wizard - Tables - Select Table, Specify Commit and Update Option ... 73
29. VB Wizard - Tables - Specify Control Type .......................... 74
30. VB Wizard - Tables - Specify Title ................................... 74
31. ADO Data Queue VB Sample Application ................................ 76
32. ADO - Data Queue VB Application .................................... 77
33. ADO - Data Queue VB Application - Get Part ......................... 78
34. Distributed Program Call Example ..................................... 82
35. ADO - Remote Program Call VB Application .......................... 82
36. ADO- Distributed Program Call - VB Get Part ....................... 84
37. The Company Structure .................................................. 87
38. RPG Application Flow .................................................... 89
39. Parts Order Entry ......................................................... 90
40. Select Customer .......................................................... 91
41. Parts Order Entry ........................................................ 92
42. Select Part ........................................................................ 93
43. Parts Order Entry .......................................................... 94
44. Parts Order Entry .......................................................... 94
45. Change Selected Order ................................................... 95
46. Completed Order .......................................................... 95
47. Printed Order ............................................................... 96
48. Table Relationships ......................................................... 97
49. The Client/Server Version of the Order Entry Application ............. 103
50. Order Entry RPG Program ................................................ 104

© Copyright IBM Corp. 1998
<table>
<thead>
<tr>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>104.</td>
<td>Delphi Application - Stored Procedures</td>
</tr>
<tr>
<td>105.</td>
<td>Get One Part - Stored Procedure</td>
</tr>
<tr>
<td>106.</td>
<td>Get All Parts - Stored Procedure</td>
</tr>
<tr>
<td>107.</td>
<td>Get One Part</td>
</tr>
<tr>
<td>108.</td>
<td>Get All Parts - Stored Procedure</td>
</tr>
<tr>
<td>109.</td>
<td>DDM Record-Level Access</td>
</tr>
<tr>
<td>110.</td>
<td>Delphi Application - Record-Level Access</td>
</tr>
<tr>
<td>111.</td>
<td>Finding a Record by Key</td>
</tr>
<tr>
<td>112.</td>
<td>All Records - Record-Level Access</td>
</tr>
<tr>
<td>113.</td>
<td>Delphi Data Queue Application</td>
</tr>
<tr>
<td>114.</td>
<td>Get One Part - Data queue</td>
</tr>
<tr>
<td>115.</td>
<td>Get All Parts - Data Queue</td>
</tr>
<tr>
<td>116.</td>
<td>Get One Part - Data Queue</td>
</tr>
<tr>
<td>117.</td>
<td>Delphi Distributed Program Call Example</td>
</tr>
<tr>
<td>118.</td>
<td>Get a Part - Program Call</td>
</tr>
<tr>
<td>119.</td>
<td>Get All Parts - Program Call</td>
</tr>
<tr>
<td>120.</td>
<td>Get a Part - Program Call</td>
</tr>
<tr>
<td>121.</td>
<td>Lotus Notes Using SQL</td>
</tr>
<tr>
<td>122.</td>
<td>Parts Form - Lotus Notes</td>
</tr>
<tr>
<td>123.</td>
<td>Parts Form</td>
</tr>
<tr>
<td>124.</td>
<td>Lotus Notes Agent Calling a Stored Procedure on AS/400</td>
</tr>
<tr>
<td>125.</td>
<td>Lotus Notes Agent that Calls a Stored Procedure on the AS/400 System</td>
</tr>
<tr>
<td>126.</td>
<td>Lotus Notes Using Record Level Access</td>
</tr>
<tr>
<td>127.</td>
<td>Record-Level Access - Lotus Notes</td>
</tr>
<tr>
<td>128.</td>
<td>Lotus Notes Agent and DataQueues</td>
</tr>
<tr>
<td>129.</td>
<td>AS/400 Program Call from a Lotus Notes Agent</td>
</tr>
<tr>
<td>130.</td>
<td>Item Discount Form</td>
</tr>
<tr>
<td>131.</td>
<td>Template of the Letter</td>
</tr>
<tr>
<td>132.</td>
<td>The Item Discount Form</td>
</tr>
<tr>
<td>133.</td>
<td>The Generated Letter</td>
</tr>
<tr>
<td>134.</td>
<td>Visual FoxPro Application Using SQL</td>
</tr>
<tr>
<td>135.</td>
<td>Get Part Using SQL</td>
</tr>
<tr>
<td>136.</td>
<td>Get All Parts Using SQL</td>
</tr>
<tr>
<td>137.</td>
<td>Launching LotusScript to Create or View a Script</td>
</tr>
<tr>
<td>138.</td>
<td>LotusScript Script Editor</td>
</tr>
<tr>
<td>139.</td>
<td>ADO - Excel 97 Sample Worksheet</td>
</tr>
<tr>
<td>140.</td>
<td>Visual Basic Editor for MS Excel</td>
</tr>
<tr>
<td>141.</td>
<td>Microsoft Access Table</td>
</tr>
<tr>
<td>142.</td>
<td>MS Access Application</td>
</tr>
<tr>
<td>143.</td>
<td>MS Access Table Populated from the ITEM File</td>
</tr>
<tr>
<td>144.</td>
<td>HTML Page with Client-Side Scripting</td>
</tr>
<tr>
<td>145.</td>
<td>HTML Page Generated by an ASP</td>
</tr>
<tr>
<td>146.</td>
<td>Edit Customer Data Using a Web Application</td>
</tr>
<tr>
<td>147.</td>
<td>Project Properties of the Business Objects DLL</td>
</tr>
<tr>
<td>148.</td>
<td>Project Explorer for the Business OBjects DLL</td>
</tr>
<tr>
<td>149.</td>
<td>Properties of the Class Module</td>
</tr>
<tr>
<td>150.</td>
<td>Registering Your Object for Use with IIS</td>
</tr>
<tr>
<td>151.</td>
<td>Find the GUID of Your Business Object</td>
</tr>
<tr>
<td>152.</td>
<td>Adding the New Implemented Categories</td>
</tr>
<tr>
<td>153.</td>
<td>Category &quot;Safe for Scripting&quot;</td>
</tr>
<tr>
<td>154.</td>
<td>Category &quot;Safe for Initializing&quot;</td>
</tr>
<tr>
<td>155.</td>
<td>Logical Connections on a LAN Environment</td>
</tr>
<tr>
<td>156.</td>
<td>Microsoft SNA Server Manager with Two Connections Defined</td>
</tr>
</tbody>
</table>
# Tables

1. The ADO Connection Object .................................................. 20
2. The ADO Command Object .................................................... 21
3. The ADO Recordset Object ................................................... 27
4. The ADO Field Object ......................................................... 31
5. Error Identifiers Used with AS/400 OLE DB Support ................... 33
6. The ADO Error Object ........................................................ 33
7. The ADO Parameter Object ................................................... 36
8. Values for the Updatability Property ..................................... 38
9. Functions to AS/400 Subsystem and Job Cross-Reference .......... 40
10. PTF List for Tables - Record-Level Access Pre-R420 ............... 41
11. PARTS File ............................................................................. 46
12. Updatability Property Setting for Different Lock Type ............ 69
13. Commitment Control ............................................................ 70
14. Details of the DPCXRPG ........................................................ 83
15. District Table Layout (Dstrct) ................................................. 98
16. Customer Table Layout (CSTMR) ............................................ 98
17. Order Table Layout (ORDERS) ............................................. 100
18. Order Line Table Layout (ORDLIN) ...................................... 100
19. Item Table Layout(ITEM) ..................................................... 100
20. Stock Table Layout(Stock) ..................................................... 100
21. Database Terminology .......................................................... 101
22. frmCustomer Details ........................................................... 120
23. frmCustomerSearch Details ................................................. 122
24. frmErrors Details ............................................................... 123
25. frmItem Details .................................................................. 124
26. frmItemSearch Details ......................................................... 125
27. frmOptions Details .............................................................. 127
28. frmOrder Details ............................................................... 128
29. PARTS File ............................................................................. 164
30. Constant Values for Direction Property of Parameters Collection ........................................................................... 175
31. Table AccessCapability ......................................................... 179
32. Different Levels of Commitment Control .............................. 185
33. Data Queue DQINPT Layout ................................................. 189
34. Data Queue DQOUPT Layout ............................................... 189
35. Details of DPCXRPG ............................................................. 195
36. PARTS File ............................................................................. 200
37. Constant values for Direction property of Parameters Collection ........................................................................... 211
38. Table Access Capability ......................................................... 216
39. Commitment Control Options .............................................. 222
40. Data Queue DQINPT Layout ............................................... 226
41. Data Queue DQOUPT Layout ............................................... 226
42. Details of DPCXRPG ............................................................. 233
43. Parts File .............................................................................. 240
44. Table Updatability Values ..................................................... 251
45. Data Queue DQINPT Layout ............................................... 256
46. Data Queue DQOUPT Layout ............................................... 256
47. Details of DPCXRPG ............................................................. 260
48. PARTS File ............................................................................. 272
49. DDMACC Network Attribute ................................................ 354
50. PCSACC Network Attribute ................................................ 354
Preface

If you have ever wanted to develop Microsoft Windows programs that work with the AS/400 system, you'll want to learn about the AS/400 OLE DB support. It provides a tremendous improvement over previously used techniques. Two of the key considerations that the AS/400 OLE DB support addresses are ease of application development while still providing acceptable performance. This redbook gives a broad understanding of the OLE DB architecture and how it can be used to access AS/400 resources.

This redbook provides examples of how to write programs using this interface. Many of the most popular Windows programming environments such as Visual Basic, PowerBuilder, Delphi, Lotus Notes and Visual FoxPro are covered. Also covered are end user tools such as Lotus 1-2-3, Excel and Microsoft Access.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization Rochester Center.

Bob Maatta is a Senior Software Engineer from the United States at the International Technical Support Organization, Rochester Center. He writes extensively and teaches IBM classes worldwide on all areas of AS/400 client/server and AS/400 application development. Before joining the ITSO three years ago, he worked in the U.S. AS/400 National Technical Support Center as a Consulting Market Support Specialist. He has over 20 years of experience in the computer field and has worked with all aspects of personal computers for the last 10 years.

Mehboob Alam is a Senior Consultant at AS/400 Support Center in IBM Rochester. He has three years of experience in the Client Access field. He holds a degree in Computer Science from Winona State University, Minnesota. His areas of expertise are Client/Server application development.

Geert Maertens is a Senior Product Developer in General Systems Europe Belgium. He has six years of experience in Client/Server development using UltimADE (a product he was instrumental in developing), C++ and Visual Basic. He has worked at GSE for 10 years. He holds a degree in Computer Science from Vrije Universiteit Brussel (Brussels Free University).

Ray McRoberts is president of his own consulting company - MIPTech, Inc. based in Rochester Minnesota. He has 30 years of experience in the computer industry, with 24 years at the Rochester IBM Lab. He holds a degree in Electrical Engineering from the University of Minnesota Institute of Technology. His areas of expertise include Visual Basic and Crystal Reports.

Craig Pelkie is a consultant based in Southern California. He is the editor of the NEWS/400 newsletter Client Access & Windows Solutions, past editor of Midrange Computing's Client Access Expert, and frequently leads seminars on Client Access, AS/400 web-enablement techniques and AS/400 client/server programming using Visual Basic. He is the author of the training manual Using Microsoft Visual Basic with the AS/400 (http://www.vb400.com) which was used in labs at IBM Rochester Partners In Development. Although he has never
programmed by wiring the boards, he started his programming career with a card sorter.

Pankaj Sharma is a Software Engineer in IBM Global Services, India. He has two years of experience in Client/Server Application Development on the AS/400 system. His areas of expertise include DB2/400, ILE C/400 and AS/400 Client/Server. He has written extensively on Client/Server Application Development on AS/400.

Thanks to the following people for their invaluable contributions to this project:

Lance Amundsen  
IBM Rochester Development Lab

Troy Bleeker  
IBM Rochester Development Lab

Brant Klepel  
IBM Rochester Development Lab

Jarek Miszczyn  
ITSO Rochester

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

• Fax the evaluation form found in “ITSO Redbook Evaluation” on page 417 to the fax number shown on the form.

• Use the electronic evaluation form found on the Redbooks Web sites:
  For Internet users http://www.redbooks.ibm.com
  For IBM Intranet users http://w3.itso.ibm.com

• Send us a note at the following address:
  redbook@us.ibm.com
Chapter 1. OLE DB and ActiveX Overview

Two major concerns that have been voiced by developers who are producing AS/400 client/server applications are:

1. Performance
2. Ease of application development

There are numerous ways to produce AS/400 client/server applications. You can do everything from writing your own program to program interfaces to using high level interfaces. Some of these interfaces require an in-depth understanding of complex API calls. In many cases, the performance you can achieve depends on how low a level you can program to. What is required is a high level programming interface that produces applications that provide a high level of performance. This is especially true with OLTP (Online Transaction Processing) type client/server applications.

If you have ever wanted to develop Microsoft Windows programs that work with the AS/400 system, you’ll want to learn about the AS/400 OLE DB support (AS/400 OLE DB). This support, which is provided with IBM’s Client Access for Windows 95/NT, is a tremendous improvement over previously used techniques. But to fully understand the benefits of AS/400 OLE DB support, you need to know about the underlying technologies that it is built on. Although some programmers resist learning about low-level details, you’ll be able to work more effectively with AS/400 OLE DB if you understand the architecture.

If you are new to AS/400 client/server programming, you’ll also want to know how AS/400 OLE DB relates to other techniques that are available. This chapter explains in detail some of your options. We will then review the foundation that AS/400 OLE DB uses and the components that you will use in your Windows programs. After reading this chapter, you’ll know where AS/400 OLE DB fits into the overall scheme of Windows to AS/400 programming techniques and why you may prefer AS/400 OLE DB to other options.

1.1 The Quick Summary

If you are already familiar with OLE DB and ADO (ActiveX Data Object) concepts, you can review this summary and move on to the following chapters. AS/400 OLE DB support is IBM’s version of an ADO provider that is specifically designed to let your programs access AS/400 data. However, AS/400 OLE DB support provides much more than SQL type access to the database. Some of the other techniques that are available with AS/400 OLE DB support include:

- Record-level access using keyed physical or logical files
- Stored-procedure calls
- SQL statement processing
- Call/Parm support for calling AS/400 programs
- OS/400 command execution
- Support for using Data Queues for send and receive processing
1.2 Past Practices for Windows to the AS/400 System

Regardless of how you feel about Windows or the AS/400 system, the fact is that most AS/400 sites also have PCs running Windows. There has been tremendous pressure on AS/400 application developers over the past several years to produce Windows applications that work with the AS/400 system.

Windows programmers can choose from several excellent programming languages. These include:

- Microsoft Visual Basic. In terms of unit-sales, this is by far the most widely used Windows programming language.
- Borland’s Delphi. This is a powerful object-oriented application development language.
- Sybase’s PowerBuilder, popular because of its DataWindows feature.
- Microsoft Visual C++ and other vendor’s C/C++ compilers for Windows.
- Embedded programming languages in end-user applications. For example, the Microsoft Office and BackOffice product suites incorporate Visual Basic for Applications (VBA) as a programming language that runs inside the application.
- For RPG programmers, IBM’s Visual Age for RPG and ASNA’s Visual RPG.

If you are an AS/400 RPG programmer, you may have investigated some of these products. If all of your programming experience has been with RPG, you probably found that you could learn the basics of these products relatively quickly (with the possible exception of C++). However, even a language like Visual Basic, with its reputation for ease-of-learning, is a very comprehensive development environment and language that can take several months to become proficient with.

The big problem that transitioning AS/400 programmers have is that even after becoming comfortable with a Windows programming language, there is still the rather large problem of getting access to the AS/400 system. For the most part, there have been three techniques used over the past several years:

- ODBC
- Client Access APIs
- Third-party products

We’ll review each of those alternatives. Although it is entirely possible to continue to use those techniques, you should understand what is involved with each choice.

1.2.1 Developing Windows to AS/400 applications with ODBC

Starting with OS/400 V3R1, AS/400 database support improved dramatically. A new ODBC driver is shipped with the Client Access for Windows 3.1 client. When the Client Access for Windows 95 product shipped (V3R1M0), a native 32-bit Windows 95 ODBC driver was also shipped.

Unfortunately, the perception of ODBC’s poor performance on the AS/400 system has persisted, even though the AS/400 ODBC interface has been able to show continual improvement. Some of the on-going criticism of AS/400 ODBC
performance is misdirected, in that much of the poor performance can now be attributed to the Windows application itself. You can write ODBC applications that perform very well. But, you can also write very poor performing ODBC applications. In many cases, it is the skill level of the application developer that determines what level of performance is achieved.

1.2.1.1 Visual Basic and the Data Control
Visual Basic programmers have insisted on using the data controls provided with Visual Basic to access AS/400 data using ODBC. This is often called the “no code” approach, since it is possible to create a functional Visual Basic database program that can retrieve and display data with only a few lines of program code.

The actual connection to the AS/400 system is set as properties of the data control. Visual Basic also includes several bound controls that can be linked to the data control. Every time a new record is fetched from the database, the bound controls update themselves to display the data. A popular bound control is the grid control, which presents a spreadsheet-like row and column display of data.

It turns out that using the Visual Basic data control is practically the worst way to access the AS/400 database through ODBC. The data control uses the Microsoft Jet database engine to translate the connection and database retrieval operations into the format used by the Client Access ODBC driver. The Jet engine, although a superb tool for desktop databases, is a terrible bottleneck when accessing an AS/400 database.

That being said, Microsoft has been improving the Jet engine. In Visual Basic 5, you can now use the “ODBC Direct” option to bypass much of Jet’s internal processing when accessing a remote database. However, this is a level of detail that many programmers do not implement.

1.2.1.2 The ODBC API
Most Visual Basic programmers working on Windows to AS/400 applications realize that the data controls included with Visual Basic are not ideal. For casual or low-usage applications, it may be acceptable to simply use the Visual Basic data control and move on. But for heavily-used applications, particularly applications that many users might be using simultaneously, it is extremely detrimental to allow those less than optimal applications to access the AS/400 system. Moving data through the network from the AS/400 system to the PC is one thing, but when a lot of the network traffic is generated from the application simply to map the structure of the database (such as happens with the Jet), it is time to use other techniques.

Many Windows programmers have learned to use the ODBC API to access the AS/400 system. Of the approximately 60 functions in the ODBC API, there are about 15 frequently used APIs that you need to know. You also need to know C, since the ODBC API is oriented towards this language.

Visual Basic and other languages include function definitions for the ODBC API, but learning to use an API effectively is still challenging. With most APIs, you are exposed to low-level details concerning memory allocation, string handling and data representation. The end result of coding an ODBC API application is that you potentially have an application that performs well. However, many Windows programmers mistakenly try to program all of their optimization on the client-side. It is true that there are techniques that might give better performance and are
only available when programming directly with the API. But some of the higher-level techniques that are available for ODBC, such as Visual Basic’s Data Access Objects (DAO) and Remote Data Objects (RDO) let you have practically the same degree of control as using the API.

The other factors that affect ODBC performance are usually more important than how you use ODBC on the PC (API, DAO, RDO). Those factors include the network itself and the AS/400 system. Often, ODBC performance can be improved simply by creating a logical file with a key that matches the WHERE and ORDER BY clauses on the SQL statement.

One of the main problems with using ODBC to the AS/400 system is that many programmers use ODBC for record-level access. The SQL statement is used to retrieve a result set of one record. Although this is certainly feasible with ODBC, the programmer’s level of expectation is that this should be as fast as record-level access in a native AS/400 application.

Since it is usually not possible to obtain near native record-level access performance using ODBC, the last resort is stored procedures. Stored procedures are ideal for the AS/400 system. A stored procedure is a compiled AS/400 program that can contain embedded SQL statements and return a result set. In addition to providing the required performance, you can also enforce database security if all of your ODBC processing is through stored procedures.

The biggest problem with stored procedures is that your Windows program is now very tightly bound to the AS/400 system. Software developers who target other platforms in addition to the AS/400 system are especially hesitant to use stored procedures. One of the premises of ODBC is that the Windows program can get the same result when accessing a different database, just by referencing a different ODBC driver in the Windows program. With platform-specific stored procedures, it is much more difficult to retarget the Windows application.

1.2.1.3 ODBC Summary
Although performance of ODBC to the AS/400 system is now competitive with other servers, there are a number of issues you need to be aware of before coding your client application. These include which coding technique to use (data controls, APIs, higher-level components such as DAO or RDO), the current structure of the database and whether or not you can make changes (such as creating additional logical files), and whether or not you should consider creating stored procedures. If you are an AS/400 RPG programmer transitioning to Windows programming, that is quite a lot to learn.

1.2.2 Client Access APIs
If your Windows application requires the utmost in Windows to AS/400 performance, then you should use Client Access APIs. That being said, using Client Access APIs is perhaps the most difficult and error-prone technique that you can use. Some of the problems you’ll need to resolve in order to use the APIs include:

- Requires proficiency in C to understand how to work with an API
- Some APIs are protocol dependent
- EBCDIC to ASCII conversion issues
- Numeric data transform issues
If you are new to the world of AS/400 client/server programming, you will find that having to learn to work with the Client Access APIs is a major detour. For example, if you stipulate that your Windows to the AS/400 application must have the absolute best performance (and what application doesn't?), here are some of the questions you need answers for:

- **What protocol are you using?** If your PC to AS/400 connection is SNA, then you'll use APPC or CPI-C (probably APPC since that is what is commonly used). If your connection is TCP/IP, you'll write a WinSock application. For the server side, you'll create a program that uses ICF files or CPI-C calls for SNA or uses AS/400 sockets support for TCP/IP.

- **Where will you do the required field-level data conversions?** If all of your data is character data, there are options in APPC and CPI-C to convert a string to and from EBCDIC and ASCII. But if you have to work with numeric fields, especially the packed fields common to AS/400 data, you need to work with the Data Transform APIs on the PC or convert the numeric data in your server program.

- **Who will perform the configuration work required on the PC and the AS/400 system?** Once you move your application from your development PC to a user’s PC, you'll need to verify that their PC is configured to connect to the AS/400 system using the protocol that you’ve coded to. One of the “golden rules” of client/server programming is that if you don’t have a connection, you don’t have a working program.

If you can successfully solve all of those problems (plus the numerous inevitable others), your application can have truly spectacular performance. If you code to a transport-level API, such as APPC, CPI-C or Sockets, you are essentially using the same technique that IBM uses to run all of their client/server applications. Everything that runs between the PC and the AS/400 system uses a transport-level protocol at its base. If you write one of these applications, then you have the best performing application.

### 1.2.2.1 Data Queues

Using a transport-level protocol sounds like a lot of work, and actually, it is. If you step back from the problem, it can be framed like this: how can we move a buffer or packet of data from the PC to the AS/400 system and back again?

A simpler technique, rather than using transport-level APIs, is to use data queues. A data queue is an object that is located on the AS/400 system. Your Windows program can send an entry to the data queue. The entry is typically a request to the AS/400 system for a particular record or set of records. It is entirely up to you to decide on the format of the request. Both your Windows client program and AS/400 server program must agree on the format and meaning of the contents of the request.

On the AS/400 system, your server program waits for an entry to arrive on the data queue. When the client application writes an entry to the data queue, the server program reads the entry from the data queue. After doing any required processing, the server program writes its response entry to the data queue.

The Windows client program, after sending its initial request to the data queue, changes its state so that it can receive from the data queue. As soon as the server response is available on the data queue, the Windows program receives and processes the entry.
Although data queues are not as fast as transport-level APIs, it is a considerably simpler technique to use. You still have to deal with the EBCDIC/ASCII and numeric data translation issues, but at least the problem of moving the buffer from the PC to the AS/400 system is resolved. Data queues are also protocol independent, so if you move from an SNA connected PC to a TCP/IP connection, you do not need to change either your client or your server program.

1.2.2.2 Data Queues OLE Automation and OCX
When you install Client Access for Windows 95/NT, there is a Toolkit component that is optionally installable. If you install the Toolkit, you have access to data queues OLE (Object Linking and Embedding) automation objects and the data queues OCX (OLE Custom Control) control.

- The **data queues OLE automation objects** are a step up from using the data queues APIs directly. When you use OLE automation objects, you are working with components that can be used in most Windows programming languages.
- The **data queues OCX control** is a visible control. For example, if you use Visual Basic, you’ll see the data queues control in the Toolbox. You can drag and drop the control onto your form and use the Properties window to set properties.

If you have recently started AS/400 client/server programming, you may have created your first programs using the data queues OCX. Prior to AS/400 OLE DB support, this was probably the best choice for learning about AS/400 client/server programming.

Many programmers who already know how to use the Client Access API do not favor using the data queues OCX. However, if you’re programming Windows applications, it is very advantageous to use OCX and OLE automation objects. For the most part, Windows application programmers are moving to component-based architectures as quickly as possible and getting away from API level coding. Knowing how to work with the data queues OCX or OLE automation objects is great preparation for learning about AS/400 OLE DB support, since it uses ADO components.

1.2.2.3 Other Client Access APIs
There are many other Client Access APIs that you can use to develop an AS/400 client/server program. Some of these APIs are:

- Data Transfer – gives you access to the file download and upload functions. If you want to create your own file transfer program, use this API.
- Optimized SQL – a native SQL interface to the AS/400 system. In addition to processing SQL requests (similar to ODBC), this set of APIs includes functions to work with files and members.
- National Language – a set of APIs that you can use to convert from one character set to another. You can use one of the functions in this set to perform EBCDIC/ASCII conversions.
- Data Transform – a set of conversion functions for numeric data fields. If you work directly with packed fields in your client program, you’ll need functions in this set to convert from packed to ASCII and back.
- Communications – several APIs that can be used to query the status of your connection to the AS/400 system, get the user ID for the connection,
determine the names of AS/400s that have connections defined and other similar functions.

- System list box and Remote Command button – these are two other OCXs that are installed with the Client Access for Windows 95/NT Toolkit.

  - The System list box provides a drop-down list of AS/400 systems that you have connections to or for which you are configured. You can include this in your applications to let your user select the AS/400 system to work with.

  - The Remote Command button is similar to a standard Windows command button. You can associate an AS/400 command and parameters with the Remote Command button. When the user clicks the Remote Command button, the command is sent to the AS/400 system.

### 1.2.2.4 Summary of Client Access API techniques

Learning how to use the Client Access APIs and using them effectively to create AS/400 client/server programs is very challenging. You should avoid using APIs if:

- You are not already familiar with C programming.
- You do not know how to translate a C function definition into the language you will be working with.
- You are not familiar with the need to translate character and numeric data.

AS/400 OLE DB support, as you will learn, solves all of these problems in a consistent, easy to use manner. Rather than forcing you to solve a series of low-level problems simply to get a result, you can use AS/400 OLE DB support to accelerate your development and put more time into the functionality of the application.

There is only one valid case that can be made for using Client Access APIs for new application development: performance. Because everything in AS/400 OLE DB support runs on top of the transport-layer APIs, you might claim that you cannot use AS/400 OLE DB support because of the additional layers. For the most part, that is a false argument. AS/400 OLE DB support supports all of the major techniques that you’ve used to access the AS/400 system: ODBC, SQL statements, stored procedures, record level access, data queues, program calls and commands. As we discovered when writing this redbook, if one AS/400 OLE DB support technique does not give satisfactory performance, it is trivial to switch to another technique. That is not the case with APIs, where changing from one technique to another usually means a lot of rework.

You should also consider the tremendous boost in programming productivity that AS/400 OLE DB support gives you. If you already work with the APIs, you might counter that you can simply copy and paste your code where needed. But now consider that perhaps your application has to evolve to a different environment. For example, instead of a Windows client/server application, maybe the application has to be hosted in a Web browser. Or perhaps your users need to work with AS/400 data directly in Excel or Lotus Notes. It would be very difficult to take your API techniques to those environments. As you’ll see in this redbook, we’ve created examples in all of those environments using AS/400 OLE DB support.
1.2.3 Third-Party Products

There are a number of third-party products available for AS/400 client/server development. These range from relatively inexpensive toolkits available from various vendors to complete application generators that require some effort to learn. Other tools are in the generic “screen-scaper” category, which takes existing AS/400 display files and maps them into Windows displays. Rather than provide a list of products here or single out any particular vendors, refer to the AS/400 trade journals that advertise the products.

Perhaps the biggest issue with these products is licensing. This is particularly important if you are developing software for resale. In most cases, your client is required to license run-time components for either the PC or the AS/400 system. If you already have a product that is serving you well, continue using it but become familiar with AS/400 OLE DB support. If you do not currently have any third-party products, include AS/400 OLE DB support in the list of products that you evaluate.

AS/400 OLE DB support fits best into the “toolkit” category of products. Apart from the Visual Basic add-in Wizards, there is nothing in AS/400 OLE DB support that generates complete applications. Even though the Wizards generate a working application, the intent of the Wizards is to give you a framework for the client/server part of the application, not the user-interface or program-logic parts.

Again, you need to be aware that user interfaces for the AS/400 system are evolving. AS/400 OLE DB support works well in a number of environments in addition to stand-alone Windows programs.

1.3 OLE DB – The Foundation of AS/400 OLE DB Support

If you were going to create a toolkit to help programmers create AS/400 client/server programs, how would you do it? Would you create AS/400 server programs that your Windows client toolkit code would call? Would you create an object-oriented tool to run on top of ODBC? What about “wrapping” the existing Client Access APIs, to make them easier to use? How would you handle the issues of protocol independence and data translation? How would you provide record-level access to the AS/400 database, a function that doesn’t have any existing Client Access API?

Those are some of the major issues that need answers before creating your toolkit. The designers of AS/400 OLE DB support also needed to answer those questions.

IBM began work on a client/server toolkit in early 1996. By May 1996, IBM was able to demonstrate an early prototype of record-level access to the AS/400 system from a PC program. The plan was to build on the earlier OLE automation object and OCX support delivered with Client Access and create additional OLE automation objects and OCXs to provide the toolkit functions. Although the ideas were sound and the feasibility of implementing the toolkit was established, this solution meant that AS/400 client/server programmers would still be working with a proprietary technology.
To understand the approach that IBM eventually adopted with AS/400 OLE DB support, we’ll explain what Microsoft was doing and how IBM adopted the newest Windows data access technologies.

1.3.1 What OLE DB Does

In Summer 1996, Microsoft published their first specifications of a new data access technology that they called OLE DB (OLE for Databases). Although ODBC was at that time firmly established as the industry standard for accessing relational databases from Windows applications, there was an emerging problem with ODBC.

Microsoft conducted several studies and found that a great deal of data that users need to work with is not stored in relational databases. For example, users store data in spreadsheets, e-mail messages and flat-file systems. In some applications, it is important to bring together data from sources like that in addition to data in the corporate relational database.

To access data in non-relational database formats, programmers need to use whatever APIs and tools are available for the particular data source. The APIs are usually completely independent of each other. For example, knowing how to use the APIs to extract data from e-mail folders does not mean that you know how to use the APIs to work with flat-file systems. For each different type of data source, you need to learn its unique API. If you need to create an application that can manipulate data from more than one type of data source, a great amount of code is required just to perform data access.

OLE DB is designed to provide a common API that can be used to present data to a consumer (an application). By itself, OLE DB does not get data from any data source. Instead, OLE DB uses a provider whose job is to know how to manipulate a particular data source and present data to OLE DB in a tabular format. (All types of data can be resolved to a tabular format. If nothing else, the data can be represented as a table with one column and one row.)

Practically all programmers who use OLE DB will be consumers; very few programmers will actually write providers. For the most part, vendors who create data sources will implement and make providers available.

OLE DB can use ODBC as a provider to access relational databases. Using different providers, OLE DB can present any type of data to the application. The advantage of using OLE DB is that you need to learn only one API, OLE DB. By directing OLE DB to use a particular provider and then issuing retrieval commands, you can get the data into a format that is easy to use in an application.

However, there is one major drawback to OLE DB: it is only usable with C++. Because most Windows application programmers work with Visual Basic or other high-level languages, Microsoft introduced a component-based architecture to provide an easier to use interface to OLE DB. That architecture is ActiveX Data Objects (ADO). ADO is a set of Component Object Model (COM) objects that are readily usable from Windows programming languages.
1.3.2 What is COM?

Most new technologies are built on existing technology and implementations. Part of the problem of learning a new technology is that you need to understand what it is based on. Although it is very easy to say that OLE DB and ADO are built on COM technologies, that statement will mean very little to you if you are uncertain what COM is.

Windows 3.0 (1990) was the first version of Windows that attracted any real interest from users. Compared with previous versions of Windows and DOS programs, it was easy to see why people were more attracted to Windows 3.0. For one thing, Windows 3.0 made use of the graphics capabilities that were available on the PC. Windows 3.0 also included support for most of the popular printers available at the time, so it was much easier to install and configure applications compared to the DOS environment.

Perhaps most importantly for Windows’ 3.0 success, Microsoft introduced Excel and Word for Windows, which demonstrated two things rather convincingly:

1. Windows applications really were easier to work with. They let the user work with the mouse, view multiple applications simultaneously and generally control the environment of the PC in ways that were unavailable with DOS.

2. Windows applications really did run slower than their DOS equivalents.

However, most people were willing to overlook the second point in favor of the first. Although some people still argue that GUIs are unnecessary and no easier to work with than character-based programs, the market has spoken. Most users prefer a GUI application to a character-based application. (The same points raised above could also be made in favor of OS/2. Rather than argue the technical merit of one choice over another, it is sufficient to say that at this point, most people are looking for Windows applications, not OS/2 applications.)

1.3.2.1 How to Program in Windows

Not many companies were interested in creating Windows applications at first. For one thing, it was still possible to sell DOS applications and use the vast assortment of tools and languages available for DOS. For another, Windows 3.0 was still not a clear winner in the market. Perhaps most importantly, programming Windows 3.0 applications required the programmer to become familiar with the Windows API. This included over 700 functions and to create a good Windows application you needed to be familiar with most of them.

Object Linking and Embedding

So Microsoft had two rather novel applications, Word and Excel. Although some people would choose to use those programs based on their GUI appeal, there was still no compelling reason to switch from DOS-based applications such as WordPerfect and Lotus 1-2-3.

To demonstrate one advantage of working in the Windows environment, Microsoft created a showcase technology. One of the things that people were doing with Excel and Word was creating a spreadsheet in Excel, then using its copy function to copy the spreadsheet to the Windows Clipboard. At that point, they would go into Word and use the paste function to paste the spreadsheet into the Word document.
That was advantageous because you didn’t have to try to create a spreadsheet in Word. However, there was a problem when you needed to update the numbers in the spreadsheet. After updating the spreadsheet, you would probably want the updated version to show up in the Word document. At that point, you would need to copy and paste the spreadsheet again.

To solve that problem, Microsoft invented a technology within Windows that they called Object Linking and Embedding (OLE). With OLE, it is possible to create a “live link” between documents. That means that for the Excel and Word applications, you can identify the portion of the spreadsheet that is to be included in Word and designate that as a live link. Upon pasting the spreadsheet into Word, Word would recognize when changes were made to the underlying spreadsheet.

Another feature of OLE is in-place activation, meaning that you can double-click the spreadsheet inside the Word application and actually invoke Excel to edit the spreadsheet.

As you might imagine, OLE is enormously complicated. It is also in addition to the Windows API. If anybody was ever going to be able to really program in Windows, there would need to be better techniques.

The Component Object Model
Using the experience of creating Windows applications and OLE, Microsoft decided to start incorporating object-oriented programming (OOP) technology into Windows. But to create an OOP-capable environment, they first had to decide on the ground-rules. Some of the questions that needed answers include:

- How can objects be used in a programming environment or application? That is, what must the application do to let the object know that it wants to use it? How does the application communicate to the object what it wants the object to do? How does the object return results to the requester?
- How should the operating system deal with objects? If a user adds new objects to their system, are they required to configure the system so that other applications can recognize the objects? Or should the operating system perform some sort of automatic registration function?
- What happens when a newer version of an object becomes available? If the user installs the newer version on their PC, what happens to the old version?
- What tools are available to create objects?

There is nothing specific to Windows about those questions. Any operating system vendor who wants to add OOP to their system needs to answer those and other questions about how their OOP environment will work. The answers to those questions dictate the implementation of the environment, which is generically called an object model.

Microsoft chose to distinguish their object model with the name Component Object Model (COM). Another object model that you might have heard of is IBM’s System Object Model (SOM). Java presents another object model.

COM is Microsoft’s definition of how OOP technologies are implemented on a Windows platform. You will often see Windows applications advertised as “COM compliant” or “OLE compliant” or “OLE/COM compliant”. What that means is that the application can use objects that were developed by other vendors, as long as
those other objects conform to the COM specification. At this point, virtually all major Windows applications support COM.

1.3.2.2 Breaking into Windows
The time frame is now 1991. Windows 3.0 has been out for over a year. OLE and COM are taking shape. If you are a PC application programmer, you have to master not only the Windows API but also potentially OLE and COM. You also need to know C and Assembler.

And then Microsoft introduces Visual Basic 1.0. Rather than typing long text files to describe the appearance of a window (source files similar to AS/400 display file DDS), you can now design a window in a GUI environment. You can use the mouse to select user interface components and position them on a form (Visual Basic’s term for a window that is under development). Double-clicking anywhere on the form opens a code window where you write code that responds to events happening on the form. Because Visual Basic 1.0 is an interpreted language, you can test the program instantly while in the development environment; you don’t need to wait while it compiles before running it.

It is fair to say that Visual Basic 1.0 had a major role in making Windows more widely accepted. All of a sudden, it was possible for application programmers to create Windows programs. One of the big factors in Visual Basic’s success and rapid adoption was its Toolbox, a window that contains a selection of objects that can be dragged and dropped onto the form. By design, Microsoft made the Toolbox extensible, meaning that other vendors could develop tools that would appear in the Toolbox.

Although this sounds similar to what COM was meant to provide, the first three versions of Visual Basic did not support COM objects but rather VBXs (Visual Basic Extensions). Although being able to acquire and add VBXs to Visual Basic was a tremendous advantage, they were inherently limiting:

- VBXs initially were not usable in programming environments other than Visual Basic.
- VBXs were not COM-compliant, meaning that the operating system was unable to provide any assistance to the VBX.

Starting with Visual Basic 4.0, Microsoft dropped support for VBXs in the 32-bit Windows 95/NT environments. In its place, Microsoft added support for COM components. Most VBX vendors converted their VBX to COM. More importantly, because the market for COM components is larger than the market for VBXs, more vendors were willing to develop tools for Windows programming.

In addition, Visual Basic 4.0 could be used to create COM components. Another change made was the extension used for the file: instead of VBX, files were now OCX (OLE Custom Control). When you see a file with the OCX extension in the Windows environment, that simply means that the file contains COM components. (Don’t be confused by the apparent general references to OLE and COM. As far as components are concerned, the references have come to mean the same thing.) At this point, virtually all of the Windows development environments (Visual Basic, Visual C++, Delphi, PowerBuilder and others) support COM components.
1.3.3 Back to OLE DB

So we now have an operating system that supports an object model. We also have several choices of development tools that let us incorporate COM components. We now want to use these tools to create great Windows applications that access data from different data sources, including the AS/400 system.

You’ve already read about some of the techniques that are available for database access. For the AS/400 system, the choices amounted to ODBC, APIs or a handful of OLE automation objects and OCXs.

You’ve also learned that the new OLE DB technology can be used to gather up all of the diverse data access techniques into one interface. And you’ve learned that you can only use OLE DB with C++.

If we have development tools that let us use COM components, it would be nice to have components to work with OLE DB. To enable OLE DB to be adopted by as many programmers as possible, Microsoft introduced ADO concurrently with OLE DB. ADO is a set of COM components that you will use to work with OLE DB.

AS/400 OLE DB support uses ADO. Because ADO drives OLE DB, and because OLE DB can be used to access data from any data source, we now have the pieces in place to implement a technology to work with the different types of access techniques available to the AS/400 system.

This time it’s different. Rather than having to learn different APIs or OLE automation objects to access the AS/400 system, you only need to learn ADO.

1.4 The ActiveX Data Object Components

Before you can use AS/400 OLE DB support, you need to know how to work with ADO. When you work with ADO, there are several components that you use to describe:

- Your connection to the AS/400 system
- The provider you’ll be using
- The type of action you want to invoke
- Records or results that you want to use in your program
- Errors that occur when attempting an operation

You need to understand how to work with components in a Windows environment. The following sections explain the important OOP concepts that you’ll need to know to work with ADO. If you are already familiar with those concepts, skip ahead to the discussion of the specific ADO components.

1.4.1 What Does ActiveX Mean?

You may be wondering what “ActiveX” means. You learned about Microsoft’s COM technology earlier in this chapter. When the Internet suddenly became popular a few years ago, it quickly became apparent that web pages that only used HTML had some limitations, especially compared with Windows
applications. Things that would be easy to do in a Windows program using a language like Visual Basic were quite difficult to do using just HTML.

In traditional Windows development, if you create a Windows program and then distribute it to other PCs, the components that you used in the program are also distributed. That distribution technique works well if you are loading software from diskette or CD-ROM. But if you want to use a component on a web page, there is no guarantee that the computer viewing the web page will have the component already installed. One solution to that problem is to download the missing component as the page is being loaded.

Because COM components were not particularly optimized for size, it could take a very long time to download those components. Microsoft did some work to revise components so they could be smaller. Also, since COM/OLE components was not a very catchy name, Microsoft applied the label “ActiveX” to the new components.

Downloading ActiveX components while loading web pages is not a widely adopted technique. There is nothing that prevents an ActiveX component from running harmful functions on a user’s machine, so most people do not allow their browser to download and install ActiveX components.

For software that you install on your PC through traditional means, ActiveX presents no more risk than any other type of software that you might use. The ADO components that you will use are completely under the control of your programs.

The bottom line is, as far as you need to be concerned, ActiveX is COM.

1.4.2 The ADO Object Model

Another term that you hear quite a lot as you start using OOP techniques is object model. Figure 3 on page 18 and Figure 2 on page 15 show the ADO object model. Most Microsoft object models are depicted this way, as an assortment of labeled boxes with connecting lines.

An object model identifies the components that are available and shows the relationships among those components. In most object models, there is a hierarchy, meaning that you need to do set-up work for the higher level components before you can use the lower level components. ADO is quite different from most other object models in that there is no strict hierarchy that dictates when you can and cannot use a component. In the ADO object model shown in the figures, the hierarchy is more a suggestion.
1.4.2.1 Instantiating Objects

Before you can use any of the ADO components, your program must first instantiate, or create an instance of, an object. You can think of instantiation as being similar to building a house from a set of blueprints. The blueprints are the object model. They are not the object; for example, you cannot live in the blueprint. But with the blueprints in hand, you can create an instance of a house that has all of the characteristics described in the blueprints.

You can also create multiple instances of objects, just as you can build more than one house from the same set of blueprints. When you create multiple instances, you identify each instance by using its unique name, just as you have a unique address associated with each house. That way, when your program needs to tell the object to do something, it can refer to the object by name.

For example, if you want to create an ADO Connection object to describe your connection to the AS/400 system, you could use this statement in Visual Basic:

```vbnet
Private Cn400 As New ADODB.Connection
```

- **Cn400** is the unique name that you assign to the connection object.
- The **New** keyword is Visual Basic’s way of saying that you want to create the **Cn400** object.
- **ADODB** is the name of the class library that contains the ADO components.
- **Connection** identifies the class (component) that you want to use as the basis of your new object.

When you work with ADO, you will usually instantiate a Connection object, one or more Command objects and one or more Recordset objects. The other objects shown in the figures are automatically instantiated for you.
1.4.2.2 Why are there singular and plural objects?

Figure 3 on page 18 contains three pairs of objects, each singular and plural:
- Error/Errors
- Parameter/Parameters
- Field/Fields

It is very important that you understand how these objects are used.

The plural form of the object is called a *collection*. A collection is a set of singular objects. For example, when you retrieve several records from the AS/400 system in an SQL statement, the records are available to your program in a Recordset object. Data is available in individual Field objects. Because you may want to work with all of the fields as a set (for example, to clear all of the field values) or get a count of the number of fields, the individual Field objects are accessed through the Fields collection.

If you are familiar with RPG, you might think of an individual Field object as being similar to a data structure. The attributes of a Field object include its name, type of data, length of the data and decimal positions.

The Fields collection would be similar to an RPG multiple occurrence data structure (MODS). Similarly to a MODS, you can loop through the Fields collection and perform operations on every individual Field object in the collection. One big difference between the RPG MODS and collections is that you can refer to an item in a collection by either an index number (like a MODS) or by a key value that you assign. For example, when you retrieve records from the AS/400 system using ADO, you can refer to a field by its position in the record or by its AS/400 field name, which is used as the key to the Fields collection.

ADO collections start numbering at zero. The first object in the collection is `object(0)`. Because each collection also has a `Count` property, you can iterate through a collection by looping from `object(0)` to `object.Collection.Count - 1`. For example, you can loop through the Fields collection like this:

```vbnet
For N = 0 To rsData.Fields.Count - 1
    FieldValue = rsData.Fields(N)
Next N
```

1.4.2.3 Properties and Methods

When you use ADO objects in your programs, you first identify the component that you need to use for a particular function. For example, to create a connection to the AS/400 system, you use a Connection object.

To describe your connection to the AS/400 system you assign a value to the `ConnectionString` property of the Connection object. A property is an attribute of an object. Most components have many properties. The list of properties associated with an object is defined in the class module that the object was instantiated from. For example, when you instantiate the `ADODB.Connection` object, the list of properties used with your object are derived from the Connection class.

Each property expects only certain types of values. For example, a `ConnectionString` property expects to be given a string containing the name of the ADO provider and the AS/400 system you want to connect to. If you set the value of the `ConnectionString` to a number instead of a valid string, you'll get an error.
when you try to use the Connection object. If your program gets input from a user and uses that input to directly set property values, you’ll want to check the value before assigning it to the property.

You tell an object to perform an operation for you by invoking one of its *methods*. Like properties, the list of methods that an object provides is defined in the class module. Methods cause code inside the object to run. For example, the Connection object has an *Open* method that causes the connection to the AS/400 system to be opened.

Most methods have one or more parameters. In many cases, you can supply the parameter values for the method either by setting properties and then invoking the method, or by supplying the parameter values as part of the method invocation. For example, here are two techniques in Visual Basic for opening a connection to the AS/400 system:

- First technique – set properties then invoke method:
  ```vbnet
  Cn400.ConnectionString = "Provider=IBMDA400; Data Source=MYAS400;"
  Cn400.Open
  ```

- Second technique – supply parameter as part of method invocation:
  ```vbnet
  Cn400.Open ConnectionString:=ConnStr
  ```

It doesn’t matter which technique you use when the components are located on your PC. If you use components across a network, the second technique is preferred since you can set the property value and invoke the method in one network transmission rather than two.
1.5 Working with ADO objects

There are four main ADO objects that you’ll work with when you use AS/400 OLE DB support:

- **Connection** – Used to describe your connection to the AS/400 system and to specify the provider that OLE DB is to use.
- **Command** – Used to identify a file to work with, an SQL statement, stored procedure or program call.
- **Recordset** – Contains the records that the AS/400 system sent back to your program.
- **Fields** – Collection of field data for the records in the Recordset.

The remaining ADO objects are:

- **Errors** – If any errors occur when using ADO objects, they are put into the Errors collection. You can iterate through the collection and retrieve information about each error.
- **Parameters** – Another collection that is used to store parameter descriptions and values. You will typically use this for stored procedures.
- **Properties** – An ADO component that is used to extend the predefined list of properties for the Connection, Command, Recordset and Field objects. AS/400 OLE DB support uses the Properties collection only on the Command component.

![Figure 3. Objects Used with ADO](image)

1.5.1 Dependencies among ADO Objects

Although Figure 3 shows the connections among the various ADO objects, there are actually very few requirements in ADO to instantiate one object before using another. For example, to run an SQL statement using ADO, you will typically do the following:

1. Open a **Connection** object to define the connection to the AS/400 system.
2. Put the SQL statement into a **Command** object. You also associate the **Connection** object with the **Command** object.
3. Invoke the **Command.Execute** method to run the SQL statement.
4. Retrieve the results from the **Recordset** object.
You can also run the SQL statement like this:

1. Put the SQL statement into a Command object. Set the Command.ActiveConnection property to a valid connection string.
2. Invoke the Command.Execute method to run the SQL statement.
3. Retrieve the results from the Recordset object.

Finally, you can run the SQL statement like this:

1. Set the Recordset.ActiveConnection property to the connection string. Set the Recordset.Source property to the SQL statement.
2. Invoke the Recordset.Open method.
3. Retrieve the results from the Recordset object.

Although it doesn’t matter to ADO or AS/400 OLE DB support which technique you use, you may find it advantageous to use the first technique, where you create a Connection object, a Command object and a Recordset object. The advantages of creating distinct objects are:

- You can use the Connection object for multiple Command objects. If you choose to open the connection directly with the Command or Recordset objects, you do not have a reusable Connection object.
- Your code will be easier to debug, since you can independently verify that you have a valid Connection and Command before trying to get results into the Recordset.
- Your code will be easier to maintain and extend. As ADO evolves, there will probably be additional properties and methods added to each of the objects. If you have independently created objects, you can add to or modify those objects more easily than if you try to perform too much activity with one component.

Most of the examples shown in this redbook use independent creation of Connection, Command and Recordset objects.
1.5.2 Connection Object

The ADO Connection object is used for the following:

- Identifying the AS/400 system you want to connect to.
- Identifying the provider that OLE DB is to use.
- Setting the isolation level for a transaction.

You can create as many ADO Connection objects in your program as you require. For example, if you need to connect to two AS/400 systems, you can create two Connection objects. Also, if you need to use more than one provider, you can create a separate Connection object for each.

Table 1 describes the Connection object as used in AS/400 OLE DB support.

<table>
<thead>
<tr>
<th>Table 1. The ADO Connection Object</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connection Object</strong></td>
</tr>
<tr>
<td>How instantiated</td>
</tr>
<tr>
<td><strong>Properties</strong></td>
</tr>
<tr>
<td>ConnectionString</td>
</tr>
<tr>
<td>Cn400.ConnectionString = _</td>
</tr>
<tr>
<td>&quot;Provider=IBMDA400;Data Source=MYAS400;&quot;</td>
</tr>
<tr>
<td>Cn400.ConnectionString = _</td>
</tr>
<tr>
<td>&quot;Provider=MSDASQL;Data Source=MY170;&quot;</td>
</tr>
<tr>
<td><strong>Methods</strong></td>
</tr>
<tr>
<td>Open</td>
</tr>
<tr>
<td>Cn400.Open ConnectionString:= _</td>
</tr>
<tr>
<td>&quot;Provider=IBMDA400;Data Source=MYAS400;&quot;</td>
</tr>
<tr>
<td>Close</td>
</tr>
<tr>
<td>Cn400.Close</td>
</tr>
<tr>
<td>Execute</td>
</tr>
<tr>
<td><strong>Note</strong>: This technique is not recommended. Create a separate Command object instead.</td>
</tr>
<tr>
<td>Set rsRecords = Cn400.Execute _</td>
</tr>
<tr>
<td>&quot;SELECT * FROM QGPL.TEST&quot;, ,adCmdText</td>
</tr>
<tr>
<td>BeginTrans, CommitTrans, RollbackTrans</td>
</tr>
<tr>
<td>Cn400.BeginTrans</td>
</tr>
<tr>
<td>Cn400.CommitTrans</td>
</tr>
<tr>
<td>Cn400.RollbackTrans</td>
</tr>
</tbody>
</table>
1.5.3 Command Object

The ADO Command object is used for the following:

- **Identify for use:**
  - AS/400 database file (physical or logical). You can also identify a specific member and record format in the file.
  - SQL statement.
  - Stored procedure to invoke.
  - AS/400 program to call.
  - Data queue to send to and receive from.
  - AS/400 command to run.
- Execute the operation to the AS/400 system using the selected technique.

You can create as many ADO Command objects in your program as you require. For example, if you need to work with two database files, you can create two Command objects. Prior to invoking the Execute method, set the value of the ActiveConnection property so that the Command object knows which AS/400 system it is to work with.

Table 2 describes the Command object as used in AS/400 OLE DB support.

**Table 2. The ADO Command Object**

<table>
<thead>
<tr>
<th>Command Object</th>
<th>How instantiated</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Private cmFile As New ADODB.Command</td>
</tr>
<tr>
<td></td>
<td>Private cmSQL As New ADODB.Command</td>
</tr>
<tr>
<td></td>
<td>Private cmStoredProc As New ADODB.Command</td>
</tr>
<tr>
<td></td>
<td>Private cmProgram As New ADODB.Command</td>
</tr>
<tr>
<td></td>
<td>Private cmDTAQ As New ADODB.Command</td>
</tr>
<tr>
<td></td>
<td>Private cmCommand As New ADODB.Command</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Properties</th>
<th>Identifies the Connection object that is to be associated with the Command object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveConnection</td>
<td>Note: You can also provide a valid connection string for this property to directly open the connection. We recommend that you create and use a separate Connection object as shown in the first line of code below.</td>
</tr>
</tbody>
</table>

```plaintext
Set cmFile.ActiveConnection = Cn400

cmFile.ActiveConnection = _
"Provider=IBMDA400;Data Source=MVAS400;"
```
Command Text

Identifies the file to open, SQL statement to run, stored procedure or program to call, data queue to use or AS/400 command.

You can open an AS/400 database file using System naming format or IFS format. SQL statements must be valid AS/400 SQL. For stored procedures, you must open and close the CALL statement with single braces \{ \}. Program calls and AS/400 commands must be opened and closed with double braces \{\{\}. Data queues must be opened with IFS syntax.

**Note:** You always need to include the \( )\) characters for a database file, even if there are no values inside. This is required because the IBMDA400 provider uses the parentheses to determine that the CommandText is being used to specify a database file name. If you use System naming format and omit the parentheses, you will actually cause an SQL SELECT * statement to be run for the database file. If you use IFS format and omit the parentheses, you will have a run-time error.

**Note:** The IBMDA400 provider uses the single and double brace characters to determine if you are requesting a stored procedure or AS/400 program or command. Although it seems arbitrary to require this syntax, it is required because ADO actually passes the CommandText value to the provider as an SQL SELECT * statement. Before passing the CommandText to OLE DB, IBMDA400 parses the CommandText and is able to determine if it should call the stored procedure, program or command. If you omit the braces, you will have a run-time SQL error.

```
cmFile.CommandText = "TESTLIB.TESTFILE(*FIRST)"
cmFile.CommandText= _
"/QSYS.LIB/TESTLIB.LIB/TESTFILE.FILE/%FIRST%.MBR()"
cmSQL.CommandText = "SELECT * FROM QGPL.MYFILE"
cmStoredProc.CommandText = _
"(CALL QGPL.MYPROC(?,?,?))"
cmProgram.CommandText = _
"{CALL /QSYS.LIB/TESTLIB.LIB/TESTPGM.PGM(?,?)})"
cmDTAQ.CommandText = "OPEN DATAQUEUE
/QSYS.LIB/TESTLIB.LIB/MYDTAQ.DTAQ(Data Char(50))
FOR SEND"
cmCommand.CommandText = _
"{{QSYS/CRTLIB LIB(TESTLIB2)}}"
```
**Command Object**

<table>
<thead>
<tr>
<th>CommandType</th>
<th>Specifies the type of command that is in CommandText. Set this property to the corresponding value shown here:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database file</td>
<td>cmFile.CommandText = adCmdTable</td>
</tr>
<tr>
<td>SQL</td>
<td>cmSQL.CommandText = adCmdText</td>
</tr>
<tr>
<td>Stored Procedure</td>
<td>cmStoredProc.CommandText = adCmdText (Note: Use this value, not adCmdStoredProc)</td>
</tr>
<tr>
<td>Program call</td>
<td>cmProgram.CommandText = adCmdText</td>
</tr>
<tr>
<td>Data Queue</td>
<td>cmDTAQ.CommandText = adCmdText</td>
</tr>
<tr>
<td>Command</td>
<td>cmCommand.CommandText = adCmdText</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Prepared</th>
<th>Set this property to cause ADO to save the SQL statement, stored procedure, program or command string before executing the Command object the first time. This is useful if you expect to repeatedly execute the Command with the same CommandText. If you will only execute the Command with the same CommandText once or twice, you should not set this property, since there is overhead to saving the CommandText.</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmSQL.Prepared = True</td>
<td>cmProgram.Prepared = False</td>
</tr>
</tbody>
</table>
CreateParameter

Methods

CreateParameter

Used to define a parameter that will be used with a stored procedure or program call. The parameters for a stored procedure or program call are shown as parameter markers (using the ? symbol) when the CommandText for the stored procedure or program is defined (see above).

**Note:** You cannot dynamically pass parameter values for AS/400 commands. You must code an AS/400 command string with all of the parameter values that you need to pass.

You can either assign the result of the CreateParameter method to a parameter object or directly append the result to the Parameters collection associated with the Command object.

Assign to a Parameter object:

```vbnet
Private P1 As New ADODB.Parameter
Set P1 = cmStoredProc.CreateParameter("STARTAT", adChar, adParamInput, 10)
cmStoredProc.Parameters.Append P1
```

Append directly to Parameters collection:

```vbnet
cmStoredProc.Parameters.Append cmStoredProc.CreateParameter("STARTAT", adChar, adParamInput, 10)
```
**OLE DB and ActiveX Overview**

<table>
<thead>
<tr>
<th>Command Object</th>
<th>Causes the activity defined in the CommandText to occur.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Execute</strong></td>
<td><strong>Note</strong>: The Execute method is not used to retrieve data from AS/400 database files using record-level access. The AD400 GetBookmark method is used to retrieve records using record level access.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: If you will be calling a stored procedure, be sure that the commitment control level on the stored procedure is correct. The default value on CRTSQLxxxx commands is COMMIT(*CHG). That value is in effect even if the SQL statements in the stored procedure are only SELECT statements. If you are only using SELECT statements or if your application does not require commitment control, create your stored procedure with COMMIT(*NONE). If you fail to do this, you may generate a run-time SQL error in your client program when you use the Execute method to fill a recordset object. You can avoid the error by destroying the recordset object prior to invoking the Execute method.</td>
</tr>
<tr>
<td></td>
<td>To fill a recordset (SQL SELECT, stored procedure, data queue):</td>
</tr>
<tr>
<td></td>
<td>Set rsSQL = cmSQL.Execute</td>
</tr>
<tr>
<td></td>
<td>Set rsStoredProc = cmStoredProc.Execute</td>
</tr>
<tr>
<td></td>
<td>Set rsDTAQ = cmDTAQ.Execute</td>
</tr>
<tr>
<td></td>
<td>To run a program or AS/400 command.</td>
</tr>
<tr>
<td></td>
<td>cmProgram.Execute</td>
</tr>
<tr>
<td></td>
<td>cmCommand.Execute</td>
</tr>
</tbody>
</table>

T o fill a recordset (SQL SELECT, stored procedure, data queue):

Set rsSQL = cmSQL.Execute
Set rsStoredProc = cmStoredProc.Execute
Set rsDTAQ = cmDTAQ.Execute

To run a program or AS/400 command.

cmProgram.Execute
cmCommand.Execute
1.5.4 Recordset Object

The ADO Recordset object is used to contain the results of your data retrieval request. For example, if you execute an SQL statement, the recordset will contain the result set that is produced by the SQL.

The recordset, although not a visible object, is similar to an AS/400 subfile. The recordset contains a set of records. You can use the Move methods to navigate to different records contained in the recordset. If you try to navigate prior to the beginning of the recordset or past the end, the BOF or EOF properties are set to True.

You can create as many recordset objects as you require in your program. If you reuse a recordset, it is cleared before new data is written to it.

When you create a recordset object, a corresponding Fields collection is automatically created. To work with a recordset, you first navigate to the record within the recordset you want to work with. You then reference data with Fields items.

---

**Attention**

If you are filling a recordset by calling an AS/400 stored procedure (program object), be sure that the commitment-control level specified when you create the program is correct. By default, the COMMIT(*CHG) value is used. If you call the stored procedure twice, you may get an SQL error in your application. You can avoid this error by doing either of the following:

- Compile the program with COMMIT(*NONE) if that is appropriate for your application. For example, stored procedures that contain only SQL SELECT statements do not require commitment control.
- Include code in your client program to destroy the recordset object before the second call to the stored procedure. You cannot simply use the Close method of the recordset object; you must actually destroy the object then create a new instance of it by calling the stored procedure.
Table 3. The ADO Recordset Object

<table>
<thead>
<tr>
<th>Recordset Object</th>
<th>How instantiated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private rsData As New ADODB.Recordset</td>
<td></td>
</tr>
</tbody>
</table>

### Properties

#### ActiveConnection

Used to set the value of the connection to use or return a string containing the connection associated with the recordset object.

You can set the value to refer to a previously created connection object or to a connection string. Using the connection object is the preferred technique.

```vba
Set rsData.ActiveConnection = Cn400
Set rsData.ActiveConnection = _
    "Provider=IBMDA400;Data Source=MYAS400;"
```

To retrieve a string containing the active connection for this Recordset:

```vba
TheConnection = rsData.ActiveConnection
```

#### BOF / EOF

These properties are used to indicate if you are at the beginning (BOF, “beginning of file”) or end (EOF, “end of file”) of the recordset. Although BOF and EOF indicate a condition associated with a file, a recordset is not a file. However, the properties are widely understood in the context of processing a group of records, so they are also used in ADO.

When you execute a command that creates a recordset, there are two possibilities:

* There are no records in the recordset. Both BOF and EOF are True.

* There are one or more records in the recordset. Both BOF and EOF are False. The current record in the recordset is set to the first record.

You usually use these properties when you iterate through the recordset. If either BOF or EOF are True and you attempt to use one of the Move operations that moves beyond the beginning or end of the recordset, you receive an error. When either BOF or EOF are True, there is no current record. You receive an error if you try to use any operations that require a current record.

If you delete all records in a recordset, the BOF and EOF properties may remain False until you attempt a Move operation on the recordset.

The following code shows a sample of using these properties to read through a recordset:

```vba
If (Not rsData.BOF) And (Not rsData.EOF) Then
    Do Until rsData.EOF
        rsData.MoveNext
        Loop
End If
```
### Bookmark

A bookmark lets you store in a variable the location of the current record in the recordset. At a later point in the program, you can return directly to the record by referring to the saved bookmark value that you stored.

- Store the value of the current record:
  
  ```vba
  BookmarkedRecord = rsData.Bookmark
  ```

- Return to the record:
  
  ```vba
  rsData.Bookmark = BookmarkedRecord
  ```

### EditMode

Used to indicate the status of editing operations on the current record in the recordset.

- The `EditMode` property returns one of these values:
  - `adEditNone` – The current record has not been changed.
  - `adEditInProgress` – Some data in the current record has been changed. The values in the recordset are not changed until you use the `Update` method of the recordset object.
  - `adEditAdd` – The `AddNew` method was invoked and there is data in the new record buffer that is not yet in the recordset.

### Methods

#### AddNew

Used to create a buffer that is used to hold data for a potential new record for the recordset.

- Sample of adding a new record to the recordset:
  
  ```vba
  Dim Fields
  Dim Values
  Fields = Array("CID", "FNAME", "LNAME")
  Values = Array(CustID, FirstName, LastName)
  rsData.AddNew Fields, Values
  ```

- **Note:** When this redbook was written, the AS/400 OLE DB support provider did not support the following technique to add a record to the AS/400 database. Use the technique shown above to add a record.
  
  ```vba
  rsData.AddNew
  rsData("CID") = CustID
  rsData("FNAME") = FirstName
  rsData("LNAME") = LastName
  rsData.Update
  ```

#### Close

This method closes the recordset, which releases any locks you have on the data. Data is cleared from the recordset. You cannot use any of the recordset `Move` operations while the recordset is closed. You can use the `Open` method to reopen the recordset.

- ```vba
  rsData.Close
  ```
<table>
<thead>
<tr>
<th>Recordset Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Delete</strong></td>
<td>Deletes the current record from the recordset. The record is removed from the AS/400 database.</td>
</tr>
<tr>
<td>rsData.Delete</td>
<td></td>
</tr>
<tr>
<td><strong>Move</strong></td>
<td>Use the Move methods to navigate through the recordset. You can only use a Move method when the move is allowed for the current record. For example, if the EOF property is True, you cannot use the MoveNext method.</td>
</tr>
<tr>
<td>MoveFirst</td>
<td></td>
</tr>
<tr>
<td>MoveLast</td>
<td></td>
</tr>
<tr>
<td>MoveNext</td>
<td></td>
</tr>
<tr>
<td>MovePrevious</td>
<td></td>
</tr>
<tr>
<td>rsData.Move 2</td>
<td>(Note: Move 1 is the same as MoveNext)</td>
</tr>
<tr>
<td>rsData.Move 0, Bookmark</td>
<td></td>
</tr>
<tr>
<td>rsData.MoveFirst</td>
<td></td>
</tr>
<tr>
<td>rsData.MoveLast</td>
<td></td>
</tr>
<tr>
<td>rsData.MoveNext</td>
<td></td>
</tr>
<tr>
<td>rsData.MovePrevious</td>
<td></td>
</tr>
<tr>
<td><strong>Open</strong></td>
<td>Use the Open method to open a recordset as an alternative to opening the recordset as the result of a command Execute method. When you use the Open method, you supply the command text or a reference to a command object as a parameter of the Open method.</td>
</tr>
<tr>
<td>rsData.Open cmSQL</td>
<td></td>
</tr>
<tr>
<td><strong>Requery</strong></td>
<td>Use the Requery method to refresh the contents of the recordset. This method is particularly important when you work with data queues. When you initially open a recordset for data queues, all of the current entries on the data queue are put into the recordset. To retrieve data queue entries that are added after you load the recordset, you use the Requery method on the recordset. The existing entries in the recordset are replaced with the new entries from the data queue.</td>
</tr>
<tr>
<td>rsData.Requery</td>
<td></td>
</tr>
<tr>
<td><strong>Supports</strong></td>
<td>The Supports method is used to determine which features are supported by the provider. If the provider supports the selected feature, a True value is returned, otherwise False is returned.</td>
</tr>
<tr>
<td>Support = rsData.Supports(adAddNew)</td>
<td></td>
</tr>
<tr>
<td><strong>Note:</strong> You can only test the adAddNew, adBookmark, adDelete, adMovePrevious and adUpdate options.</td>
<td></td>
</tr>
<tr>
<td><strong>Update</strong></td>
<td>Use the Update method to save changes to a record in the recordset. The record is updated in the AS/400 database.</td>
</tr>
</tbody>
</table>
1.5.5 Fields Collection, Field Object

The ADO Fields collection and Field object are used to contain the set of fields for a record in a recordset and to hold the value and attributes of each individual field in the record. The collection and objects are created automatically when you create a recordset object. Each recordset has its own Fields collection. You indicate which Fields collection you are using by specifying the recordset name. You work with a specific field by using either the field name or its ordinal position within the record. The field name is the AS/400 field name:

```
rsData.Fields("CUSTID")
rData.Fields(2)
```

When you use AS/400 OLE DB support to retrieve records from the AS/400 system, the records are available to you in the recordset. You usually iterate over the recordset and work with one record at a time. For each record, you can extract the field values and work with them in your program.

```
Set rsCustomer = cmCustomer.Execute

'------------------------------------------------------------------------------
'iterate through recordset, add to customer collection
'------------------------------------------------------------------------------
With rsCustomer
    If (Not .BOF) And (Not .EOF) Then
        Do Until .EOF
            Set Customer = Add(.Fields(rsData.CustID),
                                .Fields(rsData.FirstName),
                                .Fields(rsData.LastName),
                                .Fields(rsData.Initial),
                                .Fields(rsData.Address1),
                                .Fields(rsData.Address2),
                                .Fields(rsData.City),
                                .Fields(rsData.State),
                                .Fields(rsData.ZipCode))
            .MoveNext
        Loop
    End If
End With
```
**Table 4. The ADO Field Object**

<table>
<thead>
<tr>
<th>Field Object</th>
<th>How instantiated</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(automatically as part of recordset)</td>
<td></td>
</tr>
</tbody>
</table>

**ActualSize**
- For character fields, returns the number of bytes contained in the field. For numeric fields, may return `adUnknown` (-1). Use the `Precision` and `NumericScale` properties to determine the size of numeric fields.
  
  ```vbscript
  FieldSize = rsData.Fields("CUSTID").ActualSize
  ```

**Attributes**
- Returns a value that indicates which of the following attributes the field possesses:
  - `adFldUpdatable` (value of 4) — the field can be updated
  - `adFldFixed` (value of 16) — the field is a fixed length field

  The value is the sum of the attributes for the field.

  ```vbscript
  FieldAttribute = rsData.Fields("CUSTID").Attributes
  ```

  If `FieldAttribute` is 20, then the field is a fixed length field that can be updated.

**DefinedSize**
- Returns a value indicating the length of the field in the field definition. For numeric fields, may return `adUnknown` (-1). Use the `Precision` and `NumericScale` properties to determine the size of numeric fields.
  
  ```vbscript
  FieldDefinedSize = rsData.Fields("CUSTID").DefinedSize
  ```

**Name**
- Returns the name of the field.

  ```vbscript
  FieldName = rsData.Fields(0).Name
  ```

**NumericScale**
- For a numeric field, returns the number of decimal positions.

  ```vbscript
  Decimals = rsData.Fields("AMTDUE").NumericScale
  ```

**OriginalValue**
- Returns the value of the field in the recordset. This value is available until you use the `Update` method to update the field within the recordset.

  ```vbscript
  ValueBefore = rsData.Fields("CUSTID").OriginalValue
  ```

**Precision**
- For a numeric field, returns a value indicating the number of digits to the left of the decimal point.

  ```vbscript
  NumDigits = rsData.Fields("AMTDUE").Precision
  ```

**Type**
- Returns a value indicating the data type of the field. Refer to the Appendix C for the list of data types used for AS/400 data.

  ```vbscript
  DataType = rsData.Fields("CUSTID").Type
  ```
1.5.6 Errors Collection, Error Object

The ADO Errors Collection and Error objects are used to contain the set of errors, if any, that occur when using the AS/400 OLE DB support or MSDASQL providers. Errors that occur from using ADO objects are not reported through the Errors collection, but are available to your program through the standard error trapping technique in the language (for example, in Visual Basic the ADO errors are reported in the Err object).

The Errors collection is automatically created when you create a connection. The Errors collection is cleared before executing any ADO method. Any errors that the provider reports during execution of an ADO method are added to the Errors collection. In your program, you should check the Errors collection after each ADO method execution. If the count of errors in the Errors collection is greater than zero, you can iterate through the Errors collection to examine each error condition. The errors in the collection may be similar to AS/400 error reporting, in that the first error may be the most significant.

**Attention**

You can only access the Errors collection if you have a Connection object.

If you open a Command or Recordset object directly (by supplying a connection string instead of a connection object) and if there are any AS/400 OLE DB support provider errors when you use the Command or Recordset objects, you will be unable to retrieve the error information. If an error occurs and you do not have access to the Errors collection, your program will receive the first error that occurred. You will have to use your language’s error handling facilities to handle the error.

This is one of the reasons why you should always explicitly create a Connection object.
The errors reported by AS/400 OLE DB support include an error identifier in the description. The identifier is a prefix to a specific error number. Some of the identifiers include:

Table 5. Error Identifiers Used with AS/400 OLE DB Support

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWBZZ</td>
<td>AS/400 OLE DB support Provider</td>
</tr>
<tr>
<td>CWBDB</td>
<td>Database</td>
</tr>
<tr>
<td>CPF</td>
<td>OS/400</td>
</tr>
<tr>
<td>SQL</td>
<td>SQL</td>
</tr>
</tbody>
</table>

See the SDK Reference manual for a list of errors that may occur when using the AS/400 OLE DB support provider. For information on how to obtain the SDK Reference, please see Appendix A.2, “Other Documentation and Resources” on page 373. The following code sample shows how to iterate over the Errors collection and display a message box for all of the errors:

```vbnet
Dim MsgText
If ADOConnection.Errors.Count > 0 Then
    For Each ADOError In ADOConnection.Errors
        MsgText = MsgText & _
        "Number= " & ADOError.Number & " " _
        "Description= " & ADOError.Description & " " _
        "Source= " & ADOError.Source & vbCrLf
    Next
    MsgBox MsgText, vbOKOnly, "AS/400 OLE DB support Provider Reported Errors"
End If
```

Table 6. The ADO Error Object

<table>
<thead>
<tr>
<th>Error Object</th>
<th>How instantiated</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(automatically when Connection object is created)</td>
<td>Contains the error text from the AS/400 OLE DB support provider. The error identifier assigned by the provider is the first word in the text.</td>
</tr>
<tr>
<td>Description</td>
<td></td>
<td>ErrDesc = ADOConnection.Errors(0).Description</td>
</tr>
<tr>
<td>Number</td>
<td></td>
<td>Contains the error number assigned to the error by the AS/400 OLE DB support provider. These numbers are documented in Appendix C.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ErrNumber = ADOConnection.Errors(0).Number</td>
</tr>
<tr>
<td>Source</td>
<td></td>
<td>Contains a string identifying the provider that is reporting the error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When reported by the AS/400 OLE DB support provider, the Source is IBMDA400 Session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When reported by the MSDSASQL provider, the Source is Microsoft OLE DB Provider for ODBC Drivers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ErrSource = ADOConnection.Errors(0).Source</td>
</tr>
</tbody>
</table>
1.5.7 Parameters Collection and Parameter Object

The ADO Parameters collection and Parameter objects are used to contain the list of parameters for a stored procedure or program call. Before calling a stored procedure or program you add parameter objects to the parameters collection. You can assign values to parameters passed from your program to the stored procedure or called program. Upon completion of the stored procedure or called program, your program can retrieve output or updated parameter values from the parameters collection.

The Parameters collection is automatically created when you create a Command object.

The following code shows how to populate a parameters collection. The collection can be used for either a stored procedure or program call.

```vba
Dim P1 As New ADODB.Parameter
Dim P2 As New ADODB.Parameter

Set P1 = cmCustomer.CreateParameter(Name:="P1", _
    Type:=adChar, _
    Direction:=adParamInput, _
    Size:=5, _
    Value:="00001")

Set p2 = cmCustomer.CreateParameter(Name:="P2", _
    Type:=adInteger, _
    Direction:=adParamInputOutput, _
    Size:=3, _
    Value:=123)

cmCustomer.Parameters.Append P1
cmCustomer.Parameters.Append P2
```

The code can also be written so that it does not require the P1 and P2 objects:

```vba
cmCustomer.Parameters.Append cmCustomer.CreateParameter(Name:="P1", _
    Type:=adChar, _
    Direction:=adParamInput, _
    Size:=5, _
    Value:="00001")

cmCustomer.Parameters.Append cmCustomer.CreateParameter(Name:="P2", _
    Type:=adInteger, _
    Direction:=adParamInputOutput, _
    Size:=3, _
    Value:=123)
```

To refer to the value of the second parameter:

```vba
txtCustID = cmCustomer.Parameters(1).Value

txtCustID = cmCustomer.Parameters("P2")
```
### Table 7. The ADO Parameter Object

<table>
<thead>
<tr>
<th>Parameter Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>How instantiated</strong></td>
<td>Dim P1 As New ADODB.Parameter</td>
</tr>
<tr>
<td><strong>Properties</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Direction</strong></td>
<td>Used to set or return a value that indicates how the parameter is used. For stored procedures, this property must be set so that the parameter defined in your program agrees with the cataloged stored procedure. For called programs, this property indicates the intended use of a parameter.</td>
</tr>
<tr>
<td></td>
<td>Possible values for this property include:</td>
</tr>
<tr>
<td></td>
<td>adParamInput — The parameter is input (sent from this program to the stored procedure or called program).</td>
</tr>
<tr>
<td></td>
<td>adParamOutput — The parameter is output (received from the stored procedure or called program).</td>
</tr>
<tr>
<td></td>
<td>adParamInputOutput — The parameter is input/output.</td>
</tr>
<tr>
<td></td>
<td>adParamReturnValue — The parameter is a return value from a stored procedure.</td>
</tr>
<tr>
<td></td>
<td>Dim P1 As New ADODB.Parameter</td>
</tr>
<tr>
<td></td>
<td>P1.Direction = adParamInputOutput</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>Used to set or return the name assigned to the parameter. You can use the name to refer to the parameter in the parameters collection. If you don’t assign a name, you must use ordinal numbers to refer to a parameter in the parameters collection.</td>
</tr>
<tr>
<td></td>
<td>The name you assign does not have to be the same name assigned to the parameter in the stored procedure or called program.</td>
</tr>
<tr>
<td></td>
<td>Dim P1 As New ADODB.Parameter</td>
</tr>
<tr>
<td></td>
<td>P1.Name = &quot;CUSTID&quot;</td>
</tr>
<tr>
<td><strong>NumericScale</strong></td>
<td>Used to set or return the number of decimals for a numeric field. This property cannot be specified on the CreateParameter method of the command object. To set the NumericScale property, create the parameter first then use an assignment statement to set the value of this property.</td>
</tr>
<tr>
<td></td>
<td>Dim P1 As New ADODB.Parameter</td>
</tr>
<tr>
<td></td>
<td>P1.NumericScale = 2</td>
</tr>
<tr>
<td><strong>Precision</strong></td>
<td>Used to set or return the number of digits for a numeric field. This property cannot be specified on the CreateParameter method of the command object. To set the Precision property, create the parameter first then use an assignment statement to set the value of this property.</td>
</tr>
<tr>
<td></td>
<td>Dim P1 As New ADODB.Parameter</td>
</tr>
<tr>
<td></td>
<td>P1.Precision = 5</td>
</tr>
</tbody>
</table>
### Parameter Object

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Code Example</th>
</tr>
</thead>
</table>
| Size      | Used to set or return the number of bytes long the parameter is. | Dim P1 As New ADODB.Parameter  
P1.Size = 10 |
| Type      | Used to set or return the data type of the parameter. The value must be one of the valid ADO data types (see Appendix C for a list of valid ADO data types). | Dim P1 As New ADODB.Parameter  
Dim P2 As New ADODB.Parameter  
P1.Type = adChar  
P2.Type = adInteger |
| Value     | Used to set or return the value for the parameter. | Dim P1 As New ADODB.Parameter  
P1.Value = "TEST"  
P1.Value = 123 |
1.5.8 Properties Collection, Property Object

The ADO Properties collection and Property objects are used to contain the list of provider defined attributes for the following ADO objects:

- Connection
- Command
- Recordset
- Field

A Properties collection is automatically created for each of those objects when the object is created.

The only property used with the AS/400 OLE DB support provider is the Updatability property associated with the Command object. The property is used to indicate the types of operations allowed on AS/400 tables opened by the command. Although you can view and change any of the other properties in any of the property collections, the AS/400 OLE DB support provider does not use the changed values.

The following code shows how to set the value of the Updatability property for a command object. The AS/400 OLE DB support provider uses the following values to determine the table operations allowed. You set the Updatability property to the sum of the options you need for file operations.

<table>
<thead>
<tr>
<th>Open table for</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>0</td>
</tr>
<tr>
<td>Change</td>
<td>1</td>
</tr>
<tr>
<td>Delete</td>
<td>2</td>
</tr>
<tr>
<td>Insert</td>
<td>4</td>
</tr>
</tbody>
</table>

**Note:** Some languages do not support referring to a property by name. The ordinal value for the Updatability property is 16.

```vbnet
cmCustomer.Properties("Updatability") = 7
cmCustomer.Properties(16) = 7
```
Chapter 2. AS/400 OLE DB Support

This chapter provides an overview of the Client Access OLE DB provider for the AS/400 system. It discusses the requirements at the Client end and the server end to use the provider. It also discusses functions that the provider supports and illustrates them with programming examples.

2.1 Introduction

IBM offers OLE DB support for the AS/400 system in two products:

- Client Access OLE DB Provider
- The Software Development Toolkit for ActiveX and OLE DB

The OLE DB specification was developed by Microsoft to provide one consistent interface to all types of data. The Client Access OLE DB provider (we refer to it as run time support) provides access to AS/400 Databases via SQL or Record Level Access. It also provides access to Stored Procedures, Data Queues, AS/400 CL Commands and AS/400 Programs. There is no reason that in the future this could be not be expanded to encompass other non-traditional data like electronic mail, audio, and images.

The Software Development Toolkit for ActiveX and OLE DB provides an additional component that is not part of the run-time support. In general we refer to this component as the SDK. The SDK was in 'Developer Product Preview' at the time this redbook was written. It is scheduled to be shipped with the follow-on version of Client Access. The SDK provides development support with a robust Visual Basic Wizard, program samples for various popular OLE compliant programming platforms like Delphi, Visual Basic, PowerBuilder, End-User tools (Lotus 123) and online documentation. At present, the SDK can only be downloaded from the Client Access Home Page on the World Wide Web. Please see the web-site http://www.as400.ibm.com/clientaccess/oledb for details.

2.1.1 Client Support

OLE DB run time support is available as part of Client Access for Windows 95/NT Version 3 Release 1 Modification 3. This version of Client Access is shipped with OS/400 Version 4 Release 2. Later versions of Client Access will continue to provide OLE DB run-time support.

It is important to point out that OLE DB run-time support is available only for the Windows 95 and NT platforms. At this time, we know of no Microsoft plans to add this support to other platforms.

OLE DB run time support is distributed as Dynamic Link Libraries (DLLs). Run time support DLLs are installed on your PC as part of the base installation of the Client Access product. For the OLE DB support to work properly you do not need to install any additional product options on your PC. Other than what you would normally do to configure Client Access, no additional configuration is required. However, once generally available, the SDK must be installed separately. The SDK may be used during code development and it is not necessary to redistribute it with your final product. The run time DLLs cannot be redistributed - they must be installed on the client PCs as part of Client Access.
The functionality offered by the Client Access OLE DB provider is:

- SQL Statement execution - Access to AS/400 DB/2 databases through the Database server via SQL.
- Stored Procedure calls - Access to AS/400 Stored Procedures through the Database Server.
- Record Level Access - Access to AS/400 DB/2 databases through the DDM Server.
- Data Queue Access - Access to AS/400 Data Queue objects through the Optimized Data Queue Server.
- Distributed Program Calls/Remote Procedure Calls - access to AS/400 CL commands and programs through the DPC/RPC Host Server.

### 2.1.2 Host Support

Client Access and the OLE DB provider require installation of the Host Servers (BOSS option 12) on the AS/400 system. Typically, Host Servers are pre-installed on the AS/400 system. The Host Servers support must be installed on your AS/400 system and started before Client Access and the OLE DB provider can use it. You must be running OS/400 Version 3 Release 1 or later. The Host Servers required for the OLE DB provider do not exist in OS/400 releases prior to Version 3 Release 1.

Depending on the function that is invoked by an application, it may use any of these AS/400 Host Servers:

- Optimized Database Server
- DDM/DRDA Server
- Optimized Data Queue Server
- Remote Command and Distributed Program Call Server

The following table summarizes mapping of AS/400 functionality, subsystems and host server jobs. For additional information on this topic, consult the AS/400 Client Access Host Servers, SC41-5740, and Distributed Data Management, SC41-5307 manuals.

<table>
<thead>
<tr>
<th>Function</th>
<th>Connection Type</th>
<th>Subsystem</th>
<th>Job Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commands/Programs</td>
<td>IP/IPX</td>
<td>QSYSWRK</td>
<td>QZRCRVSVD, QZRCRVS</td>
</tr>
<tr>
<td>Command/Programs</td>
<td>SNA</td>
<td>QCMN/ QBASE</td>
<td>QZRCRVR</td>
</tr>
<tr>
<td>Data queues</td>
<td>IP/IPX</td>
<td>QSYSWRK</td>
<td>QZHQSRSVD, QZHQS SRV</td>
</tr>
<tr>
<td>Data queues</td>
<td>SNA</td>
<td>QCMN/ QBASE</td>
<td>AP PC device name (user ID/workstation)</td>
</tr>
<tr>
<td>SQL/Stored procedures</td>
<td>IP/IPX</td>
<td>QSERVER</td>
<td>QZDASRSVD, QZDASOINIT</td>
</tr>
<tr>
<td>SQL/Stored procedures</td>
<td>SNA</td>
<td>QSERVER</td>
<td>QZDAINIT</td>
</tr>
</tbody>
</table>
OLE DB run-time support is available over any connectivity that Client Access supports. This includes SNA, IP and IPX. If the client connects with the AS/400 system using IPX, you cannot use the Tables Record Level Access functionality. On an IP network, Record Level Access support could be of concern if you are running Pre-Version 4 Release 2 of OS/400. Only Version 4 Release 2 and future Operating system releases implement Record Level Access over native TCP/IP. Prior versions require additional setup.

### 2.1.2.1 Record Level Access - Pre-V4R2M0

When using Tables Record Level Access over an IP network, Pre-V4R2 versions of OS/400 do not offer this support via Native TCP/IP. You need to perform additional setup before this option will work. On Pre-V4R2 Operating Systems you will need to download additional PTFs as well as perform extra steps as described in the PTF Cover letter. Below is a list of version-specific PTFs required. Download the PTF for the version specified only. You must obtain the most recent supersede of a PTF if one is available:

<table>
<thead>
<tr>
<th>Function</th>
<th>Connection Type</th>
<th>Subsystem</th>
<th>Job Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record-Level Access</td>
<td>IP</td>
<td>QSYSWRK</td>
<td>QRWTLSTN, QRWTSRVR</td>
</tr>
<tr>
<td>Record-Level Access</td>
<td>IP Pre-V4R2</td>
<td>QCMN/ QBASE</td>
<td>QCNSKSVT</td>
</tr>
<tr>
<td>Record-Level Access</td>
<td>SNA</td>
<td>QCMN/ QBASE</td>
<td>APPC device name (user ID/workstation)</td>
</tr>
</tbody>
</table>

Table 10. PTF List for Tables - Record-Level Access Pre-R420

<table>
<thead>
<tr>
<th>OS/400 Versions</th>
<th>PROD ID / PTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>V3R1</td>
<td>5763SS1 / SF46301</td>
</tr>
<tr>
<td>V3R2</td>
<td>5763SS1 / SF46302</td>
</tr>
<tr>
<td>V3R6</td>
<td>5716SS1 / SF46306</td>
</tr>
<tr>
<td>V3R7</td>
<td>5716SS1 / SF46303</td>
</tr>
<tr>
<td>V4R1</td>
<td>5769SS1 / SF46313</td>
</tr>
</tbody>
</table>

### 2.1.2.2 Starting Host Server Jobs

Most of the Host server jobs run as Pre-Start jobs. These jobs are started when the subsystem starts. Usually after an IPL, the QSERVER, QSYSWRK, QCMN or QBASE subsystems are started by a startup program. The name of this startup program can be found in the QSTRUPPGM System Value. For IP (or IPX) connections, you may need to start additional host-server jobs. These jobs can be started only after the respective subsystems are active and the IP Server jobs have been started. You may issue the `STRTCPSVR *ALL` command to start all the TCP/IP servers. Once the required subsystems are active and the TCP Server jobs are running, use the `STRHOSTSVR *ALL` command to start the Host Server jobs required by IP. In Version 4 Release 2, you must issue the `STRTCPSVR *CDM` for Table Record Level-Access functionality.
2.1.3 One Common Interface

Before the availability of the Client Access OLE DB provider, developing AS/400 client/server application was not easy. Two major concerns that have surfaced from developers writing AS/400 Client/Server applications are:

1. Performance
2. Ease of application development

Developing an application that performed well often required in-depth understanding of very a complex and ever-changing set of APIs. For example, to develop an application that used AS/400 Data Queues, you not only had to know the specifics of the programming platform/language that you were using, but were required to have a good understanding of the Data Queue APIs. Even if you are an expert in Data Queue APIs, that does not help if you want to develop another application using Optimized SQL APIs, Remote Command APIs or Distributed Program Call APIs.

Client Access OLE DB support makes developing AS/400 client/server application much simpler. All the AS/400 specific implementations are done at the run-time DLL level and are “hidden” from the programmer. A typical programmer has access to AS/400 data and services via one common interface, ADO. Since you are interfacing at the OLE DB layer, AS/400 specific APIs like Data Queues, Optimized SQL, and so on, are not directly used by your program. You have the option to code directly to the OLE DB specification, use ADO, or use a third party ActiveX control that uses OLE DB.

Coding directly to the OLE DB specification requires that you to have a good understanding of complex OLE DB APIs. Most programmers, we believe, would prefer using ActiveX Data Objects(ADO) since it simplifies coding by providing an additional layer of abstraction above the OLE DB layer. You could further simplify your coding by using the Visual Basic Wizard shipped with the SDK.

2.1.4 Final Product Distribution

You cannot redistribute the Client Access run-time DLLs with your ADO/OLE DB application. You can use the following steps to deploy the application:

- Create your product for distribution without including any Client Access run-time DLLs.
- Install Client Access Version 3 Release 1 Mod 3 (or later) with the latest Service Pack on the end-user PCs.
- Install your product.

2.2 OLE DB Provider

When writing your application with ADO, you need to specify the name of the OLE DB provider. The OLE DB provider is comparable to the ODBC driver. Microsoft does not offer a driver manager for OLE DB. The OLE DB provider name, as well as the platform specific information, must be hard-coded in your application. Prompting an end-user for a provider name can cause problems.

Each OLE DB provider registers itself with Windows95/NT when it is installed on the PC. Registering is a process by which a run-time DLL makes the components
it contains known to the Windows registry. The Client Access OLE DB Provider is registered as IBMDA400. This is also known as the PROGID, short for Program ID. This PROGID value for the Client Access OLE DB Provider is added to the Windows Registry during Client Access installation. The Corresponding CLSID is 54E19B41-1D13-11D0-9B95-0020AF344E0A.

If you write an application at the OLE DB level you may programmatically derive the CLSID from the PROGID in your application and supply this CLSID to all the APIs that require it.

2.2.1 Registering OLE DB Provider DLL

If for some reason a DLL is installed, but not registered, your program may not be able to use it and you will get an 'Unable to find Server DLL' error. Although we recommend that you reinstall in such cases, you may try to register the DLL(s) by using the REGSVR32.EXE manually. REGSVR32.EXE utility is shipped by Microsoft. It can be found on almost any Windows 95/NT installation. IBM redistributes it with the SDK.

2.2.2 Naming Conventions

When using the IBMDA400 provider, you have the option of choosing from any one of three naming conventions - *SQL, *SYS or IFS. *SQL naming convention separates the library and the object name by a period. The general syntax of this is:

```
[<library>.] <object>
```

*SYS naming convention uses a forward slash ('/'), instead of a period ('.'). General syntax for *SYS naming convention is:

```
[<library>/]<object>
```

IFS naming refers to objects by their relative location from the system root (also known as the IFS root) directory:

```
/QSYS.LIB/<library.LIB>/<object>.<type>
```

If you have a file MYFILE in a library called MYLIBRARY using *SQL, *SYS and IFS naming convention, you can reference it as:

```
MYLIBRARY.MYFILE (*SQL)
MYLIBRARY/MYFILE (*SYS)
/QSYS.LIB/MYLIBRARY.LIB/MYFILE.FILE (*IFS)
```

*SQL naming must always be used for SQL statements with ADO. This is the only option that IBMDA400 will allow for SQL statements. The same is true for Stored Procedure calls. Data Queue, Commands and Programs do not allow *SQL naming - so you are limited to IFS naming or *SYS naming for these functions. Record level access requests to AS/400 database files can be sent via any of the three naming conventions mentioned here.

Note

Throughout this chapter, we refer to the Client Access OLE DB provider as IBMDA400.
2.2.3 Supported Properties and Methods

An OLE DB provider may implement a number of ADO methods and properties while omitting many others. You must be sure that a specific method or property is implemented before using it in your code. As OLE DB and ADO matures you are likely to see new methods and properties being supported. IBMDA400 was written using ADO Version 1.0 and it supports the OLE DB 1.1 specification. As new versions of ADO and OLE DB become available, IBM will update the provider to incorporate these enhancements.

Tip

For a List of Supported Properties and Methods refer to the SDK Technical Reference manual and the ADO online help. For information about how to obtain this information, see Appendix A.2, “Other Documentation and Resources” on page 373.

2.2.4 Error Handling

Lack of efficient error reporting is one of the major drawbacks of ADO. ADO does not provide very good error handling. Often your program will return an eight digit Hex number and, like most people, you will be left confused. These cryptic error codes do not lend themselves to explaining where the problem is originating from. This is mostly because COM returns errors in the form of a HRESULT format. Please consult Microsoft Knowledgebase article Q168354 for further information.

IBM provides a trace tool that is installed on every PC that has Client Access Version 3 Release 1 Mod 3 or later installed. The name of the trace tool is CWBZZTRC.EXE and it is located in the directory where you installed Client Access. If you run the **cwbzztrc** command from the command line without any parameters it will list all valid options:

![Command Prompt](image)

**Figure 4. cwbzztrc Command Options**
Once you turn on the trace it may generate traces as well as a log file. You can find the traces in Windows 95/NT under Start -> Programs -> IBM AS/400 Client Access -> Service -> Trace Files. The following is an output of the trace generated by using cwbzztcr +t.

![IBMAD400.trc](image)

**Figure 5. IBMAD400.trc**

If you download the OLE DB SDK from the Microsoft Web site (see [www.microsoft.com](http://www.microsoft.com)), you will notice that it provides a spy tool called ITest Spy Tool. Although we did not find the trace that it generates as useful or user-friendly as the ODBC Spy (or SQL Log), for a low OLE DB level understanding (of the error), this tool is useful.

### 2.3 ADO Object Reference

Most people prefer declaring the ADO object variables globally. It is easier to keep track of them that way. It is very important to de-reference and destroy any OLE objects when you exit your program. Declaring the ADO objects globally makes this process simpler. Before you quit the application, set each object that you created to Nothing and close the Connect object either from the Form Unload. Or, if you are using a class module, do it from the Class_Terminate subroutine.

Use a meaningful name for each object that you create, so you can easily recognize the object from the name itself. In all the samples that are shown in this chapter, we use the following naming conventions.

- The Connection object is cnTCPASM05.
- The Command objects are cm_APILIB_PARTS or cm_APILIB_SPROC2.
- The RecordSet objects are rs_APILIB_PARTSDATA or rs_APILIB_SPROC2.
The following code shows how we declare and instantiate these objects.

```vbnet
' Declare and instantiate the Connection object
Public cnTCPASM05 As New ADODB.Connection
' declare and instantiate the Command object
Public cm_APILIB_PARTS As New ADODB.Command
Public cm_APILIB_SPROC2 As New ADODB.Command
' declare the Recordset object
Public rs_APILIB_SPROC2 As ADODB.Recordset
Public rs_APILIB_PARTSDATA As ADODB.Recordset
```

In each of the DA400Links.CLS module we have a Class_Terminate subroutine that we use to de-reference and destroy the objects that we created globally. The following code shows how we destroy global objects.

```vbnet
Private Sub Class_Terminate()
    Set rs_APILIB_PARTS = Nothing
    Set rs_APILIB_SPROC2 = Nothing
    Set cm_APILIB_SPROC2 = Nothing
    Set cm_APILIB_PARTS = Nothing
    If Not cnTCPASM05 Is Nothing then cnTCPASM05.Close
End Sub
```

## 2.4 Client Access OLE DB Functions

In this section we discuss all of the AS/400 functions that the Client Access OLE DB provider supports.

The Visual Basic programming examples discussed in this section allow retrieval of a single record or of all the records from a PARTS file on the AS/400 system. Some examples allow update to a particular record.

**Tip**

For additional information about programming with ADO, see the *SDK Technical Reference* manual and the Microsoft online help. See Appendix A.2, "Other Documentation and Resources" on page 373 for details on how to obtain this documentation.

The PARTS database file in library APILIB is defined as:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Description</th>
<th>Length</th>
<th>Decimals</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARTNO</td>
<td>Part Number</td>
<td>5</td>
<td>0</td>
<td>Zoned</td>
</tr>
<tr>
<td>PARTDS</td>
<td>Part Description</td>
<td>25</td>
<td></td>
<td>Char</td>
</tr>
<tr>
<td>PARTQY</td>
<td>Quantity</td>
<td>5</td>
<td>0</td>
<td>Packed</td>
</tr>
<tr>
<td>PARTPR</td>
<td>Price</td>
<td>6</td>
<td>2</td>
<td>Packed</td>
</tr>
<tr>
<td>PARTDT</td>
<td>Part Shipment Date</td>
<td>10</td>
<td></td>
<td>Date</td>
</tr>
</tbody>
</table>
All the examples discussed in this chapter are available for download on the http://www.redbooks.com web-site. See Appendix A.1, “Downloading the Samples” on page 373 for details.

2.4.1 SQL Statements

Using the Client Access OLE DB provider, you can send SQL statements to the AS/400 system for processing. Executing most SQL statements on the AS/400 system is supported. We assume that you already have the necessary background for writing SQL statements. If you need additional information about AS/400 SQL and SQL statements refer to the DB2 for AS/400 SQL Programming Manual, SC41-5611.

The SDK does not include a Visual Basic Wizard for SQL statements. In this section we skip any discussion that involves the VB Wizard.

![Diagram of SQL Application](image)

Figure 6. Sample ADO SQL Application

This sample ADO application sends SQL statements to the AS/400 system. This sample program provides two functions:

- It returns a record with a matching part number from the PARTS database.
- It returns information about all the parts in the PARTS database.

When you run the Visual Basic Application, you see a Window like the one shown in Figure 6 on page 47. The interface presents several options.

- If you press the Get Part button, it will display a record that matches the part number in 'Part Number' text box.
- The "Get All Parts" option returns all the records from Parts database.
- Cancel exits the application.

The 'Start Time' reflects that the time query is sent and the 'End Time' reflects when a record or all the records are returned to the application.
2.4.2 Opening a Connection

The first step is to define a connection object. You need to define at least one Connection object in an application.

First, you create the connection object. After creating the connection object, you invoke its open method. The ConnectionString part contains the provider and data source names. The Data Source name must be the name of a valid AS/400 connection. For example, the name of a connection configured using the Client Access AS/400 connection program.

Empty strings are passed for the User ID and Password parts. The IBMDA400 provider does not support the passing of User ID and Password when you open a connection. Client Access prompts the user for connection information as it is needed. The Third statement sets the ActiveConnection property for the Command object. This indicates to ADO which Connection object to use for this command. There can be more than one Connection object active at any time in your program.

Next, we discuss the code that supports this application. This code is executed when the 'Get All Parts’ button is pressed.

---

Public cnTCPASM05 As New adodb.Connection
    cnTCPASM05.Open "Provider=IBMDA400;Data Source=cnTCPASM05;", "", ""
    Set cm_APILIB_PARTS.ActiveConnection = cnTCPASM05

---

---

***Simple SQL***

    cm_APILIB_PARTS.CommandType = adCmdText
    Set rs_APILIB_PARTSDATA= cm_APILIB_PARTS.Execute
The CommandText property specifies the SQL statement that we want to execute on the AS/400 system. This statement can be as simple as selecting all the fields, or you may send a more complicated statement. Here we ask for all the fields in each row from the PARTS database file.

The CommandType property for SQL statements is always ‘adCmdText’.

The last statement executes the SQL statement on the AS/400 system and returns the result set to the rsAPILIB_PARTSDATA Recordset object. We can now display or process the Recordset data. We use a Visual Basic ListBox to display all the rows.

Our application pops up the following VB Listbox to display all the records returned from the AS/400 system in response to pressing the Get All Parts button.

![Figure 8. ADO - SQL VB Application - Get All Parts](image)

If the Get Part button is pressed, the application Window is updated with the record returned by the AS/400 system. When a matching part number is not found in the database a Record not found for the key value that you specified message is returned. If you type 12303 in Part Number text box, and then click on the Get Part button, the following Window is displayed.
When AS/400 system receives an SQL statement in the form of a Command object’s CommandText, it compiles the statement and executes it. After an execution the host normally discards the compiled statement. If you execute the same SQL statement (CommandText) over and over again without using prepared statements, valuable processing-time is wasted because the host must re-compile it each time before it is executed. An alternate option is to use a prepared statement. A prepared statement is compiled once, but it can be executed many times. The host keeps track of all of your prepared statements.

Parameter markers give you the ability to reuse the same prepared statements even when it might appear that you are running a different query. Parameter markers can be thought of as place holders. Suppose you are running a query:

```
SELECT * FROM MYLIB.MYFILE WHERE CUSTID=12345
```

Next time you change the CUSTID to 54321, you can send a new query. Better yet, use parameter markers and change the query to:

```
SELECT * FROM MYLIB.MYFILE WHERE CUSTID=?
```

Now you can use the same prepared statement by resetting the parameter value with a different CUSTID each time. This benefit becomes more apparent when you need to INSERT multiple rows into a table. Being able to reset the parameter value gives you flexibility in programming. If you need to reset the parameter after each time you call Execute, you can do that in a loop.

Let's now go back to our application. We set our CommandText to return all the fields from the PARTS database file for the specified PARTNO. The Value of PARTNO is known only at run time. So, instead of using actual values for the PARTNO we use parameter markers. Since the default behavior of Prepared property is False, we set the Prepared property value to True.
At run time we use a Variant array Parm1 to pass in the user parameter. A user enters a PARTNO in the Part Number textbox. When the Get Part button is pressed, the following code gets executed. The `txtPartNumber.Text` statement returns the value a user inputs in the Part Number text box. We execute the query using this parameter and return the result data to the `rs_APILIB_PARTDATA` Recordset object.

It is not a good idea to create an application without using some combination of prepared statements and parameter markers. Instead of using a Variant array to pass parameters you can manually create the parameter as well. But, in any case, after the subsequent Execute, you only reset the parameter and the host does not need to recompile the SQL statement.

Creating Parameters manually takes a little more effort. The following two statements can be used to create the parameters manually:

```vbnet
Set Parm1 = cm_APILIB_PARTS.CreateParameter("P1", adChar, adInput, 5)
Set Parm1 = cm_APILIB_PARTS.Append Parm1
Set rs_APILIB_PARTDATA = cmdAPILIB_PARTS.Execute(NumRecs, Parm1, -1)
```

The `CreateParameter` method creates a parameter object with the specified properties. In the above example, `P1` is the name of the parameter object, `adChar` is the parameter data type, `adInput` indicates the direction type of the parameter (it will send to host), 5 is the length of the parameter. The `Append` method adds the Parameter Object to the Parameter collection. The `Append` method is called only after you specify the parameter data type and size (or length). Note you need to invoke the `CreateParameter` and `Append` method only once for each of the parameter objects in your program. We create a Variant array which returns the value entered in the Part Number text box the same way as we did in our previous
example. Then we execute the query using this parameter and return result data to the rs_APILIB_PARTDATA object.

The above sample demonstrates that programming in ADO is very simple. The ability to write code quickly and efficiently is one of main reasons to use ADO.

We use *SQL naming convention in this ADO SQL application. This is the only naming convention IBMDA400 allows for SQL requests. When you do not specify a library name in the SQL statement (such as SELECT * FROM MYFILE), you may get an SQL error. This is because IBMDA400 does not implement the DefaultDatabase property of the Connection object. The Host lookup for MYTABLE is thus limited to the USER library (USER is the same as your user ID that you used when signing on to the AS/400 system). SQL cannot use the job’s library list. The query fails when the USER library does not exist, MYFILE does not exist in USER library or when a matching record is not found.

**Tip**

SQL always returns a Static Cursor. So any changes you make to the data in the returned Recordset object are not reflected in the AS/400 file.

IBMDA400 uses a static cursor for ADO SQL statements. You cannot use the AddNew, Update, Remove method on the Recordset object and update the Table on the AS/400. Any modification to the AS/400 logical or physical file data must be done through SQL statements.

### 2.4.2.1 Restrictions

When using SQL functions with IBMDA400, you cannot use Commitment control. Commitment control is available only when using Record Level Access. If this is a requirement, consider using the MSDASQL provider that is the Microsoft OLE DB provider for ODBC. This allows you to use the Client Access ODBC driver for your ADO application. You have to configure a Client Access ODBC Data Source for this to work. MSDASQL may also be your choice if you want to run SQL statements from Active Server Pages using the Microsoft Internet Information Server. This requires that the provider run as a service. The Client Access ODBC driver can be configured to run with a service. For additional information on Active Server Pages refer to Chapter 10, “Internet-Based Applications” on page 303.

### 2.4.3 Stored Procedures Calls

When using the Client Access OLE DB provider, you can call AS/400 Stored Procedures. A Stored Procedure is a program on the AS/400 system that has been cataloged. You can write this program in any programming languages that is available to AS/400 - RPG, ILE C, COBOL, CL and so on. The program is first compiled, then cataloged. The SQL CREATE PROCEDURE statement catalogs the Stored Procedure. You need to call the CREATE PROCEDURE only once, or as long as the parameter types, and numbers do not change. A good way to do this is by using interactive SQL (strsql). You can also use the SDK Visual Basic wizard to create or drop an AS/400 stored procedure.

Stored Procedure can be used to improve the performance and security of client/server applications. The main reason for improved performance is that Stored Procedure greatly reduces communication flow between the Client and the Server. Most, if not all, of the database logic is contained in the AS/400 program. This may also be vital from a security point-of-view. You are not
exposing your database structure, file names as well as your application’s communication flow to a casual network sniffer (that is a hacker), who may not have direct access to this information.

The general syntax CommandText for a Stored Procedure call from ADO is:

{CALL <stored procedure> [<parameters>]}

The braces around the command are required. It indicates to IBM DA400 that this is a Stored Procedure call. Note that parameters are optional, it depends on your program. A Stored Procedure supports input, output and input/output parameters, as well as result sets. You may have up to 256 parameters in your Stored Procedure. You may return data from your Stored procedure in an output parameter (adParamOutput or adParamInOutOut type) or you may use a SQL result set. It is important to point out that only single Result sets are usable from ADO, multiple result sets are not supported. If your program returns multiple result sets, only the first one is available to you from ADO. For additional information on SQL statements and Stored Procedures, refer to the DB2 for AS/400 SQL Programming Manual, SC41-5611.

Tip

IBM DA400 does not support multiple result sets. You can call a stored procedure with multiple result sets, but only the first result set can be processed.

If your Stored Procedure is expecting parameters you may either hard code them in your ADO application or supply them dynamically during run time. If you decide to dynamically pass the parameters, use prepared statements and parameter markers in your program.

Figure 10. ADO Stored Procedure Sample

In the this example "APILIB.SPROC2" is called. This Stored Procedure takes two parameters - P1 and P2. The first Parameter, P1, may have a value of 1 or 2. When P1 is 1, P2 contains a part number. In this case, the Stored Procedure returns one record from the PARTS database. This record matches the part
number contained in P2. If a matching record is not found an error is returned. When P1 is 2, P2 contains 0. For this SPROC2 returns all records from the PARTS database. All records are returned in a SQL Result Set.

In this case, the name of the stored procedure and the name of the AS/400 program are the same, SPROC2. They could be different. For a listing of the RPG program, SPROC2, see Appendix B.2, “SPROC2/SQLRPGLE” on page 375.

When you run the Visual Basic Application, you see a Window like the one that follows. The interface presents several options. If you press the Get Part button, it displays a record for the key in ‘Part Number’ box. ‘Get All Parts’ returns all the records for Parts database and the Cancel button exits the application. The ‘Start Time’ reflects the time query is sent and ‘End Time’ reflects when a record or all the records are returned to the application.

![Figure 11. ADO - Stored Procedure VB Application](image)

### 2.4.3.1 Using the Visual Basic Wizard

The Visual Basic Wizard contained in the SDK gives you three options to manipulate Stored Procedures.

1. Work with Stored procedure...
2. Link Stored Procedure...
3. Create Form from Links...

Figure 12 displays all the Wizard options available to Visual Basic.
The 'Work with Stored Procedure' option provides wizard support to catalog a Stored Procedure. Assuming that you have compiled a program (SQLRPG, SQLCL and so on) on the AS/400 system, using this option allows you to create a Stored Procedure. We mentioned before that you can use interactive SQL or any comparable tool to create a SQL Stored Procedure on the AS/400 system.

Before you invoke this menu option create the AS/400 program. You can use the CRTSQLXXX command to create the SQL XXX Program. Run this command from an AS/400 command line. XXX can be RPG, CL, and so on. Since we are using a SQLRPG program for our Stored Procedure we create the SQLRPG program using this following command:

```
CRTSQLRPG PGM(APILIB/SPROC2) SRCFILE(APILIB/QRPGSRC) COMMIT(*NONE) OPTION(*SQL)
```

Here the PGM parameter specifies the name of program object when compiled. You must specify *NONE for COMMIT option. Otherwise you may have an unwanted lock on your Table. OPTION(*SQL) specifies the naming convention used in our SQLRPG program. We use *SQL to be consistent.

Once the program compiles without any error we select AS/400 SDK -> Work with Stored procedures... from the Add-Ins menu. In the resulting dialog box highlight the AS/400 system name (TCPASM05 in our case) and press the Create button. We now have the windows that is displayed on Figure 13 on page 56.
Figure 13. Specify the AS/400 Program Option

Here we specify Program Name (SPROC2), Library Name (APILIB), Language RPG and Parameter Style for Calling program General. When we press the Next button, it displays the dialog box on Figure 14. In this dialog box we specify the Stored Procedure Name and the Library name.

Figure 14. Specify the Stored Procedure Name and Return Type

Our Stored Procedure name is SPROC2 and Library Name is APILIB. Since SPROC2 returns one result set, we check the Results check-box and specify 1 for the number of result sets. SPROC2 always returns one result set, so we check Same for
Results Returned. Pressing **Next** displays the dialog box shown in the Figure 15 on page 57.

![Create a Stored Procedure](image)

*Figure 15. Specify the Parameter Type*

Here we specify the Parameter name, Type, Data Type, Length and Decimals. You can specify any identifier for the Parameter Name. We use P1 to indicate the first parameter and P2 for the second parameter. Since both parameters are input and output capable, we specify Input/Output for Type. For Data Type we specify INTEGER for both parameters. For length we specify 1 and 5 for P1 and P2 respectively. Finally, for decimal places we specify 0 for both parameters. When creating a Stored Procedure these values must correspond to the AS/400 program that you are using for the Stored Procedure.

When we click on the **Finish** button, the Stored Procedure is created. Note that you may create a Stored Procedure only once. If for some reason a Stored Procedure needs to be created again, you must run the SQL DROP PROCEDURE to remove the Stored Procedure from the catalog. You can also drop a stored procedure from the Visual Basic wizard.

Next we discuss the 'Link Stored Procedure' option of the Wizard. This option generates the code that are required to invoke a call to a Stored procedure. Select **AS/400 SDK -> Link Stored Procedures...** from the Add-Ins menu. You get a dialog box similar to the one on Figure 16 on page 58. Click on the **Add** button.
You are shown a dialog box similar to Figure 17 on page 58. Here we select SPROC2 under APILIB from TCPASM05. We created the SPROC2 Stored procedure using the Wizard in earlier steps. Note that this information may vary in your real-life application. Click on Next.

You should now have a dialog box similar to Figure 18 on page 59. In this dialog box you may specify a default parameter value. Note that default parameter value is optional. In our case we specify a default parameter value of 2 for P1. This indicates to SPROC2 to return all records from the PARTS database.
We click on **Next** and we are presented with a dialog box similar to Figure 19 on page 59. We select **Yes**, because **SPROC2** return results.

Click on **Finish**. This creates a link to **SPROC2** on the AS/400 system. Creating a link in the context of the VB Wizard means generating code that will invoke a call to this Stored Procedure. This code is then placed in the DA400Links.CLS class object and is executed during the class initialization.

We discuss Create Form from Links next. Before we can take this option we must create a link to the Stored Procedure. Taking this option adds the code required
to navigate and display the Recordset. We can display the Recordset using one of these controls - VB Grid, List, ListBox or a ComboBox.

![Image of VB Wizard - Stored Procedure - Specify Control Type](image1)

**Figure 20. VB Wizard - Stored Procedure - Specify Control Type**

When you select **AS/400 SDK -> Create Form from Links...** from the Add-Ins menu, you see a dialog box similar to Figure 21 on page 60. From the Control Type drop-list box, you can choose any of the control types that are displayed there. We chose ListBox in our example. Click on **Next**. We specify **All Parts** for title.

![Image of VB Wizard - Stored Procedure - Specify Data Control Title](image2)

**Figure 21. VB Wizard - Stored Procedure - Specify Data Control Title**

We click on **Finish**. Now we have added the code to display the data in the Recordset object. This data is returned by the Stored procedure call.
Next we save the project. We call this new form Form2. We add code to display this form. We accomplish this by adding a command button on Form1. When pressed it loads Form2 which contains a VB Listbox that displays all records that are on the Recordset object. The code is:

Form2.show 1

We have used the SDK Visual Basic wizard to generate much of the application. Let’s look at what it generated.

Initialization

```vbnet
Public cnTCPASM05 As New ADODB.Connection
Public cmAPILIB_SPROC2 As New ADODB.Command
Public rsAPILIB_SPROC2 As ADODB.Recordset

cnTCPASM05.Open "Provider=IBMDA400;Data Source=TCPASM05;", "", ""
Set cm_APILIB_SPROC2.ActiveConnection = cnTCPASM05
cm_APILIB_SPROC2.CommandText = "CALL APILIB.SPROC2(?,?)"
cm_APILIB_SPROC2.Prepared = True
cm_APILIB_SPROC2.Parameters(0).Direction = adParamInputOutput
cm_APILIB_SPROC2.Parameters(1).Direction = adParamInputOutput
```

The wizard first created a connection object, a command object and a Recordset object. We open the connection object specifying IBMDA400 as the OLE DB Provider and the name of the connection to the AS/400 system as the Data Source. In the next statement we specify the CommandText, following by the syntax we mentioned before, needed for a Stored Procedure call. SPROC2 takes two parameters and we want to pass these parameters dynamically, so we use question marks as place holders for these two parameters. The braces around the statement are required.

In the following statement we set the Prepared property to True. Since we are using parameter markers in our Stored Procedure, this is required.

Each parameter may have one of three valid values for Direction - adParamInput, adParamOutput or adParamInOut.

- adParamInput is used to send input to the Stored Procedure. The value is lost once a call is made to the Stored Procedure.
- adParamOutput is used to hold output values returned from a Stored Procedure call. The Stored Procedure must reset this parameter once a call is made to it.
- Option adParamInOut is used when the same parameter is used to send an input value to the Stored Procedure and receive a return value (output) from it. The parameter value may be optionally reset by the Stored Procedure, otherwise it contains the program supplied value.

In this example we use adParamInOut type for both parameters. All this code was generated by the SDK Visual Basic wizard.

We also used the wizard to generate the code to display all the parts from the PARTS database. In order to retrieve all the parts, we pass the AS/400 stored procedure program a "2" in the first parameter. It will return all the parts in an SQL
result set. We use the wizard to tie this processing to the loading of Form2. The only code that we add is to show Form2 when the Get All Parts button is pressed.

```
Form Load
Private Sub Form_Load()
   Dim Rcds As Variant
   Dim Parms As Variant
   Set Links = Form1.Links
   Parms = Array(2, 0)
   Set Links.rsAPILIB_SPROC2 = Nothing
   Set Links.rsAPILIB_SPROC2 = Links.cmAPILIB_SPROC2.Execute(Rcds, Parms, adCmdText)
   If Links.rsAPILIB_SPROC2.EOF And Not Links.rsAPILIB_SPROC2.BOF Then
      Links.rsAPILIB_SPROC2.MoveFirst
      Links.LoadListViewHdgs ListView1, Links.rsAPILIB_SPROC2
      Links.LoadListView ListView1, Links.rsAPILIB_SPROC2
   End Sub
End Sub
```

We use a Variant array to pass the parameters during execution. Since the first parameter value is 2, after a successful execution the Recordset object contains all the records that the Stored Procedure returned in the form of an SQL result set. Below is the output displayed when the Get All Parts button is pressed.

![Figure 22. ADO - Stored Procedure VB Application - Get All Parts](image)

**2.4.3.2 Adding the Get Part Support**

In this section, we complete the application. We add the code to run when the user presses the Get Part button.
If we enter 12301 for part number and press the Get Part button, the P1 value is 1 and P2 is set to the part number. So, the variant array PARMS now looks like this:

`Parms=Array(1, 12301).`

When a matching part number is not found, the EOF property of the Recordset is True. This indicates to you that the part number was not found. You may now display error or prompt for a valid part number. For a valid part number the Window is populated with the record.

![Lightning Stored Procedure Example](image)

*Figure 23. ADO - Stored Procedure VB Application - Get a Part*

### 2.4.4 Record-Level Access

The AS/400 OLE DB provider allows you to access AS/400 logical and physical files using the DDM server. You can have record level access to AS/400 files. When you open a file, you have the option of specifying a Database file member

```vba
Private Sub Command2_Click()
    Dim Rcds As Variant
    Dim Parms As Variant
    Dim PartNumber As Long
    PartNumber = Val(txtPartNumber)
    Parms = Array(1, PartNumber)
    Set Links.rsAPILIB_SPROC2 =
        Links.cmAPILIB_SPROC2.Execute(Rcds, Parms, -1)
    If Not Links.rsAPILIB_SPROC2.EOF Then
        txtPartNumber.Text = Links.rsAPILIB_SPROC2.Fields(0).Value
        txtDescription.Text = Links.rsAPILIB_SPROC2.Fields(1).Value
        txtQuantity.Text = Links.rsAPILIB_SPROC2.Fields(2).Value
        txtPrice.Text = Format(Links.rsAPILIB_SPROC2.Fields(3).Value, _
            "currency")
        txtReceiveDate.Text = Links.rsAPILIB_SPROC2.Fields(4).Value
    Else
        "part not found"
    End If
End Sub
```
name and record format. You can also specify file-level commitment control, which overrides the IsolationLevel property of the ADO Connection object. In our earlier discussion we mentioned that you cannot specify commitment control when using the SQL interface from ADO.

The SDK includes a Visual Basic Wizard for Record Level Access. If you are using Visual Basic, this wizard could be beneficial. It will give you a good start in developing your application. For Tables Record Level Access we can use IFS or *SYS naming convention.

The general syntax for IFS:
/QSYS.LIB/<library>.LIB/<file>.FILE[/<member>.MBR] [{<record format>, <commitment control}>]

The general syntax for *SYS naming:
[<library>/]<file>({<member>, <record format>, <commitment control>})

Member name, record format, and commitment control are optional values. But the left and right parenthesis (’,’) are required, even when it contains no values for member name, record format and commitment control.

Our Visual Basic sample application accesses the PARTS database using the DDM server. It either requests for all records or one record from this file. It can also update, delete or insert a record into the file.

![Diagram](image)

Figure 24. ADO- DDM Record Level Access Sample

When you run the Visual Basic Application, you see a window like the one that follows. The interface presents several options. When you press the Get Part button, it displays a record for the key in Part Number box. The Get All Parts option returns all the records from PARTS database and the Cancel button exits the application. The Start Time reflects the time request is sent to the AS/400 system and End Time reflects when a record or all the records are returned to the application.
When we press the Get Part button, we are prompted for a Part number. We enter a part number. After that, we either see a record that matches the part number we specified, or we get an error when a matching part number cannot be found in PARTS database file. The following code gets executed when we start the application. Refer to Section 2.4.4.3, “Using the Visual Basic Wizard” on page 72 to see how to use a Wizard to generate the code.

We invoke the `Open` method of the `Connection` object. In `ConnectionString` we specify the Provider name as well as the Data Source name using `Provider=` and `Data Source=` keywords. Data Source name is the name of the AS/400 system as defined in the AS400 Connection program in Client Access. We are already familiar with this statement because we use it in all of our previous examples.
In the next statement we invoke the Open method of the ixTCPASM05 object. This object is new to us; we have not used it in any of our previous examples. We need to carefully examine this new object. Unlike cnTCPASM05, ixTCPASM05 is an AD400 Connection object. (cnTCPASM05 is an ADODB Connection object.)

ADO 1.0, which IBMDA400 implements, does not have a Find or Seek method for a Recordset object. Without a Find or Seek method, key based record search is impossible. The AS/400 OLE DB developers had to create additional Automation Objects to implement this support. AD400 is a custom OLE Automation Object that implements index search capability. In our program we use the AD400 Connection object, as well as the AD400 Index object. Whether you are programming in VB or any other programming environment, for Keyed record search you have to use AD400 objects.

When we invoke the Open method for the ixTCPASM05 object, we only need to specify the AS/400 system name. Here you cannot use the "Provider=" keyword and specify "IBMDA400" or use "Data Source=" keyword and specify the AS/400 system name.

We then set cnTCPASM05 as the ActiveConnection object for the cm_APIILIB_PARTS. It is very important to remember that for the Command ActiveConnection is always the ADODB.Connection object. This Command is never associated with the AD400.Connection object.

The next statement specifies the Update mode of the Recordset object. We will skip discussing what it means for now, but we will come back to it shortly. The next statement sets the Command Text using the IFS naming convention. Here we specify the file name PARTS in the APIILIB library. We specify *FIRST for member and *NONE for the Commitment Control level. The next statement invokes the Execute method. After this statement is executed the table is open and we have a Recordset object rs_APIILIB_PARTS containing all the records from the PARTS database file.

Next we invoke the OpenIndex method of the AD400 Connection object ixTCPASM05. The general syntax for the index object is following:

```
Set <AD400 index>=<AD400 connection>.OpenIndex "<filename>", "<indexname>"
```

The OpenIndex method takes two parameters - Filename and indexname. Both parameters refer to the same AS/400 object that is the physical/logical file, that you open. This method returns an AD400 Index object. We use /QSYS.LIB/APIILIB.LIB/PARTS.FILE() for the file and QSYS.LIB/APIILIB.LIB/PARTS.FILE() for the index. Note that filename and index name both must point to the same AS/400 logical or physical file. After a successful execution an AD400 Index object is returned to ix_APIILIB_PARTS.

### Tables - Find a Record

```vba
Key1 = InputBox("Enter value for key PARTNO")
Keys = Array(Key1)
BookMark = ix_APIILIB_PARTS.GetBookmark(Keys, ad400SeekFirstEQ)
rs_APIILIB_PARTS.Move 0, BookMark
```

To find a record in the PARTS database, we use the GetBookmark method of the index Object. The GetBookmark method takes two parameters: Keys and SeekOption. The Keys value is a variant array of key field values that include all,
some, or at least one of the key fields of the file. SeekOption indicates criteria for the search. This method returns a Bookmark, which the ADO Recordset object can use to position the cursor.

If you have multiple fields in an index or key you must pass the key values in the same order as they are defined in the AS/400 logical or physical file. The IBMDA400 provider accepts a partial value only for the last field value of a key. For example, assume your key consists of two fields Fld1 and Fld2. If you pass "IBM" for Fld1, a single character "A" for Fld2 and use ad400SeekFirstEQ for the SeekOption, the GetBookMark operation will return the first record that has the Fld1 value "IBM" and the first character of Fld2 "A". In this case, the value for Fld1 would be padded, while the value for Fld2 would not be padded.

The SeekOption can be:

- ad400SeekFirstEQ (first value in the table equal to the specified value)
- ad400SeekGE (first value in the table greater than or equal to the specified value)
- ad400SeekGT (first value in the table greater than the specified value)
- ad400SeekLastEQ (last value in the table equal to the specified value)
- ad400SeekLE (first value in the table less than or equal to the specified value)
- ad400SeekLT (first value in the table less than the specified value).

In our application SeekOption is ad400SeekFirstEQ and the Keys value is set to the value entered when prompted, for example 12303. Since this record does exist in the PARTS database file a Bookmark is returned.

We then use the Move method to navigate through the Recordset. The ADO Move method takes two parameters; number of records and the Bookmark. Since the Bookmark identifies the current record in the Recordset, any non zero values for number of records shifts the current record position. Assuming the bookmark to be the starting position, any positive number of records moves the current record positioning in the forward direction and any negative number of records moves the current record in the backward direction. Any move is relative to the Bookmark value. Since we just want to display the current record, we use 0 for number of records.

In our code example we show the following window if a record matching the key value is found.
Let's summarize how Keyed record search works:

1. Create ADODB Connection, Command and Recordset objects.
2. Create AD400 Connection and Index objects.
3. Open the ADODB and AD400 Connection Objects.
4. Set the ActiveConnection to the ADODB Connection.
5. Open the table using the ADODB Command object. The Command type is adCmdTable. The records are returned in an ADODB Recordset.
6. Use the OpenIndex method of the AD400 Connection object to set the index to the AD400 Index object.
7. Use the GetBookmark for the AD400 Index object.
8. Position the Recordset using the bookmark.

Now that you understand how the IBMDA400 compensates the ‘missing’ Find method in ADO 1.0, we are ready to move on to discuss how to add, remove and update records. These functions are accomplished by invoking the AddNew, Remove and Update methods of the ADO Recordset object.

**Tip**

Files are always opened with a dynamic cursor for record-level access. Any update, delete or insert operations are immediately reflected.

AddNew takes two parameters, a List of Fields and a list of Values for the fields. If you are using an Array to supply the list of fields, you must use an Array for the Values. In our sample we use a Variant Array to pass the values. When using the AddNew method you do not need to specify all the fields of the file. Some additional considerations like default field values, null capable fields, padding, and so on, may apply in a particular circumstance.

```vbc
Flds = Array("PARTNO", "PARTDS", "PARTQY", "PARTPR", "PARTDT")
rs_APILIB_PARTS.AddNew Flds, Vals
```
In a Delete method call you need to pass the adAffectCurrent parameter or a value of 1. This indicates that the delete request is for the current record. The adAffectedGroup parameter, which removes a group of records using a Filter is not supported by IBMDA400.

```plaintext
***Delete Method***

rs_APILIB_PARTS.Delete adAffectCurrent
```

Update also takes a list of fields and a list of values to set. The changes are made to the current record only. When using Update you can pass the fields and values in an array or by using lists. This must be consistent with what you use to pass the parameters for the fields. Note that the BatchUpdate method is not supported by IBMDA400.

```plaintext
*** Update Method***

Flds = Array("PARTNO", "PARTDS", "PARTQY", "PARTPR", "PARTDT")
rs_APILIB_PARTS.Update Flds, Vals
```

### 2.4.4.1 Record Locking and Commitment Control

In the above section we discussed adding, removing or updating records without using any Commitment control. Now we discuss how you can add commitment control in your program. You may want to write your application in a way such that your data can be accessed from multiple locations but any modifications are serialized. That is, only one job modifies a record at a time. You may also want an option to rollback changes in case there is a problem.

“Updatability” is an IBMDA400 custom property of the ADO Command object. This property determines if the file is opened read-only or with update capability. If you use the default “Updatability” value, IBMDA400 opens the file for read-only and places a Shared Read lock on the file. To perform any Update, Delete or AddNew operations on a file, the “Updatability” property must be set accordingly. Note that this only applies if you opened the file using the `Command.Execute` method.

<table>
<thead>
<tr>
<th>Table Open Mode</th>
<th>Updatability value</th>
<th>Member Lock Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>0</td>
<td>Shared Read</td>
</tr>
<tr>
<td>Update</td>
<td>1</td>
<td>Shared Update</td>
</tr>
<tr>
<td>Delete</td>
<td>2</td>
<td>Shared Update</td>
</tr>
<tr>
<td>Add</td>
<td>4</td>
<td>Shared Update</td>
</tr>
</tbody>
</table>

The AS/400 database file member is locked shared read or shared update when it is opened, depending on the Updatability property setting. You cannot change the member lock type while the member is open. You must close the member and reopen it after setting a different value for Updatability.

If you open a file with a Updatability of read, the AS/400 database does not perform record locking. When the file is opened with an Updatability property other than read (for example, insert, update or delete), the AS/400 database always takes a pessimistic record-locking strategy. A record is locked for update as soon as you are positioned on that record, even though you have not made
any actual changes. This prevents others that have opened the file for update from positioning on that record.

In ADO programming, there is an option to use optimistic record locking when you open the file using the Recordset.Open method. You can specify the record lock type to be Optimistic at the Recordset level. The Recordset Open has the following syntax:

```vb
recordset.Open Source, ActiveConnection, CursorType, LockType, Options
```

```
cnTCPASM05.open "Provider=IBMDA400;Data Source=cnTCPASM05","",""
source1 = "/QSYS.LIB/APILIB.LIB/PARTS.FILE(*FIRST, *NONE)"
rs_APILIB_PARTS.Open
source1,cnTCPASM05,adOpenDynamic,adLockOptimistic,adCmdTable
```

**Note**

CursorType and LockType options in the Open event of a Recordset are not supported by the IBM OLE DB provider for AS/400. Please see SDK Technical Reference for details.

We are now ready to refer back to the statement that we skipped in our programming example of Tables Find Record:

```vbscript
cm_APILIB_PARTS.Properties("Updatability") = DBPROPVAL_UP_INSERT + DBPROPVAL_UP_DELETE + DBPROPVAL_UP_CHANGE
```

`DBPROPVAL_UP_INSERT`, `DB_PROPVAL_UP_DELETE` and `DBPROPVAL_UP_CHANGE` are constant values declared elsewhere in the program and, as you can imagine, they were set to 4, 2 and 1, respectively. This tells IBMDA400 not to block any update, delete or add operations for this Command object. Note that under the same Connection you may have different Command objects with varying “Updatability” capability.

Because a record is read using record-level access, you can specify only the field values that you want to change when you update the current record. You do not need to pass all of the record values when you update the current record.

Commitment level can be specified either at the connection level or from a Command object when you open the file. Commitment level is often referred to as the transaction Isolation Level. ADO Connection object’s IsolationLevel property may have any one of these following values:

<table>
<thead>
<tr>
<th>ADO IsolationLevel Property</th>
<th>AS/400 Commitment Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chaos (adXactChaos)</td>
<td>*NONE</td>
</tr>
<tr>
<td>Browse (adXactBrowse)</td>
<td>*CHG</td>
</tr>
<tr>
<td>Read Uncommitted (adXactReadUncommitted)</td>
<td>*CHG</td>
</tr>
<tr>
<td>Cursor Stability (adXactCursorStability)</td>
<td>*CS</td>
</tr>
<tr>
<td>Read Committed (adXactReadCommitted)</td>
<td>*CS</td>
</tr>
<tr>
<td>Repeatable Read (adXactRepeatableRead)</td>
<td>*ALL</td>
</tr>
</tbody>
</table>
The default value of the IsolationLevel property is adXactCursorStability or adXactReadCommitted. This is the DB2/400 equivalent of *CS. The following statement, for example, will set the IsolationLevel to *None when executed:

```plaintext
cnTCPASM05.IsolationLevel = adXactChaos
```

*adXactChaos* is equivalent to Decimal 16 or 0x10 in Hex notation.

You may also specify or override the IsolationLevel property value during the file open. Commitment level cannot be changed for a connection while any files are open under that connection.

On the AS/400 system, all AS/400 files that are being opened under commitment control require the use of journals. Create a journal and a journal receiver for all of the files that you open using the IsolationLevel property. If your files are not journaled when you open them by using Commitment control, an error is returned. You first create a journal receiver and then a journal, then journal any physical (or logical files) using following sets of commands:

```
CRTJRNRCV JRNRCV(APILIB/MYJRNRCV)
CRTJRN JRN(APILIB/MYJRN) JRNRCV(APILIB/MYJRNRCV)
STRJRNPF FILE(APILIB/PARTS) JRN(APILIB/MYJRN)
```

ADO does an automatic commit whenever a transaction is not explicitly started (by using BeginTrans). This causes a commit to be processed after every transaction so that all changes are automatically committed to the database files. This automatic commit is controlled by the OLE DB provider AutoCommit property and cannot be set from ADO. To enable the user to set the AutoCommit property, the IBMDA400 sets this property to match the last transaction’s isolation level. The IBMDA400 default value for the AutoCommit property is Chaos (*NONE).

The Commitment control for the DB2/400 database file is specified when the file is opened. Because ADO holds the isolation level from the OLE DB provider until it is needed in a BeginTrans, and since you will want to set the AutoCommit property to match the IsolationLevel property under which you will be running, you need to explicitly begin and end a transaction before opening any AS/400 files.

```
*** Commitment Control***
```

```plaintext
cnTCPASM05.IsolationLevel = adXactReadCommitted
cnTCPASM05.BeginTrans
cnTCPASM05.CommitTrans
On Error goto ProcessError
  'Open the File here
  cnTCPASM05.BeginTrans
  'Run update, delete or Insert
  ' if no errors were encountered during the transaction - commit changes
  cnTCPASM05.CommitTrans
ProcessError:
  'Display error messages here
  cnTCPASM05.RollbackTrans
```
Note that in the above code segment we perform a BeginTrans and a CommitTrans even before the file is opened. This allows IBMDA400 to pass the right commitment control mode, in this case adXactReadCommitted (or "CS") during the file open. If you examine the lock that is placed on the file after it is opened, you will find that it is set to "CS for the above example.

2.4.4.2 Restrictions

• The IBMDA400 provider does not support blocked reads of tables opened for record-level access. Reading multiple records at once does not work. You need to read the records one at a time from the AS/400 system. Consider using SQL SELECT *, or a call to an SQL stored procedure to get large recordset objects back to the PC. Once you have all of your data back, use record-level access support to do your individual record insert, update, and delete operations.

• If you are using record-level access and running over a TCP/IP connection to a pre-V4R2 AS/400 system, you will notice significantly improved performance with the new TCP/IP DDM server that is available in OS/400 V4R2. For additional information, refer to Section 2.1.2.1, “Record Level Access - Pre-V4R2M0” on page 41.

2.4.4.3 Using the Visual Basic Wizard

The Visual Basic Wizard contained in the SDK gives two options for Record-Level Access to AS/400 database files:

1. Link Tables...
2. Create Form from Links...

The Link Tables menu item generates record-level access code to existing AS/400 logical and physical database files. Each database file that you work with is called a Link.

If you choose AS/400 SDK -> Link Tables ... from the Add-Ins menu, you get a dialog box similar to Figure 27 on page 72. Click on the Add button to add a new link to a physical or logical file on the AS/400 system. You should now see a dialog box similar to Figure 27 on page 72.

Select PARTS for the AS/400 physical file, Check Inset, Update and Delete and *None for Commit Control. Click on Finish.
We have now created a link to the PARTS database file on the AS/400 system. Using similar steps you can create links to other AS/400 logical or physical files.

The VB Wizards next option, Create Form from Links, generates code for data display. From Add-Ins menu, choose AS/400 SDK -> Create Form from Links.... You get a similar dialog box to the one in Figure 29. Specify List Box for control type and click on Next.
Specify **All Parts** for the title on the next window and click on **Finish**.

Next, we add the code to display this form (the generated form was named Form2). When this form is loaded it displays all the records from the PARTS database. When the Get All Parts button is pressed, we execute the code:

```vbnet
Form2.show 1
```
2.4.5 AS/400 Data Queues

Using the Client Access OLE DB provider, you can interface with AS/400 data queues. The SDK Visual Basic wizard provides support to work with data queues. A data queue on the AS/400 is simply an AS/400 object that allows a program (host or client) to place data into it and retrieve data from it. When you open a data queue with IBMDA400 you can specify a record format for the data queue entries.

A data queue on the AS/400 system can be created with FIFO, LIFO, or keyed entry sequence. There is also a maximum entry length that is defined when the data queue is created. Typically, a client program places a request (entry) on an AS/400 data queue. Running on the AS/400 system, is a program that pull the request off of the data queue and does some work. In response, the AS/400 program might place some return data (entry) on the same or on another data queue from which the client program is reading.

To create a new AS/400 data queue, use the Create Data Queue (CRTDTAQ) CL command. You can also use the Visual Basic wizard to create or delete AS/400 data queues. Once a data queue is created on the AS/400 system, there is no easy method to retrieve the data queue properties. It requires a program that calls an AS/400 system API to retrieve the data queue properties. Although a bit complicated, you can use the DMPOBJ command and read the BINARY formatted output to get the property information about the data queue.

Before you can read or write to a data queue from ADO, it must be opened. You can use the CommandText to specify the command to open a data queue. Following is the syntax for this:

```
OPEN DATAQUEUE <data queue> [ [<record format>] [KEY(<key format>)] ] FOR <option>
```

Record format is an optional property. It has the following format:

```
<field name> <type> [CCSID <ccsid>]
```

Key format is also an optional field. It can only be used when the data queue is a keyed data queue. It contains a comma-separated list of field name, type and optional CCSID.

Option is either SEND or RECEIVE. To write to a data queue use the SEND option. To read from it use the RECEIVE option. Once you have opened a data queue, you can only use it for whatever option you specified on the Open. Thus, if a data queue requires both read and write operation, you should open it twice, once for sending and once for receiving.

Data queues with multiple record formats are not supported. The read operation performed is destructive. You must read the entire content at once. Reading single entries is not supported.

Note that you must use IFS naming or *SYS naming convention for data queues.

A data queue is always opened with a static cursor. Since the open performs a destructive read operation, the data queue is cleared as a result of the read. The Recordset object maintains a static copy of the record that was read from the
data queue. It may not reflect any subsequent changes to the data queue on the AS/400 system (made by another program).

Once a data queue is open you may read fields and current entry values from it. After an open for RECEIVE, all entries are read into the Recordset object, and you are positioned at the first entry. You can read the current entry field values by ordinal position or by field name. The Recordset is not updatable, and the AddNew, Update or Delete methods will return errors. To re-read any new data queue entries you can use the requery method of the Recordset object. All existing entries of the Recordset object will be deleted and replaced with the reread entries from the data queue. If there are no new entries on the AS/400 data queue, the resulting Recordset will be empty.

To write to the data queue it must be opened for SEND. The Recordset of a data queue opened for SEND is always empty. BOF and EOF returns true after the open. You can execute the AddNew method, but the Update and Delete methods will return errors.

When you close an open data queue the AS/400 object lock that exists on the data queue is freed. You can choose to reuse the Recordset object or release the object from memory.

Now that we have completed the minimum background needed to understand how data queues work, we are ready to examine our sample program. Besides our ADO client application, we have two data queues - DQINPT and DQOUPT and a RPG program named DQXRPG on the AS/400 system. We use the DQINPT data queue to send a 'signal' to our host program. This signal is a FLAG and a part number. When the FLAG is "A" (a request for all the parts), we do not send a part number. Otherwise we send a part number. The host program monitors the DQINPT data queue and depending on the flag or a part number it updates DQOUPT with a matching record for the part number or with all the records from PARTS database. The client ADO program then retrieves that information from the DQOUPT data queue. For a complete listing of the AS/400 RPG program, refer to Appendix B.4, “DQXRPG/RPGLE” on page 378.

![Figure 31. ADO Data Queue VB Sample Application](image)
When you run the Visual Basic Application, you will see a Window like the one that follows. The interface presents several options. If you press the Get Part button, it displays a record for the key in 'Part Number' box. The 'Get All Parts' button returns all the records for Parts database and the Cancel button ends the application. The 'Start Time' reflects the time the request is sent to the AS/400 system and 'End Time' reflects when the record or all the records were returned to the application.

![Figure 32. ADO - Data Queue VB Application](image)

Let's look at the code sample. Note that the Open method is as usual. You need to pass the Provider name IBMDA400 and Data Source name using the `Provider=` and `Data Source=` keyword in the `ConnectionString`.

```vba
Data Queue Open

    cnTCPASM05.Open "Provider=IBMDA400;Data Source=TCPASM05;", ",", ""
    Set rs_APILIB_DQINPT = cnTCPASM05.Execute("open dataqueue APILIB.DQINPT
    (option CHARACTER(1), part CHARACTER(5) ) for Send",
        Rcds, adCmdText)

    Set rs_APILIB_DQOUPT = cnTCPASM05.Execute("open dataqueue APILIB.DQOUPT
    (option CHARACTER(1), part CHARACTER(5), desc CHARACTER(25), qty
    DECIMAL(5,0), price DECIMAL(7,2), date CHARACTER(10) ) for Receive",
        Rcds, adCmdText)
```

Using this `Connection` object, we perform two Execute operations. The first `Execute` method sets the `rs_APILIB_DQINPT` recordset. It sends a OPEN DATA QUEUE with a for Send request to IBMDA400. We specify the Data Queue name APILIB.DQINPT using *SYS naming convention. We also specify the record format. The Option 'for Send' is used to indicate that this Data Queue is opened for write operation alone.

The next `Execute` operation opens the APILIB.DQOUPT Data Queue with a for Receive option. We set the `rs_APILIB_DQOUPT` Recordset to receive the output from this Data Queue. We also specify the record format of the Recordset object.
After these two statements are executed we have two Data Queues 'opened' - one for read and one for write operations.

---

### Get One Record

```vba
Flds = Array("option", "part")
Vals = Array("", TxtPartNumber.Text)
If Len(TxtPartNumber.Text) = 0 Then
    MsgBox "A Part# must be specified", vbOK, "Send Data Queue"
Else
    Links.rsAPILIB_DQINPT.AddNew Flds, Vals
    Links.rsAPILIB_DQOUPT.Requery
End If
```

To get one part back we need to write a part number to DQINPT. We set the Flag to a blank and the part field to the part number read from the screen. We accomplish this by using a Variant array Flds for the field names and a Variant array Vals for the values for these fields. We then call the AddNew method and pass these two Variant arrays to the method. This writes the entry into the APILIB_DQINPT Data Queue. At this point our host RPG program, which is monitoring this Data Queue, reads the request and accordingly places a record from the PARTS database into the APILIB.DQOUPT Data Queue. So, when we perform the Requery operation on the rsAPILIB_DQOUPT Recordset object it reads from the APILIB.DQOUPT Data Queue. Upon successful read, the record is displayed in a Window like the one that follows.

![Figure 33. ADO - Data Queue VB Application - Get Part](image)

To get all the parts we invoke the AddNew method in the rs_APILIB_DQINPT Recordset object and pass two Variant arrays to it, the Flds and Vals. Flds contain "Option" and "part". Vals contain "A" and "" since there is no part number required when the Option is "A". This causes IBMDA400 to write "A" for option in the DQINPT Data Queue. Our host application is still monitoring this Data Queue. Since it reads "A" for the option, it updates Data Queue DQOUPT with all the records from the PARTS database. When we invoke the Requery method on the DQOUPT data queue, it updates our rs_APILIB_DQOUPT Recordset with the
new entries from DQOUPT. Issuing a Requery method is necessary only when
rsAPILIB_DQOUPT.EOF returns True. This is important, because when we opened the
Data Queue DQOUPT with the 'for Receive' option we reset the rs_APILIB_DQOUPT
Recordset object. Thus, it is possible that rs_APILIB_DQOUPT will return False
and we can skip issuing a Requery.

```vbnet
Flds = Array("Option", "part")
Vals = Array("A", "")
rs_APILIB_DQINPT.AddNew Flds, Vals
If rs_APILIB_DQOUPT.EOF Then rsAPILIB_DQOUPT.Requery
' display the records that are returned here
```

The following Window is returned when the 'Get All Parts' button is pressed.

<table>
<thead>
<tr>
<th>option</th>
<th>part</th>
<th>desc</th>
<th>qty</th>
<th>price</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>11000</td>
<td>30 Video Card</td>
<td>11</td>
<td>200.25</td>
<td>1998-05-08</td>
</tr>
<tr>
<td>Y</td>
<td>12301</td>
<td>Quad speed ...</td>
<td>42</td>
<td>120.00</td>
<td>1996-01-12</td>
</tr>
<tr>
<td>Y</td>
<td>12302</td>
<td>SCSI II Cable</td>
<td>25</td>
<td>30.00</td>
<td>1995-11-13</td>
</tr>
<tr>
<td>Y</td>
<td>12303</td>
<td>17 inch SVG ...</td>
<td>42</td>
<td>350.00</td>
<td>1996-03-04</td>
</tr>
<tr>
<td>Y</td>
<td>12304</td>
<td>Ethernet PC ...</td>
<td>30</td>
<td>85.30</td>
<td>1995-12-17</td>
</tr>
<tr>
<td>Y</td>
<td>12305</td>
<td>Home mouse</td>
<td>47</td>
<td>25.50</td>
<td>1996-02-18</td>
</tr>
<tr>
<td>Y</td>
<td>12306</td>
<td>Gonder bendor</td>
<td>75</td>
<td>8.50</td>
<td>1951-08-27</td>
</tr>
<tr>
<td>Y</td>
<td>12307</td>
<td>800 dpi fatbe...</td>
<td>12</td>
<td>875.33</td>
<td>1996-03-01</td>
</tr>
<tr>
<td>Y</td>
<td>12308</td>
<td>100 MHZ Pe...</td>
<td>4</td>
<td>1875.20</td>
<td>1996-02-24</td>
</tr>
<tr>
<td>Y</td>
<td>12309</td>
<td>LasEdJet Toner</td>
<td>12</td>
<td>88.45</td>
<td>1995-12-17</td>
</tr>
<tr>
<td>Y</td>
<td>12310</td>
<td>Logo mouse ...</td>
<td>376</td>
<td>7.25</td>
<td>1994-11-24</td>
</tr>
<tr>
<td>Y</td>
<td>12311</td>
<td>Screen wipes</td>
<td>4750</td>
<td>1.50</td>
<td>1996-01-10</td>
</tr>
<tr>
<td>Y</td>
<td>12312</td>
<td>V34 Modem</td>
<td>58</td>
<td>120.45</td>
<td>1996-03-06</td>
</tr>
<tr>
<td>Y</td>
<td>12313</td>
<td>Games joystick</td>
<td>32</td>
<td>42.75</td>
<td>1995-11-12</td>
</tr>
<tr>
<td>Y</td>
<td>12314</td>
<td>3m printer ca...</td>
<td>20</td>
<td>12.40</td>
<td>1996-01-23</td>
</tr>
<tr>
<td>Y</td>
<td>12315</td>
<td>Antiglare scr...</td>
<td>45</td>
<td>34.77</td>
<td>1996-02-27</td>
</tr>
<tr>
<td>Y</td>
<td>12316</td>
<td>Quad speed ...</td>
<td>14</td>
<td>151.38</td>
<td>1996-01-12</td>
</tr>
<tr>
<td>Y</td>
<td>12317</td>
<td>SCSI II Cable</td>
<td>25</td>
<td>37.84</td>
<td>1995-11-13</td>
</tr>
<tr>
<td>Y</td>
<td>12318</td>
<td>17 inch SVG</td>
<td>6</td>
<td>1388.59</td>
<td>1996-03-04</td>
</tr>
</tbody>
</table>

```
To summarize this section:

- Open a Data Queue with 'For Open' or 'For Receive' option. The same Recordset object cannot be used for both Read and Write operation.
- Once opened, you perform AddNew to add entry to the Data Queue or Requery to read from a Data Queue.

2.4.6 Remote Commands and Distributed Program Calls

You can use the Client Access OLE DB provider to send commands to the AS/400 system or call AS/400 programs. IBMDA400 allows you to interface with existing AS/400 CL commands and programs. The SDK Visual Basic wizard provides support for working with commands and program calls. You can call an AS/400 command and pass command parameters. You can use the CommandText property to specify the command. The following syntax is used to invoke commands:

```
{{<command> <command parameters>}}
```

The two braces around the whole command are required. You can specify the command and the required parameters using this syntax. Command parameters values are optional. They are required depending on the CL command that you are invoking.

For commands, you may use IFS or *SYS naming convention. *SQL naming is not allowed.

We should mention that calling interactive CL commands are not supported. You can only call CL commands that can be called in batch mode. Also, you cannot prepare commands and pass substitution variables for parameters dynamically. You must concatenate the command and the parameter values.
together to form the complete command. When you send a CL command, IBMDA400 does not support a return parameter or a Recordset. For a complete list of AS/400 CL commands, refer to OS/400 CL Reference Manual, SC41-5722.

**Send a message**

```vbnet
cnTCPASM05.Open "Provider=IBMDA400;Data Source=tcpasm05;", ",", ""
Set cmDDM.ActiveConnection = cnTCPASM05
cmDDM.CommandText = "\"SNDMSG TOUSR(ALAM) MSG("Lightning is great")\""
cmDDM.CommandType = adCmdText
cmDDM.Execute adCmdText
```

In the above code sample we send a message from ADO using the **SNDMSG** command to an AS/400 user. We open the Connection object by specifying the provider and Data Source. Next we bind the Command object with this Connection object. We set the CommandText property with the exact syntax of the **SNDMSG** CL command. We set the CommandType to adCmdText and execute the command. Depending on whether that user is enrolled in the System Directory as an Office Vision user or as a MAPI Mail user (Lotus Mail, ccMail, Exchange and so on), the mail will be delivered to OV/400 or to a PC desktop.

Following this same method you may invoke similar CL commands. Remember that the only restriction that applies here is that the command cannot be an interactive command.

Besides the CL commands you may also invoke an AS/400 program using IBMDA400. You can pass input parameters and return output parameters from the program. However you cannot return a Recordset. You can prepare a call to the program and pass all the required parameters dynamically.

The general syntax of the program call is:

```
{{CALL <program> [(<program parameter>)]
```

Program parameters can be passed dynamically or statically. You can either hardcode all the required parameters in your call or you can use place holders and then pass the parameter values dynamically during run time. For a program call, a limit of 40 parameters applies. This restriction is imposed by the Distributed Program Call and Remote Procedure Call Server. Upon return from the program call, output parameter values are set.

The following is an illustration of how to use IBMDA400 to execute an AS/400 program. In this sample the client program calls an AS/400 RPG program named DPCXRPG with a parameter that is value of part number. This program queries the database file and returns details of the record using output parameter. For a complete listing of DPCXRPG, see Appendix B.3, “DPCXRPG/RPGLE” on page 377.
When you run the Visual Basic application, you see a window like the one that follows. The interface presents several options. If you press the **Get Part** button, it displays a record for the key in 'Part Number' box and **Cancel** button exits the application. Notice that there are no Get All Parts option here - since a program call does not return a Recordset. You can 'simulate' the ‘Get All Parts’ by calling the DPCXRPG program once for every record on the database - but this is expensive and we suggest you use SQL or Record-Level Access for this functionality. The ‘Start Time’ reflects the time the request is sent to the AS/400 system and ‘End Time’ reflects when the record or all the records were returned to the application.

Our Visual Basic program makes a call to a RPGLE program - DPCXRPG. This program takes two input parameter and return four output parameters.
The details of the RPG program DPCXRPG are as follows:

<table>
<thead>
<tr>
<th>Seq</th>
<th>Description</th>
<th>Length Type</th>
<th>I/O</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Action Flag</td>
<td>1 Character</td>
<td>I/O</td>
<td>S - Retrieve single record</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A - Position to the start of file</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F - Fetch the next record</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OUTPUT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y - Operation succeeded</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X - Operation failed or EOF found</td>
</tr>
<tr>
<td>2</td>
<td>Part Number</td>
<td>5.0 Packed</td>
<td>I/O</td>
<td>Part Number to be retrieved or retrieved</td>
</tr>
<tr>
<td>3</td>
<td>Description</td>
<td>25 Character</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Quantity</td>
<td>5.0 Packed</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Price</td>
<td>6.2 Packed</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Ship Date</td>
<td>10 Date</td>
<td>O</td>
<td></td>
</tr>
</tbody>
</table>

Let us now review the code sample that drives this application.

```vbnet
Public cnTCPASM05 As New ADODB.Connection
Public cm_APILIB_DPCXRPG As New ADODB.Command
Public cnTCPASM05.Open "Provider=IBMDA400;Data Source=TCPASM05;", "", ""
Set cm_APILIB_DPCXRPG.ActiveConnection = cnTCPASM05
cm_APILIB_DPCXRPG.CommandText = "'\{CALL /QSYS.LIB/APILIB.LIB/DPCXRPG.PGM(?,?,?,?,?,?)\}'"
Set cm_APILIB_DPCXRPG.Prepared = True
cm_APILIB_DPCXRPG.Parameters.Append cm_APILIB_DPCXRPG.CreateParameter("P1", adChar, adParamInputOutput, 1)
cm_APILIB_DPCXRPG.Parameters.Append cm_APILIB_DPCXRPG.CreateParameter("P2", adDecimal, adParamInputOutput)
cm_APILIB_DPCXRPG.Parameters(1).Precision = 5
cm_APILIB_DPCXRPG.Parameters(1).NumericScale = 0
cm_APILIB_DPCXRPG.Parameters.Append cm_APILIB_DPCXRPG.CreateParameter("P3", adChar, adParamOutput, 25)
cm_APILIB_DPCXRPG.Parameters.Append cm_APILIB_DPCXRPG.CreateParameter("P4", adDecimal, adParamOutput)
cm_APILIB_DPCXRPG.Parameters(3).Precision = 5
cm_APILIB_DPCXRPG.Parameters(3).NumericScale = 0
cm_APILIB_DPCXRPG.Parameters.Append cm_APILIB_DPCXRPG.CreateParameter("P5", adDecimal, adParamOutput)
cm_APILIB_DPCXRPG.Parameters(4).Precision = 6
cm_APILIB_DPCXRPG.Parameters(4).NumericScale = 2
cm_APILIB_DPCXRPG.Parameters.Append cm_APILIB_DPCXRPG.CreateParameter("P6", adChar, adParamOutput, 10)
```

We create a ADODB Connection object - cnTCPASM05. We use this same Connection object in all our sample programs in this chapter. We create a Command object cm_APILIB_DPCXRPG next. Notice that we do not have a Recordset object. In the next statement, we invoke the Open method of cnTCPASM05 Connection object. Note that the Open method is just as usual. You pass the Provider name IBMDA400 and
Data Source name using the Provider= and Data Source= keyword in the ConnectionString. We then set cnTCPASM05 to be the ActiveConnection for our Command object. This binds the Command object to the Connection object. This is required since you may have more than one Connection object active at any time.

Next we set the CommandText property of the Command object. We use the IFS naming convention. Note that the double braces around the command is required. It indicates to IBMDA400 that this is a program call. We have six parameters; two input/output and four output. Since we know the parameter values only at run time, we use question marks as place holders. On the next statement we set the Command Object’s Prepared property to True. It indicates that we are sending a prepared statement.

In the next few statements, we create and append each of the six parameters - P1 through P6. Data type for P1, P3 and P6 is adChar or Character type and the rest are adDecimal (Packed). We also specify the Parameter direction matching the information from Table 14 on page 83. We also set the precision and Numeric scale value for the adDecimal type parameters (P2, P4 and P5).

```vba
PartNumber = val(txtPartNumber)
Parms = Array("S", PartNumber)
cm_APILIB_DPCXRPG.Execute Rcds, Parms, adCmdText
```

When the Get Part button is pressed, we read the user input in the txtPartNumber textbox. In our sample-run this value is 12303. Next, we create a Variant array for all the parameters that need to be passed when calling the DPCXRPG program. Note that only P1 and P2 are input capable. The rest is output only. So, in our array we just need to pass P1 and P2.

Next we perform the Execute method. This causes the DPC/RPC server to run the DPCXRPG program with the specified parameters and send the output back to IBMDA400. Using this output, IBMDA400 resets parameters P3 through P6 with the values from the PARTS database. We display them on the screen.
2.4.6.1 Restrictions
User-defined parameters or parameters that are structures are not supported for Program calls.
Chapter 3. Overview of the Order Entry Application

In this chapter, we cover an example RPG order entry application. This application is representative of a commercial application, although it does not include all the necessary error handling a business application would require.

This section introduces the application and specifies the database layout. In Chapter 4, “Application Development Using AS/400 OLE DB Support” on page 103, we convert the RPG Order Entry application to an AS/400 client/server application which uses the Client Access OLE DB provider to access the AS/400 system. In other chapters in this redbook, we use the database described in this chapter to demonstrate AS/400 OLE DB support techniques and to analyze and measure performance of AS/400 client/server applications that use the AS/400 OLE DB support.

3.1 Overview of the Order Entry Application

This section provides an overview of the application and a description of how the application database is used.

3.1.1 The ABC Company

The ABC Company is a wholesale supplier with one warehouse and 10 sales districts. Each district serves 3000 customers (30,000 total customers for the company). The warehouse maintains stock for the 100,000 items sold by the Company.

The following diagram illustrates the company structure (warehouse, district, and customer).

![Figure 37. The Company Structure](image)

3.1.2 The ABC Company Database

The company runs its business with a database. This database is used in a mission critical, OLTP (Online Transaction Processing) environment.
The database includes tables with the following data:

- District information (next available order number, tax rate, and so on)
- Customer information (name, address, telephone number, and so on)
- Order information (date, time, shipper, and so on)
- Order line information (quantity, delivery date, and so on)
- Item information (name, price, item ID, and so on)
- Stock information (quantity in stock, warehouse ID, and so on)

3.1.3 A Customer Transaction

1. Customers telephone one of the 10 district centers to place an order.
2. The district customer service representative answers the telephone, gets the following information, and enters it into the application:
   - Customer number
   - Item numbers of the items the customer wants to order
   - The quantity required for each item
3. The customer service representative may prompt for a list of customers or a list of parts.
4. The application then:
   - Reads the customer last name, customer discount rate, and customer credit status from the Customer Table (CSTMR).
   - Reads the District Table for the next available district order number. The next available district order number is incremented by one and updated.
   - Reads the item names, item prices, and item data for each item ordered by the customer from the Item Table (ITEM).
   - Checks if the quantity of ordered items is in stock by reading the quantity in the Stock Table (STOCK).
5. When the order is accepted:
   - Inserts a new row into the Order Table to reflect the creation of the new order (ORDERS).
   - A new row is inserted into the Order Line Table to reflect each item in the order.
   - The quantity is reduced by the quantity ordered.
   - A message is written to a data queue to initiate order printing.

3.1.4 Application Flow

---

Note

To download the sample code used in this redbook, please refer to Appendix A.1, “Downloading the Samples” on page 373 for more information

---

The RPG Order Entry Application consists of the following components:

- ORDENTD - Parts Order Entry - display file
- ORDENTR - Parts Order Entry - main RPG processing program
• PRTORDERP - Parts Order Entry - print File
• PRTORDERR - Print Orders - RPG server job
• SLTCUSTD - Select Customer - display file
• SLTCUSTR - Select Customer - RPG SQL stored procedure
• SLTPARTD - Select Part - display file
• SLTPARTR - Select Part - RPG stored procedure

ORDENTR is the main RPG program. It is responsible for the main line processing. It calls two supporting RPG programs that are used to prompt for and select end-user input. They are SLTCUSTR which handles selecting a customer, and SLTPARTR which handles selecting part numbers. PRTORDERR is an RPG program that handles printing customer orders. It reads order records that were placed on a data queue and prints them in a background job.

3.1.5 Customer Transaction Flow

The following scenario steps through a customer transaction showing the application flow. Understanding the flow of the AS/400 application will assist in understanding the changes made to this application to support a graphical client.

3.1.5.1 Starting the Application

The application is started by calling the main program from an AS/400 command line.

CALL ORDENTR
When the order entry application is started, the following display is shown:

![AS/400 Interface](image)

Type choices, press Enter.  
2=Change  
Customer number . . . . . . . . . . . . . . Order number . . . . . . . . . . . . . .

F3=Exit  F4=Prompt  F6=Accept Order  F12=Cancel

Figure 39. Parts Order Entry

The user is expected to type in a customer number and press the Enter key, but they may choose to end the program by pressing either F3 or F12.

If the user does not know the customer number, F4 may be pressed to show a window containing a list of available customers.
The user may press **F12** to remove the window and return to the initial panel. They may roll through the items in the list until they find the customer they want. By typing a 1 in the option field and pressing the Enter key they indicate their choice and the selected customer is returned to the initial panel.
After selecting a customer from the list or typing a valid customer number and pressing the Enter key, the customer details are shown and an order number is assigned. An additional prompt is displayed allowing the user to type a part number and quantity.

If the user does not know the part number the F4 function key may be pressed to present a window containing a list of available parts.
The user may press F12 to remove the window and return to the initial panel. They may roll through the items in the list until they find the part they want. By typing a 1 in the option field and pressing the Enter key they indicate their choice and the selected part is returned to the initial panel.
After selecting a customer from the list or typing a valid customer number and pressing the **Enter** key, the part and quantity ordered are added to the list section below.

The user may type a **2** beside an entry in the list to change the order. Pressing the **Enter** key causes a window to appear allowing the order line to be changed.
The user may press **F12** to cancel the change, press **F4** to list parts, or type a new part identifier or different quantity. Pressing the **Enter** key causes the part identifier and quantity to be validated. If valid, the order line is changed in the list and the window is closed.

Here you see the quantity for *Zoo Season Pass* has been changed to 3. When the order is complete, the user may press F6 causing the database to be updated and an order to be placed on the data queue for printing.
The printed order is created by a batch process. It shows the customer details and the items, quantities and cost of the order.

### 3.1.6 Database Table Structure

The ABC Company database has eight tables:

- District
- Customer
- Order
- Order Line
- Item
- Stock
- Warehouse
- History

The relationship among these tables are shown in the following diagram:
3.1.7 Order Entry Application Database Layout

The sample application uses the following tables of the database:
- District
- Customer
- Order
- Order Line
- Stock
- Item (catalog)

The following sections describe in detail the layout of the database.
### 3.1.7.1 Tables

**Table 15. District Table Layout (Dstrct)**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Real Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>DID</td>
<td>District ID</td>
<td>Decimal</td>
<td>3</td>
</tr>
<tr>
<td>DWID</td>
<td>Warehouse ID</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>DNAME</td>
<td>District Name</td>
<td>Character</td>
<td>10</td>
</tr>
<tr>
<td>DADDR1</td>
<td>Address Line 1</td>
<td>Character</td>
<td>20</td>
</tr>
<tr>
<td>DADDR2</td>
<td>Address Line 2</td>
<td>Character</td>
<td>20</td>
</tr>
<tr>
<td>DCITY</td>
<td>City</td>
<td>Character</td>
<td>20</td>
</tr>
<tr>
<td>DSTATE</td>
<td>State</td>
<td>Character</td>
<td>10</td>
</tr>
<tr>
<td>DZID</td>
<td>Zip Code</td>
<td>Character</td>
<td>5</td>
</tr>
<tr>
<td>DTAX</td>
<td>Tax</td>
<td>Decimal</td>
<td>13</td>
</tr>
<tr>
<td>CYTD</td>
<td>Year to Date Balance</td>
<td>Decimal</td>
<td>9</td>
</tr>
<tr>
<td>DNXTOR</td>
<td>Next Order Number</td>
<td>Decimal</td>
<td>9</td>
</tr>
</tbody>
</table>

Primary Key: DID,DWID

**Table 16. Customer Table Layout (CSTMR)**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Real Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>CID</td>
<td>Customer ID</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>CDID</td>
<td>District ID</td>
<td>Decimal</td>
<td>3</td>
</tr>
<tr>
<td>CWID</td>
<td>Warehouse ID</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>CFIRST</td>
<td>First Name</td>
<td>Character</td>
<td>16</td>
</tr>
<tr>
<td>CINIT</td>
<td>Middle Initials</td>
<td>Character</td>
<td>2</td>
</tr>
<tr>
<td>CLAST</td>
<td>Last Name</td>
<td>Character</td>
<td>2</td>
</tr>
<tr>
<td>CADDR1</td>
<td>Address Line 1</td>
<td>Character</td>
<td>20</td>
</tr>
<tr>
<td>CADDR2</td>
<td>Address Line 2</td>
<td>Character</td>
<td>20</td>
</tr>
<tr>
<td>CDCT</td>
<td>Discount</td>
<td>Decimal</td>
<td>5</td>
</tr>
<tr>
<td>CCITY</td>
<td>City</td>
<td>Character</td>
<td>20</td>
</tr>
<tr>
<td>CSTATE</td>
<td>State</td>
<td>Character</td>
<td>2</td>
</tr>
<tr>
<td>CZIP</td>
<td>Zip Code</td>
<td>Character</td>
<td>10</td>
</tr>
<tr>
<td>CPHONE</td>
<td>Phone Number</td>
<td>Character</td>
<td>16</td>
</tr>
<tr>
<td>CBAL</td>
<td>Balance</td>
<td>Decimal</td>
<td>7</td>
</tr>
<tr>
<td>CCRDLM</td>
<td>Credit Limit</td>
<td>Decimal</td>
<td>7</td>
</tr>
<tr>
<td>CYTD</td>
<td>Year to Date</td>
<td>Decimal</td>
<td>13</td>
</tr>
<tr>
<td>CPAYCNT</td>
<td>Payment</td>
<td>Decimal</td>
<td>5</td>
</tr>
<tr>
<td>Field Name</td>
<td>Real Name</td>
<td>Type</td>
<td>Length</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------</td>
<td>------------</td>
<td>--------</td>
</tr>
<tr>
<td>CDELCNT</td>
<td>Delivery Qty</td>
<td>Decimal</td>
<td>5</td>
</tr>
<tr>
<td>CLTIME</td>
<td>Time of Last Order</td>
<td>Numeric</td>
<td>6</td>
</tr>
<tr>
<td>CDATA</td>
<td>Customer Information</td>
<td>Character</td>
<td>500</td>
</tr>
</tbody>
</table>

Primary Key: CID,CDID,CWID
### Table 17. Order Table Layout (ORDERS)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Real Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWID</td>
<td>Warehouse ID</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>ODID</td>
<td>District ID</td>
<td>Decimal</td>
<td>3</td>
</tr>
<tr>
<td>OCID</td>
<td>Customer ID</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>OID</td>
<td>Order ID</td>
<td>Decimal</td>
<td>9</td>
</tr>
<tr>
<td>OENTDT</td>
<td>Order Date</td>
<td>Numeric</td>
<td>8</td>
</tr>
<tr>
<td>OENTTM</td>
<td>Order Time</td>
<td>Numeric</td>
<td>6</td>
</tr>
<tr>
<td>OCARID</td>
<td>Carrier Number</td>
<td>Character</td>
<td>2</td>
</tr>
<tr>
<td>OLINES</td>
<td>Number of Order Lines</td>
<td>Decimal</td>
<td>3</td>
</tr>
<tr>
<td>OLOCAL</td>
<td>Local</td>
<td>Decimal</td>
<td>1</td>
</tr>
</tbody>
</table>

Primary Key: OWID, ODID, OID

### Table 18. Order Line Table Layout (ORDLIN)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Real Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>Order ID</td>
<td>Decimal</td>
<td>9</td>
</tr>
<tr>
<td>ODID</td>
<td>District ID</td>
<td>Decimal</td>
<td>3</td>
</tr>
<tr>
<td>OWID</td>
<td>Warehouse ID</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>ONLNBR</td>
<td>Order Line Number</td>
<td>Decimal</td>
<td>3</td>
</tr>
<tr>
<td>OLSPWH</td>
<td>Supply Warehouse</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>OLIID</td>
<td>Item ID</td>
<td>Character</td>
<td>6</td>
</tr>
<tr>
<td>OLMNT</td>
<td>Quantity Ordered</td>
<td>Numeric</td>
<td>3</td>
</tr>
<tr>
<td>OAMNT</td>
<td>Amount</td>
<td>Numeric</td>
<td>7</td>
</tr>
<tr>
<td>OLVD</td>
<td>Delivery Date</td>
<td>Numeric</td>
<td>6</td>
</tr>
<tr>
<td>OLSTI</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
</tbody>
</table>

Primary Key: OLWID, OLDID, OLOID, OLNBR

### Table 19. Item Table Layout (ITEM)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Real Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>IID</td>
<td>Item ID</td>
<td>Character</td>
<td>6</td>
</tr>
<tr>
<td>INAME</td>
<td>Item Name</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>IPRICE</td>
<td>Price</td>
<td>Decimal</td>
<td>5</td>
</tr>
<tr>
<td>IDATA</td>
<td>Item Information</td>
<td>Character</td>
<td>50</td>
</tr>
</tbody>
</table>

Primary Key: IID

### Table 20. Stock Table Layout (Stock)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Real Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>STWID</td>
<td>Warehouse ID</td>
<td>Character</td>
<td>4</td>
</tr>
</tbody>
</table>
3.1.8 Database Terminology

This redbook concentrates on the use of the AS/400 system as a database server in a client/server environment. In some cases, we use SQL to access the AS/400 database, in other cases, we use native database access.

The terminology used for the database access is different in both cases. In Table 21, you find the correspondence between the different terms.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Real Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>STIID</td>
<td>Item ID</td>
<td>Character</td>
<td>6</td>
</tr>
<tr>
<td>STQTY</td>
<td>Quantity in Stock</td>
<td>Decimal</td>
<td>5</td>
</tr>
<tr>
<td>STDI01</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>STDI02</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>STDI03</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>STDI04</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>STDI05</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>STDI06</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>STDI07</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>STDI08</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>STDI09</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>STDI10</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
<tr>
<td>STYTD</td>
<td>Year to Date</td>
<td>Decimal</td>
<td>9</td>
</tr>
<tr>
<td>STORDERS</td>
<td>Quantity</td>
<td>Decimal</td>
<td>5</td>
</tr>
<tr>
<td>STREMORD</td>
<td>Quantity</td>
<td>Decimal</td>
<td>5</td>
</tr>
<tr>
<td>STDATA</td>
<td>Item Information</td>
<td>Character</td>
<td>50</td>
</tr>
</tbody>
</table>

Primary Key:STWID,STIID

<table>
<thead>
<tr>
<th>AS/400 Native</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library</td>
<td>Collection</td>
</tr>
<tr>
<td>Physical File</td>
<td>Table</td>
</tr>
<tr>
<td>Field</td>
<td>Column</td>
</tr>
<tr>
<td>Record</td>
<td>Row</td>
</tr>
<tr>
<td>Logical File</td>
<td>View or Index</td>
</tr>
</tbody>
</table>
Chapter 4. Application Development Using AS/400 OLE DB Support

In Chapter 3, “Overview of the Order Entry Application” on page 87 we discuss a sample AS/400 Order Entry application that uses RPG programs and DDS display files. This chapter presents a client/server version of that application. The client/server version lets you work with the same database files on the AS/400 system as the RPG application and invoke the same processes to post a completed order. One of the goals of the client/server version is to reuse as much of the already written AS/400 code as possible. By using this approach, we were able to concentrate on the user interface part of the application, since most of the processing logic was already available on the AS/400 system.

Figure 49 shows a sample of the client/server Order Entry application. It was developed using the Client Access OLE DB provider.
4.1 Migrating the RPG Program to Use AS/400 OLE DB Support

The original Order Entry application consists of four RPG programs:

- **ORDENTR** - the main processing program
- **SLTPARTR** - supports prompting for Part numbers
- **SLTCUSTR** - supports prompting for Customer information
- **PRTORDERR** - monitors the Order data queue and prints order records

Our goal is to migrate this application to a client/server environment. We want to reuse as much of the existing application as possible in the client/server version. We also want the original application to continue to function as a 5250 application.
We migrate the application to an ADO Client that provides a graphical user interface. We modify the original RPG code to allow it to be used from either the new ADO client interface or from the original 5250 interface. Figure 51 shows the new design. This version of the application is a thin client implementation. Most of the processing is done on the AS/400 system. The client provides a graphical user interface.

We also provide a fat client version of the application as shown in Figure 52. In this case, the processing logic is done at the client and the AS/400 system is used to store the databases.
The client/server Order Entry application was designed to show some of the techniques that are available to you if you use the AS/400 OLE DB Provider. The application uses the following techniques:

- DDM record-level access
- SQL statement processing
- Stored procedure calls
- Distributed program calls
- Data queue send operation

The only Client Access OLE DB Provider technique not illustrated is AS/400 Command execution, which is a variation of the distributed program call technique.
4.2 Working with the Program

The Order Entry using ADO program is written using Microsoft Visual Basic 5.0 (the program will work with the Professional or Enterprise editions). You can start the program from the Visual Basic development environment or compile it to an EXE program and run it in Windows.

When you start the program, the order entry form fields are blank.

- If you know a customer number, you can enter it in the Customer ID field. When you tab out of the field, the program retrieves the customer information from the database.
- If the customer is not found, an error message is displayed in the status bar at the bottom of the form.

You can also click the **Customer Search** toolbar button. The program then displays the Customer Search form (see Figure 54).

- Enter a character in the entry field then click the **Search** command button.
  
  **Note:** The starting with feature is not implemented in this program. All customer records in the database are loaded and displayed.
- You can click a customer ID in the Cust ID column to return the customer data to the order entry form.
- You can right-click on a customer ID to display the pop-up menu of customer options (see Figure 55 on page 108). You can add a new customer, delete, update or choose the selected customer, or close the customer search form.

![Figure 54. The Customer Search Dialog](#)
The Customer Details form (see Figure 56) is displayed with the selected customer data if you select the update or delete options. If you select the Add option the customer details form is displayed with blank fields.

- To add a customer, enter values for the customer data. The program displays an error message if you enter a duplicate customer ID.

- If you select a customer for update, you can change all of the customer data except the customer ID.

- If you select a customer for delete, you cannot change any of the customer data fields. Your only options are to confirm or cancel the delete request.

- If you click OK for the add, update or delete request, the customer search dialog is refreshed (see Figure 54 on page 107). You will be able to see the new or changed customer data in the customer search dialog.
The Item Search and Item Details dialogs are similar to the customer dialogs (see Figure 57 and Figure 58). The only difference in functionality is that the item search does not support the *add new* request.

*Figure 57. Item Search Dialog, Showing Pop-Up Menu*

*Figure 58. The Item Details Dialog*
4.2.1 First Time Processing

The Options for ADOTest dialog (see Figure 59) is displayed the first time you run the program. You need to enter or select the following options:

- **Libraries** – Specify the name of the library or libraries where AS/400 objects used by the program are located. If you installed the source code and database file samples provided with this redbook in the suggested library, enter APILIB as the library name. You can click the **Make all the same** command button to put the same value into the second through fourth library entry fields.

- **Customers and Items Database** – Select the processing technique that you want to use to work with the two databases. The program uses your selections to determine if it should use the AS/400 OLE DB Provider Record Level Access or SQL support.

- **Order Processing** – Select the processing technique that you want to use to process a completed order. Order processing updates files on the AS/400 with the order data and prints a listing of the order. You can select the **Use Program Call** technique or the **Use SQL** technique.
  
  - The Program Call technique passes all order information in a parameter list to an AS/400 program. The AS/400 program puts the order information into the database and puts an order entry on a data queue. A batch program receives the data queue entry and then prints the order listing.
  
  - The SQL technique uses SQL **INSERT** and **UPDATE** statements to put order information into the AS/400 database. This technique also uses AS/400 OLE DB Provider data queue support to put an order entry on a data queue. On the AS/400 system, a batch program receives the data queue entry and then prints the order listing.

![Options for ADOTest](image)

**Figure 59. Set Libraries and Processing Options in the Options for ADOTest Dialog**

You can review the currently selected processing techniques in the lower-right corner of the Order Entry using ADO dialog (see Figure 60 on page 111). The selected techniques are displayed from left to right:
• Customers Database technique
• Items Database technique
• Order Processing technique

The abbreviations used for the techniques are:
• RLA – Record-Level Access
• SQL – SQL processing
• DPC – Distributed Program Call

4.2.2 AS/400 System Name

The AS/400 system name is displayed in the status bar at the bottom of the Order Entry using ADO dialog. In Figure 60 the system name is **TCPASM05**. If you are connected to multiple AS/400 systems, the system names are displayed in a listbox. You can select the AS/400 system to use in the listbox.

![Order Entry using ADO 1.0](image)

Figure 60. Order Entry Dialog, See Option Selections in Lower Right Corner
4.2.3 Effect of Changing Options

You can change program options or select a different AS/400 system while the program is running. If you make any changes on the Options for ADO Test dialog (Figure 59 on page 110) and click the **OK** command button or select a different AS/400 system on the Order Entry using ADO dialog (see Figure 60 on page 111), the program reinitializes.

You should not change any of the options when there is any pending order data. If you make changes and have pending order data, the data is lost.

4.2.4 Procedure Trace Option

The Enable procedure trace checkbox (lower left corner of Figure 59 on page 110) is used to enable tracing of the program in the Visual Basic development environment. This trace is in addition to the debugging facilities provided with Visual Basic. A sample of the trace is shown in Figure 61. The trace is not displayed if you run the program as a compiled EXE.

![Immediate Window, Showing Procedure Trace when Running ADOTest](image)

*Figure 61. Immediate Window, Showing Procedure Trace when Running ADOTest*
4.2.5 Changing Options

Options that you specify remain in effect until you change them, even if you exit the program and then rerun it later. To change options, use the File | Set program options menu item (see Figure 62). That menu option displays the Options dialog again (see Figure 59 on page 110).

![Figure 62. Use the File | Set Program Options Menu Item to Go to the Options Dialog](image)

4.2.6 Use of the Registry

Your option selections are recorded in the Windows 95/NT registry. Figure 63 shows the location of the options in the registry and a sample set of values for the options.

If you delete the registry key for the options, the program displays the Options dialog (see Figure 59 on page 110) the next time you start the program.

![Figure 63. Registry Entries for ADOTest Program](image)
4.2.7 Error Report

The last dialog used in the program is the ADO Error Report (see Figure 64). This dialog is displayed for ADO or Visual Basic errors that cannot otherwise be handled in the program. For example, Figure 64 shows the error message for an invalid library name selection.

Many error messages include additional text description. You can view the additional text by clicking on the error number.

If the error is not severe enough to cause a program failure, the error is considered to be recoverable. If continuing the program with the error would cause program failure, the error is not recoverable. If there are any errors in the ADO Error Report dialog that are not recoverable, the OK button is disabled. Your only option with unrecoverable errors is to click the Cancel button to end the program.

Figure 64. The ADO Error Report Dialog
4.3 Setting References to ADO in Visual Basic

Before you can run the Visual Basic program, you need to check the project references to be sure that the project is configured for the version of ADO that you want to test.

Follow these steps to set the reference to ADO:

- Open the ADOTEST.VBP project in the Visual Basic development environment.
- Go to the Project | References menu item (see Figure 65 on page 116).
- On the References dialog (see Figure 66 on page 116), verify that the entry Microsoft OLE DB ActiveX Data Objects 1.0 Library is checked if you want to test ADO 1.0 (the version that is distributed with the AS/400 OLE DB Provider). You should also verify that file MSADO10.DLL is selected, as shown in the figure.
- If you download and install ADO 1.5, you can select the Microsoft ActiveX Data Objects 1.5 Library (see Figure 67 on page 116). Verify that the file name is MSADO15.DLL.

Note: As of June 1, 1998 you can freely download the ADO 1.5 library from the following Web site:

http://www.microsoft.com/data/mdacl5.htm

- If you are unable to locate that link, try either of these links:

  http://www.microsoft.com/data/ado
  http://www.microsoft.com/data

Note: You cannot select both ADO 1.0 and ADO 1.5 simultaneously in the Visual Basic program. The References dialog prevents you from selecting both libraries.

You can also set a conditional compile value that the program uses to display the selected version of ADO (see the captions of Figure 49 on page 103 or Figure 60 on page 111).

- Go to the Project | ADOTEST Properties menu item (see Figure 68 on page 117).
- In the ADOTEST – Project Properties dialog (see Figure 69 on page 117), enter the ADO_VERSION conditional compile value. Because you can only enter an integer value, use 10 for ADO Version 1.0 and 15 for ADO Version 1.5.
Figure 65. Use the Project | References Menu Item to Check the Version of ADO

Figure 66. Reference for ADO Version 1.0

Figure 67. Reference for ADO Version 1.5
Figure 68. Use the Project | ADOTEST Properties Menu Item to Set Project Properties

Figure 69. The Project Properties Dialog
4.4 Preparing AS/400 Program Objects

In order to run the application, you will have to catalog two stored procedures.

- Catalog the Stored Procedures:

  STRSQL (requires 5769-ST1)

  ```sql
  create procedure apilib/sltcustr (in dummy1 char(1),
    in dummy2 char(1))
  result set 1
  external name apilib/sltcustr
  language rpgle general variant

  create procedure apilib/sltpartr (in dummy1 char(1),
    in dummy2 char(1))
  result set 1
  external name apilib/sltpartr
  language rpgle general variant
  ```
4.5 Overview of the Visual Basic Application

The Visual Basic Project Explorer window (see Figure 70) shows an outline view of the forms, modules, classes and related documents used in this project. You have already seen pictures of the forms.

![Figure 70. The Visual Basic Project Explorer Showing the ADOTest Project](image)

There is a clear distinction in the project between user interface code and database access code. All database access occurs in the class modules. Each database file on the AS/400 has at least one class module associated with it. The Customer and Item database files have two class modules, one for access to sets of records (CCustomers and CItems), the other for access to individual records (CCustomer, CItem).

The program starts execution in the modMain module. That module executes initialization code and then passes control to the frmOrder form to display the order entry form. Other forms are loaded as needed depending upon your actions when running the program.

Error message text is kept in the ADOTEST.RES resource file. This is the compiled version of the file. You can review and change the text of the resource file messages by editing file ADOTEST.RC. If you make any changes, you will have to run the resource compiler to update the .RES file (see the Visual Basic references included with Visual Basic for more information on using and compiling a resource file).
4.6 Detailed Description of Forms Used in the Project

This section describes each of the forms used in the project. The form descriptions are presented in alphabetical order.

The “Events Monitored” description includes only events that occur in other forms or classes that the form needs to know about. Events Monitored does not describe events that pertain only to the form itself. For example, Events Monitored does not describe command button click events for command buttons that are on the form.

4.6.1 frmCustomer — Customer Details

![Customer Details Dialog]

Figure 71. The Customer Details Dialog

Table 22. frmCustomer Details

<table>
<thead>
<tr>
<th>Form</th>
<th>frmCustomer Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td></td>
</tr>
<tr>
<td>This form is used for the following purposes:</td>
<td></td>
</tr>
<tr>
<td>1.) Add a new customer. If you enter a customer ID that is already in the database an error message is displayed in the left status bar at the bottom of the form.</td>
<td></td>
</tr>
<tr>
<td>2.) Update data for an existing customer. The customer ID cannot be changed.</td>
<td></td>
</tr>
<tr>
<td>3.) Delete an existing customer.</td>
<td></td>
</tr>
<tr>
<td>No edits are performed on the data.</td>
<td></td>
</tr>
<tr>
<td><strong>Events Raised</strong></td>
<td>CustomerChange — This event is raised if you add a new customer, update an existing customer or delete a customer. This event is used in the frmCustomerSearch form to control the refresh of the customer list when any changes occur to customer data.</td>
</tr>
<tr>
<td><strong>Events Monitored</strong></td>
<td>CustomerError — From class CCustomer. This event is raised in class CCustomer if there is an error when working with customer data. For example, the CustomerError event is raised if you attempt to add a customer to the database and the customer ID is already used. By monitoring this event, the frmCustomer form can display an error message.</td>
</tr>
</tbody>
</table>
Public Data Members

**CS_OPTION — enumeration**
Use this enumeration to select the type of operation that is to be performed on customer data. Values are:

- `CS_OPTION.AddCustomer`
- `CS_OPTION.UpdateCustomer`
- `CS_OPTION.DeleteCustomer`

Public Procedures

**Initialize**
Call the `Initialize` procedure before passing control to this form. The `Initialize` procedure is used to set values for the public properties associated with the form, if the values are passed as arguments to the `Initialize` procedure.

Public Properties

**Mode** (Get/Let)
The `Mode` property is used to set or retrieve the value of the currently selected operation. Before passing control to this form, set `Mode` to one of the values in the `CS_OPTION` enumeration.
4.6.2 frmCustomerSearch — Customer Search

![Customer Search Dialog](image)

**Figure 72. The Customer Search Dialog**

**Table 23. frmCustomerSearch Details**

<table>
<thead>
<tr>
<th>Form</th>
<th>frmCustomerSearch</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This form is used for the following purposes:</td>
</tr>
<tr>
<td></td>
<td>1.) Display a list of customers in the database.</td>
</tr>
<tr>
<td></td>
<td>2.) Allow selection of a customer. Customer data for the selected customer is returned to the caller through the MyCustomer object based on the CCustomer class.</td>
</tr>
<tr>
<td></td>
<td>3.) Allow selection of file maintenance operations: Add new, Delete, Update.</td>
</tr>
<tr>
<td><strong>Events Raised</strong></td>
<td>HaveCustomer — this event is raised if you select a customer. You select a customer by clicking on the customer ID or by selecting the Return selected customer menu item from the pop-up menu.</td>
</tr>
<tr>
<td></td>
<td>Other forms can monitor for this event. When the event is raised, the data for the selected customer can be retrieved from the MyCustomer object (defined as Public in modMain.bas)</td>
</tr>
<tr>
<td><strong>Events Monitored</strong></td>
<td>CustomerChange — From form frmCustomer. This event is raised in form frmCustomer if there are any changes made to customer detail data. A change can be an addition to the database, deletion of a customer record or update to a customer record. When the CustomerChange event occurs, this form refreshes itself so that current data is displayed.</td>
</tr>
<tr>
<td><strong>Public Data Members</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Public Procedures</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Public Properties</strong></td>
<td>None</td>
</tr>
</tbody>
</table>
4.6.3 frmErrors — ADO Error Report

![Figure 73. The ADO Error Report Dialog](image)

Table 24. frmErrors Details

<table>
<thead>
<tr>
<th>Form</th>
<th>frmErrors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This form is used for the following purposes:</td>
</tr>
<tr>
<td></td>
<td>1.) Display a list of errors that are encountered when running the program. The errors are reported from Visual Basic or by ADO.</td>
</tr>
<tr>
<td></td>
<td>2.) Display detail information about a selected error.</td>
</tr>
<tr>
<td></td>
<td>3.) Allow execution to continue if possible or cancel the program.</td>
</tr>
<tr>
<td>Events Raised</td>
<td>ErrorCancel — this event is raised if you click the Cancel button. Other forms in the project can monitor this event and take whatever steps are required to end the program.</td>
</tr>
<tr>
<td>Events Monitored</td>
<td>None</td>
</tr>
<tr>
<td>Public Data Members</td>
<td>None</td>
</tr>
<tr>
<td>Public Procedures</td>
<td>LoadErrors</td>
</tr>
<tr>
<td></td>
<td>Call this procedure to load the list of errors into the error list display (the upper part of the ADO Error Report dialog). This procedure also shows the form and positions it to the front.</td>
</tr>
<tr>
<td>Public Properties</td>
<td>None</td>
</tr>
</tbody>
</table>
4.6.4 frmItem — Item Details

The following table illustrates frmItem details:

Table 25. frmItem Details

<table>
<thead>
<tr>
<th>Form</th>
<th>frmItem</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This form is used for the following purposes:</td>
</tr>
<tr>
<td></td>
<td>1.) Update data for an existing item. The item ID cannot be changed.</td>
</tr>
<tr>
<td></td>
<td>2.) Delete an existing item.</td>
</tr>
<tr>
<td></td>
<td>No edits are performed on the data.</td>
</tr>
<tr>
<td><strong>Events Raised</strong></td>
<td>ItemChange — This event is raised if you update an existing item or delete an item. This event is used in the frmItemSearch form to control the refresh of the item list when any changes occur to item data.</td>
</tr>
<tr>
<td><strong>Events Monitored</strong></td>
<td>ItemError — From class CItem. This event is raised in class CItem if there is an error when working with item data. For example, the ItemError event is raised if you attempt to add an item to the database and the item ID is already used. By monitoring this event, the frmItem form can display an error message.</td>
</tr>
<tr>
<td><strong>Public Data Members</strong></td>
<td>IT_OPTION — enumeration</td>
</tr>
<tr>
<td></td>
<td>Use this enumeration to select the type of operation that is to be performed on item data. Values are:</td>
</tr>
<tr>
<td></td>
<td>IT_OPTION.AddItem</td>
</tr>
<tr>
<td></td>
<td>IT_OPTION.UpdateItem</td>
</tr>
<tr>
<td></td>
<td>IT_OPTION.DeleteItem</td>
</tr>
<tr>
<td><strong>Public Procedures</strong></td>
<td>Initialize</td>
</tr>
<tr>
<td></td>
<td>Call the Initialize procedure before passing control to this form. The Initialize procedure is used to set values for the public properties associated with the form, if the values are passed as arguments to the Initialize procedure.</td>
</tr>
</tbody>
</table>
4.6.5  frmItemSearch — Item Search

![The Item Search Dialog](image)

The following table illustrates frmItem search details:

<table>
<thead>
<tr>
<th>Form</th>
<th>frmItemSearch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This form is used for the following purposes:</td>
</tr>
<tr>
<td>1.) Display a list of items in the database.</td>
<td></td>
</tr>
<tr>
<td>2.) Allow selection of an item. Item data for the selected item is returned to the caller through the MyItem object based on the CItem class.</td>
<td></td>
</tr>
<tr>
<td>3.) Allow selection of file maintenance operations: Add, Delete, Update.</td>
<td></td>
</tr>
<tr>
<td>Events Raised</td>
<td>HaveItem — This event is raised if you select an item. You select an item by clicking on the item ID or by selecting the Return selected item menu item from the pop-up menu. Other forms can monitor for this event. When the event is raised, the data for the selected item can be retrieved from the MyItem object (defined as Public in modMain.bas)</td>
</tr>
</tbody>
</table>
Events Monitored

- **ItemChange** — From form `frmItem`. This event is raised in form `frmItem` if there are any changes made to item detail data. A change can be an addition to the database, deletion of an item record or update to an item record. When the ItemChange event occurs, this form refreshes itself so that current data is displayed.

<table>
<thead>
<tr>
<th>Form</th>
<th>frmItemSearch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events Monitored</td>
<td>ItemChange</td>
</tr>
<tr>
<td>Public Data Members</td>
<td>None</td>
</tr>
<tr>
<td>Public Procedures</td>
<td>None</td>
</tr>
<tr>
<td>Public Properties</td>
<td>None</td>
</tr>
</tbody>
</table>

A Fast Path to AS/400 Client/Server Using AS/400 OLE DB Support
### 4.6.6 frmOptions — Options for ADOTest

**Figure 76. The Options for ADOTest Dialog**

**Table 27. frmOptions Details**

<table>
<thead>
<tr>
<th>Form</th>
<th>frmOptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This form is used for the following purposes:</td>
</tr>
<tr>
<td></td>
<td>1.) Set the library names where AS/400 objects used in the project are located.</td>
</tr>
<tr>
<td></td>
<td>2.) Select the processing technique to use for the Customers and Items database access and for Order Processing.</td>
</tr>
<tr>
<td></td>
<td>3.) Enable the procedure trace option.</td>
</tr>
<tr>
<td></td>
<td>When you start the program, the current values are retrieved from the Windows 95/NT registry. If you change any options, the registry is updated.</td>
</tr>
<tr>
<td></td>
<td>The MyOptions object (constructed from the COption class) is used to communicate the current option selections to other forms and modules in this program.</td>
</tr>
<tr>
<td></td>
<td>No edits are performed on the data. If you specify an invalid library name, the error is not reported until you try to access objects in the library. The Options dialog also does not verify your selection for a processing technique. For example, if you do not have the stored procedures used in the searches, you should not select the Record Level Access option.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Events Raised</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events Monitored</td>
<td>None</td>
</tr>
<tr>
<td>Public Data Members</td>
<td>None</td>
</tr>
<tr>
<td>Public Procedures</td>
<td>None</td>
</tr>
<tr>
<td>Public Properties</td>
<td>None</td>
</tr>
</tbody>
</table>
4.6.7 frmOrder — Order Entry Using ADO

Figure 77. The Order Entry Using the ADO Dialog

Table 28. frmOrder Details

<table>
<thead>
<tr>
<th>Form</th>
<th>frmOrder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This form is used for the following purposes:</td>
</tr>
<tr>
<td></td>
<td>1.) Allow entry of a customer ID or request a customer search. Upon entering a valid customer ID or returning from the customer search with a selected customer, data for the customer is displayed in the Customer Information frame.</td>
</tr>
<tr>
<td></td>
<td>2.) Allow entry of order line items. The item number can be specified or the item search can be started. If a valid item number is entered or returned from the item search, the item description is displayed in the Item Information frame. After entering the quantity the item can be added to the list of items in the order.</td>
</tr>
<tr>
<td></td>
<td>3.) Allow update of the quantity of a line item.</td>
</tr>
<tr>
<td></td>
<td>4.) Allow deletion of a line item from the order.</td>
</tr>
<tr>
<td></td>
<td>5.) Invoke the order processing application on the AS/400 to write the order information to the database and print a listing of the order.</td>
</tr>
<tr>
<td></td>
<td>6.) Show the Options dialog (selection in the File menu).</td>
</tr>
<tr>
<td></td>
<td>Although you can enter any number of order line items, the processing programs only work with the first 50 order line items.</td>
</tr>
<tr>
<td>Events Raised</td>
<td>None</td>
</tr>
</tbody>
</table>
### Application Development Using AS/400 OLE DB Support

#### Events Monitored

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CustomerError</strong></td>
<td>From object MyCustomer. This event is raised in object MyCustomer (based on class CCustomer) when there is an error with customer data. The event handling routine in frmOrders checks for the record not found condition and displays a message in the status bar.</td>
</tr>
<tr>
<td><strong>ErrorCancel</strong></td>
<td>From form frmErrors. This event is raised in form frmErrors if the Cancel command button is clicked. The event handler in frmOrders ends the program in response to this event.</td>
</tr>
<tr>
<td><strong>HaveCustomer</strong></td>
<td>From form frmCustomerSearch. This event is raised in form frmCustomerSearch when a customer is selected. Data for the selected customer is available in the MyCustomer object. The frmOrders form moves the data from the MyCustomer object to the Customer Information frame.</td>
</tr>
<tr>
<td><strong>HaveError</strong></td>
<td>From collection MyErrors. This event is raised in collection MyErrors (based on class CErrors) when there is a non-recoverable error. The frmOrders event handling routine displays the frmErrors form to display the error list.</td>
</tr>
<tr>
<td><strong>HaveItem</strong></td>
<td>From form frmItemSearch. This event is raised in form frmItemSearch when an item is selected. Data for the selected item is available in the MyItem object. The frmOrders form moves the data from the MyItem object to the Item Information frame.</td>
</tr>
<tr>
<td><strong>ItemError</strong></td>
<td>From object MyItem. This event is raised in object MyItem (based on class CItem) when there is an error with item data. The event handling routine in frmOrders checks for the record not found condition and displays a message in the status bar.</td>
</tr>
<tr>
<td><strong>OptionsChanged</strong></td>
<td>From object MyOptions. This event is raised if you change any of the program options (on frmOptions) and save the changes. The event handling routine in frmOrders calls the Form_Load procedure to reinitialize the program so that the new options will be in effect.</td>
</tr>
<tr>
<td><strong>OrderError</strong></td>
<td>From object MyOrder. This event is raised in object MyOrder (based on class COrder) when there is an error when processing the order. The event handling routine in frmOrders loads the frmErrors form to display the error list.</td>
</tr>
</tbody>
</table>

#### Public Data Members

<table>
<thead>
<tr>
<th>Data Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IT_OPTION</strong></td>
<td>enumeration</td>
</tr>
<tr>
<td>Use this enumeration to select the type of operation that is to be performed on item data. Values are:</td>
<td></td>
</tr>
<tr>
<td>IT_OPTION.AddItem</td>
<td></td>
</tr>
<tr>
<td>IT_OPTION.UpdateItem</td>
<td></td>
</tr>
<tr>
<td>IT_OPTION.DeleteItem</td>
<td></td>
</tr>
</tbody>
</table>
4.7 Review of ADO Code Used in the Order Entry Using ADO Application

With the exception of object initialization code used in the modMain.bas code module, all of the ADO code used in the application is contained in the class modules. The forms that display and act upon the database never deal directly with ADO operations. Instead, those forms make requests to the objects to retrieve, update, delete or add to the database.

By separating the database access functions from the user interface, it should be easier to modify the application to use different ADO techniques than if the database access code was included in the forms. For example, if you change the access technique in one of the classes from SQL to record level access, you will not need to make any changes to the form that uses the data. Because the form’s only interface to the database is through the properties and methods defined by the class, you can change the underlying processing without having to change anything else.

This section shows the code used in the code module and each of the class modules. Only the code pertaining to ADO is shown. When you review the code in the Visual Basic editor you will see more source code than is shown in this redbook.

4.7.1 ADO Code in modMain.bas

The following code is used in this module. This code is used to define the ADO Connection object and the AD400 Connection object, instantiate the objects and destroy the objects.

4.7.1.1 Code Used in the General Declarations Section

The code used in this section defines the ADO Connection and AD400 Connection objects. These are used in all of the classes in the program.

```vbnet
Public CnAD400 As AD400.Connection
Public Cn400 As ADODB.Connection
```

4.7.1.2 Code Used in the CreateObjects Procedure

The Connection objects are instantiated in this procedure.

```vbnet
Set Cn400 = New ADODB.Connection
Set CnAD400 = New AD400.Connection
```

4.7.1.3 Code Used in the DestroyObjects Procedure

The Connection objects are destroyed in this procedure.

```vbnet
Set CnAD400 = Nothing
Set Cn400 = Nothing
```
4.7.2 ADO Code in Class CADO

The CADO class contains a utility procedure that creates an ADO Parameter object.

4.7.2.1 Code Used in the CreateCommandParm Procedure

This code is used to create an ADO command parameter. A temporary ADO Command object is created to allow creation of the ADO Parameter object. The Parameter object is returned from this procedure. Parameter objects can be used to pass values to programs, stored procedures and SQL statements with parameter markers.

```vbscript
Public Function CreateCommandParm(ByVal ParmName, ByVal ParmType, ByVal ParmSize, Optional ByVal ParmNumericScale = 0, Optional ByVal ParmDirection = adParamInput) As ADODB.Parameter
    '-------------------------------------------------------------------------
    'create a parameter, return new parameter object
    '-------------------------------------------------------------------------
    Dim P1 As ADODB.Parameter
    Dim CommandObject As New ADODB.Command
    '-------------------------------------------------------------------------
    'create the parameter
    '-------------------------------------------------------------------------
    Set P1 = CommandObject.CreateParameter(Name:=ParmName, Type:=ParmType, Direction:=ParmDirection)
    Select Case ParmType
    '-------------------------------------------------------------------------
    'set additional properties for Character parameter
    '-------------------------------------------------------------------------
    Case adChar
        P1.Size = ParmSize
    '-------------------------------------------------------------------------
    'set additional properties for Decimal / Numeric parameter
    'use DECIMAL for AS/400 Packed data
    'use NUMERIC for AS/400 Zoned data
    '-------------------------------------------------------------------------
    Case adDecimal, adNumeric
        P1.Precision = ParmSize
        P1.NumericScale = ParmNumericScale
    End Select
    Set CreateCommandParm = P1
End Function
```
4.7.3 ADO Code in Class CCustomer

This class is used to act on a record in the customer database. Add, delete, get and update operations are supported. The class supports two types of access:

- Record level access using the AD400 Index class
- SQL statement processing using ADO

4.7.3.1 Code Used in the General Declarations Section

The code used in this section is used to define the ADO Connection, Command and Recordset objects and the AD400 Connection and Index objects.

```vbscript
Private m_ADOConnection As ADODB.Connection
Private cmCustomer As ADODB.Command
Private cmCustomerRead As ADODB.Command
Private rsCustomer As ADODB.Recordset
Private rsCustomerRead As ADODB.Recordset

Private ixCustomer As AD400.Index
Private m_AD400Connection As AD400.Connection
```

4.7.3.2 Code Used in the AddCustomerUsingRLA Procedure

This procedure is used to add a customer to the database using the record level access technique provided with the AS/400 OLE DB Provider. Note that the AddNew technique used is different from the AddNew documentation provided by Microsoft. You must use the technique illustrated here to obtain correct results.

```vbscript
Private Sub AddCustomerUsingRLA()

'-------------------------------------------------------------------------
'add a new customer using Record Level Access
'-------------------------------------------------------------------------
Dim Fields
Dim Values

'create Fields array - field names in Customer file
' If you do not specify all of the field names and a default initialization value, fields that are defined in the database file but NOT included in this list are initialized to x'00'. This causes an error when you use a program like DFU to review the record, and may also cause problems with other programs that access the record.
'-------------------------------------------------------------------------

Fields = Array("CID", _,
    "CDID", _,
    "CFIRST", _,
    "CWID", _,
    "CINIT", _,
    "CLAST", _,
    "CLDATE", _,
    "CADDR1", _,
    "CCREDT", _,
    "CADDR2", _,
    "CDCT", _,
    "CCITY", _,
    "CSTATE", _,
    "CZIP", _,
    "CPHONE", _,
    "CBAL", _
```
Values = Array(CStr(Format(CustomerID, "0000")), _, ADOTEST_DISTRICTID, _, FirstName, _, ADOTEST_WAREHOUSEID, _, Initial, _, LastName, _, 0, _, Address1, _, "OK", _, Address2, _, 0, _, City, _, State, _, ZipCode, _, " ", _, 0, _, 0, _, 0, _, 0, _, 0, _, 0, _, 0, _, 0, _
" ")

End Sub

4.7.3.3 Code Used in the AddCustomerUsingSQL Procedure

This procedure is used to add a customer to the database using an SQL INSERT statement. The MakeSQLNameValue procedure in the CMakeString class is called to format the name/value list used for the INSERT statement.

Private Sub AddCustomerUsingSQL()

' add a customer using SQL

Dim FieldAlpha
Dim FieldNames
Dim FieldValues
Dim SQL

' create the SQL statement

SQL = "INSERT INTO"
SQL = SQL & BLANK
SQL = SQL & Options.FilesLibrary
SQL = SQL & SQLDELIMITER
SQL = SQL & DBFILE_CUSTOMER
SQL = SQL & BLANK

'---------------------------------------------------------------
'send field names, values, type to formatting routine (create INSERT list)
'---------------------------------------------------------------

FieldNames = Array("CFIRST", _
   "CINIT", _
   "CLAST", _
   "CADDR1", _
   "CADDR2", _
   "CCITY", _
   "CSTATE", _
   "CZIP", _
   "CLDATE", _
   "CLTIME", _
   "CBAL", _
   "CYTD", _
   "CID", _
   "CDID", _
   "CWID")

FieldValues = Array(FirstName, _
   Initial, _
   LastName, _
   Address1, _
   Address2, _
   City, _
   State, _
   ZipCode, _
   DateLastUpdated, _
   TimeLastUpdated, _
   CurrentBalance, _
   YearToDate, _
   CStr(Format(CustomerID, "0000")), _
   ADOTEST_DISTRICTID, _
   ADOTEST_WAREHOUSEID)

FieldAlpha = Array(True, _
   True, _
   True, _
   True, _
   True, _
   True, _
   True, _
   True, _
   True, _
   True, _
   False, _
   False, _
   False, _
   False, _
   False, _
   True, _
   False, _
   True)

SQL = SQL & MakeString.MakeSQLNameValue(FieldName:=FieldNames, _
   FieldValue:=FieldValues, _
   IsAlpha:=FieldAlpha, _
   SQLStringType:=MS_SQL.Insert)
execute the SQL statement

```
cmCustomer.CommandText = SQL
Set rsCustomer = cmCustomer.Execute
```

End Sub

### 4.7.3.4 Code Used in the DeleteCustomerUsingRLA Procedure

This procedure is used to delete a customer using the record-level access technique provided with the AS/400 OLE DB Provider. The `ReleaseLock` procedure is also shown following this procedure. `ReleaseLock` is used to position the database cursor to the beginning of file.

```
Private Sub DeleteCustomerUsingRLA()
    'delete customer record using Record Level Access
    Call GetCustomerUsingRLA(ReadOption:=Read.Update)
    rsCustomer.Delete
    Call ReleaseLock

End Sub
```

```
Private Sub ReleaseLock()
    'position the recordset at BOF (releases any locks)
    On Error GoTo 0                               'disable error handling
    rsCustomer.MoveFirst
    rsCustomer.MovePrevious

End Sub
```

### 4.7.3.5 Code Used in the DeleteCustomerUsing SQL Procedure

This procedure is used to delete a customer using an SQL `DELETE` statement.

```
Private Sub DeleteCustomerUsingSQL()
    'delete a customer record using SQL
    Dim SQL

    'format SQL statement
    SQL = "DELETE FROM" SQL = SQL & BLANK
    SQL = SQL & Options.FilesLibrary SQL = SQL & SQLDELIMITER
    SQL = SQL & DBFILE_CUSTOMER SQL = SQL & BLANK
    SQL = SQL & "WHERE CID=" SQL = SQL & QUOTE
    SQL = SQL & CStr(Format(CustomerID, "0000")) SQL = SQL & QUOTE
    SQL = SQL & BLANK
```
SQL = SQL & "AND CDID="
SQL = SQL & ADOTEST_DISTRICTID
SQL = SQL & BLANK
SQL = SQL & "AND CWID="
SQL = SQL & QUOTE
SQL = SQL & ADOTEST_WAREHOUSEID
SQL = SQL & QUOTE

'-------------------------------------------------------------------------------
'execute the SQL statement
'-------------------------------------------------------------------------------

cmCustomer.CommandText = SQL
Set rsCustomer = cmCustomer.Execute

End Sub
4.7.3.6 Code Used in the GetCustomerUsingRLA Procedure

This procedure is used to retrieve a specific customer record from the database. The AD400 Index support is used to get the customer. This procedure demonstrates the technique of using two recordsets to read the database:

- One recordset is for read-only. When a record is retrieved using this option, the record is not locked.
- The other recordset is for read-update. The record is locked when this option is used.

If the requested record is not found in the file, an error condition is raised. The procedure traps the error and raises it again so that the calling procedure will handle it.

Private Sub GetCustomerUsingRLA(ByVal ReadOption)
    '-------------------------------------------------------------------------
    'get bookmark for selected record, position recordset to the record
    '-------------------------------------------------------------------------
    Dim Bookmark
    Dim CustomerKey
    On Error GoTo ErrorHandler

    '-------------------------------------------------------------------------
    'ixCustomer - retrieves requested record (in CustomerKey)
    '-------------------------------------------------------------------------
    CustomerKey = CStr(Format(CustomerID, "0000"))
    Bookmark = ixCustomer.GetBookmark(Key:=CustomerKey, dwSeekOptions:=ad400SeekFirstEQ)

    '-------------------------------------------------------------------------
    'rsCustomerRead - recordset containing selected record, read only
    'rsCustomer - recordset containing selected record, all operations allowed
    '-------------------------------------------------------------------------
    Select Case ReadOption
        Case Read.NoUpdate
            rsCustomerRead.Move NumRecords:=0, _
            Start:=Bookmark
            Call MoveDataFileToField(CurrentRecordset:=rsCustomerRead)
        Case Read.Update
            rsCustomer.Move NumRecords:=0, _
            Start:=Bookmark
            Call MoveDataFileToField(CurrentRecordset:=rsCustomer)
    End Select
    Exit Sub

ErrorHandler:
    Err.Raise Err.Number
End Sub
4.7.3.7  Code Used in the GetCustomerUsingSQL Procedure

The code in this procedure uses an SQL SELECT statement to retrieve the customer record. Note the use of the BOF/EOF test to determine if a record was retrieved.

Private Sub GetCustomerUsingSQL()
   '-------------------------------------------------------------------------
   'get customer record using SQL
   '-------------------------------------------------------------------------
   Dim SQL
   On Error GoTo ErrorHandler
   '-------------------------------------------------------------------------
   'format the SQL statement
   '-------------------------------------------------------------------------
   SQL = "SELECT * FROM"
   SQL = SQL & BLANK
   SQL = SQL & Options.FilesLibrary
   SQL = SQL & SQLDELIMITER
   SQL = SQL & DBFILE_CUSTOMER
   SQL = SQL & BLANK
   SQL = SQL & "WHERE CID="
   SQL = SQL & QUOTE
   SQL = SQL & CStr(Format(CustomerID, "0000"))
   SQL = SQL & QUOTE
   SQL = SQL & BLANK
   SQL = SQL & "AND CDID="
   SQL = SQL & ADOTEST_DISTRICTID
   SQL = SQL & BLANK
   SQL = SQL & "AND CWID="
   SQL = SQL & QUOTE
   SQL = SQL & ADOTEST_WAREHOUSEID
   SQL = SQL & QUOTE
   '-------------------------------------------------------------------------
   'execute the SQL statement
   '-------------------------------------------------------------------------
   cmCustomer.CommandText = SQL
   Set rsCustomer = cmCustomer.Execute
   '-------------------------------------------------------------------------
   'raise error if no record found
   '-------------------------------------------------------------------------
   If (rsCustomer.BOF And rsCustomer.EOF) Then
      Err.Raise Errors.ERRID_NO_RECORD_FOUND
   End If
   '-------------------------------------------------------------------------
   'record found, process
   '-------------------------------------------------------------------------
   Call MoveDataFileToField(CurrentRecordset:=rsCustomer)

   Exit Sub

ErrorHandler:
   Err.Raise Err.Number

End Sub
4.7.3.8 Code Used in the Initialize Procedure

The Initialize procedure is used to set values for the command objects used to work with the customer database. Note the use of the ReleaseLock procedure when opening the database using record-level access. If you fail to release the lock, the first record in the file is locked if the database is open for change.

Public Sub Initialize(Optional ByVal vADOConnection, _
    Optional ByVal vAD400Connection, _
    Optional ByVal vErrors, _
    Optional ByVal vOptions)
    'initializations for CCustomer - put here rather than Class.Initialize so that this code is not run each time a Customer object is created
    Dim FileWork
    'if object references not passed in, use properties
    If Not IsMissing(vADOConnection) Then
        Set ADOConnection = vADOConnection
        End If
    If Not IsMissing(AD400Connection) Then
        Set AD400Connection = vAD400Connection
        End If
    If Not IsMissing(vErrors) Then
        Set Errors = vErrors
        End If
    If Not IsMissing(vOptions) Then
        Set Options = vOptions
        End If
    'open ADO objects for customer, depending on processing type
    Select Case Options.CustomerTechnique
    'open for record level access - Lock/NoLock versions
    Case Options.USE_RECORDLEVELACCESS
        FileWork = _
            MyMakeString.MakeFileName(vLibrary:=Options.FilesLibrary, _
                vFile:=DBFILE_CUSTOMER, _
                vCommitmentControl:=MS_CommitmentControl.None)
    'open customer recordset for all file operations (Read/Change/Delete/Add)
    'When you open a recordset for Change, the first record in the file is locked for update. To avoid record locking, you can use the MoveFirst/MovePrevious technique shown here to move to BOF.
    Set cmCustomer = New ADODB.Command
    With cmCustomer
        Set .ActiveConnection = ADOConnection
        .CommandText = FileWork
        .CommandType = adCmdTable
        .Properties("Updatability") = MyADO.FILEOP_RCDA
    End With
    End Case
End Sub
Set rsCustomer = .Execute
End With

Call ReleaseLock

'-------------------------------------------------------------------------
'open customer recordset for read only
'-------------------------------------------------------------------------
Set cmCustomerRead = New ADODB.Command
With cmCustomerRead
    Set .ActiveConnection = ADOConnection
    .CommandText = FileWork
    .CommandType = adCmdTable
    .Properties("Updatability") = MyADO.FILEOP_R
    Set rsCustomerRead = .Execute
End With

Set ixCustomer = AD400Connection.OpenIndex(TableName:=FileWork, _
    IndexName:=FileWork)

'-------------------------------------------------------------------------
'open for SQL
'-------------------------------------------------------------------------
Case Options.USE_SQL
    Set cmCustomer = New ADODB.Command
    Set cmCustomer.ActiveConnection = ADOConnection
    cmCustomer.CommandType = adCmdText
End Select
Exit Sub
ErrorHandler:
End Sub

4.7.3.9 Code Used in the UpdateCustomerUsingRLA Procedure
This procedure updates the customer record using record level access support. The customer record is first retrieved with the GetCustomerUsingRLA procedure. After moving the new field values to the recordset, the Recordset Update method is used to update the database.

Private Sub UpdateCustomerUsingRLA()
    '-------------------------------------------------------------------------
    'set fields, update record
    '-------------------------------------------------------------------------
    Call MoveDataFieldToFile
    Call GetCustomerUsingRLA(ReadOption:=Read.Update)
    Call MoveDataSaveToFile
    rsCustomer.Update
    Call ReleaseLock
End Sub
4.7.3.10 Code Used in the UpdateCustomerUsingSQL Procedure

The code used in this procedure updates the customer record in the database using an SQL UPDATE statement. The field names, values and types are passed to the MakeSQLNameValue procedure in the CMakeString class to create the list of fields and values required for the UPDATE statement.

Private Sub UpdateCustomerUsingSQL()
    Dim FieldAlpha
    Dim FieldNames
    Dim FieldValues
    Dim SQL
    '-------------------------------------------------------------------------
    'format SQL statement
    '-------------------------------------------------------------------------
    SQL = "UPDATE"
    SQL = SQL & BLANK
    SQL = SQL & Options.FilesLibrary
    SQL = SQL & SQLDELIMITER
    SQL = SQL & DBFILE_CUSTOMER
    SQL = SQL & BLANK
    SQL = SQL & "SET"
    '-------------------------------------------------------------------------
    'send field names, values, types to routine to format field/value list
    '-------------------------------------------------------------------------
    FieldNames = Array("CFIRST", _
                        "CINIT", _
                        "CLAST", _
                        "CADDR1", _
                        "CADDR2", _
                        "CCITY", _
                        "CSTATE", _
                        "CZIP", _
                        "CLDATE", _
                        "CLTIME", _
                        "CBAL", _
                        "CYTD")

    FieldValues = Array(FirstName, _
                        Initial, _
                        LastName, _
                        Address1, _
                        Address2, _
                        City, _
                        State, _
                        ZipCode, _
                        DateLastUpdated, _
                        TimeLastUpdated, _
                        CurrentBalance, _
                        YearToDate)

    FieldAlpha = Array(True, _
                        True, _
                        True, _
                        True, _
                        True, _
                        True, _
                        True, _
                        True, _
                        True, _
                        True, _
4.7.4 ADO Code in Class CCustomers

The CCustomers class is used to provide a collection class of customer data. This is used when you need to work with one or more customers at a time; for example, when displaying a list of customers on a search form.

The class provides a GetCustomers procedure that is used to retrieve customers using one of these techniques that you select:

- Stored procedure call
- SQL statement

Both techniques fill a Recordset object. The customer data is extracted from the Recordset and used to fill the customer collection. At that point, the data is available to the application.

4.7.4.1 Code Used in the General Declarations Section

The code used in this section is used to define the ADO Connection, Command and Recordset objects.

```
Private m_ADOConnection As ADODB.Connection
Private cmCustomer As ADODB.Command
Private rsCustomer As ADODB.Recordset
```
4.7.4.2 Code Used in the Initialize Procedure

This procedure is used to initialize the `Command` object. The `Command` object will execute either a stored procedure call or an SQL `SELECT` statement to fill the `Recordset` object.

The stored procedure call requires two dummy parameters. This is because the RPG program was modified so that it can continue to work with AS/400 workstation I/O or respond as a stored procedure and return a result set. The two parameters are used to indicate to the program which technique it is to use when called. The procedure uses the `CreateCommandParm` method in class `CADO` to create the Parameter objects.

```vbnet
Public Sub Initialize(Optional ByVal vADOConnection, _
                     Optional ByVal vErrors, _
                     Optional ByVal vOptions)

    Dim PI As New ADODB.Parameter
    Dim SQL
    Dim StoredProcWork

    '---------------------------------------------------------------
    'if object references not passed in, use properties
    '---------------------------------------------------------------
    If Not IsMissing(vADOConnection) Then
        Set ADOConnection = vADOConnection
    End If

    If Not IsMissing(vErrors) Then
        Set Errors = vErrors
    End If

    If Not IsMissing(vOptions) Then
        Set Options = vOptions
    End If

    '-------------------------------------------------------------------------
    'initializations for CCustomers - put here rather than Class_Initialize so
    'that this code is not run each time a Customer collection is created
    '-------------------------------------------------------------------------
    Set mCol = New Collection
    Set cmCustomer = New ADODB.Command

    With cmCustomer
        Set .ActiveConnection = ADOConnection
        .CommandType = adCmdText
        .Properties("Updatability") = MyADO.FILEOP_R
    End With

    'select the processing technique to use based on option selection on Options
    'form available options for Customer search:
    '  Options.USE_DATAQUEUES     (not currently implemented)
    '  Options.USE_PROGRAMCALL    (not currently implemented)
    '  Options.USE_RECORDLEVELACCESS  (not currently implemented)
    '  Options.USE_SQL
    '  Options.USE_STOREDPROCEDURES
    '-------------------------------------------------------------------------
    Select Case Options.CustomerTechnique
```
Case Options.USE_STOREDPROCEDURES,
  Options.USE_DATAQUEUES, _
  Options.USE_PROGRAMCALL, _
  Options.USE_RECORDLEVELACCESS

' set two dummy parameters (required by this stored procedure on AS/400)
' set CommandText property to identify stored procedure to use on AS/400
'------------------------------------------------------------------------
Set P1 = MyADO.CreateCommandParm(ParmName:="DUMMY1", _
  ParmType:=adChar, _
  ParmSize:=1)
 
.Parameters.Append P1
.Parameters("DUMMY") = "$"

Set P1 = MyADO.CreateCommandParm(ParmName:="DUMMY2", _
  ParmType:=adChar, _
  ParmSize:=1)

.Parameters.Append P1
.Parameters("DUMMY2") = "$"

  vStoredProcedure:=STOREDPROC-customers, _
  vParmCount:=2)

' create SQL statement, assign to CommandText
'------------------------------------------------------------------------
Case Options.USE_SQL

SQL = "SELECT CID, CFIRST, CINIT, CLAST, CADDR1, CADDR2, CCITY,
  CSTATE, CZIP, CDID FROM"
SQL = SQL & BLANK
SQL = SQL & Options.FilesLibrary
SQL = SQL & SQLDELMITERS
SQL = SQL & DBFILE-CUSTOMER
SQL = SQL & BLANK
SQL = SQL & "WHERE CDID="
SQL = SQL & ADOTEST-DISTRICTID
SQL = SQL & BLANK
SQL = SQL & "AND CWID="
SQL = SQL & QUOTE
SQL = SQL & ADOTEST-WAREHOUSEID
SQL = SQL & QUOTE
SQL = SQL & BLANK
SQL = SQL & "ORDER BY CID"
SQL = SQL & BLANK
SQL = SQL & "FOR FETCH ONLY"

.CommandText = SQL
 .Prepared = True
End Select
End With
End Sub

4.7.4.3 Code Used in the GetCustomers Procedure
This procedure executes the CommandText that was formatted in the Initialize procedure. The stored procedure or SQL statement fills the Recordset object. All of the rows in the Recordset are added to the customers collection as individual customer objects.

Note: the StartAt option is not implemented in this program. If you want to use this feature, you will have to modify the stored procedure or the SQL statement to work with the StartAt value.

Public Sub GetCustomers(ByVal StartAt,  
  Optional ByVal vADOConnection, _  
  Optional ByVal vErrors)
  '------------------------------------------------------------------------
  'get the selected set of customers
  '------------------------------------------------------------------------

  'if object references not passed in, use properties
  '------------------------------------------------------------------------
  If Not IsMissing(vADOConnection) Then
    Set ADOConnection = vADOConnection
  End If

  If Not IsMissing(vErrors) Then
    Set Errors = vErrors
  End If

  'execute command to get data into customer recordset
  '------------------------------------------------------------------------
  Set rsCustomer = Nothing
  Set rsCustomer = cmCustomer.Execute

  'iterate through recordset, add to customer collection
  '------------------------------------------------------------------------
  Call Clear

  With rsCustomer
    If (Not .BOF) And (Not .EOF) Then
      Do Until .EOF
        Set Customer = Add(.Fields("CID"), _  
          .Fields("CFIRST"), _  
          .Fields("CLAST"), _  
          .Fields("CINIT"), _  
          .Fields("CADDR1"), _  
          .Fields("CADDR2"), _  
          .Fields("CCITY"), _  
          .Fields("CSTATE"), _  
          .Fields("CZIP"))

        .MoveNext
      End If
    End If
  End With

End Sub
Loop
End If

End With

End Sub

4.7.5 ADO Code in Class CDistrict

The code in this class is used to access the district database. This database is used to retrieve the next available order number. The code also increments the order number.

The code is similar to the examples shown in the CCustomer class:

- The General Declarations section contains definitions of the ADO Command and Recordset objects used for the class.
- The Initialize procedure is used to initialize the Command object.
- The GetNextOrder procedure uses an SQL SELECT statement to get the next available order number, followed by an SQL UPDATE statement to update the order number.

4.7.6 ADO Code in Class CErrors

This class is used to hold an errors collection. The errors collection exposed by this class is a superset of the ADO Errors collection, in that CErrors includes both Visual Basic errors and ADO errors. Multiple errors returned from an ADO operation in the ADO Errors collection are available through CErrors.

The primary function of the CErrors class is to store all of the errors that result from an operation, add descriptive information to the error if available and categorize the severity of the error. Error severity is categorized as:

- **Recoverable** — The error will not halt execution of the program. An example of a recoverable error is a record not found.
- **Nonrecoverable** — The program cannot continue because of the error that occurred. An example of this type of error is an invalid Connection string for an ADO Connection object.

4.7.6.1 Code Used in the General Declarations Section

The code used in this section is used to define the ADO Connection object.

```
Private m_ADOConnection As ADODB.Connection
```

4.7.6.2 Code used in the AddError procedure

This procedure is used to add all available errors (Visual Basic and ADO errors) to the Errors collection exposed by this class.

The ADO Connection object can optionally be passed to this procedure. If the Connection object reference is not passed, the assumption is that only Visual Basic errors should be added to the collection. If the Connection object reference is passed, all ADO errors currently associated with the Connection object are added to the collection.

```
Public Sub AddError(ByVal ModuleName, ByVal RoutineName, ByVal Connection
```

```
Public Sub AddError(ByVal ModuleName, ByVal RoutineName, ByVal Connection
```
ByVal ErrorNumber, _
ByVal ErrorDescription, _
ByVal ErrorSource, _
Optional ByVal vConnection)

Dim ADOError As ADODB.Error
Dim LogError
Dim RaiseErrorFlag

' set value for optional parameters -- if not passed in use property
If Not IsMissing(vConnection) Then
    Set ADOConnection = vConnection
End If

' initialize RaiseErrorFlag -- if any unrecoverable errors are encountered,
' set to True to raise the error event (displays errors form)
RaiseErrorFlag = False

' add VB error to the error collection.
' m_ShortDescription, m_Recovery are set in SetErrorParms.
Call SetErrorParms (ErrorNumber:=ErrorNumber, _
    ErrorDescription:=ErrorDescription)

If m_LogError Then
    Add ModuleName:=ModuleName, _
        RoutineName:=RoutineName, _
        Number:=ErrorNumber, _
        Description:=ErrorDescription, _
        Source:=ErrorSource, _
        RecoveryPossible:=m_RecoveryPossible, _
        ReportedBy:=ERROR_VB, _
        MessageID:=m_MessageID, _
        MessageIDShort:=m_MessageIDShort
    If m_RecoveryPossible = ERROR_RECOVERY_NO Then
        RaiseErrorFlag = True
    End If
End If

' add ADO error(s) to the error collection
' tests for existence of connection -- if connection object not yet
' instantiated, then just report VB errors.
If Not (ADOConnection Is Nothing) Then
    For Each ADOError In ADOConnection.Errors
        Call SetErrorParms (ErrorNumber:=ADOError.Number, _
            ErrorDescription:=ADOError.Description)
    End For
End If
RoutineName:=RoutineName, _
Number:=ADOError.Number, _
Description:=ADOError.Description, _
Source:=ADOError.Source, _
RecoveryPossible:=m_RecoveryPossible, _
ReportedBy:=ERROR_ADO, _
MessageID:=m_MessageID, _
MessageIDShort:=m_MessageIDShort

If m_RecoveryPossible = ERROR_RECOVERY_NO Then
    RaiseErrorFlag = True
End If

End If
Next
End If

'--------------------------------------------------------------------------
'advise sinks of error(s) if any were logged
'--------------------------------------------------------------------------
If (mCol.Count > 0) And (RaiseErrorFlag = True) Then
    RaiseEvent HaveError
End If

End Sub

4.7.7 ADO Code in Classes CItem and CItems
The CItem and CItems classes are clones of the CCustomer and CCustomers class.

The CItem class is used to perform database operations on an individual record in
the item database. The processing techniques include record-level access and
SQL statements.

• The CItems class is used to work with a set of item records. Processing
  techniques include stored procedure call and SQL statements.

4.7.8 ADO Code in Class CLineItems
The CLineItems collection class is used to contain the list of items in an order and
to add those items to the order lines database. An SQL INSERT statement is used
to add each of the line items to the database.

The Stock database is also updated when the line items are added to the
database. For each line item added, the quantity on hand in the corresponding
item record in the Stock database is decremented.

4.7.8.1 Code Used in the General Declarations Section
The code used in this section is used to define the ADO Connection, Command and
Recordset objects.

Private cmOrderLines As ADODB.Command
Private rsOrderLines As ADODB.Recordset
Private m_ADOConnection

4.7.8.2 Code Used in the AddToDatabase Procedure
The `AddToDatabase` procedure is used to add each order line item to the order line item database. The procedure first defines the SQL statement and creates ADO Parameter objects for each of the parameter markers in the SQL statement. After preparing the SQL statement, the procedure iterates over the line items collection and performs the `INSERT` statement for each line item.

For each item inserted, the `UpdateStock` method of the `CStock` collection is called to decrement the quantity on hand for the item.

```
Public Sub AddToDatabase(ByVal OrderNumber, ByVal CustomerDiscount)
    '--------------------------------------------------------------------------
    'add all of the line items in this collection to the Order Lines database
    '--------------------------------------------------------------------------
    Dim ItemCount
    Dim P1 As ADODB.Parameter
    Dim SQL
    '--------------------------------------------------------------------------
    'create command object, set for use
    '--------------------------------------------------------------------------
    Set cmOrderLines = New ADODB.Command
    With cmOrderLines
        Set .ActiveConnection = ADOConnection
        .CommandType = adCmdText

        Set .ActiveConnection = ADOConnection
        .CommandType = adCmdText
```
SQL = "INSERT INTO"
SQL = SQL & BLANK
SQL = SQL & Options.FilesLibrary
SQL = SQL & SQLDELEIMITER
SQL = SQL & DBFILE_ORDERLINES
SQL = SQL & BLANK
SQL = SQL & "(OLOID, OLDID, OLWID, OLNBR, OLSPWH, OLIID, OLQTY, OLAMNT, OLDLVD, OLDLVT)"
SQL = SQL & BLANK

.CommandText = SQL
.Prepared = True

'create/append parameters to command object
'------------------------------------------------------------------------

'------------------------------------------------------------------------

'order ID - packed(9,0)
'------------------------------------------------------------------------

Set P1 = MyADO.CreateCommandParm(ParmName:="OLOID", _
ParmType:=adDecimal, _
ParmSize:=9)

.Parameters.Append P1

'district ID - packed(3,0)
'------------------------------------------------------------------------

Set P1 = MyADO.CreateCommandParm(ParmName:="OLDID", _
ParmType:=adDecimal, _
ParmSize:=3)

.Parameters.Append P1

'warehouse ID - alpha(4)
'------------------------------------------------------------------------

Set P1 = MyADO.CreateCommandParm(ParmName:="OLWID", _
ParmType:=adChar, _
ParmSize:=4)

.Parameters.Append P1

'order line number - packed(3,0)
'------------------------------------------------------------------------

Set P1 = MyADO.CreateCommandParm(ParmName:="OLNBR", _
ParmType:=adDecimal, _
ParmSize:=3)

.Parameters.Append P1

'supply warehouse - alpha(4)
'------------------------------------------------------------------------

Set P1 = MyADO.CreateCommandParm(ParmName:="OLSPWH", _
ParmType:=adChar, _
ParmSize:=4)

.Parameters.Append P1
Set P1 = MyADO.CreateCommandParm(ParmName:="OLIID", _
      ParmType:=adChar, _
      ParmSize:=6)
 Parameters.Append P1

Set P1 = MyADO.CreateCommandParm(ParmName:="OLQTY", _
      ParmType:=adNumeric, _
      ParmSize:=3)
 Parameters.Append P1

Set P1 = MyADO.CreateCommandParm(ParmName:="OLAMNT", _
      ParmType:=adNumeric, _
      ParmSize:=7, _
      ParmNumericScale:=2)
 Parameters.Append P1

Set P1 = MyADO.CreateCommandParm(ParmName:="OLDLVD", _
      ParmType:=adNumeric, _
      ParmSize:=8)
 Parameters.Append P1

Set P1 = MyADO.CreateCommandParm(ParmName:="OLDLVT", _
      ParmType:=adNumeric, _
      ParmSize:=6)
 Parameters.Append P1

End With

ItemCount = 0
For Each LineItem In mCol
  ItemCount = ItemCount + 1
'set parameter values, execute INSERT to add line item to ORDLIN database

With cmOrderLines

    .Parameters("OLOID") = OrderNumber
    .Parameters("OLDID") = ADOTEST_DISTRICTID
    .Parameters("OLWID") = ADOTEST_WAREHOUSEID
    .Parameters("OLNBR") = ItemCount
    .Parameters("OLSPWH") = "OLED"
    .Parameters("OLIID") = CStr(Format(LineItem.ItemID, "000000"))
    .Parameters("OLQTY") = LineItem.Quantity
    .Parameters("OLAMNT") = LineItem.Price - (LineItem.Price * CustomerDiscount / 100)
    .Parameters("OLDLVD") = 1231999
    .Parameters("OLDLVT") = 235959

    Set rsOrderLines = Nothing
    Set rsOrderLines = .Execute

End With

'update STOCK database for this item

With MyStock

    .ItemID = CStr(Format(LineItem.ItemID, "000000"))
    .WarehouseID = ADOTEST_WAREHOUSEID
    .Quantity = Format(LineItem.Quantity, "000")

    .UpdateStock

End With

Next

End Sub
4.7.9 ADO Code in Class COrder

The COrder class is used to submit an order for batch processing on the AS/400. The class supports two techniques for processing the order:

**Distributed Program Call** — A call is made to an AS/400 program to process the order. Two parameters are passed to the program to indicate the customer number and the line items from the order.

**SQL Statements with Data Queue** — This technique directly updates the AS/400 database files using SQL statements. It then sends an entry to a data queue which is being monitored by an AS/400 batch program. The AS/400 batch program prints the order.

4.7.9.1 Code Used in the General Declarations Section

The code used in this section is used to define the ADO Connection object.

Private m_ADOConnection

4.7.9.2 Code used in the ProcessOrderProgramCall procedure

This procedure is called if the selected processing technique is Distributed Program Call. The procedure formats the program name and creates two Parameter objects that are appended to the Parameters collection associated with the ADO Command object that executes the program call.

Note that the second parameter is actually a list of the line items that were entered in the order. Because all of the line items are put into one parameter string, there is a limit to the number of line items that can be processed for the order.

Private Sub ProcessOrderProgramCall()

'process the order - use Distributed Program Call technique

    Dim CustParm
    Dim N
    Dim OrderParm
    Dim ParmLength
    Dim ParmWork
    Dim StartPos

    Dim cmProcessOrder As New ADODB.Command
    Dim P1 As New ADODB.Parameter
    Const PARM_SET_LENGTH = 40

    'set properties for Program Call command object
    With cmProcessOrder
        Set .ActiveConnection = ADOConnection
        .CommandType = adCmdText
        .Prepared = True
    End With

    'Prepare the command object
    cmProcessOrder.CommandText = "Call subroutine" & N & "@" & CustParm & OrderParm
    cmProcessOrder.Parameters.Append(P1, ParmLength, ParmWork, StartPos)

    'Execute the command object
    cmProcessOrder.Execute
End Sub
'format name of program to call
 '-------------------------------------------------------------------------
 '.CommandText = MyMakeString.MakeProgramName(vLibrary:=MyOptions.ProgramsLibrary, _
 vProgram:=PROGRAM_ORDERPROCESS, _
 vParmCount:=2)
 '-------------------------------------------------------------------------
 'set parameter 1 for program call
 '    Positions 1 - 4: customer ID
 '    Positions 5 - 9: number of line items (up to 50)
 '-------------------------------------------------------------------------
 CustParm = CStr(Format(CustomerID, "0000")) & _
 CStr(Format(IIf(LineItems.Count > ADOTEST_LINE_ITEM_LIMIT, _
 ADOTEST_LINE_ITEM_LIMIT, _
 LineItems.Count), _
 "00000"))

Set P1 = MyADO.CreateCommandParm(ParmName:="CUSTPARM", _
 ParmType:=adChar, _
 ParmSize:=9)

'.Parameters.Append P1
'.Parameters("CUSTPARM") = CustParm
 '-------------------------------------------------------------------------
 'set parameter 2 for program call (up to 50 in set)
 '    Positions 1 -  6: item number
 '    Positions  7 - 30: item description
 '    Positions 31 - 35: price
 '    Positions 36 - 40: quantity
 '-------------------------------------------------------------------------
 OrderParm = Space(PARM_SET_LENGTH * ADOTEST_LINE_ITEM_LIMIT)
 N = 0
 StartPos = 1
 For Each LineItem In LineItems
  N = N + 1
  If (N <= ADOTEST_LINE_ITEM_LIMIT) Then
   ParmWork = Space(PARM_SET_LENGTH)
   Mid(ParmWork, 1, 6) = CStr(Format(LineItem.ItemID, "000000"))
   Mid(ParmWork, 7, 24) = LineItem.ItemName
   Mid(ParmWork, 31, 5) = CStr(Format(LineItem.Price * 100, _
   "00000"))
   Mid(ParmWork, 36, 5) = CStr(Format(LineItem.Quantity, "00000"))
   Mid(OrderParm, StartPos, PARM_SET_LENGTH) = ParmWork
   StartPos = StartPos + PARM_SET_LENGTH
  End If
 Next

Set P1 = MyADO.CreateCommandParm(ParmName:="ORDERPARM", _
 ParmType:=adChar, _
 ParmSize:=PARM_SET_LENGTH * ADOTEST_LINE_ITEM_LIMIT)
.Parameters.Append P1
.Parameters("ORDERPARM") = OrderParm

'execute the command (calls the program)
.Execute

End With
Exit Sub

End Sub

4.7.9.3 Code Used in the ProcessOrderSQL Procedure

This procedure is called if the selected processing technique is SQL. The procedure calls procedures in the other classes associated with each database file to perform the inserts or updates required for the order.

The procedure finishes by formatting the CommandText used to process a data queue using the AS/400 OLE DB Provider. Note that the data being passed with the data queue is formatted in the MakeDTAQName procedure in class CMakeString.

Private Sub ProcessOrderSQL()

'-------------------------------------------------------------------------
'process the order - use SQL, also Data Queue for order printing
'-------------------------------------------------------------------------
    Dim dqCommandText
    Dim FieldName
    Dim FieldType
    Dim FieldLength
    Dim FieldValues
    Dim OrderNumber
    Dim OrderTotal

    Dim MyOrderHeader As New COrderHeader
    Dim cmDTAQ As New ADODB.Command
    Dim rsDTAQ As New ADODB.Recordset

    '-------------------------------------------------------------------------
    'get next order number
    '-------------------------------------------------------------------------
    OrderNumber = MyDistrict.GetNextOrder(vDistrictID:=ADOTEST_DISTRICTID, _
        vWarehouseID:=ADOTEST_WAREHOUSEID)

    '-------------------------------------------------------------------------
    'get customer record
    '-------------------------------------------------------------------------
    Call MyCustomer.GetCustomer(vCustomerID:=CustomerID)

    '-------------------------------------------------------------------------
    'add line items to order lines database
    '-------------------------------------------------------------------------
    Call LineItems.Initialize(vADOConnection:=ADOConnection, _
        vErrors:=MyErrors, _
        vOptions:=MyOptions)
    Call LineItems.AddToDatabase(OrderNumber:=OrderNumber, _
        CustomerDiscount:=MyCustomer.Discount)
'add order header record
Call MyOrderHeader.Initialize(vADOConnection:=ADOConnection, _
    vErrors:=MyErrors, _
    vOptions:=MyOptions)
Call MyOrderHeader.AddToDatabase(vOrderNumber:=OrderNumber, _
    vCustomerID:=CustomerID, _
    vOrderLines:=LineItems.Count)
'calculate order total
For Each LineItem In LineItems
    OrderTotal = OrderTotal + _
        (LineItem.Price * LineItem.Quantity)
Next
'update customer record
With MyCustomer
    .DateLastUpdated = Format(Month(Now), "00") & _
        Format(Day(Now), "00") & _
        Format(Year(Now), "0000")
    .TimeLastUpdated = Format(Hour(Now), "00") & _
        Format(Minute(Now), "00") & _
        Format(Second(Now), "00")
    .CurrentBalance = .CurrentBalance + OrderTotal
    .YearToDate = .YearToDate + OrderTotal
    .UpdateCustomer
End With
'write entry to data queue to tell batch order processing program to process
'this order
FieldName = Array("CUSTOMERID", _
    "DISTRICTID", _
    "WAREHOUSEID", _
    "ORDERID")
'***ERROR NOTE
'5/21/98-CP
'numeric fields written to dtaq have negative sign (last half-byte is x'C')
'example: DistrictID=1, packed(3,0) is output to dtaq as x'000C' not x'000F'
'problem also occurs for zoned, output is x'F0F0C1'
'THIS ONLY OCCURS FOR SOME AS/400s DEPENDING ON OS/400 AND PTF.
'CHECK FOR CURRENT DTAQ PTFs. CP-5/23/98

FieldType = Array(MyMakeString.DTAQ_CHAR, _
    MyMakeString.DTAQ_NUMERIC, _
    MyMakeString.DTAQ_CHAR, _
    MyMakeString.DTAQ_NUMERIC)

FieldLength = Array("4", _
    "3,0", _
    "4", _
    "9,0")

FieldValues = Array(CustomerID, _
    ADOTEST_DISTRICTID, _
    ADOTEST_WAREHOUSEID, _
    OrderNumber)

With cmDTAQ
    .CommandText = MyMakeString.MakeDTAQName(vLibrary:=Options.DataQueuesLibrary, _
        vDTAQ:=DTAQ_ORDERPROCESS, _
        FieldName:=FieldName, _
        FieldType:=FieldType, _
        FieldLength:=FieldLength, _
        Direction:=MS_DTAQDirection.Send)
    .CommandType = adCmdText
    Set .ActiveConnection = ADOConnection

'first Execute opens the data queue
Set rsDTAQ = .Execute
rsDTAQ.AddNew FieldName, FieldValues

'this Execute sends the entry to the data queue
Set rsDTAQ = .Execute

End With

Exit Sub

End Sub
4.7.10 ADO Code in Class COrderHeader

An order header record is inserted into the database when the order is processed. This class uses an SQL INSERT statement to insert the order header record.

4.7.11 ADO Code in Class CStock

The CStock class is used to update inventory records for items when the order is processed. The update is accomplished using an SQL SELECT and UPDATE statement.

4.7.11.1 Code Used in the General Declarations Section

The code used in this section is used to define the ADO Connection, Command and Recordset objects.

Private m_ADOConnection As ADODB.Connection
Private cmStockSelect As ADODB.Command
Private cmStockUpdate As ADODB.Command
Private rsStockSelect As ADODB.Recordset

4.7.11.2 Code Used in the Initialize Procedure

The Initialize procedure is used to prepare the SQL SELECT and UPDATE statements and to initialize the Command objects that will execute the statements. Because it is likely that the UpdateStock procedure in this class will be called multiple times for an order, the Initialize routine is used so that the preparation of the Command objects only happens once.

Public Sub Initialize(Optional ByVal vADOConnection, _
Optional ByVal vErrors, _
Optional ByVal vOptions)
'--------------------------------------------------------------------------
'initializations for CStock - put here rather than Class_Initialize so that
'this code is not run each time a Stock object is created
'--------------------------------------------------------------------------
Dim SQL
Dim P1 As ADODB.Parameter
'if object references not passed in, use properties
'--------------------------------------------------------------------------
If Not IsMissing(vADOConnection) Then
    Set ADOConnection = vADOConnection
End If
If Not IsMissing(vErrors) Then
    Set Errors = vErrors
End If
If Not IsMissing(vOptions) Then
    Set Options = vOptions
End If
'--------------------------------------------------------------------------
'create objects used in this class
'--------------------------------------------------------------------------
Set MakeString = New CMakeString
Set cmStockSelect = New ADODB.Command
Set cmStockUpdate = New ADODB.Command
'create the SQL statement for retrieval
-------------------------------------------------------------------------
SQL = "SELECT STQTY FROM"
SQL = SQL & BLANK
SQL = SQL & Options.FilesLibrary
SQL = SQL & SQLDELMITER
SQL = SQL & DBFILE_STOCK
SQL = SQL & BLANK
SQL = SQL & "WHERE STWID="
SQL = SQL & QUOTE
SQL = SQL & ADOTEST_WAREHOUSEID
SQL = SQL & QUOTE
SQL = SQL & BLANK
SQL = SQL & "AND STIID="
SQL = SQL & SQLMARKER

'-------------------------------------------------------------------------

'set the command object for select operation
-------------------------------------------------------------------------
With cmStockSelect
    Set .ActiveConnection = ADOConnection
    .CommandType = adCmdText
    .CommandText = SQL
    .Prepared = True

    Set P1 = MyADO.CreateCommandParm(ParmName:="STIID", _
        ParmType:=adChar, _
        ParmSize:=6)

    .Parameters.Append P1
End With

'-------------------------------------------------------------------------

'create the SQL statement for update
-------------------------------------------------------------------------
SQL = "UPDATE"
SQL = SQL & BLANK
SQL = SQL & Options.FilesLibrary
SQL = SQL & SQLDELMITER
SQL = SQL & DBFILE_STOCK
SQL = SQL & BLANK
SQL = SQL & "SET STQTY=?"
SQL = SQL & BLANK
SQL = SQL & "WHERE STWID="
SQL = SQL & QUOTE
SQL = SQL & ADOTEST_WAREHOUSEID
SQL = SQL & QUOTE
SQL = SQL & BLANK
SQL = SQL & "AND STIID="
SQL = SQL & SQLMARKER

'-------------------------------------------------------------------------

'set the command object for update operation
-------------------------------------------------------------------------
With cmStockUpdate
    Set .ActiveConnection = ADOConnection
    .CommandType = adCmdText
    .CommandText = SQL
    .Prepared = True
Set P1 = MyADO.CreateCommandParm(ParmName:="STQTY", _
            ParmType:=adDecimal, _
            ParmSize:=5)
.Parameters.Append P1

Set P1 = MyADO.CreateCommandParm(ParmName:="STIID", _
            ParmType:=adChar, _
            ParmSize:=6)
.Parameters.Append P1

End With

End Sub

4.7.11.3 Code Used in the UpdateStock Procedure

This procedure is used to retrieve the item record and update the quantity on hand. The SQL SELECT statement is executed to retrieve the record. The quantity on order is then subtracted from the on-hand quantity. The SQL UPDATE statement is executed to update the on-hand quantity in the database.

Public Sub UpdateStock(Optional ByVal vWarehouseID, _
                      Optional ByVal vItemID, _
                      Optional ByVal vQuantity)
    '-------------------------------------------------------------------------
    'update Stock record
    '-------------------------------------------------------------------------
    Dim NewQuantity
    '-------------------------------------------------------------------------
    'set values if optional parameters passed
    '-------------------------------------------------------------------------
    If Not IsMissing(vWarehouseID) Then
        WarehouseID = vWarehouseID
        End If
    If Not IsMissing(vItemID) Then
        ItemID = vItemID
        End If
    If Not IsMissing(vQuantity) Then
        Quantity = vQuantity
        End If
    '-------------------------------------------------------------------------
    'update stock quantity for this item
    '-------------------------------------------------------------------------
    With cmStockSelect
        .Parameters("STIID") = CStr(Format(ItemID, "000000"))
    Set rsStockSelect = Nothing
    Set rsStockSelect = .Execute
    NewQuantity = rsStockSelect.Fields("STQTY") - Quantity
    End With

End Sub
With cmStockUpdate
  .Parameters("STQTY") = rsStockSelect.Fields("STQTY") - Quantity
  .Parameters("STIID") = CStr(Format(ItemID, "000000"))
  .Execute
End With

End Sub

4.7.12 Final Notes about this Sample

The sample Order Entry using ADO application was designed to show some of the techniques that are available to you if you use the AS/400 OLE DB Provider. The sample includes the following techniques:

- Record-Level Access
- SQL statement processing
- Stored procedure calls
- Distributed program calls
- Data queue send operation

The only AS/400 OLE DB Provider technique not illustrated is AS/400 Command execution, which is a variation of the distributed program call technique.

4.7.12.1 Error Handling

Early in the project, it became apparent that a comprehensive error handler was required. This is because there are multiple sources of error when using ADO. You need to be prepared to handle both Visual Basic errors and the errors that are reported in the ADO Errors collection.

When working with the ADO Errors collection, you need to examine all of the errors. In some instances, the most descriptive error is the last error in the collection, not the first. You will have to decide how to handle multiple errors in your program. You may find it worthwhile to closely examine the CErrors class and frmErrors form provided with this sample to see how multiple errors can be handled and reported.

4.7.12.2 SQL Statement Coding

If you look at any of the procedures where an SQL statement is created you will see that the statement is built up an element at a time. Also, constants are used in place of literals such as spaces, apostrophes (single quote marks) and commas.

It is usually very difficult to code an SQL statement correctly, especially when you have to include variables in the statement. You must remember to put single quotation marks around character literals and not around numerics, for example. Also, when you are building an SQL statement, you need to be sure to have blank spaces between SQL clauses.

Although it may look very cumbersome to construct the SQL statements as shown in the sample program, it is very easy to make corrections to the statement when an error was encountered.

You may also want to examine the MakeSQLNameValue procedure in class CMakeString to see if you can use it to create name/value pairs in your INSERT and
UPDATE statements. One of the primary advantages of `MakeSQLNameValue` is that it correctly handles embedded apostrophes in data values and returns them as double apostrophes.

4.7.12.3 Use of Variants and ByVal
Practically all Visual Basic programmers will strongly object to the use of untyped variables in the sample program. This was purposely done after careful consideration. Data typing in this application was not necessary and given the nature of the application, there is absolutely no discernable performance penalty in using Variants.

The use of `ByVal` for procedure parameters was also done deliberately. By using `ByVal`, any inadvertent modifications to parameters in the called procedure are not reflected back to the calling procedure. In all cases where a value needed to be returned, the procedure was either used as a function or values were set in objects that the calling procedure could then use.

If you are interested in a much more comprehensive discussion of these Visual Basic programming issues, refer to the book *Advanced Visual Basic 5* (The Mandlebrot Set, Ltd., Microsoft Press, ISBN 1-57231-414-1), especially Chapter 5, "Changing Your Approach to Visual Basic Coding".

4.7.12.4 Use of Classes for All Database Access
The most significant design decision in this application was the removal of all of the database access code from the forms. Data is only available to forms through the objects, which are used as proxies to the database.

This technique provides an abstracted view of the database to the forms: they can only see data that is provided through the objects and they can only manipulate data by using methods in the objects.

Using this technique it will be relatively simple to change the database access technique if required. It will also be easy to reuse the code, since there is nothing specific about this application in the classes that access the database.
Chapter 5. PowerBuilder Using OLE/DB

PowerBuilder is a graphical application development environment that can be used to develop AS/400 client/server applications. The PowerBuilder ODBC interface lets you access the DB2/400 database using Open Database Connectivity (ODBC), Microsoft's standard for database connectivity. This can be done by installing the Client Access ODBC driver and defining an ODBC data source.

OLE DB is a set of application programming interfaces (APIs) that provide access to a wide range of data sources. ODBC provides access only to information in a relational database through SQL, while OLE DB is a method to access data via a standard COM (Component Object Model) Interface. OLE DB provides both the SQL functionality defined in ODBC and interfaces suitable to access non-SQL resources.

IBMDA400 is an OLE DB provider to AS/400 data and resources. It is a part of Client Access version V3R1M3. It includes support to interface with the following AS/400 resources:

- Table record-level access
- Data queues
- Programs
- CL commands
- SQL statements
- SQL stored procedures

IBMDA400, the Client Access OLE DB provider, has been written to the OLE DB version 1.1 specification and has been tested to work with ADO 1.0. For more details on the ADO programming model, see Chapter 1, “OLE DB and ActiveX Overview” on page 1. In this chapter we show you, through some examples the way to use IBMDA400, the Client Access OLE DB provider, with PowerBuilder Version 6.0. It is assumed that you are familiar with the ADO programming model.

5.1 Introduction to Application Examples

The rest of this chapter covers application examples which demonstrate:

AS/400 access via:

- SQL
- Stored Procedures
- Table Record Level Access
- AS/400 Data Queues
- Remote Commands
- Program Calls on the AS/400

The database access example applications discussed in the remainder of this chapter allow retrieval of a single record or all the records from a PARTS file on the AS/400 system. Some examples allow an update to the current Record in the
window. These examples have been tested to work with PowerBuilder 6.0 and ADO 1.0.

Tip

For additional information about programming with ADO, see the SDK Technical Reference Manual and the Microsoft online help. See Appendix A.2, “Other Documentation and Resources” on page 373 for details on how to obtain this documentation.

We use the PARTS database file in the APILIB library in all the examples. The PARTS file is defined as:

Table 29. PARTS File

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Description</th>
<th>Length</th>
<th>Decimals</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARTNO</td>
<td>Part Number</td>
<td>5</td>
<td>0</td>
<td>Zoned</td>
</tr>
<tr>
<td>PARTDS</td>
<td>Part Description</td>
<td>25</td>
<td></td>
<td>Char</td>
</tr>
<tr>
<td>PARTQY</td>
<td>Quantity</td>
<td>5</td>
<td>0</td>
<td>Packed</td>
</tr>
<tr>
<td>PARTPR</td>
<td>Price</td>
<td>6</td>
<td>2</td>
<td>Packed</td>
</tr>
<tr>
<td>PARTDT</td>
<td>Part Shipment Date</td>
<td>10</td>
<td></td>
<td>Date</td>
</tr>
</tbody>
</table>

All the examples discussed in this chapter are available for download on the www.redbooks.ibm.com web-site. See Appendix A.1, “Downloading the Samples” on page 373 for details.

5.2 An SQL Application Example

IBMDA400, the Client Access OLE DB provider, allows you to send remote SQL statements to the AS/400 server. There is support for executing most SQL statements on the AS/400 system. For more information on SQL statements see the DB2 for AS/400 SQL Programming Manual, SC41-5611.

You can execute SQL statements using the Execute method of the Connection Object. If you are executing SQL statements with parameters, for example in a where clause, you should use parameter markers and prepare the SQL statement before executing it. This improves performance because the SQL statement is not prepared every time you execute it.

The SQL SELECT statement and SQL CALL statement, for calling a stored procedure, are the only two SQL statements that generate a result set, and thus a Recordset object. The Recordset Object created is not updatable. Therefore, the AddNew, Update, and delete methods are not valid.

Note

- SQL naming is required.
- IBMDA400 does not support the SELECT FOR UPDATE statement.
- IBMDA400 does not support commitment control for SQL statements.
This example is an illustration of how to use IBMDA400, the Client Access OLE DB provider, to access DB2/400 data using SQL. In this example, the client program requests data from the AS/400 database by sending SQL statements to the AS/400 system. The host server executes the SQL statement and returns the results to the client program in an SQL result set. Two SQL statements are used:

```
Select * from apilib.parts
Select * from apilib.parts where partno = ?
```

to retrieve all columns for all records and to retrieve all columns for a given part number, respectively.
The following screen shows the result of requesting one part (12345) from the PARTS file:

![Figure 79. Get Part Using SQL](image1)

The above screen shows the result of requesting all the parts from the PARTS file.

![Figure 80. Get All Parts Using SQL](image2)

In the following sections, we discuss the step by step implementation of the application we just described.
5.2.1 Opening a Connection
The first step is to define a connection object. You need to define at least one Connection Object in an application.

```java
Long ll_status
Partsrcddata = CREATE OLEObject
ll_status = cn400.ConnectToNewObject ("adodb.connection")
cn400.Open("Provider=IBMDA400;Data Source="TCPASM05;">", "", "")
```

First you create the connection object. You should declare these ADO objects as global variables of type OleAuto.

After creating the connection object, you invoke its open method. The ConnectionString part contains the provider and data source names. The Data Source name must be the name of a valid AS/400 connection. For example, the name of a connection configured using the Client Access AS/400 connection program.

Empty strings are passed for the User ID and Password parts. The IBMDA400 provider does not support the passing of User ID and Password when you open a connection. Client Access prompts the user for connection information as it is needed.

5.2.2 Get One Part-SQL
To get one part you have to query the PARTS file based on the part number that you enter.

For this you have to create Command and Recordset objects. Then you have to prepare the Command object such that it is ready to be executed.

```java
Partsrcd = CREATE OLEObject
ll_status = Partsrcd.ConnectToNewObject ("adodb.command")
```

```java
Partsrcddata = CREATE OLEObject
ll_status = Partsrcddata.ConnectToNewObject ("adodb.recordset")
```

```java
Partsrcd.ActiveConnection = cn400
Partsrcd.CommandText = "select * from APILIB.PARTS where PARTNO = ?"
Partsrcd.Prepared = True
```

You have to set the ActiveConnection property of the Command object to the Connection that you opened in the first step. The next step is to set the CommandText property of Command Object to the SQL statement to be executed. Use parameter markers(?) for the variables that will be used. Using parameter
markers improves performance as the prepared SQL statement is used at the time of execution.

**Note**

The ADO support shipped with the Client Access OLE DB provider is ADO 1.0. If you upgrade to a higher level, for example ADO 1.5, you should add the following statement after executing the Prepared method:

```
Partsrcd.Parameters.Refresh
```

![Image of a form with fields for Part Number, Part Description, Quantity, Price, Received Date, Start Time, and End Time, with buttons for Get Part and Get All Parts.](image)

*Figure 81. Get One Part Information Based on Part Number Entered*

The next step is to execute the prepared SQL statement. When you enter the value of part number and press the button *Get Part*, the SQL that has been prepared above gets executed. The parameter marker is replaced by the value of part number you entered. See Figure 81.

```vbnet
Any Rcds, Parms
Long parm
parm = Long(sle_1.Text)
Parms = parm
parm = Parms
Rcds = "0"
Partsrcddata = Partsrcd.Execute(Rcds, Parms, 1)
```

The parameters passed to the *Execute* method of Command Object are `RecordsAffected`, `parameter value` and `cmdText`. `RecordsAffected` is not supported by IBMDA400, but you have to pass it. The second parameter is the value that you want to be substituted for parameter marker used in the SQL statement that you prepared in the first step. The third parameter indicates the type of `cmdText`. A value of 1 indicates that it is a textual definition of a command. For a list of constant values, see the *Execute* Method topic of the *SDK Technical Reference* and ADO online help. See Appendix A.2, “Other Documentation and
The SQL statement, when executed, returns an SQL result set. This result set is loaded into the ADO Recordset object. Then we populate the fields in the form as shown below.

```plaintext
If Partssrcddata.EOF Then
  sle_2.Text = ""
  sle_3.Text = ""
  sle_4.Text = ""
  sle_5.text = ""
Else
  sle_2.Text = Partssrcddata.Fields(1).Value
  sle_3.Text = Partssrcddata.Fields(2).Value
  sle_4.Text = Partssrcddata.Fields(3).Value
  sle_5.Text = Partssrcddata.Fields(4).Value
End If
```

5.2.3 Get All Parts - SQL

To get all the parts, you have to select all the records from the database file.

```plaintext
***Executing SQL***

Partsdata = cn400.Execute("select * from APILIB.PARTS", Rcds, 1)
```

You can execute the SQL statement to select all the records by using the Execute method of Connection object. The parameters passed are CommandText, RecordsAffected and adCmdText. For details, see the Execute Method of Connection Object in the SDK Technical Reference and ADO online help.
You can prepare this SQL statement as illustrated in Section 5.2.2 "Get One Part-SQL" on page 167. If you choose to prepare this SQL statement, then you won’t have parameter markers and you won’t have to pass parameters to this SQL statement at the time of execution.

This SQL statement returns a result set that is loaded into the ADO Recordset object.

5.2.3.1 Navigating the Result Set
If the execution of the SQL statement produces a result set, the result set data is loaded into the ADO Recordset object. If the result set is empty, the Recordset also is empty and BOF and EOF are true. You can use the MoveFirst, MoveNext and MovePrevious methods to navigate the result set.

The field names in the Recordset can be read like this:

```vbnet
For indx = 1 To <recordset>.Fields.Count - 1
    <value> = <recordset>.Fields(Indx).Name
Next
```

The field values in the current record can be read like this:

```vbnet
If Not <recordset>.BOF And Not <recordset>.EOF Then
    If IsNull(<recordset>.Fields(0).Value) Then
        <value> = ""
    Else
        <value> = <recordset>.Fields(0).Value
    End If
End If
```

Note
- The MoveLast method is not supported for an SQL Recordset.
- An SQL Recordset is not updatable. As a result, AddNew, Update and Delete methods are not valid.
- An SQL Recordset does not support bookmarks.

5.2.3.2 Populate the ListView
Now you have to navigate through the Recordset and populate the data control of your choice. In this application we have chosen to use a ListView data control, and we use a function that populates the ListView.

This function takes the name of the ListView and the Recordset as arguments and populates the ListView.

First, it reads the field names from the Recordset and adds column to the ListView. In the second step, it loops through the Recordset and adds one record at a time.
5.2.4 Error Handling

The Connection object contains the errors collection. All PC client and AS/400 error and warning messages are returned in the error collection at the connection level. You can check for any exception by navigating through the error collection as shown below:

```powerbuilder
For indx = 1 To rs.Fields.Count - 1
    lv.AddColumn(rs.Fields(Indx).Name, Left!, 400)
Next

If rs.EOF And Not rs.BOF Then
    rs.MoveFirst
End If

Do While Not rs.EOF
    lv.AddItem((Trim(rs.Fields(0).Value) + ("" + Trim(rs.Fields(1).Value)) + 
            ("" + Trim(rs.Fields(2).Value)) + ("" + 
            String(rs.Fields(3).Value))+("" + String(rs.Fields(4).Value))), 0)
    rs.MoveNext
Loop

FOR i = 0 TO cn400.errors.count - 1
    msg = ""
    msg = msg + "Number : " + string(cn400.errors.Item(i).Number)
    msg = msg + "Description: " + string(cn400.errors.Item(i).Description)
    MessageBox("Error Message", msg)
Next
```

For more information on Errors, see the Error Object and Errors Collection topics in the SDK Technical Reference and ADO online help.

Use the cwbzztrc.exe program to work with error logging. You can run the program from a DOS command prompt. Run the program without any parameters to get the options and syntax. This program is installed when you install Client Access. The following is the default installation path and log file path:

C:\Program Files\IBM\Client Access\cwbzztrc.exe
C:\Program Files\IBM\Client Access\Service\Trace Files\IBMDA400.log

5.3 A Stored Procedure Application Example

IBMDA400, the Client Access OLE DB provider, allows you to call stored procedures on the AS/400 system. A Stored Procedure is a cataloged program that may be written in most of the programming languages on the AS/400 system. To catalog an AS/400 program you use the CREATE PROCEDURE SQL statement. For
more information on SQL statements and Stored Procedure, see the DB2 for AS/400 SQL Programming Manual, SC41-5611.

A Stored Procedure can be used to control multiple database operations on the server thereby reducing the line turn-arounds. Thus, in client/server applications Stored Procedures can be used to improve performance by reducing the communication overhead. A Stored Procedure may take up to 256 parameters. The output is returned in OUT (or INOUT) parameters or via SQL result sets.

Figure 83. PowerBuilder Application - Stored Procedures
This screen shows the result of requesting one part from the PARTS file.

Figure 84. Get One Part - Stored Procedure

Figure 85. Get All Parts - Stored Procedure

Figure 85 shows the result of requesting all the parts from the PARTS file through a stored procedure call.
Following is an illustration of how to use the Client Access OLE DB provider to call Stored Procedures. In this example, the client program calls a stored procedure SPROC2 in APILIB library. The stored procedure can be created by using the interactive SQL environment on the AS/400 system and entering the following statement.

```
***Create Procedure***

CREATE PROCEDURE APILIB/SPROC2 (  
  INOUT P1 INTEGER,  
  INOUT P2 INTEGER)  
EXTERNAL NAME APILIB/SPROC2 LANGUAGE RPG GENERAL
```

The stored procedure is named SPROC2. It calls an AS/400 RPG program also named SPROC2. This program takes in two integer input/output parameters. Based on the parameters passed it returns either one part or all the parts. For a complete listing of the RPG program, SPROC2, see “SPROC2/SQLRPGLE” on page 375.

A value of 1 for the first parameter returns a single record in the result set with the part number field matching the part number supplied in the second parameter. A value of 2 in the first parameter returns all records from the parts database in a result set. The second parameter is ignored in this case.

### 5.3.1 Opening a Connection

First we create a connection object and open it. This is same as we did in the SQL example of this chapter.

### 5.3.2 Preparing for Stored Procedure Call

Since we use the same stored procedure for both getting one part and getting all parts, we prepare it once and then execute with different input parameters.

After you have opened the connection object, you have to create a Command object and prepare it for a Stored Procedure call as shown below:

```
Partsrsdcmd = CREATE OLEObject  
ll_status = Partsrsdcmd.ConnectToNewObject ("adodb.command")  
Partsrsdcmd.ActiveConnection = cn400  
Partsrsdcmd.CommandText = "{call APILIB.SPROC2(?,?)}"  
Partsrsdcmd.Prepared = True
```

You have seen in the earlier section a discussion on how to prepare a Command object.

Let’s take a look at the CommandText property of the Command object. The braces around the call statement are required as they indicate that it is a stored procedure call. The question marks (?) indicate the parameters to be passed when the Command is executed.
Next you have to set the Direction property of the Command Parameters. This indicates whether the parameters are of type IN, OUT or INOUT. A constant value of 3 indicates that parameters have type INOUT. See Table 30.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>adParanInput</td>
<td>1</td>
<td>IN</td>
</tr>
<tr>
<td>adParamOutput</td>
<td>2</td>
<td>OUT</td>
</tr>
<tr>
<td>adParamInputOutput</td>
<td>3</td>
<td>INOUT</td>
</tr>
<tr>
<td>adParamReturnValue</td>
<td>4</td>
<td>Return value</td>
</tr>
</tbody>
</table>

### 5.3.3 Get One Part - Stored Procedure

We have to pass a value of 1 in the first parameter and a part number in the second parameter to get one record. Then the stored procedure returns back a single record matching the part number passed.

**Figure 86. Get One Part**
To get one part, execute the Stored procedure as shown below:

```vbnet
any Rcds = 0, parms, parm[2]
parm[1] = 1
parm[2] = s1e_1.Text // value of the part number that you entered above
parms = parm
Partsrcddata = Partsrcdcmd.Execute(Rcds,Parms,1)
```

With PowerBuilder, use the constant value (1) in place of the constant name (adCmdText). For a list of constant values, please see the Execute Method in the SDK Technical Reference and ADO online help.

The stored procedure returns a result set that is loaded in ADO Recordset object. From that we populate the form as shown in Section 5.2.2 "Get One Part-SQL" on page 167.

5.3.4 Get All Parts- Stored Procedure

As Figure 87 shows, we pass a value of 2 and 0 for the first and the second parameters respectively and execute the same stored procedure.

```
//Get all parts
any Rcds = 0, parms, parm[2]
parm[1] = 2
parm[2] = 0
parms = parm
Partsrcddata = Partsrcdcmd.Execute(Rcds,Parms,1)
```
The only difference here is the value of parameters passed to the Stored Procedure. The stored procedure returns a result set which is loaded in an ADO Recordset object. This Recordset is not updatable. That is, you can’t use AddNew, Delete or Update methods of the ADO Recordset object.

We use the same function as discussed in Section 5.2.3.2 "Populate the ListView" on page 170 to populate the data control in this application.

For error handling, you can use the same routine as discussed in section 5.2.4 "Error Handling" on page 171.

5.3.5 Restrictions

The IBMDA400 provider does not support multiple result sets. You can call a stored procedure that returns multiple result sets, but you will be able to process only the first result set.

5.4 A Table Record-Level Access Application

The Client Access OLE DB provider allows you to interface with existing AS/400 logical and physical database files through record-level access. When you open a database file you may specify a database file member name and its record format. You can specify file-level commitment control, which overrides the IsolationLevel property of the Connection object.

You can find more information about the file using the Display File Description (DSPF) CL command or Display file Field Description (DSPFFD) CL command.

The DDM record level interface allows record-level access to AS/400 databases. The host server support is provided by the OS/400 DDM server. It provides the ability to read by key, update, delete and insert records into the database. V4R2 provides both SNA and TCP/IP connectivity support, whereas connection to prior releases of OS/400 is through SNA only. If you require a TCP/IP connection while
using an OS/400 release prior to V4R2, you can obtain PTFs which provide this support. See “Record-Level Access” on page 63 for details.

Figure 88. DDM Record Level Access

Figure 88 is an illustration of how to use IBMDA400 to access DB2/400 data using record level access.

This application opens the PARTS database file using record-level access. You can add new records, update or delete existing records and find a particular record by key. You can navigate through all the records in the file.
5.4.1 Opening a Connection

The first step is to create a `Connection` object and open it. See the Section 5.2.1 "Opening a Connection" on page 167 for details.

5.4.2 Opening the PARTS Table

After you have opened the Connection to the AS/400 system, you can open the table. First, you have to create a `Command` object and a `Recordset` object.

```plaintext
cmParts.ActiveConnection = cn400
cmParts.Properties[16] = 7
cmParts.CommandText = "/QSYS.LIB/APILIB.LIB/PARTS.FILE/*FIRST.MBR(*FIRST, *NONE)"
cmParts.Parameters.Append(cmParts.CreateParameter("P1", 129, 1, 1))
rsPartsdata = cmParts.Execute(Rcds, PArms, 2)
```

The first statement sets the `ActiveConnection` property of the `Command` object to the open Connection.

The second statement sets the `Updatabiltiy` property of the file that you are going to open. You should assign the `Updatability` property a sum of the `Updatability` values that apply. See Table 31 for details.

<table>
<thead>
<tr>
<th>Open table for</th>
<th>Updatability value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>0</td>
</tr>
<tr>
<td>Change</td>
<td>1</td>
</tr>
<tr>
<td>Delete</td>
<td>2</td>
</tr>
<tr>
<td>Insert</td>
<td>4</td>
</tr>
</tbody>
</table>

Thus a value of ( 7 ) opens the file for all operations.

In the third statement you set the `CommandText` parameter of the `Command` object to the file name that you want to open. You can use either IFS naming or system naming convention.

`/QSYS.LIB/<library>.LIB/<file>.FILE[/<member>.MBR]([<record format>, <commitment control>])`

The value of the `commitment control` parameter overrides the `IsolationLevel` property of the `Connection` Object.

In the fourth statement, a parameter `P1` is created and appended to the `Command` object. You have to do this for your table to be opened if you are using ADO1.0. For details about the parameters for the `CreateParameter` method of the `Command` object, refer to `SDK Technical Reference` and ADO online help.

**Member Locking**

The AS/400 database file members are locked shared read or locked shared-update when they are opened, depending on the `Updatability` property setting. You cannot change the member lock while the member is open. You have to close the file member and reopen it to do so.
**Record Locking**

If you open a file with an **Updatability** property of Read, the AS/400 database will not perform record locking. If the file is opened with an **Updatability** property other than Read (for example, insert, update, or delete), the AS/400 database will always take a pessimistic record-locking strategy. That is, a record will be locked for update as soon as you are positioned on that record, even though you have not made any actual changes. This will prevent others who have the file opened for update from positioning to the same record. For more information and a list of other record lock types that are not supported, see the LockType Property topic in the *SDK Technical Reference* and ADO online help.

**Considerations**

Because ADO always positions you at the first record when you open a file, and the AS/400 database uses pessimistic record locking, this may cause contention if you have opened the file for update operations. You will have the first record locked for update. No others will be able to open the file for update while you remain on this first record. Therefore, you may want to do a MovePrevious and sit at BOF (no current record) until you know which record you want to update.

Another technique is to open the file twice within the application; once for read and once for update. You always use the file opened for read to display records, navigate, and so on. In the file opened for update, you will always keep at BOF or EOF to avoid any record locking. Once you know what record you need to update, you position there, update the record, and move back to BOF or EOF. This technique limits the record locking to the actual time that it takes to update the record values.

### 5.4.3 Navigating the Table

You can read all the field names after the table has been opened like this:

```plaintext
For indx = 1 To <recordset>.Fields.Count - 1
<value> = <recordset>.Fields(Indx).Name
NEXT
```

The field values in the current record can be read like:

```plaintext
If Not <recordset>.BOF And Not <recordset>.EOF Then
    If IsNull(<recordset>.Fields(0).Value) Then
        <value> = ""
    Else
        <value> = <recordset>.Fields(0).Value
    End If
End If
```

Since the AS/400 database opens a file with a dynamic cursor, all the Recordset navigational methods are valid. You can use **MoveFirst**, **MoveNext**, **MoveLast**, and **MovePrevious** methods to position to a particular record.

### 5.4.4 Finding a Record

Figure 90 shows the Get Parts using record the level access screen.
If the file has defined key fields, you can find a specific record in the file provided you have a full key value or a partial key value.

You have to use the AD400 Connection object to define, open, and close a connection from the PC client to the AS/400 server. Because you can reuse AD400 connections to the AS/400 system within your application, only one is required for each AS/400 system. This connection enables Find support on keyed files, and is used to manage open indexes on the AS/400 system. An index on the AS/400 system is either a keyed physical or logical file.

**Open an AD400 Connection**

Following is the way to declare and then open an AD400 connection to the AS/400 system:

```
oleobject cn400index
cn400index = CREATE OLEObject
ll_status = cn400index.ConnectToNewObject ("ad400.connection")
cn400index.Open("TCPASM05", ",", ")
```

Since User ID and Password are not supported, empty strings have been passed in the last two parameters.

**Open an Index**

You have to use the AD400 Index object to open an index. An index is a keyed physical or a logical file on the AS/400 system. Once the AD400 index is open, you can find a record by key value and retrieve a bookmark for that record. The bookmark can be used to position to that record in the ADO Recordset object.
You have to pass the <filename> and <indexname> as parameters to the OpenIndex method. Both <filename> and <indexname> must be existing AS/400 keyed physical or logical files. Both support system as well as IFS naming convention.

**Getting a bookmark**
Here you find the key value and then move to that value in the Recordset.

```plaintext
any Keys
any Bookmark
Keys = sle_findpart.text
Bookmark = ixParts_Index.GetBookmark(Keys, ad400SeekFirstEQ)
rsPartsdata.Move(0,Bookmark)
' Read the current record values
```

The Keys value is a variant array of key field values that include all, some, or at least one of the key fields of the file. You must pass the key values in the order that matches the defined key field order of the file.

### 5.4.5 Table Operations

If you have opened the file for all the operations, then all the updatable Recordset operations are valid.

#### 5.4.5.1 Adding a new record

To add a new record, use the Addnew method of the Updatable Recordset object:
5.4.5.2 Deleting a Record

Delete the current record using delete method of Recordset object:

```vbnet
If Not rsPartsdata.BOF And Not rsPartsdata.EOF Then
    rsPartsdata.Delete(1)
    MessageBox ("Current Record","Current record was deleted")
else
    // do something
```
5.4.5.3 Updating a Record

Use the `Update` method of the updatable `Recordset` object to update a record you are currently positioned at:

```vbnet
any Fields[5], Flds, ValuesA[5], Vals
If Not rsPartsdata.BOF And Not rsPartsdata.EOF Then
    Fields[1] = "PARTNO"
    Fields[2] = "PARTDS"
    Fields[3] = "PARTQTY"
    Fields[4] = "PARTPR"
    Fields[5] = "PARTDT"
    Flds = Fields
    ValuesA[1] = sle_partno.text
    Vals = ValuesA
    rsPartsdata.Update( Flds, Vals )
Else
    MessageBox("User Position","You are currently not positioned on a record. Press next or previous button to be positioned at a record.")
End If
```

5.4.6 Get All Parts - Record-Level Access

To get all the records from the database file, we have to open the file as illustrated in Section 5.4.2 "Opening the PARTS Table" on page 179. When you open the table, it returns a result set with record pointer positioned at the first record.

To get all the records from the table, we navigate through the Recordset and populate values from the Recordset to the data control of our choice. We discussed this earlier in Section 5.2.3.2 "Populate the ListView" on page 170.

The only difference is that result set here is dynamic, unlike the result set returned by the execution of the SQL statement. Every time you do a `MoveNext`, `MovePrevious`, `MovePrevious`, or `MoveLast`, it is actually a trip to the server.
5.4.7 Commitment Control

You can specify the Commitment control at the Connection level. Following are the valid values for the IsolationLevel property of the ADO Connection object. The default, if the IsolationLevel property is not specified, is Cursor Stability (adXactCursorStability).

<table>
<thead>
<tr>
<th>ADO IsolationLevel Property</th>
<th>AS/400 Commitment Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chaos (adXactChaos)</td>
<td>NONE (*NC)</td>
</tr>
<tr>
<td>Browse (adXactBrowse)</td>
<td>*CHG(*UR)</td>
</tr>
<tr>
<td>CursorStability(adXactCursorStability)</td>
<td>*CS</td>
</tr>
<tr>
<td>Repeatable Read (adXactRepeatableRead)</td>
<td>*ALL</td>
</tr>
</tbody>
</table>

All files are opened under commitment control in ADO. This is true if you specify a value for IsolationLevel or simply take the default. On the AS/400 system, all AS/400 files that are opened under commitment control require the use of journals. You must create a journal and a journal receiver for all of the files that you open by using the IsolationLevel property. If your files are not journaled when you open them by using the connection isolation level, an error is issued.

Also, ADO does an automatic commit whenever a transaction (BeginTrans) is not explicitly started. This causes a commit to be processed after every transaction so that all changes are automatically committed to the database file. The commitment-control level that is used for this automatic commit is controlled by
the OLE DB provider AutoCommit property, which is not available and cannot be
set through ADO. To enable the user to set the AutoCommit property, the
IBMDA400 provider sets this property to match the last transaction-isolation level.
The IBMDA400 default for the AutoCommit property is Chaos (*NONE).

When you use the DB2/400 database, commitment control for the file is specified
when the file is opened. Because ADO holds the isolation level from the OLE DB
provider until it is needed on a BeginTrans, and since you will want to set the
AutoCommit property to match the IsolationLevel property under which you will be
running, you need to explicitly begin and end a transaction before opening any
AS/400 files. However, you can override the commitment control specified at
Connection level by specifying the file-level commitment control. If you do not
specify a file-level commitment control when you open the file, the IsolationLevel
property of the ADO connection object is used.

Transactions are started, committed, and rolled back at the ADO connection object
level. It is illustrated as:

<connection>.IsolationLevel = 256
<connection>.BeginTrans
<connection>.EndTrans
<connection>.CommitTrans
if errors
<connection>.RollbackTrans

With PowerBuilder, use the constant value (256) instead of the constant name
(adXactBrowse). For a list of constant values, see the IsolationLevel Property
topic in the SDK Technical Reference and ADO online help.

5.4.7.1 Restrictions
The IBMDA400 provider only supports commitment control or transaction control
for record-level access tables. It does not support transaction control for SQL
statements.

The IBMDA400 provider does not support nested transactions. If you have
already have started a transaction with <connection>.BeginTrans, you cannot
start another transaction until you have either done a commit or rollback
operation.

5.4.8 Closing the Table
You can close an open table and free the AS/400 file locks and file record locks
that exist on the file. This can be done by:
<recordset>.Close

You can choose to reuse the Recordset object or release the object from memory.
SetNull(<recordset>)

This releases all of the Recordset object resources and eliminates the object from
memory. See the Closing a Recordset and the Releasing a Recordset object
topics of the ADO Recordset object for more information.

5.4.9 Error Handling
See Section 5.2.4 "Error Handling" on page 171.
5.5 A Data Queue Application Example

The Client Access OLE DB provider allows you to interface with existing AS/400 data queues. It allows you to place records on and receive records off of AS/400 data queues. The support also handles conversions from AS/400 formats (for example, Packed Decimal) to PC formats.

A data queue is an AS/400 object that allows a program (host or client) to place data into it and retrieve data from it. When you open a data queue you can specify a record format for the data queue entries.

A data queue on the AS/400 system can have FIFO (First In First Out), LIFO (Last In First Out), or keyed-entry sequence. There also is a maximum entry length that is defined when the data queue is created. In the example we discuss, a client program places a request (entry) on an AS/400 data queue. Running on the AS/400 system is a program (DQXRPG) that reads the request off of the data queue and accesses the PARTS file. In response, the AS/400 program places records from the PARTS file on the output data queue.

You can use the Create Data Queue (CRTDTAQ) CL command to create a data queue on the AS/400 system.

Data queues have following characteristics:
• Data queues can be accessed by many jobs simultaneously.
• They can be used to hold data being passed back and forth between jobs.
• Messages on a Data Queue are free format.

Figure 92 is an illustration of how to use IBMDA400 to read or write to Data Queues.

In this example, the client program requests data from the server program (DQXRPG) by writing to input data queue DQINPT. The input is a flag
and a number, for example "S12301" is requesting a single record of part number 12301. If requesting all parts (FLAG = "A") the part number is not necessary. The server program searches for the information in the database and writes the result to the output data queue. See “DQXRPG/RPGLE” on page 378 for a complete listing of DQXRPG.

Figure 93 shows the result of requesting one part from the PARTS database.

Figure 93 shows the result of requesting one part from the PARTS database.

Figure 94 on page 189 shows the result of requesting all parts from the PARTS file.
5.5.1 Data Queue Server Program Background

An input queue and output queue were created with the commands:

```
CRTDTAQ DTAQ(APILIB/DQINPT) MAXLEN(6) TEXT('Data Queue for Parts Input)
CRTDTAQ DTAQ(APILIB/DQOUPT) MAXLEN(48) TEXT('Data Queue for Parts Output)
Program Name : APILIB/DQXRPG
```

The following tables illustrate data queue layouts.

**Table 33. Data Queue DQINPT Layout**

<table>
<thead>
<tr>
<th>Queue Position</th>
<th>Description</th>
<th>Length Type</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 1</td>
<td>Action Flag</td>
<td>1 character</td>
<td>S - Retrieve single record</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A - Position to start of file</td>
</tr>
<tr>
<td>2 - 6</td>
<td>Part Number</td>
<td>5.0 Zoned</td>
<td>Part number to retrieve</td>
</tr>
</tbody>
</table>

**Table 34. Data Queue DQOUPT Layout**

<table>
<thead>
<tr>
<th>Queue Position</th>
<th>Description</th>
<th>Length Type</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 1</td>
<td>Action Flag</td>
<td>1 character</td>
<td>Y - operation succeeded</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X - record not found or EOF</td>
</tr>
<tr>
<td>2 - 6</td>
<td>Part Number</td>
<td>5.0 Zoned</td>
<td>part to be retrieved</td>
</tr>
</tbody>
</table>
This server program should be active before you use this application. You can call this program using **SBMJOB** command on the AS/400 system:

```
SBMJOB CMD(CALL PGM(APILIB/DXRPG))
```

### 5.5.2 Opening the Data Queue

By now you know that you have to create a *Connection* object and open it. See Section 4.1.1, “Opening a Connection” on page 30 for details.

You will also need to create two *Recordset* objects, one each for the input data queue and output data queue.

Data queues are opened by using the *Execute* method of the *Connection* object as shown:

```pascal
<recordset> = <connection>.Execute("open dataqueue <data queue> [(record format)] [KEY(<key format>)] for <option>
```

**Record format**

A comma-separated list of field name, type and optional CCSID.

**Key format**

This is valid only for keyed data queues. It is a comma separated list of a field name, type, and optional CCSID.

**Option**

The value can be set as SEND or RECIEVE to write to a data queue or read from a data queue.

---

This table describes the queue positions and their corresponding descriptions:

<table>
<thead>
<tr>
<th>Queue Position</th>
<th>Description</th>
<th>Length Type</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 - 31</td>
<td>Description</td>
<td>25 character</td>
<td></td>
</tr>
<tr>
<td>32 - 34</td>
<td>Quantity</td>
<td>5.0 packed</td>
<td></td>
</tr>
<tr>
<td>35 - 38</td>
<td>Price</td>
<td>6.2 packed</td>
<td></td>
</tr>
<tr>
<td>39 - 48</td>
<td>Ship date</td>
<td>10 character</td>
<td></td>
</tr>
</tbody>
</table>

Rcds is *RecordsAffected*, and it is not supported by IBMDA400. You have to pass (0) in this variable as is done in the earlier examples.
With PowerBuilder, use the constant value (1) in place of the constant name (adCmdText). For a list of constant values, see the Execute Method topic of the SDK Technical Reference and ADO online help.

5.5.2.1 Reading Field Names
Once the data queue is open, you can read all of the field names that compose the data queue entry format. These names will be the same as those you specified on the open of the data queue.

This is like reading the field names from a Recordset object. See section 5.2.3.1 "Navigating the Result Set" on page 170.

5.5.2.2 Navigating Read Entries
When you open a data queue for RECEIVE, all the entries are read into the Recordset object, and you are positioned at the first entry.

You can use Recordset object methods (for example MoveFirst, MoveNext, MovePrevious, MoveLast) for navigating through the Recordset. The Recordset is not updatable and thus the AddNew, Update, and Delete will generate errors.

Please see Section 5.2.3.1 "Navigating the Result Set" on page 170.

5.5.2.3 Rereading Entries
If you open a data queue for RECEIVE, you can reread any new data queue entries by using the Requery method of the Recordset object. All existing records in the Recordset object are deleted and replaced with the reread entries from the data queue. If there are no new entries on the data queue, the resulting Recordset will be empty.

If <recordset>.EOF Then
    <recordset>.Requery
End If
<navigate the read entries>

5.5.2.4 Adding an Entry
If you open a data queue for SEND, you can add new entries to the data queue using the AddNew method of the Recordset object. See Section 5.4.5 "Table Operations" on page 182. But, the Update and Delete methods are not supported and will produce errors.

5.5.3 Get One Part - Data Queue
The following screen is used to get one part using the data queue.
When you enter the part number and press the Get Part button, the following code is executed:

```csharp
any Fields[2], Flds, ValuesA[2], Vals
Fields[1] = "Option"
Fields[2] = "part"
Flds = Fields
ValuesA[1] = "S"
ValuesA[2] = sle_1.text
Vals = ValuesA
DqInpt.AddNew( Flds, Vals )
Dqoupt.ReQuery
```

In this code we are placing a record containing "S" (option for single record) and the part number (corresponding to which we want data). The server program searches the database and places the requested record on the output queue. If the record is not found it places "X" in the first field of the output data queue(DQOUPT).

The record is loaded in the Recordset object. We can populate the field values from the Recordset object. See Section 5.2.2 “Get One Part-SQL” on page 167 for details.

### 5.5.4 Get All Parts - Data Queue

When you request for all the parts, the following code is executed:
The only difference in this code from what you saw earlier is the values that we placed on the Input data queue(DQINPT). Here we place a value of "A" for the option field. This means that now the server program retrieves all the records and places them on the output data queue.

When we execute the Requery method on the output data queue, the whole result set is loaded in the Recordset object.

Finally, we populate the ListView from this Recordset as shown in Section 5.2.3.2 "Populate the ListView" on page 170.

5.5.5 Closing a Data Queue

Closing a data queue releases the lock that is on the data queue object on the AS/400 system. This can be accomplished by closing the Recordset and releasing all the resources by eliminating the object from the memory.

```
//Closing the Recordset
<recordset>.Close
//Releasing the Recordset
SetNull(<recordset>)
```

5.6 Remote Command Execution Example

IBMDA400, the Client Access OLE DB provider, allows you to interface with AS/400 commands. There is a support to call an AS/400 command and pass command parameters.

5.6.1 Executing a Remote Command

You use the Execute method of the Connection object to execute the AS/400 commands. You need to create a Connection object and open a connection to the AS/400 system as illustrated in earlier sections.

The parameters passed to the Execute method are CommandText, RecordsAffected, and adCmdText.

CommandText is the command surrounded by two pairs of curly braces. If you want to accept user input as part of the command, you will have to concatenate the user input to the actual command.
The parameter `adCmdText` is the name of the constant which determines the type of CommandText. With PowerBuilder, use the constant value (1) instead of the constant name (`adCmdText`). For a list of constant values, see the Execute Method topic in the *SDK Technical Reference* and ADO online help.

```csharp
//Execute CL command
<connection>.Execute("QSYS/CRTLIB LIB(APILIB)",,1)
```

### 5.6.2 Restrictions

You can’t call interactive commands. Only commands that can be called in batch can be executed. IBM DA400 does not support preparing commands as can be done for SQL statements. Therefore, you should concatenate the command text and the command values together to form the complete command.

### 5.7 A Distributed Program Call (DPC) Application Example

IBMDA400, the Client Access OLE DB provider, lets you call AS/400 programs. You can pass both input and output parameters. For more information on creating programs on the AS/400 system, see the appropriate AS/400 language reference manual for your particular programming language.

![Diagram of Distributed Program Call Example](image)

The following is an illustration of how to use IBM DA400 to call an AS/400 program. In this example the client program calls an AS/400 program (DPCXRPG) with a parameter that is the value of a part number. This program queries the database file and returns back the details of the PART.
For a listing of DPCXRPG, see “DPCXRPG/RPGLE” on page 377. The details of the RPG program DPCXRPG are:

### Table 35. Details of DPCXRPG

<table>
<thead>
<tr>
<th>Seq</th>
<th>Description</th>
<th>Length Type</th>
<th>I/O</th>
<th>Values</th>
</tr>
</thead>
</table>
| 1   | Action Flag   | 1 Character | I/O | S - Retrieve single record  
A - Position to the start of file  
F - Fetch the next record  
OUTPUT  
Y - Operation succeeded  
X - Operation failed or EOF found |
| 2   | Part Number   | 5.0 Packed  | I/O | Part Number to be retrieved or retrieved                              |
| 3   | Description   | 25 Character| O   |                                                                        |
| 4   | Quantity      | 5.0 Packed  | O   |                                                                        |
| 5   | Price         | 6.2 Packed  | O   |                                                                        |
| 6   | Ship Date     | 10 Date     | O   |                                                                        |

### 5.7.1 Defining the Program Call

You need to create a Connection object and open it. Also you have to create a Command object. Go back to earlier sections for details on how to do that.

```java
Partsrccmd.ActiveConnection = cn400
Partsrccmd.CommandText = "{{call apilib/dpcxrpg(?,?,?,?,?,?)}}"
Partsrccmd.Prepared = True
```
See Section 5.3.2 "Preparing for Stored Procedure Call" on page 174 for details. The only difference is that you have to put two pairs of curly braces around a program call. This is the way IBM D400 differentiates between a program call and a Stored Procedure call. You have to specify a list of comma separated question marks as place holders for each of the program parameters.

Note that when a call to an AS/400 program is prepared, there is no existing information on the AS/400 system that describes the program parameters. To create program parameter information, you should catalog and call the program as a stored procedure. Since AS/400 program information does not exist, you must create and specify all of the program parameters in your application as shown here:

```csharp
Partsrcdcmd.Parameters.Append(Partsrcdcmd.CreateParameter("opt", 129, 3, 1))
Partsrcdcmd.Parameters.Append(Partsrcdcmd.CreateParameter("partno", 14, 3, 0))
Partsrcdcmd.Parameters.Append(Partsrcdcmd.CreateParameter("qty", 14, 2, 0))
Partsrcdcmd.Parameters.Append(Partsrcdcmd.CreateParameter("price", 14, 2, 0))
Partsrcdcmd.Parameters.Append(Partsrcdcmd.CreateParameter("pdate", 129, 2, 10))
```

With PowerBuilder, use the constant values in place of the constant names (adParamOutput, adParamInput, ad data types) while creating the parameters. For a list of constant values, see the Direction Property and Type Property topics in the SDK Technical Reference and ADO online help.

Next you have to set the Precision and NumericScale for those data types (adDecimal and adNumeric) that require them. This is only necessary when calling an AS/400 program.

```csharp
//Here you set the precision and Numeric Scale of the "partno" parameter that
//you created in earlier step.
Partsrcdcmd.Parameters[1].Precision= 5
Partsrcdcmd.Parameters[1].NumericScale=0

Partsrcdcmd.Parameters[3].Precision= 5
Partsrcdcmd.Parameters[3].NumericScale=0

Partsrcdcmd.Parameters[4].Precision= 6
Partsrcdcmd.Parameters[4].NumericScale=2
```

5.7.2 Get Part - Program Call

The next step is to execute the Command. It is like executing the Stored Procedure call. See Section 5.3 "A Stored Procedure Application Example" on page 171 for details. The code below illustrates how to execute the prepared Command object.
Here we pass a value of ‘S’ in the first parameter and part number in the second parameter. This means that we are requesting a single record matching the value of part number entered as shown in the Figure below.

![Figure 98. Get A Part - Program Call](image)

The next step is to read the output parameters returned by the AS/400 program. It is accomplished like this:

```plaintext
If (Partsrcdcmd.Parameters[0].Value = 'Y') Then
    sle_2.Text = Partsrcdcmd.Parameters[2].value
    sle_3.Text = Partsrcdcmd.Parameters[3].value
    sle_4.Text = Partsrcdcmd.Parameters[4].value
    sle_5.Text = Partsrcdcmd.Parameters[5].value
```

### 5.7.3 Restrictions

Calling programs on the AS/400 system that have user defined parameters or parameters that are structures are not supported.
5.8 DataWindow Considerations

In the examples, you have seen the way to access resources on the AS/400 system using IBMDA400, the Client Access OLE DB provider. The data you get back is either in the form of result sets or parameters. This data can be populated in the data controls of your choice.

DataWindows is one of the most powerful controls available in the PowerBuilder development environment. It is an object that can be used to retrieve and manipulate data from a relational database or other data source. While using DataWindows you can choose from several presentation styles. Also you can choose to display values for each column in particular display formats and edit styles.

A DataWindow in PowerBuilder can be based on an SQL statement, a query, a stored procedure or an external data source. You can do that if you are using the Client Access ODBC driver to connect to the AS/400 system.

If you are using IBMDA400, you will not be able to create DataWindows based on Quick select, SQL and Query in the same manner as you do when you are using the Client Access ODBC driver. This is because PowerBuilder doesn’t support DataWindows based on ADO objects.

The PowerBuilder DataWindow control has been optimized to work with ODBC data sources. It has not been updated to be optimized for new technologies such as ADO. We expect that the developers of PowerBuilder will update their DataWindow control to support ADO data sources in a future release.

To use ADO you will have to create DataWindows based on an external data source. Then you can retrieve data using any of the methods discussed in this chapter into a result set. The result set can be used to populate the DataWindow control. In order to update the database you can’t use DataWindow functions. Instead, you have to write routines to do this.
Chapter 6. Delphi Using OLE DB

Delphi is a Windows application development environment to build Client/Server applications. It is based on Object Pascal. The Delphi Client/Server Suite provides an integration of tools necessary for building, testing, and deploying two-tiered and multi-tiered applications that work with both local and remote SQL databases. For this you have to use third-party ODBC drivers to access local and remote ODBC-compliant databases. This can be done by installing the Client Access ODBC driver and defining an ODBC data source.

This chapter focuses on how to develop database applications using OLE DB with the AS/400 system. OLE DB is a set of application programming interfaces (APIs) that provide access to a wide range of data sources. ODBC provides access only to information in a relational database through SQL, while OLE DB is a method to access data via a standard COM (Component Object Model) Interface. OLE DB provides both the SQL functionality defined in ODBC and interfaces suitable to access non-SQL resources.

IBMDA400 is an OLE DB provider to AS/400 data and resources. It is a part of Client Access Version V3R1M3. It includes support to interface with the following AS/400 functions:

- Table record-level access
- Data queues
- Programs
- CL commands
- SQL statements
- SQL stored procedures

IBMDA400, the Client Access OLE DB provider, has been written to OLE DB Version 1.1 specification and has been tested to work with ADO 1.0. For more details on ADO programming model, please see Chapter 1, “OLE DB and ActiveX Overview” on page 1.

In this chapter we will show you through programming examples how to use IBMDA400 with Delphi Version 3.0.

6.1 Introduction to Application Examples

The rest of this chapter covers application examples which demonstrate:

AS/400 resource access via:

- SQL
- Stored Procedures
- Table Record Level Access
- AS/400 Data Queues
- Program Calls on the AS/400

The database access example applications discussed in the remainder of this chapter allow retrieval of a single record or all the records from a PARTS file on
the AS/400 system. Some examples allow an update to the current Record in the window. These examples have been tested to work with Delphi 3.0 and ADO 1.0.

We use the PARTS database file in the APILIB library in all the examples. The PARTS file is defined as:

Table 36.  PARTS File

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Description</th>
<th>Length</th>
<th>Decimals</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARTNO</td>
<td>Part Number</td>
<td>5</td>
<td>0</td>
<td>Zoned</td>
</tr>
<tr>
<td>PARTDS</td>
<td>Part Description</td>
<td>25</td>
<td></td>
<td>Char</td>
</tr>
<tr>
<td>PARTQY</td>
<td>Quantity</td>
<td>5</td>
<td>0</td>
<td>Packed</td>
</tr>
<tr>
<td>PARTPR</td>
<td>Price</td>
<td>6</td>
<td>2</td>
<td>Packed</td>
</tr>
<tr>
<td>PARTDT</td>
<td>Part Shipment Date</td>
<td>10</td>
<td></td>
<td>Date</td>
</tr>
</tbody>
</table>

All the examples discussed in this chapter are available for download on the www.redbooks.ibm.com web-site. See Appendix A.1, “Downloading the Samples” on page 373 for details.

Tip
For additional information about programming with ADO, see the SDK Technical Reference Manual and the Microsoft online help. See Appendix A.2, “Other Documentation and Resources” on page 373 for details on how to obtain this documentation.

6.2 An SQL Application Example

You can execute remote SQL statements on the AS/400 server using IBMDA400, the Client Access OLE DB provider. There is support for executing most SQL statements on the AS/400 system. For more information on SQL statements see the DB2 for AS/400 SQL Programming Manual, SC41-5611.

You can execute SQL statements using the Execute method of the Connection Object. If you are executing SQL statements with parameters (for example if you are using a where clause that has a parameter which varies in value from one call to another) you should use parameter markers and prepare the SQL statement before executing it. This improves performance since the overhead of preparing the SQL statement every time you execute it is reduced.

The SQL SELECT statement and SQL CALL statement, for calling a stored procedure, are the only two SQL statements that generate a result set, and thus a Recordset object. The Recordset Object created is not updatable. Therefore, the AddNew, Update, and Delete methods are not valid.
This example is an illustration of how to use IBMDA400 to access DB2/400 data using SQL. In this example, the client program requests data from the AS/400 database by sending SQL statements to the AS/400 system. The host server executes the SQL statement and returns the results to the client program in a SQL result set. Two SQL statements are used:

```sql
Select * from apilib.parts
Select * from apilib.parts where partno = ?
```

to retrieve all columns for all records in PARTS file and to retrieve all columns for a given part number in PARTS file, respectively.
This screen shows the result of requesting one part (12346) from the PARTS file.

In the following sections, we discuss the step by step implementation of the application we just described. In this application, we have two forms. One main form where we can query for a part number and a second one where we display all the parts.
6.2.1 Opening a Connection

First step is to define a Connection object and invoke its Open method to start a session with the AS/400 system. We do that in the Project Source.

```delphi
begin
  Application.Initialize;
  Con400 := CreateOleObject('adodb.connection');
  Con400.Open('Provider=IBMDA400;Data Source=TCPASM05;','');
  // some more code
end
```

First you create the Connection object. You should declare this as a global variable of Variant type. All ADO objects are declared as Variant type.

After creating the Connection object, you invoke its Open method. The ConnectionString part contains the provider and data source names. The Data Source name must be the name of a valid AS/400 connection. For example, the name of a connection configured using using the Client Access AS/400 connection program.

Empty strings are passed for the User ID and Password parts. The IBMDA400 provider does not support the passing of User ID and Password when you open a connection. Client Access prompts the user for connection information as it is needed.

6.2.2 Get One Part-SQL

To get one part you have to query the PARTS file based on the part number that you enter. For this, you have to create Command and Recordset objects. Then you have to prepare the Command object such that it is ready to be executed.

```delphi
begin
  //open the connection as done in the earlier section
  AllPartsData := CreateOleObject('adodb.recordset');
  PartData := CreateOleObject('adodb.recordset');
  PartCmd := CreateOleObject('adodb.command');
  PartCmd.ActiveConnection := 'Provider=IBMDA400;Data Source=TCPASM05;';
  PartCmd.CommandText := 'select * from apilib.parts where partno = ?';
  PartCmd.Prepared := True;
  // some code
end.
```

You have to set the ActiveConnection property of the Command object to a new connection as shown in the code above.
With Delphi, don’t attempt to reuse an open connection. ActiveConnection can only be set to a new connection.

The next step is to set the CommandText property of Command Object to the SQL statement to be executed. Use parameter markers(?) instead of the variables that will have different value every time the SQL is executed. Using parameter markers improves performance since the prepared SQL statement is used at the time of execution.

The parameters passed to the Execute method of Command Object are RecordsAffected, parameter value and adCmdText. RecordsAffected is not supported by IBMDA400, but you have to pass it. The second parameter is the value that you want to be substituted for parameter marker used in the SQL statement that you prepared in the first step. The third parameter indicates the type of CommandText. A value of 1 indicates that it is a textual definition of a

```pascal
procedure TAppli.GetPartClick(Sender: TObject);
begin
  //define Partnumber and Parms of Variant type
  Partnumber := PartNumb.Text;
 Parms := Partnumber;
  PartData := PartCmd.Execute(Rcds, Parms, 1);

  //Load the data
end;
```
command. For a list of constant values, see the Execute Method topic in the *SDK Technical Reference* and the Microsoft ADO online help.

The SQL statement, when executed, returns an SQL result set. This result set is loaded into the ADO Recordset object. Then we populate the fields in the form as:

```delphi
If Not PartData.EOF Then
  Begin
    Quant.Text := PartData.Fields(2).Value;
    RcvDate.Text := PartData.Fields(4).Value;
  end
Else
  ShowMessage('Part not found!');
```

### 6.2.3 Get All Parts - SQL

![Sql Example](image)

To get all the parts, you have to select all the records from the database file.

```delphi
AllPartsData := Con400.Execute('select * from apilib.parts',Rcds, 1);
```

You can execute the SQL statement to select all the records by using the `Execute` method of the `Connection` object. The parameters passed are `CommandText`,
RecordsAffected and adCmdText. For details, see the Execute method of Connection object in the SDK Technical Reference and the ADO online help.

You can prepare this SQL statement as illustrated in Section 6.2.2 "Get One Part-SQL" on page 203. If you choose to prepare this SQL statement, then you won’t have parameter markers and you won’t have to pass parameters to this SQL statement at the time of execution.

This SQL statement returns a result set that is loaded into the ADO Recordset object.

6.2.3.1 Navigating the Result Set
If the execution of the SQL statement produces a result set, the result set data is loaded into the ADO Recordset object. If the result set is empty, the Recordset is also empty and BOF and EOF is true. You can use MoveFirst, MoveNext and MovePrevious methods to navigate the result set.

The field names in the Recordset can be read as:

```pascal
 indx: integer;
 For indx:= 0 To rs.Fields.Count - 1 do
 <value> := <recordset>.Fields(Indx).Name;
```

The field values in the current record can be read as:

```pascal
 If Not <recordset>.BOF And Not <recordset>.EOF Then
 begin
   If VarType(<recordset>.Fields(0).Value) = Null Then
     <value> := ''
   Else
     <value> := <recordset>.Fields(0).Value;
 end;
```

Note

- MoveLast method is not supported for an SQL Recordset.
- An SQL Recordset is not updatable. As a result AddNew, Update and Delete methods are not valid.
- An SQL Recordset does not support bookmarks.

6.2.3.2 Populate the ListView
Now you have to navigate through the Recordset and populate the data control of your choice. In this application we have chosen to use TListView data control and we use a procedure that populates the TListView control.

This function takes the name of the TListView and the Recordset as arguments and populates the TListView control.

First it reads the field names from the Recordset and adds columns to the TListView control. In the second step, it loops through the Recordset and adds one record at a time to the control.
6.2.4 Error Handling

The Connection object contains the errors collection. All PC client and AS/400 error and warning messages are returned in the error collection at the connection level.

For more information on Errors, see the Error Object and Errors Collection topics in Chapter 1, “OLE DB and ActiveX Overview” on page 1.

Use the cwbbztrc.exe program to work with error logging. You can run the program from a DOS command prompt. Run the program without any parameters to get the options and syntax. This program is installed when you install Client Access. The following is the default installation path and log file path:

```pascal
procedure LoadListView(lv: TListView; rs: Variant);
begin
    // Set type of list view
    lv.ViewStyle := vsReport;
    lv.Columns.Clear;
    For indx:= 0 To rs.Fields.Count - 1 do
        begin
            hdr := lv.Columns.Add;
            hdr.caption := rs.Fields(Indx).Name;
            hdr.width := ColumnHeaderWidth;
        end;
    // Make sure we are at the beginning of the Recordset
    If rs.EOF AND Not rs.BOF Then
        rs.MoveFirst;
    // Clear data from control
    lv.Items.Clear;
    // Load the data
    While Not rs.EOF do
        begin
            For indx:= 0 To rs.Fields.Count - 1 do
                begin
                    If indx = 0 Then
                        begin
                            itm := lv.Items.Add;
                            itm.caption := TrimRight(rs.Fields(indx).Value);
                            end
                    Else
                        itm.SubItems.Add(TrimRight(rs.Fields(indx).Value));
                    end;
                    rs.MoveNext;
                end;
        end.
end.
```
6.3 A Stored Procedure Application Example

The AS/400 OLE DB provider allows you to call Stored Procedures on the AS/400 system. A Stored Procedure is a cataloged program which may be written in most of the programming languages on the AS/400 system. To catalog an AS/400 program you can use the CREATE PROCEDURE SQL statement. For more information on SQL statements and Stored Procedure, see the *DB2 for AS/400 SQL Programming Manual*, SC41-5611.

A Stored Procedure can be used to control multiple database operations on the server, thereby reducing the line turn-arounds. Thus, in client/server applications, Stored Procedures can be used to improve performance by reducing the communication overhead. A Stored Procedure may take one to up to 256 parameters. The output is returned in OUT (or INOUT) parameters or via SQL result sets.

Following is an illustration of how to use the Client Access OLE DB provider to call Stored Procedures. In this example, the client program calls a stored procedure SPROC2 in the APILIB library. The stored procedure can be created by using the the interactive SQL environment on the AS/400 system and entering the following statement.

Figure 104. Delphi Application - Stored Procedures
The stored procedure is named SPROC2. It calls an AS/400 RPG program also named SPROC2. This program takes in two integer input/output parameters. Based on the parameters passed it returns either one part on all the parts. For a complete listing of the RPG program, SPROC2, see “SPROC2/SQLRPGLE” on page 375.

A value of 1 for the first parameter returns a single record in the result set with the part number field matching the part number supplied in the second parameter. A value of 2 in the first parameter returns all records from the parts database in a result set. The second parameter is ignored in this case.

The following screen shows the result of requesting one part from the PARTS file.

---

**Create Procedure**

```sql
CREATE PROCEDURE APILIB/SPROC2 (
INOUT P1 INTEGER,
INOUT P2 INTEGER)
EXTERNAL NAME APILIB/SPROC2 LANGUAGE RPG GENERAL
```

---

**Figure 105. Get One Part - Stored Procedure**

The following screen shows the result of requesting one part from the PARTS file.
6.3.1 Opening a Connection

First we create a Connection object and open it. This is same as we did in the previous section of this chapter.

6.3.2 Preparing for Stored Procedure Call

Since we use the same stored procedure for both getting one part or getting all the parts, we prepare it once and then execute it with different input parameters.

After you have opened the connection object, you have to create a Command object and prepare it for a Stored Procedure call as shown below:

```.delphi
// Create Connection object
Cn400 := CreateOleObject('adodb.connection');
Cn400.Open('Provider=IBMDA400;Data Source=TCPASM05;','','');

// Create Recordset and Command objects
AllPartsData := CreateOleObject('adodb.recordset');
PartData := CreateOleObject('adodb.recordset');
AllPartsCmd := CreateOleObject('adodb.command');

// Prepare for the Stored Procedure call
AllPartsCmd.ActiveConnection := 'Provider=IBMDA400;Data Source=TCPASM05';
AllPartsCmd.CommandText := '{call APILIB.SPROC2(?),?}';
AllPartsCmd.Prepared := True;
```

You have seen in the earlier SQL example section a discussion on how to prepare a Command object. You have to set the ActiveConnection property of the Command object to a new connection as shown in the code above.

With Delphi, don’t attempt to reuse an open connection. The ActiveConnection only can be set to a new connection.
Let's take a look at the CommandText property of the Command object. The braces around the call statement are required since they indicate that it is a Stored Procedure call. The question marks (?) indicate the parameters to be passed when the Command is executed.

Next you have to set the Direction property of Command Parameters. This indicates whether the parameters are of type IN, OUT or INOUT. A constant value of 3 indicates that parameters have type INOUT. See the Table below.

**Table 37. Constant values for Direction property of Parameters Collection**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>adParamInput</td>
<td>1</td>
<td>IN</td>
</tr>
<tr>
<td>adParamOutput</td>
<td>2</td>
<td>OUT</td>
</tr>
<tr>
<td>adParamInputOutput</td>
<td>3</td>
<td>INOUT</td>
</tr>
<tr>
<td>adParamReturnValue</td>
<td>4</td>
<td>Return value</td>
</tr>
</tbody>
</table>

`AllPartsCmd.Parameters[0].Direction := 3;`  
`AllPartsCmd.Parameters[1].Direction := 3;`

### 6.3.3 Get One Part - Stored Procedure

To retrieve information about an individual part, we have to pass a value of 1 in the first parameter and a part number in the second parameter. The stored procedure returns a single record matching the part number passed in the second parameter.
To get one part you click on the **Get Part** button, which causes the execution of the Stored procedure as shown below:

```delphi
procedure TAppli.GetPartClick(Sender: TObject);
begin
  Partnumber := PartNum.Text;
 Parms := VarArrayCreate([0, 1], varVariant);
 Parms[0] := 1;
 Parms[1] := Partnumber;
 PartData := AllPartsCmd.Execute(Rcds, Parms, 1);

  // Load the data
  If Not PartData.EOF Then
    Begin
      Quant.Text := PartData.Fields(2).Value;
      RcvDate.Text := PartData.Fields(4).Value;
      EndTime.Text := TimeToStr(Time);
    end
  Else
    ShowMessage('      Part not found !      ');
    PartData := Null;
  end;
end;
```

With Delphi, use the constant value (1) in place of the constant name (adCmdText) while executing the prepared Command. For a list of constant values, please see the Execute method in the *SDK Technical Reference* and ADO online help. For information on how to obtain the *SDK Reference Manual*, see Appendix A.2, “Other Documentation and Resources” on page 373.

The stored procedure returns a result set which is loaded in the ADO Recordset object. From that, we populate the form as shown in the code above.
6.3.4 Get All Parts - Stored Procedure

Here we pass a value of 2 and 0 for the first and the second parameters respectively, and execute the same stored procedure.

```
procedure TAllParts.FormCreate(Sender: TObject);
begin
  Parms := VarArrayCreate([0, 1], varVariant);
  Parms[0] := 2;
  Parms[1] := 0;
  AllPartsData := AllPartsCmd.Execute(Rcds, Parms, 1);
  LoadListView(AllPart,AllPartsData);
  AllPartsData := Null
end;
```

The only difference here is the value of the parameters passed to the stored procedure. The stored procedure returns a result set that is loaded in an ADO Recordset object. This Recordset is not updatable, that is you can’t use the AddNew, Delete or Update methods of the ADO Recordset object.

We use the same function LoadListView as discussed in Section 6.2.3.2 "Populate the ListView" on page 206 to populate the data control in this application.

For error handling you can use the same method as discussed in Section 6.2.4 "Error Handling" on page 207.
6.3.5 Restrictions

The IBMDA400 provider does not support multiple result sets. You can call a stored procedure that returns multiple result sets, but you will be able to process only the first result set.

6.4 A Table Record-Level Access Application

IBMDA400, Client Access OLE DB provider, allows you to interface with existing AS/400 logical and physical database files through record-level access. When you open a database file you may specify a database file member name and its record format. You can specify file-level commitment control, which overrides the IsolationLevel property of the Connection object.

You can find more information about the file using Display File Description (DSPF) CL command or Display file Field Description (DSPFD) CL command.

The DDM record-level interface allows record-level access to AS/400 databases. The host server support is provided by the OS/400 DDM server. It provides the ability to read by key, update, delete and insert records into the database. V4R2 provides TCP/IP connectivity support whereas connection to prior releases of OS/400 is through SNA only. If you require a TCP/IP connection while using an OS/400 release prior to V4R2, you can obtain PTFs which provide this support. See “Record-Level Access” on page 63 for details.

Figure 109. DDM Record-Level Access
Following is an illustration of how to use IBMDA400 to access DB2/400 data using Record-Level Access.

![GET AND MAINTAIN PARTS (USING RECORD LEVEL ACCESS)](image)

This application opens the PARTS database file using Record-Level Access. You can add new records, update or delete existing records and find a particular record by key. You can navigate through all the records in the file.

6.4.1 Opening a Connection

The first step is to create a Connection object and open it. See the section 6.2.1 "Opening a Connection" on page 203 for details.

6.4.2 Opening the PARTS Table

After you have opened the Connection to AS/400, you can open the table. First you have to create the Command object and a Recordset object.

```pascal
PartCmd := CreateOleObject('adodb.command');
PartCmd.ActiveConnection := 'Provider=IBMDA400;Data Source=TCPASM05';
PartCmd.Properties('Updatability') := 7;

PartData := CreateOleObject('adodb.recordset');
PartData := PartCmd.Execute(Rcds, , adCmdTable);
```

The first statement sets the ActiveConnection property of the Command object to the new connection. With Delphi, don’t attempt to reuse an open connection. The ActiveConnection only can be set to a new connection.
The second statement sets the Updatability property of the file that you are going to open. You should assign the Updatability property a sum of the Updatability values that apply.

Figure 38 shows table access capabilities.

<table>
<thead>
<tr>
<th>Table 38. Table Access Capability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Open table for</strong></td>
</tr>
<tr>
<td>Read</td>
</tr>
<tr>
<td>Change</td>
</tr>
<tr>
<td>Delete</td>
</tr>
<tr>
<td>Insert</td>
</tr>
</tbody>
</table>

Thus a value of (7) opens the file for all operations.

In the third statement you set the CommandText parameter of the Command object to the file name that you want to open. You can use either IFS naming or system naming convention.

/\SYS.LIB/<library>.LIB/<file>.FILE[/<member>.MBR]([<record format>, <commitment control>])

The value of the commitment control parameter overrides the IsolationLevel property of the Connection Object.

In the fourth statement, we create a Recordset object. Then we invoke the Execute method of the Command Object to open the file. The parameters passed are RecordsAffected, Parameters and the type of Command Text. In Delphi, we pass a constant value for the last parameter. In this example we have passed the name of a constant, adCmdTable. This value indicates that the CommandText actually represents the name of a physical file on the AS/400 system. You have to declare the constant like this:

adCmdTable: integer = 2;

For a list of constants please see Execute method of Command object in the SDK Technical Reference and the ADO online help.

**Member Locking**

The AS/400 database file member is locked shared-read or locked shared-update when they are opened, depending on the Updatability property setting. You cannot change the member lock while the member is open. To do so, close the file member and reopen it.

**Record Locking**

If you open a file with an Updatability property of Read, the AS/400 database will not perform record locking. If the file is opened with an Updatability property other than Read (for example, insert, update, or delete), the AS/400 database always will take a pessimistic record-locking strategy. In addition, a record will be locked for update as soon as you are positioned on that record, even though you have not made any actual changes. This will prevent others who have the file opened for update from positioning to the same record. For more information and a list of other record-lock types that are not supported, see the LockType Property topic in the SDK Technical Reference and the ADO online help.
Considerations
ADO always positions you at the first record when you open a file, and the AS/400 database uses pessimistic record locking. This may cause contention if you have opened the file for update operations. You will have the first record locked for update. Others will be able to open the file for update while you remain on this first record. Therefore, you may want to do a MovePrevious and sit at BOF (Beginning of File) until you know which record you want to update.

Another technique used by AS/400 developers is to open the file twice within the application; once for read and once for update. You always use the file opened for read to display records, navigate, and so on. You would always keep the file open for update at BOF or EOF to avoid any record locking. Once you know what record you need to update, you position there, update the record, and move back to BOF or EOF. This technique limits the record-locking to the actual time that it takes to update the record values.

6.4.3 Navigating the Table
You can read all the field names after the table has been opened like this:

```pascal
indx: integer;
For indx:= 0 To rs.Fields.Count - 1 do
  <value> := <recordset>.Fields(Indx).Name;
```

The field values in the current record can be read like this:

```pascal
If Not <recordset>.BOF And Not <recordset>.EOF Then
  begin
    If VarType(<recordset>.Fields(0).Value) = Null Then
      <value> := ''
    Else
      <value> := <recordset>.Fields(0).Value;
  end;
```

Since the AS/400 database opens a file with a dynamic cursor, all the Recordset navigational methods are valid. You can use the MoveFirst, MoveNext, MoveLast and MovePrevious methods to position at a particular record.
6.4.4 Finding a Record

Figure 111. Finding a Record by Key

If the file has defined key fields, you can find a specific record in the file, provided you have a full key value or a partial key value.

You have to use the AD400 Connection object to define, open, and close a connection from the PC client to the AS/400 server. Because you can reuse AD400 connections to the AS/400 system within your application, only one is required for each AS/400 system. This connection enables Find support on keyed files, and is used to manage open indexes on the AS/400 system. An index on the AS/400 system is either a keyed physical or logical file.

Open an AD400 Connection
The following describes how to declare and then open an AD400 connection to the AS/400 system.

```c
//opens a connection to AS/400 server and supports find on key
Cn400Index := Create0leObject('ad400.connection');
// Open the connections
Cn400Index.Open('TCPASM05', '', '');
```

Since User Id and Password are not supported, empty strings have been passed in last two parameters.
Open an Index
You have to use the AD400 Index object to open an index. Once the AD400 index is open, you can find a record by key value and retrieve a bookmark for that record. The bookmark can be used to position to that record in the ADO Recordset object.

You have to pass the <filename> and <indexname> as parameters to the OpenIndex method. Both <filename> and <indexname> must be existing AS/400 keyed physical or logical files. Both system as well as IFS naming convention can be used.

Getting a bookmark
Here is how to find the key value and then move to that value in the Recordset.

```delphi
procedure TForm1.FindClick(Sender: TObject);
begin
  Key1 := InputBox('Find key', 'Enter value for key PARTNO', '');
  Keys := VarArrayCreate([0, 0], varVariant);
  Keys[0] := Key1;
  BookMark := CustMaintIndex.GetBookmark(Keys, 1);
  PartData.Move(0, BookMark);
  //Fill the Form
end;
```

The Keys value is a variant array of key field values that include all, some, or at least one of the key fields of the file. You must pass the key values in the order that matches the defined key field order of the file. In the above code, PartData is the recordset that we opened earlier in Section 6.4.2 "Opening the PARTS Table" on page 215.

6.4.5 Table Operations
If you have opened the file for all the operations (as we have done in this example) then all the updatable Recordset operations are valid.

6.4.5.1 Adding a new record
You can use the Addnew method of the Updatable Recordset object:
6.4.5.2 Deleting a Record

You can delete the current record using the delete method of the Recordset object:

```pascal
procedure TForm1.DeleteClick(Sender: TObject);
begin
  If Not PartData.BOF And Not PartData.EOF Then
  begin
    PartData.Delete(1);
    application.messagebox('Current record was deleted', 'Message', IDOK);
  end
  Else
  application.messagebox('You are currently not positioned on a record. Press next or previous button to be positioned at a record.', 'Message', IDOK);
end;
```
6.4.5.3 Updating a Record
You use the `Update` method of the updatable `Recordset` object to update the record you are currently positioned at.

```pascal
procedure TForm1.UpdateClick(Sender: TObject);
begin
  If Not PartData.BOF And Not PartData.EOF Then
    begin
      Flds := VarArrayCreate([0, 4], varVariant);
      Flds[0] := 'PARTNO';
      Flds[1] := 'PARTDS';
      Vals := VarArrayCreate([0, 4], varVariant);
      Vals[0] := Edit1.Text;
      PartData.Update(Flds, Vals);
    end
  Else
    application.messagebox('You are currently not positioned on a record. Press next or previous button to be positioned at a record.', 'Message', IDOK);
end;
```

6.4.6 Get All Parts - Record-Level Access
To get all the records from the database file, we have to open the table as illustrated in Section 6.4.2 "Opening the PARTS Table" on page 215. When you open the table, it returns a result set with the record pointer positioned at the first record.

To get all the records from the table, we navigate through the Recordset and populate values from the Recordset to the data control of our choice. We discussed this earlier in Section 6.2.3.2 "Populate the ListView" on page 206.

The only difference is that the result set here is dynamic, unlike the result set returned by execution of the SQL statement, and every time you do a MoveNext, MovePrevious, MovePrevious or MoveLast, data is actually retrieved from the server.
6.4.7 Commitment Control

You can specify the Commitment control at the Connection level. The following are the valid values for the IsolationLevel property of the ADO Connection object. The default, if the IsolationLevel property is not specified, is Cursor Stability (adXactCursorStability).

Table 39. Commitment Control Options

<table>
<thead>
<tr>
<th>ADO IsolationLevel Property</th>
<th>AS/400 Commitment Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chaos (adXactChaos)</td>
<td>NONE (*NC)</td>
</tr>
<tr>
<td>Browse (adXactBrowse)</td>
<td>*CHG(*UR)</td>
</tr>
<tr>
<td>CursorStability(adXactCursorStability)</td>
<td>*CS</td>
</tr>
<tr>
<td>Repeatable Read (adXactRepeatableRead) *</td>
<td>*ALL</td>
</tr>
</tbody>
</table>

All files are opened under commitment control in ADO. This is true if you specify a value for IsolationLevel or simply take the default. On the AS/400 system, all AS/400 files that are opened under commitment control require the use of journals. Create a journal and a journal receiver for all of the files that you open by using the IsolationLevel property. If your files are not journaled when you open them by using the connection isolation level, an error is issued.
Also, ADO does an automatic commit whenever a transaction (BeginTrans) is not explicitly started. This causes a commit to be processed after every transaction so that all changes are automatically committed to the database file. The commitment control level that is used for this automatic commit is controlled by the OLE DB provider AutoCommit property, which is not available and cannot be set through ADO. To enable the user to set the AutoCommit property, the IBMDA400 provider sets this property to match the last transaction isolation level. The IBMDA400 default for the AutoCommit property is Chaos (^NONE).

When you use the DB2/400 database, commitment control for the file is specified when the file is opened. Because ADO holds the isolation level from the OLE DB provider until it is needed on a BeginTrans, and since you will want to set the AutoCommit property to match the IsolationLevel property under which you will be running, you need to explicitly begin and end a transaction before opening any AS/400 files.

However, you can override the commitment control specified at Connection level by specifying the file-level commitment Control. If you do not specify a file-level commitment control when you open the file, the IsolationLevel property of the ADO connection object is used.

Transactions are started, committed, and rolled back at the ADO connection object level. It is illustrated as:

```pascal
<connection>.IsolationLevel := 256;
<connection>.BeginTrans;
// some code
<connection>.EndTrans;
// if error
<connection>.RollbackTrans;
//else
<connection>.CommitTrans;
```

With Delphi, use the constant value (256) instead of the constant name (adXactBrowse). For a list of constant values, see the IsolationLevel Property topic in the SDK Technical Reference and the ADO online help.

### 6.4.7.1 Restrictions
The IBMDA400 provider only supports commitment control or transaction control for record-level access tables. It does not support transaction control for SQL statements.

The IBMDA400 provider does not support nested transactions. If you have already started a transaction with `<connection>.BeginTrans`, you cannot start another transaction until you have either done a commit or rollback operation.

### 6.4.8 Closing the Table
You can close an open table and free the AS/400 file locks and file record locks that exist on the file. This can be done by:

```pascal
<recordset>.Close;
```

You can choose to reuse the `Recordset` object or release the object from memory.

```pascal
<recordset> := Null;
```
This releases all of the Recordset object resources and eliminates the object from memory. See the Closing a Recordset and the Releasing a Recordset object topics of the ADO Recordset Object for more information.

### 6.4.9 Error Handling

See Section 6.2.4 "Error Handling" on page 207

### 6.5 A Data Queue Application Example

IBMDA400, the Client Access OLE DB provider, allows you to interface with existing AS/400 data queues. It allows you to place records on and receive records off of AS/400 data queues. The support also handles conversions from AS/400 formats (for example, Packed Decimal) to PC formats.

A data queue is an AS/400 object that allows a program (host or client) to place data into it and retrieve data from it. When you open a data queue you can specify a record format for the data queue entries.

A data queue on the AS/400 system can have FIFO (First In First Out), LIFO (Last In First Out), or keyed-entry sequence. There is also a maximum entry length that is defined when the data queue is created. In the example that we discuss in this section, a client program places a request (entry) on an AS/400 data queue. Running on the AS/400 system is a program (DQXRPG) that reads the request off of the data queue and accesses the PARTS file. In response, the AS/400 program places records from the PARTS file on the output data queue.

You can use the Create Data Queue (CRTDTAQ) CL command to create a data queue on the AS/400 system.

Data queues have following characteristics:

- Data queues can be accessed by many jobs simultaneously.
- They can be used to hold data being passed back and forth between jobs.
- Messages on a Data Queue are free format.

The following is an illustration of how to use IBMDA400 to read or write to Data Queues.

In this example, the Delphi application requests data from the server program by placing a request on the input data queue DQINPT. The input is a flag and a number, for example "S12301" is requesting a single record of part number 12301. If you have requested all parts (FLAG = "A") the part number is not necessary. The server program searches for the information in the database and writes the result to the output data queue. For a listing of DQXRPG, see "DQXRPG/RPGLE" on page 378.
This screen shows the result of requesting information about one part (12325).
This screen shows the result of requesting information about all the parts in the PARTS file.

### 6.5.1 Data Queue Server Program Background

An input queue and output queue were created with the commands:

```
CRTDTAQ DTAQ(APILIB/DQINPT) MAXLEN(6) TEXT('Data Queue for Parts Input)
CRTDTAQ DTAQ(APILIB/DQOUPT) MAXLEN(48) TEXT('Data Queue for Parts Output)
```

Program Name: APILIB/DQXRPG

#### Table 40. Data Queue DQINPT Layout

<table>
<thead>
<tr>
<th>Queue Position</th>
<th>Description</th>
<th>Length Type</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 1</td>
<td>Action Flag</td>
<td>1 character</td>
<td>S - Retrieve single record</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A - Position to start of file</td>
</tr>
<tr>
<td>2 - 6</td>
<td>Part Number</td>
<td>5.0 Zoned</td>
<td>Part number to retrieve</td>
</tr>
</tbody>
</table>

#### Table 41. Data Queue DQOUPT Layout

<table>
<thead>
<tr>
<th>Queue Position</th>
<th>Description</th>
<th>Length Type</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 1</td>
<td>Action Flag</td>
<td>1 character</td>
<td>Y - operation succeeded</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X - record not found or EOF</td>
</tr>
<tr>
<td>2 - 6</td>
<td>Part Number</td>
<td>5.0 Zoned</td>
<td>part to be retrieved</td>
</tr>
<tr>
<td>7 - 31</td>
<td>Description</td>
<td>25 character</td>
<td></td>
</tr>
<tr>
<td>32 - 34</td>
<td>Quantity</td>
<td>5.0 packed</td>
<td></td>
</tr>
</tbody>
</table>
This server program should be active before you use the application. You can run this program using the **SBMJOB** command on the AS/400 system.

<table>
<thead>
<tr>
<th>Queue Position</th>
<th>Description</th>
<th>Length Type</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>35 - 38</td>
<td>Price</td>
<td>6.2 packed</td>
<td></td>
</tr>
<tr>
<td>39 - 48</td>
<td>Ship date</td>
<td>10 character</td>
<td></td>
</tr>
</tbody>
</table>

6.5.2 Opening the Data Queue

By now you know that you have to create a **Connection** object and open it. See Section 4.1.1, “Opening a Connection” on page 30 for details.

You will also need to create two **Recordset** objects, one each for the input data queue and the output data queue.

Data queue are opened by using `Execute` method of **Connection** object as shown:

```delphi
<recordset> = <connection>.Execute("open dataqueue <data queue> [(record format)] [KEY(<key format>)] for <option>"
```

**Record format**

It is a comma-separated list of field name, type and optional CCSID.

**Key format**

This is valid only for keyed data queue. It is a comma-separated list of a field name, type, and optional CCSID.

**Option**

You have to set this as `SEND` or `RECEIVE` to write to a data queue or read from a data queue.

```delphi
begin
  Application.Initialize;
  Cn400 := CreateOleObject(’adodb.connection’);
  Cn400.Open(’Provider=IBMDA400;Data Source=TCPASM05;’,’’);’
  DqInpt := CreateOleObject(’adodb.recordset’);
  DqOupt := CreateOleObject(’adodb.recordset’);
  //open data queues for send and recieve
  DqInpt := Cn400.Execute(’open dataqueue apilib.dqinpt(Option CHARACTER(1),part CHARACTER(5) )for Send’,Rcds,1);
  DqOupt := Cn400.Execute(’open dataqueue apilib.dqoupt(Option CHARACTER(1),Part CHARACTER(5),Desc CHARACTER(25),Qty DECIMAL(5,0),Price DECIMAL(6,2), Date CHARACTER(10) )for Receive’,Rcds,1);
  // some code
end.
```

**Rcds** is `RecordsAffected` and it is not supported by IBMDA400. With Delphi, use the constant value (1) in place of the constant name (adCmdText). For a list of...
constant values, see the Execute Method topic in the *SDK Technical Reference* and the ADO online help.

### 6.5.2.1 Reading Field Names
Once the data queue is open, you can read all of the field names that compose the data queue entry format. These names will be the same as those you specified on the open of the data queue. This is like reading the field names from a *Recordset* object. See Section 6.2.3.1 "Navigating the Result Set" on page 206.

### 6.5.2.2 Navigating Read Entries
When you open a data queue for RECEIVE, all the entries are read into the *Recordset* object, and you are positioned at the first entry.

You can use the *Recordset* object methods (for example MoveFirst, MoveNext, MovePrevious, MoveLast) for navigating through the Recordset. The Recordset is not updatable and thus the AddNew, Update, and Delete will generate errors.

Please see Section 6.2.3.1 "Navigating the Result Set" on page 206.

### 6.5.2.3 Rereading Entries
If you open a data queue for RECEIVE, you can reread any new data queue entries by using the Requery method of the *Recordset* object. All existing records in the *Recordset* object are deleted and replaced with the reread entries from the data queue. If there are no new entries on the data queue, the resulting Recordset will be empty.

```vbscript
If <recordset>.EOF Then
   <recordset>.Requery;
   <navigate the read entries>
```

### 6.5.2.4 Adding an Entry
If you open a data queue for SEND, you can add new entries to the data queue using the AddNew method of the *Recordset* object. See Section 6.4.5.1 "Adding a new record" on page 219. The Update and Delete methods are not supported and will produce errors.
6.5.3 Get One Part - Data Queue

When you enter the part number and press the button "Get Part" the following of code is executed.

```pascal
procedure TAppli.GetPartClick(Sender: TObject);
begin
  Flds := VarArrayCreate([0,1], varVariant);
  Flds[0] := 'Option';
  Flds[1] := 'part';
  Vals := VarArrayCreate([0,1], varVariant);
  Vals[0] := 'S';
  DqInpt.AddNew(Flds, Vals);
  DqOupt.ReQuery;

  //load the data
end
```

In this code, we are placing a record containing "s" (option for single record) and the part number (for which we want data). The server program (DQXRPG) searches the database and places the requested record on the output queue(DQOUPT). If the record is not found, the server program places an "X" in the first field of the output data queue.

The client program executes a ReQuery command on the data queue and the record is loaded in the Recordset object. We then populate the field values from the Recordset object. See Section 6.2.2 "Get One Part-SQL" on page 203 for details.
6.5.4 Get All Parts - Data Queue

When we request all the parts, following is the code is executed.

```pascal
procedure TAllParts.FormCreate(Sender: TObject);
begin
  Flds := VarArrayCreate([0,1], varVariant);
  Flds[0] := 'Option';
  Flds[1] := 'part';
  Vals := VarArrayCreate([0,1], varVariant);
  Vals[0] := 'A';
  Vals[1] := '';
  DqInpt.AddNew(Flds, Vals);
  LoadListView(AllPart, DqOupt);
end;
```

The only difference in the code from what you saw earlier are the values that we place on the Input data queue. Here we place value of "A" for the option field. This means that the server program retrieves all the records and places them on the output data queue.

When we execute the ReQuery method on the output data queue, the whole result set is loaded in the Recordset object.

Finally, we populate the ListView from this Recordset object as shown in Section 6.2.3.2 "Populate the ListView" on page 206.

6.5.5 Closing a Data Queue

Closing a data queue releases the lock that is on the DTAQ object on the AS/400 system. This can be accomplished by closing the Recordset and releasing all the resources by eliminating the object from the memory.

```pascal
//Closing the Recordset
<recordset>.Close

//Releasing the Recordset
SetNull(<recordset>)
```

6.6 Remote Commands Execution Example

IBMDA400, the Client Access OLE DB provider, allows you to interface with AS/400 CL commands. There is a support to call an AS/400 command and pass command parameters.

6.6.1 Executing a Remote Command

You use the Execute method of the Connection object to execute the CL commands. You need to create a Connection object and open a connection to the AS/400 system as illustrated in the earlier examples.
The parameters passed to the Execute method are CommandText, RecordsAffected and adCmdText.

CommandText is the CL command surrounded by two pairs of curly braces. If you want to accept user input as part of the command, you will have to concatenate the user input to the actual command.

The parameter adCmdText is the name of the constant which determines the type of CommandText. With Delphi use the constant value (1) instead of the constant name (adCmdText). For a list of constant values, see the Execute Method topic of the SDK Technical Reference and the ADO online help.

```csharp
//Execute CL command
<connection>.Execute("{{QSYS/CRTLIB LIB(APILIB)}}",,1)
```

### 6.6.2 Restrictions

You can’t call interactive CL commands. Only CL commands which can be called in batch can be executed. IBMDA400 does not support preparing commands as can be done for SQL statements. Therefore, you should concatenate the command text and the command values together to form the complete command.

### 6.7 A Distributed Program Call (DPC) Application Example

IBMDA400, the Client Access OLE DB provider, lets you call AS/400 programs. You can pass both input and output parameters. For more information on creating programs on the AS/400, see the appropriate AS/400 language reference manual for your particular programming language.

![Diagram of Distributed Program Call Example](image)

*Figure 117. Delphi Distributed Program Call Example*
Following is an illustration of how to use IBMDA400 to call an AS/400 program. In this sample, the Client program calls an AS/400 program, DPCXRPG, with a parameter that is the value of a part number. This program queries the database file and returns back details about the PART in output parameters. See “DPCXRPG/RPGLE” on page 377 for a complete listing of the program DPCXRPG.

Figure 118. Get a Part - Program Call

This screen shows the result of requesting information about one part from the PARTS file. There is an option to get all the parts from PARTS file.

Figure 119. Get All Parts - Program Call
The details of the RPG program DPCXRPG are:

**Table 42. Details of DPCXRPG**

<table>
<thead>
<tr>
<th>Seq</th>
<th>Description</th>
<th>Length Type</th>
<th>I/O</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Action Flag</td>
<td>1 Character</td>
<td>I/O</td>
<td>S - Retrieve single record</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A - Position to the start of file</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F - Fetch the next record</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OUTPUT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y - Operation succeeded</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X - Operation failed or EOF found</td>
</tr>
<tr>
<td>2</td>
<td>Part Number</td>
<td>5.0 Packed</td>
<td>I/O</td>
<td>Part Number to be retrieved or retrieved</td>
</tr>
<tr>
<td>3</td>
<td>Description</td>
<td>25 Character</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Quantity</td>
<td>5.0 Packed</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Price</td>
<td>6.2 Packed</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Ship Date</td>
<td>10 Date</td>
<td>O</td>
<td></td>
</tr>
</tbody>
</table>

### 6.7.1 Defining the Program Call

You need to create a **Connection** object and open it. Also you have to create a **Command** object. Please go back to earlier sections for details on how to do that.

```delphi
dpcxrpgCmd := CreateOleObject('adodb.command');
dpcxrpgCmd.ActiveConnection := 'Provider=IBMDA400;Data Source=TCPASM05;';
dpcxrpgCmd.CommandText := '{{call apilib/dpctest(?,?,?,?,?,?)}}';
dpcxrpgCmd.Prepared := True;
```

Please see Section 6.3.2 "Preparing for Stored Procedure Call" on page 210 for details. The only difference is that you have to put two pairs of curly braces around a program call. This is the way, IBMDA400 differentiates between a program call and a Stored Procedure call. You have to specify a list of comma-separated question marks as placeholders for each of the program parameters.

Note that when a call to an AS/400 program is prepared, there is no existing information on the AS/400 system that describes the program parameters. To create program parameter information, you should catalog and call the program as a stored procedure. Since AS/400 program information does not exist, you must create and specify all of the program parameters in your application as shown here:
With Delphi, use the constant values in place of the constant names (adParamOutput, adParamInput, ad data types) while creating the parameters. For a list of constant values, see the Direction Property and Type Property topics in the Microsoft ADO online help. This documentation is shipped with the SDK and the SDO online help, see Appendix A.2, “Other Documentation and Resources” on page 373 for details.

Next you have to set the Precision and NumericScale, for those data types (adDecimal and adNumeric) that require them. This only is necessary when calling an AS/400 program.

### 6.7.2 Get a Part Using Program Call

The next step is to Execute the Command. It is similar to executing a Stored Procedure call. Please see Section 6.3 “A Stored Procedure Application Example” on page 208 for details. The code below illustrates how to execute the prepared Command object.

```delphi
DpcxrpgCmd.Parameters.Append(DpcxrpgCmd.CreateParameter('opt', 129{adChar}, 3{adParamInputOutput}, 1));

DpcxrpgCmd.Parameters.Append(DpcxrpgCmd.CreateParameter('partno', 14{adDecimal}, 3{adParamInputOutput}, 0));
    DpcxrpgCmd.Parameters[1].Precision := 5;
    DpcxrpgCmd.Parameters[1].NumericScale := 0;

DpcxrpgCmd.Parameters.Append(DpcxrpgCmd.CreateParameter('partds', 129{adChar}, 2{adParamOutput}, 25));

DpcxrpgCmd.Parameters.Append(DpcxrpgCmd.CreateParameter('qty', 14{adDecimal}, 2{adParamOutput}, 0));
    DpcxrpgCmd.Parameters[3].Precision := 5;
    DpcxrpgCmd.Parameters[3].NumericScale := 0;

DpcxrpgCmd.Parameters.Append(DpcxrpgCmd.CreateParameter('price', 14{adDecimal}, 2{adParamOutput}, 0));
    DpcxrpgCmd.Parameters[4].Precision := 6;
    DpcxrpgCmd.Parameters[4].NumericScale := 2;

DpcxrpgCmd.Parameters.Append(DpcxrpgCmd.CreateParameter('pdate', 129{adChar}, 2{adParamOutput}, 10));
```
Figure 120. Get a Part - Program Call

Here we pass a value of 's' in the first parameter and part number in the second parameter. This means that we are requesting a single record matching the value of the part number entered as shown.

```pascal
procedure TAppli.GetPartClick(Sender: TObject);
begin
  Partnumber := PartNumb.Text;
  Parms := VarArrayCreate([0,5], varVariant);
  Parms[0] := 'S';
  Parms[1] := StrToFloat(Partnumber);
  Parms[3] := 0;
  Parms[4] := 0;

  DpcxrpgCmd.Execute(Rcds, Parms,1);
  // read the output parameters
end;
```

The next step is to read the output parameters returned by the AS/400 program. It is accomplished like this:
6.7.3 Get All Parts Using Program Call

To get all the records, we first call the DPCXRPG program with value 'A' for the first parameter. It positions to the first record in the file.

```
If (DpcxrpgCmd.Parameters[0].Value = 'Y') Then
  Begin
    PartDes.Text := DpcxrpgCmd.Parameters[2];
    Quant.Text := DpcxrpgCmd.Parameters[3];
    Price.Text := DpcxrpgCmd.Parameters[4];
    RcvDate.Text := DpcxrpgCmd.Parameters[5];
  end
```

```
Parms := VarArrayCreate([0,5], varVariant);
Parms[0] := 'A';
Parms[1] := 0;
Parms[3] := 0;
Parms[4] := 0;
DpcxrpgCmd.Execute(Rcds,Parms,1);```

Then we call DPCXRPG with a value 'F' for the first parameter. This fetches the next record in file. We populate the values that the program returns in the data component which is a ListView in this case. We continue this process in a loop until we are returned EOF(End of File). See the code below:
6.7.4 Restrictions

test : boolean;
test := false
While Not test do
begin
  Parms[0] := 'F';
  Parms[1] := 0;
  Parms[3] := 0;
  Parms[4] := 0;
  DpcxrpCmd.Execute(Rcds, Parms, 1);
  If DpcxrpCmd.Parameters[0].Value = 'X' then test := true
  Else
    begin
      For i := 1 to 5 do
        begin
          If i = 1 Then
            begin
              itm := AllPart.Items.Add;
            end
          Else
            itm.SubItems.Add(DpcxrpCmd.Parameters[i].Value);
          end;
        end;
    end;
end;

Calling programs on the AS/400 system that have user defined parameters or parameters that are structures are not supported.
Chapter 7. Lotus Notes with OLE DB

Lotus Notes is a groupware software solution that combines messaging, groupware and the Internet. It is based on a client/server architecture.

Lotus Notes provides you the facility to access data in relational databases and to interface with transaction processing systems, with LotusScript extensions for ODBC. LotusScript is an object oriented structured programming language that interfaces to Notes through predefined object classes. Lotus Notes has a LotusScript:Data Object, which gives you read and write access via the SQL language to relational databases.

This chapter covers using LotusScript with IBMDA400, the Client Access OLE DB provider. We assume that you are familiar with the ADO Programming Model. For a detailed discussion of the ADO programming Model see Chapter 1, “OLE DB and ActiveX Overview” on page 1.

Using IBMDA400, you can access AS/400 databases using SQL. Also you can access data queues, call programs and stored procedures, and execute AS/400 commands from the Lotus Notes environment.

This chapter provides examples that demonstrate how to use IBMDA400 to access AS/400 resources from Lotus Notes.

7.1 Introduction to Application Examples

The remainder of this chapter covers application examples (Lotus Notes Forms and Agents) that demonstrate:

AS/400 resource access via:

• SQL
• Stored Procedures
• Table Record-Level Access
• AS/400 Data Queues
• Remote Commands
• Program Calls

The database access example applications discussed in this chapter allow retrieval of a single record or all the records from a PARTS file on the AS/400 system. Some examples allow update of the current Record in the form. Please note that all these examples have been tested to work with ADO 1.5 and Lotus Notes 4.6.

Note

The level of ADO shipped with the Client Access OLE DB provider is ADO1.0. To obtain ADO 1.5, see www.microsoft.com/data/mdac15.htm.
We have used the PARTS table in the APILIB library in all the examples except the Notes application in Section 7.8 "A Sample Notes Application" on page 262. The PARTS file is defined as:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Description</th>
<th>Length</th>
<th>Decimals</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARTNO</td>
<td>Part Number</td>
<td>5</td>
<td>0</td>
<td>Zoned</td>
</tr>
<tr>
<td>PARTDS</td>
<td>Part Description</td>
<td>25</td>
<td></td>
<td>Char</td>
</tr>
<tr>
<td>PARTQY</td>
<td>Quantity</td>
<td>5</td>
<td>0</td>
<td>Packed</td>
</tr>
<tr>
<td>PARTPR</td>
<td>Price</td>
<td>6</td>
<td>2</td>
<td>Packed</td>
</tr>
<tr>
<td>PARTDT</td>
<td>Part Shipment Date</td>
<td>10</td>
<td></td>
<td>Date</td>
</tr>
</tbody>
</table>

All the examples discussed in this chapter are available for download on the www.redbooks.ibm.com web-site. See Appendix A.1, “Downloading the Samples” on page 373 for details.

7.2 An SQL Example

![Diagram of Lotus Notes Using SQL]

Figure 121. Lotus Notes Using SQL
In this example we have a simple Notes form that has five fields. When a user enters the Part Number (which is the first field in the form) and exits out of that field, we retrieve a record with the matching part number from PARTS file in the AS/400 database and populate the form. We have written that script in the Exiting event of the Part Number field.

We execute a script when the user exits from the Part Number field in a Parts Form document. The script executes a query using the value of Part Number from the form in the where clause of an SQL SELECT statement. It then fills in the PartName, Cost, and Description fields based on values in the retrieved record.

We will discuss the LotusScript written for this purpose. We have used two ADO objects, Connection and Recordset. First, we create a Connection object and invoke its Open method to establish a connection to the AS/400 system.

The ConnectionString part contains the provider and data source names. The Data Source name must be a valid AS/400 connection name. You should have a valid connection configured to an AS/400 system.

Empty strings are passed for the User ID and Password parts. The IBMDA400 provider does not support the passing of the User ID and Password when you open a connection. Client Access prompts the user for connection information as it is needed.
When a user inputs the Part Number and exits the field, the following code is executed. In this code, we prepare an SQL query with a parameter marker and then we execute that query based on the value that the user inputs. Here, we use the `Execute` method of the `Command` object.

We set the `ActiveConnection` property to the open `Connection`. We set the `CommandText` to the SQL statement in which we use a parameter marker. Now we are ready to execute the prepared SQL statement. It is recommended that you prepare the SQL statements before executing as it results in better performance.
The parameters passed to the **Execute** method are **CommandText**, **RecordsAffected** and **adCmdText**.

The parameter **adCmdText** is the name of the constant which determines the type of **CommandText**. With Lotus Notes use the constant value (1) instead of the constant name (**adCmdText**). For a list of constant values, see the **Execute Method** topic in the *SDK Technical Reference* and the ADO online help. For information on how to obtain these references, see Appendix A.2, “Other Documentation and Resources” on page 373.

```vbnet
' if result set is not empty and we are at first row
If rs.BOF And rs.EOF Then
    MessageBox ("No record found")
Else
    Call uidoc.FieldSetText("Partno", _
        rs.fields(0).value)
    Call uidoc.FieldSetText("PartDesc", _
        rs.fields(1).value)
    Call uidoc.FieldSetText("PartQty", _
        rs.fields(2).value)
    Call uidoc.FieldSetText("PartPrc", _
        rs.fields(3).value)
    Call uidoc.FieldSetText("PartDt", _
        rs.fields(4).value)
End If
```

After we have executed the SQL statement, the result set is returned in a **Recordset** object. In this example only one record is returned. We load and display the values in the form as shown:
In the terminate event, we close both the Connection and Recordset objects. Doing so releases all the resources held by these objects. It also releases locks, if any, on the AS/400 objects.

7.2.1 Error Handling

The Connection object contains the errors collection. All PC client and AS/400 error and warning messages are returned in the error collection at the connection level. You can check for any exception by navigating through the error collection as shown below:

```
'In the terminate event

Sub Terminate
    cn.close
    rs.close
End Sub
```

In the terminate event, we close both the Connection and Recordset objects. Doing so releases all the resources held by these objects. It also releases locks, if any, on the AS/400 objects.
For more information on Errors, see the Error Object and Errors Collection topics of the SDK Technical Reference and the ADO online help.

Use the cwbbzztrc.exe program to work with error logging. You can run the program from a DOS command prompt. Run the program without any parameters to get the options and syntax. This program is installed when you install Client Access. The following is the default installation path and log file path:

C:\Program Files\IBM\Client Access\cwbbzztrc.exe
C:\Program Files\IBM\Client Access\Service\Trace Files\IBMDA400.log

7.3 A Stored Procedure Example

The AS/400 OLE DB provider allows you to call Stored Procedures on the AS/400 system. A Stored Procedure is a cataloged program that may be written in most of the programming languages supported on the AS/400 system. To catalog an AS/400 program, you can use the CREATE PROCEDURE SQL statement. For more information on SQL statements and Stored Procedure see the DB2 for AS/400 SQL Programming Manual, SC41-5611.

A Stored Procedure can be used to control multiple database operations on the server thereby reducing the line turn-arounds. Thus, in client/server applications, Stored Procedures can be used to improve performance by reducing the communication overhead. A Stored Procedure may take one to up to 256 parameters. The output is returned in OUT (or INOUT) parameters or via SQL result sets.

```
processerror:
  j = cn.errors.count
  j = j - 1
  For i = 0 To cn.errors.count - 1
    errmsg = ""
    errmsg = errmsg + "Number : " + Str (cn.errors.Item(i).Number) + " 
    " + "Native error:" + Str (cn.errors.Item(i).NativeError) + " 
    + "SQL State:" + cn.errors.Item(i).SQLState + " 
    + "Source:" + cn.errors.Item(i).Source + " 
    + "Description:" + cn.errors.Item(i).Description
    MessageBox errmsg, MB_OK, "Error Occured"
  Next
  Call cn.errors.clear()
Exit Sub
```
The following is an illustration of how to use the AS/400 OLE DB provider to call stored procedures. In this example, a Lotus Notes agent calls a stored procedure SPROC2 in the APILIB library.

The Stored Procedure can be created from the interactive SQL prompt on the AS/400 system using the following statement:

```
***Create Procedure***
CREATE PROCEDURE APILIB/SPROC2 (
  INOUT P1 INTEGER,
  INOUT P2 INTEGER)
EXTERNAL NAME APILIB/SPROC2 LANGUAGE RPG GENERAL
```

The stored procedure is named SPROC2. It calls an AS/400 RPG program also named SPROC2. This program takes in two integer input/output parameters. Based on the parameters passed, it returns either one part on all the parts. For a complete listing of the RPG program, SPROC2, see “SPROC2/SQLRPGLE” on page 375.

This program takes in two integer input parameters. Based on the parameters passed it returns either one part on all the parts. A value of ‘1’ in the first parameter returns a single record in the result set with the part number field matching the part number supplied in the second parameter. A value of ‘2’ in the first parameter returns all records from the parts database in a result set. The second parameter is ignored in this case.

In this Lotus Notes agent, we pass a value of ‘2’ in the first parameter so that we get all the records back.
We first declare a **connection object**, a **resultset object** and a **command object**.

We open a connection to the AS/400 system using the `Open` method of the **Connection object**. We create a **Command object** and prepare it for the stored procedure call. The **CommandText** property is set to the stored procedure call with parameter markers. These parameters are passed in when it is executed. Please note that the stored procedure call statement has been enclosed by a pair curly braces.
In the next step we set the Direction property of the Command Parameters. This indicates the type of parameters that IN, OUT or INOUT. A constant value of 3 indicates that parameters are of type INOUT.

We create a variant array and pass values of 2 and 0 for the first and second parameters respectively. When we execute the Command, a result set is loaded into the Recordset object. We navigate through the result set and display it as shown below.

![Figure 125. Lotus Notes Agent that Calls a Stored Procedure on the AS/400 System](image)

The IBMDA400 provider does not support multiple result sets. You can call a stored procedure that returns multiple result sets, but you will be able to process only the first result set.
7.4 A Record Level Access Example

IBMDA400 allows you to interface with AS/400 physical files and logical files through record-level access. This interface allows record-level access to AS/400 databases. The host server support is provided by the OS/400 DDM server. It provides the ability to read by key, update, delete and insert records into the database. V4R2 provides TCP/IP connectivity support, whereas connection to the prior releases of OS/400 is through SNA only. If you require a TCP/IP connection while using an OS/400 release prior to V4R2, you can obtain PTFs which provide this support. See “Record-Level Access” on page 63 for details.

In this example we have a simple form based on the PARTS table in the APILIB library on the AS/400 system. We use record-level access to interface with the PARTS file. We can navigate through the Recordset and update any record in the AS/400 database.
Next, we take a look at the step-by-step implementation of this example.

7.4.1 Opening the File for Record-Level Access

The first step is to open a connection to the AS/400 system and then open the physical file. For this we need to create a `Connection` object and use its `Open` method.
Here we create Connection, Recordset and Command objects. Then we invoke the Open method of the Connection object to establish a connection to the AS/400 system.

Then we associate the ActiveConnection property of the Command object to the open connection. The next statement sets the Updatability property of the file that we open. You should assign the Updatability property a sum of the Updatability values that apply. For example, a value of "7", indicates open with all capabilities.

Table 44. Table Updatability Values

<table>
<thead>
<tr>
<th>Open table for</th>
<th>Updatability value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>0</td>
</tr>
<tr>
<td>Change</td>
<td>1</td>
</tr>
<tr>
<td>Delete</td>
<td>2</td>
</tr>
<tr>
<td>Insert</td>
<td>4</td>
</tr>
</tbody>
</table>

Next we set the CommandText parameter of the Command object to the file name that we want to open. You can use either IFS naming or system naming convention.

/QSYS.LIB/<library>.LIB/<file>.FILE[/<member>.MBR]([<record format>, <commitment control>])

A parameter P1 is created and appended to the Command object. You have to do this in order to open the table. For details about the parameters to the CreateParameter method of the Command object please refer to the SDK Technical Reference and the ADO online help.
We invoke the `Execute` method of the `Command` object to get the file opened. This returns a result set that is loaded into an ADO Recordset object.

### 7.4.2 Navigating the Recordset

Here we use the `MoveFirst`, `MoveNext`, and `MoveLast` methods of ADO Recordset object to navigate through the result set.

The code below demonstrates moving to the first record and populating the form from the result set.

```vba
' code for move to the first record.
Sub Click(Source As Button)
    Dim ws As New Notesuiworkspace
    Dim uidoc As Notesuidocument
    Set uidoc = ws.CurrentDocument

    On Error Goto label1
    rs.MoveFirst
    If rs.BOF And rs.EOF Then
        Messagebox ("No record found")
    Else
        Call uidoc.FieldSetText("Partno", _
            rs.fields(0).value)
        Call uidoc.FieldSetText("PartDesc", _
            rs.fields(1).value)
        Call uidoc.FieldSetText("PartQty", _
            rs.fields(2).value)
        Call uidoc.FieldSetText("PartPrc", _
            rs.fields(3).value)
        Call uidoc.FieldSetText("PartDt", _
            rs.fields(4).value)
        End If
    label1:
        Call cn.errors.clear()
End Sub
```

### 7.4.3 Updating the Table

We use the `Update` method of the Recordset object to update any of the records in the table.
In this code we create two variant arrays, one containing the field names as defined in the physical file and the other having values to be updated in a particular record. Then we invoke the `Update` method of the `Recordset` object to update these values in the database.

Likewise, we can use the `Delete` and the `AddNew` methods of the `Recordset` object to delete and add records to the database file. Please note that all these operations are allowed if the file is opened with the `Updatability` property equal to 7.

### 7.4.4 Finding a Record

IBMDA400 uses the AD400 Connection and AD400 Index object to locate a particular record based on a key value. The sample code for finding a record is shown below:

```vba
Sub Click(Source As Button)
    Dim ws As New Notesuiworkspace
    Dim uidoc As Notesuidocument
    Set uidoc = ws.CurrentDocument

    Dim flds As Variant
    Dim vals As Variant
    Dim flds1(4) As Variant
    Dim vals1(4) As Variant

    flds1(0) = "PARTNO"
    flds1(1) = "PARTDS"
    flds1(2) = "PARTQY"
    flds1(3) = "PARTPR"
    flds1(4) = "PARTDT"

    vals1(0) = uidoc.FieldGetText("Partno")
    vals1(1) = uidoc.FieldGetText("PartDesc")
    vals1(2) = uidoc.FieldGetText("PartQty")
    vals1(3) = uidoc.FieldGetText("PartPrc")
    vals1(4) = uidoc.FieldGetText("PartDt")

    flds = flds1
    vals = vals1
    rs.Update flds, vals

End Sub
```
7.5 A Data Queue Application Example

The Client Access OLE DB provider allows you to interface with existing AS/400 data queues. It allows you to place records on and receive records off of AS/400 data queues. The support also handles conversions from AS/400 formats (for example, Packed Decimal) to PC formats.

A data queue is an AS/400 object that allows a program (host or client) to place data into it and retrieve data from it. When you open a data queue you can specify a record format for the data queue entries.

A data queue on the AS/400 system can have FIFO (First In First Out), LIFO (Last In First Out), or keyed entry sequence. There is also a maximum entry length that is defined when the data queue is created. In the example we discuss, a client program places a request (entry) on an AS/400 data queue. Running on the AS/400 system is a program (DQXRPG) that reads the request off of the data queue and accesses the PARTS file. In response, the AS/400 program places records from the PARTS file on the output data queue.

You can use the Create Data Queue (CRTDTAQ) CL command to create a data queue on the AS/400 system.

Data queues have the following characteristics:

- Data queues can be accessed by many jobs simultaneously.
- They can be used to hold data being passed back and forth between jobs.
- Messages on a Data Queue are free format.
Figure 128 on page 255 is an illustration of how to use IBMDA400 to read or write to data queues.

In this example, the client program requests data from the server program (DQXRPG) by writing to input data queue DQINPT. The input is a flag and a part number. For example, "S12301" is a request for a single record of part number 12301. If requesting all parts (FLAG = "A") the part number is not necessary. The server program searches for the information in the database and writes the result to the output data queue. See “DQXRPG/RPGLE” on page 378 for a complete listing of DQXRPG.

7.5.1 **Data Queue Server Program Background**

An input queue and output queue were created with the commands:

```plaintext
CRTDTAQ DTAQ(APILIB/DQINPT) MAXLEN(6) TEXT('Data Queue for Parts Input')
CRTDTAQ DTAQ(APILIB/DQOUPT) MAXLEN(48) TEXT('Data Queue for Parts Output')
Program Name : APILIB/DQXRPG
```
This server program should be active before you use this application. You can run this program using SBMJOB command on the AS/400 system.

7.5.2 Opening the Data Queue

First we create a Connection object and open it. Then we create two Recordset objects, one each for the input data queue and the output data queue.

Data queues are opened by using the Execute method of the Connection object as shown:

```vbnet
<recordset> = <connection>.Execute("open dataqueue <data queue> [(record format)] [KEY(<key format>)] for <option>
```

**Record format**

A comma-separated list of field name, type and optional CCSID.

**Key format**

This is valid only for keyed data queue. It is a comma-separated list of a field name, type, and optional CCSID.

**Option**

This value is set to SEND to write to a data queue or RECIEVE to read from a data queue.

7.5.2.1 Reading Field Names

Once the data queue is open, you can read all of the field names that compose the data queue entry format. These names will be the same as those you specified on the open of the data queue.
7.5.2.2 Navigating Read Entries
When you open a data queue for RECEIVE, all the entries are read into the Recordset object, and you are positioned to the first entry.

You can use the Recordset object methods (for example MoveFirst, MoveNext, MovePrevious, MoveLast) for navigating through the Recordset. The Recordset is not updatable and thus the AddNew, Update, and Delete will generate errors.

7.5.2.3 Rereading Entries
If you open a data queue for RECEIVE, you can reread any new data queue entries by using the Requery method of the Recordset object. All existing records in the Recordset object are deleted and replaced with the reread entries from the data queue. If there are no new entries on the data queue, the resulting Recordset will be empty.

7.5.2.4 Adding an Entry
If you open a data queue for SEND, you can add new entries to the data queue using the AddNew method of the Recordset object. The Update and Delete methods will produce errors.

7.5.3 Get All Parts - Data Queue
When we run the agent manually from the Notes agent list, the following code is executed:

```vba
' Declarations
Dim cn As Variant
Dim rs As Variant
Dim rd As Variant
Dim Rcds As Variant
Dim ValuesA(1) As Variant
Dim Fields(l) As Variant
Dim Flds As Variant
Dim Vals As Variant
Dim system As String
Dim msg As String
```

In the Initialize subroutine, we create a Connection object and use its Open method to establish a connection to the AS/400 system. Then we create two Recordset objects one each for the input data queue and the output data queue.

We open the input and output data queues for send and receive respectively using the Execute method of the Connection object. We have discussed the opening of data queues in Section 7.5.2 "Opening the Data Queue" on page 256.

After we have opened the data queues, we place a record on the input data queue. This record has value of "A" and "" for its first and second parameters respectively. These values indicate to the server program DQXRPG that we are requesting all the records.

The server program DQXRPG reads the record from the data queue and places all the records from the PARTS file on the output data queue. When we do a Requery on the output data queue, we get a result set back. This result set is
loaded into a Recordset object. We navigate through the Recordset object and display the values one record at a time.

```vbscript
Sub Initialize
  system = Inputbox$("Enter the AS400 system name.","System Name",,1500,1500)
  Set cn = CreateObject("ADODB.Connection")
  Set rs = CreateObject("ADODB.Recordset")
  Set rd = CreateObject("ADODB.Recordset")
  cn.open "Provider=IBM\DA400;Data Source=" + system + ";", "", ""
  Rcds = 0
  Set rs = cn.Execute("open dataqueue apilib.dqinpt(Option CHARACTER(1),part CHARACTER(5) )for Send",Rcds,1)
  Set rd = cn.Execute("open dataqueue apilib.dqoupt(Option CHARACTER(1),Part CHARACTER(5),Desc CHARACTER(25),Qty DECIMAL(5,0),Price DECIMAL(6,2), Date CHARACTER(10) )for Receive",Rcds,1)
  Fields(0) = "Option"
  Fields(1) = "part"
  Flds = Fields
  ValuesA(0) = "A"
  ValuesA(1) = ""
  Vals = ValuesA
  rs.AddNew Flds, Vals
  rd.ReQuery
  Do While Not rd.EOF
    MessageBox msg
    rd.MoveNext
  Loop
End Sub
```

7.5.4 Closing a Data Queue

Closing a data queue releases the lock that is on the data queue object on the AS/400 system. This can be accomplished by closing the Recordset and releasing all the resources by eliminating the object from the memory.

In the terminate event of the agent we close both the Recordset objects:

```vbscript
rs.close
rd.close
```
7.6 Remote Command Execution Example
The Client Access OLE DB provider allows you to interface with AS/400 CL
commands. There is a support to call AS/400 commands and pass command
parameters.

7.6.1 Executing a Remote cCommand
You use the Execute method of the Connection object to execute the CL commands.
You need to create a Connection object and open a connection to the AS/400
system as illustrated in earlier sections.
Here we show a Lotus Notes agent which executes a CL command on the AS/400
system.

Dim cn As Variant
Sub Initialize
Dim system As String
Dim cm As String
On Error Goto errorProcess
system = Inputbox$("Enter the AS400 system name.","System
Name",,1500,1500)
Set cn = CreateObject("ADODB.Connection")
cn.open "Provider=IBMDA400;Data Source=" + system + ";", "", ""
cm = Inputbox$("Enter the AS400 CL Command you want to execute","CL
Command",,1500,1500)
cm = "{{" + cm + "}}"
cn.Execute cm ,"",1
Exit Sub
errorProcess:
Messagebox "Error " & Err() & ": " & Error()
Exit Sub
End Sub

The parameters passed to the Execute method are CommandText, RecordsAffected
and adCmdText.
CommandText is the CL command surrounded by two pairs of curly braces. If you
want to accept user input as part of the Command, you will have to concatenate
the user input to the actual command.

The parameter adCmdText is the name of the constant which determines the type
of CommandText. With Lotus Notes use the constant value (1) instead of the
constant name (adCmdText). For a list of constant values, see the Execute Method
topic in the SDK Technical Reference and the ADO online help.

7.6.2 Restrictions
You can’t call interactive CL commands. Only CL commands that can be called in
batch can be executed. The Client Access OLE DB provider does not support
preparing commands as can be done for SQL statements. Therefore, you should
concatenate the command text and the command values together to form the
complete command.
Lotus Notes with OLE DB

259


7.7 A Distributed Program Call Example

The Client Access OLE DB provider, lets you call AS/400 programs. You can pass both input and output parameters. For more information on creating programs on the AS/400 system, see the appropriate AS/400 language reference manual for your particular programming language.

The following is an illustration of how to use IBMDA400 to call an AS/400 program. In this sample a Lotus Notes agent calls an AS/400 program named DPCXRPG with a parameter that is the value of a part number. This program queries the database file and returns back details of the part. See “DPCXRPG/RPGLE” on page 377 for a complete listing of the program.

The details of the RPG program DPCXRPG are:

Table 47. Details of DPCXRPG

<table>
<thead>
<tr>
<th>Seq</th>
<th>Description</th>
<th>Length Type</th>
<th>I/O</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Action Flag</td>
<td>1 Character</td>
<td>I/O</td>
<td>S - Retrieve single record</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A - Position to the start of file</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F - Fetch the next record</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OUTPUT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y - Operation succeeded</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X - Operation failed or EOF found</td>
</tr>
<tr>
<td>2</td>
<td>Part Number</td>
<td>5.0 Packed</td>
<td>I/O</td>
<td>Part Number to be retrieved or retrieved</td>
</tr>
<tr>
<td>3</td>
<td>Description</td>
<td>25 Character</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Quantity</td>
<td>5.0 Packed</td>
<td>O</td>
<td></td>
</tr>
</tbody>
</table>
We create a **Connection** object and open it. Also we create a **Command** object. We prepare the **Command** object for the program call. Please note that the **CommandText** property of the **Command** object, in case of a program call, is surrounded by two pairs of curly braces. This is the way IBMDA400 differentiates between a program call and a Stored Procedure call. You have to specify a list of comma-separated question marks as placeholders for each of the program parameters.

```
' Opening the connection
cn.open "Provider=IBMDA400;Data Source=" + system + ";", ",", ","

'Preparing the Command
Set cm.ActiveConnection = cn
cm.CommandText = "{{call apilib/dpcxrpg(?,?,?,?,?,?)}}"
cm.Prepared = True
```

Note that when a call to an AS/400 program is prepared, there is no existing information on the AS/400 system that describes the program parameters. To create program parameter information, you should catalog and call the program as a stored procedure. Since AS/400 program information does not exist, you must create and specify all of the program parameters in your application as shown here:

```
'Creating the program Parameters
cm.Parameters.Append(cm.CreateParameter("opt",129,3, 1))
cm.Parameters.Append(cm.CreateParameter("partno",14,3, 0))
cm.Parameters(1).Precision = 5
cm.Parameters(1).NumericScale=0
cm.Parameters.Append(cm.CreateParameter("partds",129,2, 25))
cm.Parameters.Append(Cm.CreateParameter("qty",14,2, 0))
cm.Parameters(3).Precision= 5
cm.Parameters(3).NumericScale= 0
cm.Parameters.Append(cm.CreateParameter("price",14,2, 0))
cm.Parameters(4).Precision = 6
cm.Parameters(4).NumericScale = 2
cm.Parameters.Append(cm.CreateParameter("pdate",129,2, 10))
```

With Lotus Notes, use the constant values in place of the constant names while creating the parameters. For a list of constant values, see the Direction Property and the Type Property topics in the *SDK Technical Reference* and the ADO online help.

<table>
<thead>
<tr>
<th>Seq</th>
<th>Description</th>
<th>Length Type</th>
<th>I/O</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Price</td>
<td>6.2 Packed</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Ship Date</td>
<td>10 Date</td>
<td>O</td>
<td></td>
</tr>
</tbody>
</table>
Next you have to set the **Precision** and **NumericScale**, for those data types (adDecimal and adNumeric) that require them. This only is necessary when calling an AS/400 program.

### 7.7.1 Get a Part Using a Program Call

The next step is to **Execute** the Command. The code below illustrates how to execute the prepared **Command** object.

```vbnet
partno = Inputbox$("Enter the part number you want.","Part Number",,1500,1500)
Parms(0)  = "S"
Parms(1) = partno
Parms(2) = " "
Parms(3)  = 0
Parms(4) = 0
Parms(5) = " "

parms = Parms1
cm.Execute Rcds, parms,1
```

Here we pass a value of ‘S’ in the first parameter and a part number in the second parameter. This means that we are requesting a single record matching the value of the part number entered. The parameters passed to the **Execute** method of **Command** object are **RecordsAffected** (not supported by IBMDA400 we pass a value of 0), **Parameters** (a variant array containing values which will replace the parameter markers) and the type of **CommandText**. You should pass only constant values in place of the third parameter. For a list of constant values and their meanings please see the **Execute** method in the **SDK Technical Reference** and the ADO online help.

The next step is to read the output parameters returned by the program. It is accomplished like:

```vbnet
If Strcompare (cm.Parameters(0).Value , "Y") = 0  Then
  'Read the parameters
Else
  Messagebox " Part not found"
End If
```

### 7.7.2 Restrictions

Calling programs on the AS/400 system that have user defined parameters or parameters that are structures are not supported.

### 7.8 A Sample Notes Application

In this section, we discuss a Notes sample application that uses the SQL interface of IBMDA400. This application generates a letter for a customer that tells the customer about a discount sale and the items which are on sale. It uses
two tables CSTMR and ITEM on the AS/400 system. See Chapter 3.1.6, “Database Table Structure” on page 96, for the format of these tables.

In this application, the application developer creates a document in a Notes database that contains information about the item on sale. See Figure 130 for the layout of the form.

![Item Discount Form](image)

**Figure 130. Item Discount Form**

This document contains details of the Item, Rate of discount and the last day of the sale.

We want to send a letter to some or all the customers telling them about the sale and the Items on sale. The template of the letter is shown in Figure 131 on page 264.
When the application is run, the user is prompted for the Customer ID to which the letter is to be sent. The form is initialized with information about the customer. The customer information is retrieved from the CSTMR table on the AS/400 system.

When the user fills in the item ID (from a list of keywords which is populated from the document that was created in the first step) and exits out of the item ID field, the information about the item is obtained from the AS/400 database. The values for discount and last date are retrieved from the Notes document that was created in the first step.

Next, we discuss the LotusScript used to implement this application. Please see Appendix A.1, “Downloading the Samples” on page 373 for instructions to how to download this sample application.
We first declare a connection object and two recordset objects.

 DECLARE the global variables
 Dim cn As Variant
 Dim rs As Variant
 Dim rd As Variant

 We open the Connection to AS/400 system
 Sub Initialize
   Dim system As String
   system = Inputbox$("Enter the AS400 system name.", "System Name", 1500, 1500)
   On Error Goto processError
   Set cn = CreateObject("ADODB.Connection")
   Set rs = CreateObject("ADODB.Recordset")
   Set rd = CreateObject("ADODB.Recordset")

   cn.open "Provider=IBM400;Data Source=\" + system + ";", "", ""
 Exit Sub

 processError:
 Messagebox "Error " & Err() & ": " & Error()
 Exit Sub
 End Sub

 We prompt the user for the name of the AS/400 system. We then create a connection object and invoke its Open method. We also create the recordset objects.

 Sub Initialize
 Dim rcds As Variant
 Dim parms As Variant
 Dim query1 As String
 rcds = 0
 On Error Goto processError
 parms = Inputbox$("Enter the Customer ID.", "Customer ID", 1500, 1500)
 query1 = "select * from apilib.cstmr where cid = " & "'" & parms & "'

 Set rs = cn.execute (query1, rcds, 1)
 Exit Sub

 processError:
 Messagebox "Error " & Err() & ": " & Error()
 Exit Sub

 End Sub
In the form `Initialize`, we prompt the user to enter a customer ID based on which we execute an SQL statement to select a record from the CSTMR table in the APILIB library.

Please note that we have used the `Execute` method of the `Connection` object to execute the SQL statements. We can also use prepared SQL statements using the `Execute` method of the `Command` object. See Section 7.2 "An SQL Example" on page 240.

```vba
Sub Postopen(Source As NotesUIDocument)
    Dim workspace As New NotesUIWorkspace
    Dim uidoc As NotesUIDocument
    Set uidoc = workspace.CurrentDocument

    On Error Goto processError

    Call uidoc.FieldSetText("Name", rs.Fields(3).value)
    Call uidoc.FieldSetText("Custaddr", rs.Fields(6).value)
    Exit Sub
    processError:
    Messagebox "Error " & Err() & ": " & Error()
    Exit Sub
End Sub
```

In the `Postopen` event of the form, we populate the fields related to the customer information (Name and Address) from the result set retrieved in the previous step.
When the user selects one item ID from a list of keywords and exits out of this field, the code in the Exiting event of the item ID field is executed. The list of keywords is formed from the information in the item discount form. See Figure 130 on page 263.

In this code, we get details about the item from the ITEM table in the APILIB library and about the item discount and last date from the Notes document that we created in the first step.

This simple application is an illustration of how we can integrate the corporate data with Lotus Notes.
Figure 132. The Item Discount Form

Figure 133. The Generated Letter
Finally, we show the generated letter. We have used the AS/400 OLE DB support to execute SQL statements on the AS/400 system. We have incorporated the information retrieved from the AS/400 system into the letter.
Chapter 8. Microsoft Visual FoxPro Using OLE/DB

Visual FoxPro is a Windows graphical application development environment used to build client/server applications. It supports accesses to its native Foxpro database or to external ODBC databases. The Visual FoxPro interface lets you access the DB2/400 database using Open Database Connectivity (ODBC), Microsoft’s standard for database connectivity. This can be done by installing the Client Access ODBC driver and defining an ODBC data source.

This chapter focuses on how to develop Visual FoxPro applications using OLE DB with the AS/400 system. OLE DB is a set of application programming interfaces (APIs) which provide access to a wide range of data sources. ODBC provides access only to information in a relational database through SQL, while OLE DB is a method to access data via a standard COM (Component Object Model) Interface. OLE DB provides both the SQL functionality defined in ODBC and interfaces suitable to access non-SQL resources.

IBMDA400 is an OLE DB provider to AS/400 data and services. It is a part of Client Access Version V3R1M3. It includes support to interface with the following AS/400 resources:
- Table record-level access
- Data queues
- Programs
- CL commands
- SQL statements
- SQL stored procedures

IBMDA400, the Client Access OLE DB provider, has been written to the OLE DB Version 1.1 specification and has been tested to work with ADO 1.0. For more details on the ADO programming model, please see Chapter 1, “OLE DB and ActiveX Overview” on page 1.

In this chapter, we will show you through a simple example the way to use IBMDA400 with Visual FoxPro Version 5.0.

Tip

For additional information about programming with ADO, see the SDK Technical Reference Manual and the Microsoft online help. See Appendix A.2, “Other Documentation and Resources” on page 373 for details on how to obtain this documentation.

8.1 Introduction to Application Examples

The rest of this chapter covers the application example that demonstrates AS/400 Database access via SQL.

The database access example application discussed in the remainder of this chapter allows retrieval of a single record or all the records from a PARTS file on
the AS/400 system. This example has been tested to work with Visual FoxPro 5.0 and ADO 1.0.

We use the PARTS database file in the APILIB library in the example. The PARTS file is defined as:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Description</th>
<th>Length</th>
<th>Decimals</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARTNO</td>
<td>Part Number</td>
<td>5</td>
<td>0</td>
<td>Zoned</td>
</tr>
<tr>
<td>PARTDS</td>
<td>Part Description</td>
<td>25</td>
<td></td>
<td>Char</td>
</tr>
<tr>
<td>PARTQY</td>
<td>Quantity</td>
<td>5</td>
<td>0</td>
<td>Packed</td>
</tr>
<tr>
<td>PARTPR</td>
<td>Price</td>
<td>6</td>
<td>2</td>
<td>Packed</td>
</tr>
<tr>
<td>PARTDT</td>
<td>Part Shipment Date</td>
<td>10</td>
<td></td>
<td>Date</td>
</tr>
</tbody>
</table>

The example discussed in this chapter is available for download on the www.redbooks.ibm.com web-site. See Appendix A.1, “Downloading the Samples” on page 373 for details.

8.2 An SQL Application Example

IBMDA400, the Client Access OLE DB provider, allows you to send remote SQL statements to the AS/400 server. There is support for executing most SQL statements on the AS/400 system. For more information on SQL statements see the DB2 for AS/400 SQL Programming manual, SC41-5611.

You can execute SQL statements using the `Execute` method of the `Connection` Object. If you are executing SQL statements with parameters, for example in a `where` clause, you should use parameter markers and prepare the SQL statement before executing it. This improves performance because the SQL statement is not prepared every time you execute it.

The SQL SELECT statement and SQL CALL statement, for calling a stored procedure, are the only two SQL statements that generate a result set, and thus a `Recordset` object. The `Recordset` Object created is not updatable. Therefore, the `AddNew`, `Update`, and `delete` methods are not valid.
This example is an illustration of how to use IBMDA400, the Client Access OLE DB provider, to access DB2/400 data using SQL. In this example, the client program requests data from the AS/400 database by sending SQL statements to the AS/400 system. The host server executes the SQL statement and returns the results to the client program in a SQL result set. Two SQL statements are used:

Select * from apilib.parts
Select * from apilib.parts where partno = ?

to retrieve all columns for all records and to retrieve all columns for a given part number, respectively.

Figure 135 on page 274 shows the result of retrieving one part(12307) from the PARTS file.
Figure 135. Get Part Using SQL

Figure 136 on page 275 shows the result of retrieving all the parts from the PARTS file.
In the following sections we discuss the step-by-step implementation of the application.

### 8.2.1 Opening a Connection

The first step is to define a `connection` object. You need to define at least one `connection` object in an application.

```vfp
cn=createobject('adodb.connection')
rs=createobject('adodb.recordset')
cn.open ("Provider=IBMDA400;Data Source=TCPASM05;", ",", ")
```

First we create a `connection` object and a `recordset` object. After creating the `connection` object, we invoke its `open` method. The `ConnectionString` part contains the provider and data source names. The Data Source name must be the name of a valid AS/400 connection. For example, the name of a connection configured using the Client Access AS/400 connection program.
Empty strings are passed for the User ID and Password parts. The IBMDA400 provider does not support the passing of User ID and Password when you open a connection. Client Access prompts the user for connection information as it is needed. The passing in of User ID and Password will be supported in a future release of IBMDA400.

8.2.2 Get One Part-SQL

To get one part we have to query the PARTS file based on the part number that was entered.

```vbnet
cm.ActiveConnection = "Provider=IBMDA400;Data Source=TCPASM05;"
cm.CommandText = "select * from APILIB.PARTS WHERE PARTNO = ?"
cm.Prepared = .T.
```

We set the `ActiveConnection` property of the `Command` object to the Connection that was opened in the first step. The next step is to set the `CommandText` property of `Command` Object to the SQL statement to be executed. We use parameter markers(?) for the variables which will be used. Using parameter markers improves performance as the prepared SQL statement is used at the time of execution.

```vbnet
parms = inputnum
rs = cm.Execute(rcds, parms, 1)
```

The next step is to execute the prepared SQL statement. When you enter the value of part number and press the button "Get Part", the SQL statement that has been prepared gets executed. The parameter marker is replaced by the value of part number you entered.

The parameters passed to the `Execute` method of Command Object are `RecordsAffected`, parameter value and `adCmdText`. `RecordsAffected` is not supported by IBMDA400 but you have to pass it in. The second parameter is the value that you want to be substituted for the parameter marker used in the SQL statement that you prepared in the first step. The third parameter indicates the type of `CommandText`. A value of 1 indicates that it is a textual definition of a command. For a list of constant values, see the `Execute` Method topic in the SDK Reference and the ADO online help. See Appendix A.2, "Other Documentation and Resources" on page 373, for information on how to obtain this documentation.

```vbnet
ThisForm.Text1.Value = rs.Fields(0).Value
ThisForm.Text2.Value = rs.Fields(1).Value
ThisForm.Text3.Value = rs.Fields(2).Value
ThisForm.Text4.Value = rs.Fields(3).Value
ThisForm.Text5.Value = rs.Fields(4).Value
ThisForm.Text7.Value = Time()
ThisForm.Text8.Value = seconds() - ssecs
```

The SQL statement, when executed, returns an SQL result set. This result set is loaded into the `ADO Recordset` object. We then we populate the fields in the form.
8.2.3 Get All Parts - SQL

To get all the parts, you have to select all the records from the database file.

### Executing SQL

```vbnet
rs = cn.Execute("select * from APILIB.PARTS", Rcds, 1)
```

You can execute the SQL statement to select all the records by using the `Execute` method of the `Connection` object. The parameters passed are `CommandText`, `RecordsAffected` and `adCmdText`. For details, see the `Execute` method of the `Connection` object in the *SDK Reference Manual* and the ADO online help.

You can prepare this SQL statement as illustrated in section 8.2.2 on page 276. If you choose to prepare this SQL statement, you won’t have parameter markers inside this statement and you won’t have to pass in the parameters at the time of execution.

This SQL statement returns a result set that is loaded into the ADO Recordset object.

#### 8.2.3.1 Navigating the Result Set

If the execution of the SQL statement produces a result set, the result set data is loaded into the ADO Recordset object. If the result set is empty, the Recordset is also empty and BOF and EOF are set to true. You can use the `MoveFirst`, `MoveNext` and `MovePrevious` methods to navigate through the result set.

The field names in the Recordset can be read like this:

```vbnet
For indx = 1 To <recordset>.Fields.Count - 1
    <value> = <recordset>.Fields(Indx).Name
Next
```

The field values in the current record can be read like this:

```vbnet
If Not <recordset>.BOF And Not <recordset>.EOF Then
    If IsNull(<recordset>.Fields(0).Value) Then
        <value> = ""
    Else
        <value> = <recordset>.Fields(0).Value
    End If
End If
```

---

**Note**

- `MoveLast` method is not supported for an SQL Recordset.
- An SQL Recordset is not updatable. As a result `AddNew`, `Update` and `Delete` methods are not valid.
- An SQL Recordset does not support bookmarks.
8.2.3.2 Populate the Grid Via Bound Control to Table

```
OPEN DATABASE light1
USE PARTS
DELETE ALL
SET DELETED ON
DO WHILE NOT rs.EOF
  Append Blank
  Replace partno with VAL(rs.Fields(0).Value)
  Replace partds with rs.Fields(1).Value
  Replace partqy with VAL(rs.Fields(2).Value)
  Replace partcr with VAL(rs.Fields(3).Value)
  Replace partdt WITH CTOD(rs.Fields(4).Value)
```

We have already created a FoxPro Table named light1 with the corresponding fields to match the AS/400 Table. The Foxpro Table is opened and purged of previous data. Then a blank record is appended and its fields are set to those of the current record in the recordset using the Replace command. We continue this process until all records in the recordset are handled. Then the form allparts is loaded, which contains the grid that is bound to the parts tables (a bound control automatically displays the associated data without the need for code).

8.2.4 Error Handling

The Connection object contains the errors collection. All PC client and AS/400 error and warning messages are returned in the error collection at the connection level. You can check for any exception by navigating through the error collection.

For more information on Errors, see the Error Object and Errors Collection topics in the SDK Reference Manual and the ADO online help.

Use the cwbzztrc.exe program to work with error logging. You can run the program from a DOS command prompt. Run the program without any parameters to get the options and syntax. This program is installed when you install Client Access. The following is the default installation path:

C:\Program Files\IBM\Client Access\cwbzztrc.exe
Chapter 9. End-User Tools

In this chapter we discuss how you can take advantage of Client Access OLE DB Provider services from popular end-user application tools like Lotus 1-2-3, MS Excel and MS Access. Samples used in this section can be downloaded from the web. For additional information on downloading samples please refer to Appendix A.1, “Downloading the Samples” on page 373.

Tip

For additional information about programming with ADO, see the SDK Technical Reference Manual and the Microsoft online help. See Appendix A.2, “Other Documentation and Resources” on page 373 for details on how to obtain this documentation.

9.1 Lotus 1-2-3

Lotus 1-2-3 97 Edition allows you to create and manipulate ADO objects. However, when using LotusScript you must create scripts to do so. In this section we discuss some sample Lotus 1-2-3 scripts that use the Client Access OLE DB provider. Creating a script using LotusScript is very simple. You may develop similar scripts using the techniques that we discuss here and take advantage of the Client Access OLE DB provider from Lotus 1-2-3.

Our sample must be run in Lotus 1-2-3 97 Edition or later. When you open the workbook it initially uses SQL to load and refresh the spreadsheet. We access the PARTS database file from the scripts. The scripts allow you to delete, update or insert the selected record into the AS/400 database file using record level access. Once the spreadsheet is loaded, you can select a record for update or delete or enter a new record to insert. The scripts can be invoked from the Actions menu.

9.1.0.1 Running the Scripts

We have created an Actions menu in the workbook. The menu has four items: Insert, Delete, Update and Get All Parts. Below is a list of the menu items and their functionality:

- Get All Parts - Reads the PARTS database file and populates the spreadsheet with records that are read. Get All Parts is called any time you load this spreadsheet.
- Insert - Adds the highlighted row into the PARTS database file on the AS/400 system. It then performs a read to display the updated information from the PARTS database file.
- Delete - Removes the highlighted row from the PARTS database file. It then performs a read to display the updated information from the PARTS database file.
- Update - Modifies the highlighted row on the PARTS database file. It then performs a read to display the updated information from the PARTS database file.
Each of these three options - Insert, Delete and Update - makes changes to the AS/400 file and then refreshes the spreadsheet.

Let's take a look at the LotusScript Script Editor.

9.1.0.2 Using LotusScript Script Editor
To view the script code select Edit->Scripts&Macros-> Show Script Editor. The Script Editor window will appear. In this Script Editor window select (Globals) in the Object drop ListBox.

Figure 137. Launching LotusScript to Create or View a Script

Once you have the Script Editor open you can view any of the scripts by selecting it from the Script; drop ListBox. The available Scripts are: (Declarations), Initialize, Terminate, GetAll, Delete, Insert, and Update.
Select Initialize from the Script Editor’s Scripts drop ListBox. The Initialize subroutine is displayed. Lotus 1-2-3 executes this subroutine everytime you load this spreadsheet.

In this subroutine, we first prompt the user to enter the AS/400 system name. You should have a valid connection defined to this system. This system should also have the PARTS database file installed in the APILIB library.

Next we call CreateObject three times. The CreateObject call creates an ADO object. The first CreateObject creates an ADODB Connection object. We have declared cn as a Variant data type. Once this call completes a new ADODB Connection object cn is created. We use this object with all the other ADO Objects discussed in this section.
Using CreateObject we then create ADODB Recordsets rs and rd. We need two Recordset objects, because one is used for SQL statements and the other for record-level access.

Next we invoke the Open method of the Connection object cn. In the ConnectionString we specify the Provider name using the Provider= keyword. The Client Access OLE DB provider name is IBMDA400. We use the Data Source= keyword to specify the AS/400 system name. This name is supplied by the user. Note that in this release, the Client Access OLE DB provider does not allow user-id and password in the ConnectionString. IBM has plans to enhance this in a future release.

Next we invoke the Recordset object’s Open method. The Recordset Open has the following syntax:

`recordset.Open Source, ActiveConnection, CursorType, LockType, Options`

Source is the AS/400 database file name that we want to open using record level access. ActiveConnection is the Connection object. We use the cn Connection object that we created earlier. For the CursorType we use 2. This indicates that an adOpenDynamic type cursor is used for navigating the record set. For LockType we use 3. This indicates that an adLockOptimistic type lock is used. For option we use 2 this indicates that the command is adCmdTable, to evaluate the Source as a table name.

Using the next two statements we clear out cells from the cell A2 to D500 range in the spreadsheet.

One reason to perform some operations from the Initialize subroutine is this way we can prevent opening and closing the Connection and other objects multiple times. Opening and closing connections multiple times is expensive and can cause errors.

The last statement of this script makes a call to GetAll. We discuss GetAll in the following section.

9.1.0.4 Get All Records
When you select Get All Parts from the Actions menu, the GetAll subroutine is invoked. This returns all records from the PARTS database file to the spreadsheet.
In this subroutine we first clear all the cell data that are in the [A2..E500] Range. Since the PARTS database file has five fields and we don't expect more than a few hundred records in the file, this range is sufficient.

Next we invoke the Execute method of the Connection object cn. The Connection.Execute method has following syntax:

```
connection.Execute CommandText, RecordsAffected, Options
```

In the CommandText we place the SQL statement that is executed on the AS/400 system. The RecordsAffected is the number of rows that are affected by the command and the Options specifies the command type.

The rest of the subroutine is devoted to navigating the Recordset, reading and placing each item from the Recordset into the spreadsheet cells. Note that navigating the Lotus 1-2-3 spreadsheet or populating the spreadsheet is not ADO specific. To accomplish this:

- We read each field value from the current row of the Recordset object and place them into the spreadsheet cells.

- We continue reading and populating the spreadsheet for all rows until 'end of file' has been reached in the Recordset.

The Range.Cell method returns a single-cell range for a specified row, column, and sheet. Our range is A2..D500. Since we begin at row 0 column 0, our start cell is A2. Once we have identified the cell, we can set the Contents property of this cell with the rs.Fields(col).Value. This places the value that is in the current column of the current row of the Recordset into the current cell of the spreadsheet. We increment the column count until all columns from the Recordsets current row is copied into the spreadsheet. Then we increment the
row count and using the MoveNext method move to the next row of the Recordset. This process continues until we reach the End Of File (EOF) in the Recordset.

Once all data is copied to the spreadsheet we delete the Recordset object by setting it to Nothing.

9.1.0.5 Insert a row

When you take the Insert option from the Actions menu, two subroutines are called - Insert and GetAll. First we call the Insert subroutine. This inserts the highlighted row from the Lotus 1-2-3 spreadsheet into the AS/400 database file. Then we call the GetAll subroutine. Refer to section 9.1.0.4, “Get All Records” on page 282 for an explanation of the GetAll subroutine. In this section we discuss the Insert subroutine.

Sub Insert
Dim temp(4) As Variant
Dim temp2(4) As Variant
Dim row As Integer
Dim col As Integer
Dim Flds As Variant
Dim Vals As Variant
temp(0) = "PARTNO"
temp(1) = "PARTDS"
temp(2) = "PARTQTY"
temp(3) = "PARTPR"
temp(4) = "PARTDT"
CurrentApplication.Calc
row = [A:F1].CellValue - 2
col = 0
For col = 0 To 4
   Set RowCell = RowRange.Cell(row,col,0)
   temp2(col) = RowCell.CellValue
Next col
Flds = temp
Vals = temp2
rd.AddNew Flds, Vals
GetAll
End Sub

For the Insert subroutine to work correctly we need to know which row the user has highlighted on the spreadsheet. This is implemented using the @CELLPOINTER function. The @CELLPOINTER(attribute) returns information about the current cell. The attribute of interest is the address of a cell. We have @CELLPOINTER("row") defined on the [A:F1] cell in the spreadsheet. From the Insert subroutine, when we call [A:F1].CellValue and offset it by two, it gives us the row number that the user has highlighted.

Once we have identified the row, we can to read each CellValue for that row. This tells us each field value that will be entered into the AS/400 file. We place each of the CellValue in a Variant array named temp2.

Next we invoke the AddNew method to add the new row into the Recordset object rd. The AddNew method takes two parameters. The first parameter is a Variant array of all the fields, and the second parameter is a Variant array of all the values.
Since record-level access uses a dynamic cursor on the Recordset, any changes that are made to the Recordset are automatically reflected on the AS/400 database file. The new row is inserted into the AS/400 file.

Finally we call GetAll to refresh the spreadsheet. Note that calling GetAll at this point is not required.

Note that the Insert method uses ADO objects that are declared Globally and then created in the Initialize subroutine. For additional information about this refer to section 9.1.0.3, “GlobalDeclarations and Initialization” on page 281.

9.1.0.6 Delete a Row
When you select Actions->Delete, two subroutines are called - delete1 and GetAll. First we call the delete1 subroutine. This removes the row that is highlighted on the Lotus 1-2-3 spreadsheet from the AS/400 database file. Then we call the GetAll subroutine. It refreshes the spreadsheet to reflect the changes that you made in the AS/400 database file. We discuss the GetAll subroutine in Section 9.1.0.4, “Get All Records” on page 282. In this section we discuss the delete1 subroutine.

```vbp
Sub delete1
    CurrentApplication.Calc
    row = [A:F1].CellValue - 2
    rd.MoveFirst
    rd.Move row
    rd.Delete
    GetAll
End Sub
```

For the delete1 subroutine to work correctly we need to know which row the user has highlighted in the spreadsheet. This is implemented using the CELLPOINTER function. The CELLPOINTER(attribute) returns information about the current cell. The attribute we are interested in is the address of the cell. We have defined CELLPOINTER("row") on [A:F1]. From the Insert subroutine, we call [A:F1].CellValue and offset it by two. It gives us the current row number. This is the row the user has highlighted.

Once we have identified the row number, we can go back and remove the same row from the Recordset. To accomplish this we move to the beginning of the Recordset object using the MoveFirst method and then using the Move method we move to the record that the user has highlighted. Then using the Delete method we delete the current row from the Recordset object.

Note that the Delete1 method uses ADO objects that are declared globally and then instantiated from the Initialize subroutine. For additional information on this refer to Section 9.1.0.3, “GlobalDeclarations and Initialization” on page 281.

9.1.0.7 Update a Row
When you select Actions->Update two subroutines are called - Update and GetAll. First we call the Update subroutine. This reads the highlighted row in the Lotus 1-2-3 spreadsheet and then modifies the corresponding record in the AS/400 database file. Then we call the GetAll subroutine. For information about
the GetAll subroutine refer to Section 9.1.0.4, “Get All Records” on page 282. In this section we discuss the Update subroutine.

```vbscript
Sub Update
    Dim temp(4) As Variant
    Dim temp2(4) As Variant
    Dim row As Integer
    Dim col As Integer
    Dim Flds As Variant
    Dim Vals As Variant
    temp(0) = "PARTNO"
    temp(1) = "PARTDS"
    temp(2) = "PARTQTY"
    temp(3) = "PARTPR"
    temp(4) = "PARTDT"
    CurrentApplication.Calc
    row = [A:F1].CellValue - 2
    For col = 0 To 4
        Set RowCell = RowRange.Cell(row, col, 0)
        temp2(col) = RowCellCellValue
    Next col
    Flds = temp
    Vals = temp2
    rd.MoveFirst
    rd.Move row-1
    rd.Update Flds, Vals
    GetAll
End Sub
```

For the Update subroutine to work correctly we need to know which row the user has highlighted on the spreadsheet. This is implemented using the @CELLPOINTER function. The @CELLPOINTER(attribute) returns information about the current cell. The attribute we are interested in is the address of the cell. We have defined @CELLPOINTER("row") on [A:F1]. From the Update subroutine, we call [A:F1].CellValue and offset it by two, to get the current row number. This is the row the user has highlighted for modification.

Once we have identified the row it is very easy to read each CellValue on that row. Some or all of the field values may be modified. We place each of the CellValue in a Variant array named temp2.

Then we call the Update method to modify the Recordset. Update takes two parameters - the field names and field values. We pass these parameters as Variant arrays.

Since the rd Recordset object uses a dynamic cursor, any changes are immediately reflected on the AS/400 database file.

Note that the Update subroutine uses ADO objects that are declared Globally and then instantiated from Initialize subroutine. For additional information about this, refer to Section 9.1.0.3, “GlobalDeclarations and Initialization” on page 281.
9.2 Microsoft Excel

In Excel you need to create macros to take advantage of the Client Access OLE DB Provider. A macro is a series of commands and functions that are stored in a Visual Basic module. The Visual Basic Editor in Excel 97 can be used to develop these macros. If you need additional information about Excel 97, the Visual Basic Editor or Visual Basic Script, consult the Microsoft documentation.

The sample included with this redbook and discussed later in this section uses SQL statements to load and refresh the spreadsheet with all of the records from the ITEM database file. Please see Chapter 3.1.6, “Database Table Structure” on page 96 for the structure of the ITEM database.

It uses record-level access to insert, update, or delete the selected record. Once the spreadsheet has been loaded, select the record or enter data for a new record and use the Actions menu items to make the desired changes in the ITEM database file.

You can view or modify the script code by selecting from the menu **Tools->Macro->Macros** and by selecting the appropriate macro. There are four different macros included in this worksheet: read, add, remove and modify. Here is a list of these macros and their functionality:

- **Read** - Reads the ITEM database file and populates the spreadsheet with these records.
- **Add** - Inserts the highlighted row into the ITEM database file. It then performs a read to display the updated information from the ITEM database file.
- **Remove** - Deletes the highlighted row from the ITEM database file. It then performs a read to display the updated information from the ITEM database file.
- **Modify** - Updates the highlighted record on the ITEM database file. It then performs a read to display the updated information from the ITEM database file.

Each of these macros -add, remove and modify- makes changes to the AS/400 file and then refreshes the spreadsheet.
Using the Visual Basic Editor you can examine the macros that are behind this Excel spreadsheet. Select Tools->Macro->Visual Basic Editor, to get a similar view as shown on Figure 140 on page 289. Each macro - add, modify, read, remove - are implemented using one or more Methods from the ADOClass class module. For instance, when you execute the read macro it invokes a call to the GetAll method which is implemented in the ADOClass. Each of these methods then creates one or more ADO objects, sets the ADO object properties and calls the ADO methods as necessary. For a discussion of the ADO object model refer to Chapter 1, “OLE DB and ActiveX Overview” on page 1.

Most of the ADO objects are declared Globally in the ADOClass class module. When you invoke a macro, the ADOClass class module is initialized.
We have added an Actions menu to this worksheet. The actions menu has four items - Get All Item, Insert, Delete and Update. To run a macro, simply select the menu item. In the following sections we discuss each of the macros that are used in this Excel worksheet.

9.2.0.1 Global Variables and Class Initialization

```vbnet
Option Explicit
Public cnTCPASM05 As New ADODB.Connection
Public ixTCPASM05 As New AD400.Connection 'for ddm index
Public ix_APILIB_ITEM As AD400.Index     'for ddm index
Public cm_APILIB_ITEM As New ADODB.Command 'for SQL command
Public dm_APILIB_ITEM As New ADODB.Command 'for DDM command
Public rd_APILIB_ITEM As ADODB.Recordset 'for DDM access
Public rs_APILIB_ITEM As ADODB.Recordset 'for SQL access
Public Parm1  As New ADODB.Parameter
Private Sub Class_Initialize()
    cnTCPASM05.Open "provider=IBMDA400;data source=tcpasm05;"
    ixTCPASM05.Open "TCPASM05", "", ""
    Set cm_APILIB_ITEM.ActiveConnection = cnTCPASM05
    cm_APILIB_ITEM.CommandText = "SELECT * FROM APILIB.ITEM ORDER BY IID"
End Sub
```

The cnTCPASM05 object is an ADODB Connection object. We use this Connection object from all the other AD0 objects. The ixTCPASM05 object is an IBM supplied AD400 Connection object. This object is used by the update and delete macro. It allows us to do a key based search. The ix_APILIB_ITEM object is an AD400 Index object.
We have two Command objects, cm_APILIB_ITEM and dm_APILIB_ITEM. We also have two Recordset objects, rs_APILIB_ITEM and rd_APILIB_ITEM. The Command object cm_APILIB_ITEM and the Recordset object rs_APILIB_ITEM are used for SQL statements. The Command object dm_APILIB_ITEM and Recordset object rd_APILIB_ITEM are used for record level access.

The Class_Initialize subroutine is invoked anytime a new instance of the class is created. In Class_Initialize, we first invoke the Open method of the Connection object cnTCPASM05. For ConnectionString we specify the Provider name as well as the Data Source name using the Provider= and Data Source= keywords. Data Source name is the name of the AS/400 system as defined in the AS400 Connection program in Client Access. Note that in this release the Client Access OLE DB provider does not allow user ID and password in the ConnectionString. IBM has plans to enhance this in a future release.

Next we call the Open method of ixTCPASM05, which is an AD400 Connection object. We then set the ActiveConnection property of the cm_APILIB_ITEM Command Object and set the CommandText property to select all the records from the ITEM database file. One reason to perform some of these operations from the Class_Initialize subroutine is this way we can prevent opening and closing the Connection object multiple times. Opening and closing connections multiple times is expensive and can cause errors.

9.2.0.2 Get All Records
When you select Actions->Get All Items, the GetAll method is invoked from the ADOClass class module. This returns all the records from the ITEM database file to your spreadsheet. In this subroutine we first invoke the Execute method of the cm_APILIB_ITEM Command object. Note that earlier we set the CommandText and other required properties in the Class_Initialize subroutine. The Class_Initialize subroutine is discussed in section 9.2.0.1, “Global Variables and Class Initialization” on page 289. The Execute method returns all records from the ITEM database file and places them in the rs_APILIB_ITEM Recordset object.

The rest of subroutine is devoted to navigating the Recordset, reading and placing each item from the Recordset into the spreadsheet cells. Note that navigating the Excel spreadsheet or populating the spreadsheet is not ADO specific. Here is an overview of the code that is used for this purpose in the subroutine.

• First we clear any data that is in a specific range on the spreadsheet.
• Next we get the column headings and add them to the spreadsheet.
• We read each field value from the current row of the Recordset object and place it in the spreadsheet.
• We continue reading and populating the spreadsheet for all records until “end of file” has been reached in the Recordset.

For additional information on how to manipulate the Excel spreadsheet from a macro, please consult the appropriate Microsoft documentation.
9.2.0.3 Insert a Row

When you select Actions -> Insert, two functions are called: Insert and GetAll. First we call the Insert method. This inserts the highlighted row from the Excel spreadsheet into the AS/400 database file. Then we call the GetAll method. Both Insert and GetAll methods are implemented in the ADOClass class module. Refer to Section 9.2.0.2, “Get All Records” on page 290, for a discussion of the GetAll method. In this section we discuss the Insert method.

Note that the Insert method uses ADO objects that are declared Globally and then instantiated from Class_Initialize subroutine. For additional information on this refer to Section 9.2.0.1, “Global Variables and Class Initialization” on page 289.

Sub GetAll()
Dim rowCount As Integer
Dim colCount As Integer
On Error GoTo errors

Set rs_APILIB_ITEM = cm_APILIB_ITEM.Execute()

Worksheets("sheet1").Activate
Range("A1").Activate
Selection.CurrentRegion.Select
Selection.ClearContents
Range("A1").Select
RowCount = 1
For colCount = 0 To rs_APILIB_ITEM.Fields.Count - 1
   Worksheets("Sheet1").Cells(rowCount, colCount + 1).Value = rs_APILIB_ITEM.Fields(colCount).Name
Next colCount

While Not rs_APILIB_ITEM.EOF
   rowCount = rowCount + 1
   For colCount = 0 To rs_APILIB_ITEM.Fields.Count - 1
      Worksheets("Sheet1").Cells(rowCount, colCount + 1).Value = rs_APILIB_ITEM.Fields(colCount).Value
   Next colCount
   rs_APILIB_ITEM.MoveNext
Wend
Set rs_APILIB_ITEM = Nothing
Exit Sub

errors:
’add error handling code here
End Sub
We first set the `ActiveConnection` property of the `Command` object `cm_APILIB_ITEM` to bind the ADO `Connection` object `cnTCPASM05` to the `cm_APILIB_ITEM` Command object.

Using the `CommandText` property we specify the AS/400 database file name that we want to open. For an insert operation we use record-level access. We specify the `CommandText` using the appropriate syntax for record-level access.

Next we set the `CommandType` property to `adCmdTable` and `Properties` ("Updatability") property to 4. The `adCmdTable` option indicates that we want to use record level access. The "Updatability" property indicates that we want to perform an insert operation and the file should have a Shared Update Lock.

Next we call the `Execute` method to open the ITEM database file and return all records to a `Recordset` object. For record-level access, the returned `Recordset` object is a dynamic Recordset. This means that any changes such as add, delete or update that are made to the `Recordset` object are immediately reflected in the AS/400 database file.

The `AddNew` method inserts a new row into the `Recordset` object. `AddNew` takes the field name and values as parameters. Each set of values are passed as a Variant array. We read the cell values from the highlighted row in the spreadsheet and place them in a Variant array. We also create a Variant array of field names.

```vba
Sub Insert()
    Dim flds As Variant
    Dim vals As Variant
    Dim numRecs As Variant
    Dim Parm As Variant
    Dim rowCount As Integer
    Dim colCount As Integer
    On Error GoTo errors:
    Set cm_APILIB_ITEM.ActiveConnection = cnTCPASM05
    cm_APILIB_ITEM.CommandText = "APILIB.ITEM()"
    cm_APILIB_ITEM.CommandType = adCmdTable
    cm_APILIB_ITEM.Properties("Updatability") = 4 ' insert=4
    Set rd_APILIB_ITEM = cm_APILIB_ITEM.Execute()
    Worksheets("Sheet1").Activate
    rowCount = Selection.Row
    colCount = 0
    flds = Array("IID", "INAME", "IPRICE", "IDATA")
    vals = Array(Worksheets("Sheet1").Cells(rowCount, 1).Value, _
                  Worksheets("Sheet1").Cells(rowCount, 2).Value, _
                  Worksheets("Sheet1").Cells(rowCount, 3).Value, _
                  Worksheets("Sheet1").Cells(rowCount, 4).Value)
    rd_APILIB_ITEM.AddNew flds, vals
    Set rd_APILIB_ITEM = Nothing
    Exit Sub
errors:
    'error handling code here
    Exit Sub
```
When we invoke the AddNew method we get a new row added into the ITEM database file.

Finally we call the GetAll method. This refreshes the spreadsheet. Note that calling the GetAll method is not required.

### 9.2.0.4 Delete a Record

When you select Actions -> Delete, two functions are called: Delete and GetAll. Delete reads the highlighted row from the Excel spreadsheet and then removes it from the AS/400 database file. Then we call the GetAll method. Refer to Section 9.2.0.2, “Get All Records” on page 290, for GetAll. In this section, we discuss the Delete method.

First, we set the `ActiveConnection` property for the Command object `dm_APILIB_ITEM`. This binds the ADO Connection object `cnTCPASM05` to the `dm_APILIB_ITEM` Command object. We use the `dm_APILIB_ITEM` Command object for all record level access functions.

Using the `CommandText` property, we specify the AS/400 database file name that we want to open. We use the record level access method to AS/400 database file for

```vba
Sub Delete()
    Dim flds As Variant
    Dim vals As Variant
    Dim numRows As Variant
    Dim Parms As Variant
    Dim rcds As Variant
    Dim Keys As Variant
    Dim Bookmark As Variant
    Dim rowCount As Integer

    Set dm_APILIB_ITEM. ActiveConnection = cnTCPASM05
    dm_APILIB_ITEM. CommandText = "APILIB.ITEM()"
    dm_APILIB_ITEM. CommandType = adCmdTable
    dm_APILIB_ITEM. Parameters. Append dm_APILIB_ITEM. CreateParameter("P1", adChar, adParamInput, 1)
    dm_APILIB_ITEM. Properties("Updatability") = 2 'delete=2
    Set rd_APILIB_ITEM = dm_APILIB_ITEM. Execute(rcds, Parms, adCmdTable)
    Set ix_APILIB_ITEM = ixTCPASM05. OpenIndex("APILIB.ITEM()", "APILIB.ITEM()")

    'get the column headings and data
    Worksheets("Sheet1"). Activate
    rowCount = Selection. Row
    Keys = Array(Worksheets("Sheet1"). Cells(rowCount, 1). Value)
    Bookmark = ix_APILIB_ITEM. GetBookmark(Keys, ad400SeekFirstEQ)
    rd_APILIB_ITEM. Move 0, Bookmark
    rd_APILIB_ITEM. Delete adAffectCurrent

    Set dm_APILIB_ITEM = Nothing
    Set rd_APILIB_ITEM = Nothing
    Set ix_APILIB_ITEM = Nothing
    Exit Sub
errors:
    'error handling code here
End Sub
```
a delete operation. We specify the CommandText using the appropriate syntax for record-level access.

Next we set the **CommandLine** property to **adCmdTable** and the **Properties** ("Updatability") property to 2. The **adCmdTable** option indicates that we want to use record level access. The "Updatability" property indicates that we want to perform a delete operation and the file should have a Shared Update Lock.

Next we invoke the **Execute** method. This opens the ITEM database file and returns all records to a **Recordset** object. For record level access, the returned **Recordset** object is a dynamic **Recordset**. This means that any changes such as add, delete or update that are made to the **Recordset** object are immediately reflected in the AS/400 database file.

The **Delete** method removes an existing row from the **Recordset** object. Since the row is already in the **Recordset**, we need to be able to find this row effectively. ADO 1.0 which the Client Access OLE DB provider implements does not have a **Seek** or **Find** method. So we need to invoke the **OpenIndex**, **GetBookmark** and **Move** methods to aid the search operation. The **OpenIndex** and the **GetBookmark** methods are implemented in the **AD400 Automation** object. We have discussed declaring and instantiating **AD400 Connection** and **Index** object in Section 9.2.0.1, “Global Variables and Class Initialization” on page 289.

The **OpenIndex** method takes two parameters: File name and Index name. Both parameters refer to the same AS/400 object - the physical or logical file that you open. This method returns an **AD400 Index** object. We use "APILIB.ITEM()" for file and "APILIB.ITEM()" for index. The braces after file and index name are required. After a successful completion an **AD400 Index** Object is returned.

Next we read the key field value from the highlighted row of the spreadsheet. In this example the IID field is used as a key for the search. The **GetBookmark** method takes two parameters: **Keys** and **SeekOption**. **Keys** is a Variant array containing the IID field value, and the seek option is ad400SeekFirstEQ. This seek option returns the first row that matches the specified key value.

We then use the **Move** method to navigate through the **Recordset** and find the record that matches the IID key value.

We then invoke the **Delete** method to modify the **Recordset**. The Client Access OLE DB provider supports only one parameter option for the **Delete** method: adAffectCurrent, which indicates that current record should be deleted from the **Recordset**. Since the changes on the **Recordset** are dynamically reflected in the AS/400 file, the row is also removed from ITEM database file after this operation completes.

Finally, we call the **GetAll** method. This refreshes the spreadsheet. Note that calling the **GetAll** method at this point is not required, but we invoke this method so that the changes made to the AS/400 database file are reflected here.

Note that the **Delete** method uses ADO objects that are declared Globally and then instantiated from the **Class.Initialize** subroutine. For additional information on this, refer to Section 9.2.0.1, “Global Variables and Class Initialization” on page 289.
9.2.0.5 Update a Record

When you select Actions -> Update, two functions are called: Update and GetAll. Update reads the highlighted row from the Excel spreadsheet then modifies the AS/400 database file accordingly. The GetAll method reads the AS/400 database file and refreshes the spreadsheet with the records that are read. Both the Update and GetAll methods are implemented in the ADOClass class object. We discuss the GetAll method in Section 9.2.0.2, “Get All Records” on page 290. In this section we discuss the Update method.

```vba
Sub Update()
    Dim flds, vals, Keys, numRecs, parm, rcds, Bookmark
    Dim rowCount As Integer
    Dim ColCount As Integer
    Set dm_APILIB_ITEM.ActiveConnection = cnTCPASM05
    dm_APILIB_ITEM.CommandText = "APILIB.ITEM()"
    dm_APILIB_ITEM.CommandType = adCmdTable
    dm_APILIB_ITEM.Parameters.Append dm_APILIB_ITEM.CreateParameter("P1", adChar, adParamInput, 1)
    dm_APILIB_ITEM.Properties("Updatability") = 1 ' change=1
    Set rd_APILIB_ITEM = dm_APILIB_ITEM.Execute(rcds, Parm, adCmdTable)
    Set ix_APILIB_ITEM = ixTCPASM05.OpenIndex("APILIB.ITEM()", "APILIB.ITEM()")
    Worksheets("Sheet1").Activate
    rowCount = Selection.Row
    ColCount = 0
    flds = Array("IID", "INAME", "IPRICE", "IDATA")
    vals = Array(Worksheets("Sheet1").Cells(rowCount, 1).Value,
                    Worksheets("Sheet1").Cells(rowCount, 2).Value,
                    Worksheets("Sheet1").Cells(rowCount, 3).Value,
                    Worksheets("Sheet1").Cells(rowCount, 4).Value)
    Keys = Array(Worksheets("Sheet1").Cells(rowCount, 1).Value)
    Bookmark = ix_APILIB_ITEM.GetBookmark(Keys, ad400SeekFirstEQ)
    rd_APILIB_ITEM.Move 0, Bookmark
    rd_APILIB_ITEM.Update flds, vals
    Set dm_APILIB_ITEM = Nothing
    Set rd_APILIB_ITEM = Nothing
    Set ix_APILIB_ITEM = Nothing
    Exit Sub
    errors:
    ’error handling code here
End Sub
```

First we set the ActiveConnection property for the Command object cm_APILIB_ITEM. This binds the ADO Connection object cnTCPASM05 to the cm_APILIB_ITEM Command object.

The CommandText property specifies the AS/400 database file name that we want to open. For the update operation we use the record level access method to the AS/400 database file. We specify the CommandText using the appropriate syntax for record level access.
Next, we set the **CommandType** property to **adCmdTable** and the **Properties** ("Updatability") property to 1. The **adCmdTable** option indicates that we want to use record-level access. The "Updatability" property indicates to that we want to perform an insert operation and the file should have a Shared Update Lock.

Next, we invoke the **Execute** method. This opens the ITEM database file and returns all records to a **Recordset** object. For record level access, the returned **Recordset** object is a dynamic Recordset. This means that any changes such as add, delete or update that are made to the **Recordset** object are immediately reflected in the AS/400 database file.

The **Update** method modifies an existing row in the **Recordset** object. Since this row is already in the Recordset, we need to be able to find this row effectively. ADO 1.0 which the Client Access OLE DB provider implements does not have a Seek or Find method. So, we need to invoke the **OpenIndex**, **GetBookmark** and **Move** methods to aid the search operation. Note that the **OpenIndex** and **GetBookmark** methods are implemented in the AD400 Automation object. We have discussed declaring and instantiating AD400 **Connection** and **Index** objects in section 9.2.0.1, “Global Variables and Class Initialization” on page 289.

The **OpenIndex** method takes two parameters: File name and Index name. Both parameters refer to the same AS/400 object - the physical or logical file that you open. This method returns an AD400 Index object. We use "APILIB.ITEM()" for file and "APILIB.ITEM()" for index. The braces after the file and index name are required. After a successful completion an AD400 **Index** Object is returned to **ix_APILIB_ITEM**.

Next we read the highlighted row from the Excel spreadsheet. In this example the first field IID is used as a key for the search. The **GetBookmark** method takes two parameters: **Keys** and **SeekOption**. **Keys** is a Variant array containing the IID field value from the highlighted row and **SeekOption** is set to the **ad400SeekFirstEQ**. This seek option returns the first row that matches the key values specified in **Keys**.

We then use the **Move** method to navigate through the **Recordset**. It makes the matching record our current record.

Next we invoke the **Update** method to modify the **Recordset**. The **update** method takes two parameters, a list of fields and a list of values for those fields. Both lists are passed as Variant arrays to the method call. We update a row in the ITEM database file after this operation completes.

Next we call the **GetAll** method. This refreshes the spreadsheet. Note that calling **GetAll** method is not required, but we invoke this method so that the changes made to the AS/400 database file are reflected on the spreadsheet.

Note that the **Update** method uses ADO objects that are declared Globally and then instantiated from the Class_Initialize subroutine. For additional information on this refer to Section 9.2.0.1, “Global Variables and Class Initialization” on page 289.

### 9.3 Microsoft Access 97

This section covers how to use Microsoft Access with the Client Access OLE DB provider. The example discussed in this section has an option to retrieve all the...
records from a physical file on the AS/400 system into a Microsoft Access table on the PC. Also it allows the user to upload the records from the MS Access table to the AS/400 physical file. This example is based on the ITEM file in the APILIB library. Please see Chapter 3, “Overview of the Order Entry Application” on page 87 for the structure of the ITEM database.

It is assumed that you have a knowledge of the ADO programming model. Please see Chapter 1, “OLE DB and ActiveX Overview” on page 1, for a detailed discussion on the ADO programming model.

![Microsoft Access Table](image)

Figure 141. Microsoft Access Table

As a first step, we create the MS Access table with the same structure as the ITEM file in the APILIB library on the AS/400 system (see Figure 141 on page 297). Next, we create a MS Access Form which is linked to the MS Access table that we created in the first step.
We write a Class Module that has all the database access logic in Visual Basic. The class module and the functions implemented are discussed in this section. We have chosen to use SQL for retrieving all the records from the ITEM file and record-level access for uploading the records to the ITEM file on the AS/400 system.

```
Option Compare Database
Option Explicit

Public cnTCPASM05 As New ADODB.Connection
Public Err As ADODB.errors
Public cm_APILIB_ITEM As New ADODB.Command 'for SQL command
Public dm_APILIB_ITEM As New ADODB.Command 'for DDM command
Public rd_APILIB_ITEM As ADODB.Recordset   'for DDM access
Public rs_APILIB_ITEM As ADODB.Recordset   'for SQL access
```

We declare all the ADO objects as global variables. We have declared a Connection object, two Command objects (one each for the SQL statement and the record-level access processing) and two Recordsets.
In the Initialize subroutine, we establish a connection to the AS/400 system. We use the `Open` method of the `Connection` object. Then we use the `Command` object to prepare the SQL statement to get all the records from the ITEM file on the AS/400 system.

```vbscript
Private Sub Class_Initialize()
    cnTCPASM05.Open "provider=IBMDA400;data source=tcpasm05;"
    Set cm_APILIB_ITEM.ActiveConnection = cnTCPASM05
    cm_APILIB_ITEM.CommandText = "SELECT * FROM APILIB.ITEM ORDER BY IID"
End Sub
```

In the Class Terminate, we release all the PC as well as AS/400 resources by closing the connection and setting it to `Nothing`.

```vbscript
Private Sub Class_Terminate()
    Set cnTCPASM05 = Nothing
End Sub
```

In the GetAll subroutine, we execute the SQL statement that we prepared in the Initialize subroutine. This returns back a result set which is loaded in the `Recordset` object. We also open a Recordset for the MS Access table `ADOTable`. We move the records from the Recordset populated by the SQL statement to the local Recordset(the MS Access table). Thus all the records from the ITEM file on the AS/400 system are populated in the MS Access table. See Figure 143 on page 300.

```vbscript
Sub GetAll()
    Dim rowCount As Integer
    Dim colCount As Integer
    Dim db As Database
    Dim AccessRS As Variant

    Set db = CurrentDb
    Set AccessRS = db.OpenRecordset("ADOTable")
    'execute the SQL query
    Set rs_APILIB_ITEM = cm_APILIB_ITEM.Execute()
    'get the all rows from the recordset
    While Not rs_APILIB_ITEM.EOF
        With AccessRS
            .AddNew
            !IID = rs_APILIB_ITEM.Fields(0).Value
            !INAME = rs_APILIB_ITEM.Fields(1).Value
            !IPRICE = rs_APILIB_ITEM.Fields(2).Value
            !IDATA = rs_APILIB_ITEM.Fields(3).Value
            .Update
        End With
        rs_APILIB_ITEM.MoveNext
    Wend
    Set rs_APILIB_ITEM = Nothing
    Set AccessRS = Nothing
End Sub
```
Figure 143. MS Access Table Populated from the ITEM File

After we have all the records in the MS Access table, we can make changes and add new records to delete records from the table locally. The UploadAll subroutine takes all the records, after you have changed them, from the MS Access table and uploads them to the ITEM file on the AS/400 system. There are numerous ways to do this. Once you are familiar with how to use OLE DB with MS Access, you can develop applications to suit your requirements.
In the UploadAll subroutine, we open the ITEM file for record-level access in updatable mode. We open the MS Access table named ADOTable. Then we delete all the records from the ITEM file on the AS/400 system. Next, we read the records one at a time from the MS Access table and write them to the ITEM file on the AS/400 system.
In the subroutine named CloseAll, we delete all the records from the MS Access table named ADOTable.

```vba
Sub CloseAll()
    Dim db As Database
    Set db = CurrentDb
    db.Execute "Delete * from ADOTable;"
    db.Close
End Sub
```
Chapter 10. Internet-Based Applications

This chapter discusses some samples of dynamic web pages that get their content from tables on the AS/400 system. In all the cases that are shown here, the HTML files contain scripting code. This code can be executed on the browser (client workstation), on the web-server or on both systems. The example code is available for download, see Appendix A.1, “Downloading the Samples” on page 373 for details.

10.1 Client Side Scripting

The scenario in this section is perhaps the easiest to understand. We show a web page that contains some scripting code that is executed by the browser, on the client system. The sample page looks like Figure 144: It allows us to access information in the PARTS file on the AS/400 system from a browser. Access to the Parts file is done through the Client Access OLE DB provider SQL interface. Pushbuttons are provided which allow the user to navigate through the file.

SQL Sample

The following fields have been generated automatically based on the schema information found in the recordset. If the data fields do not appear below this message you may have to reduce the level of security in your browser.

<table>
<thead>
<tr>
<th>PARTNO</th>
<th>12302</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARTDS</td>
<td>SCSI II Cable</td>
</tr>
</tbody>
</table>

Figure 144. HTML Page with Client-Side Scripting

Let’s take a look on how this page is constructed. First of all, there is the code that makes up the HTML header, the three pushbuttons and the text paragraph on the page. This code is included here for completeness:
Next we see the HTML tag to start a section for client-side scripting code. This tag looks like:

```html
<script language="VBScript">
```

It indicates the start of scripting code and that we want to run the code using the VBScript interpreter. This tag is recognized by web-browsers such as Microsoft Internet Explorer 4.0 (MSIE) or Netscape Navigator. It should be noted here that MSIE supports VBScript directly. When using Netscape Navigator, you need to install a plug-in to support VBScript.

The first piece of VBScript code is not defined as a procedure or function. The VBScript Scripting Engine recognizes this and executes this code while the page is starting up. As we will show, this type code can be used to generate HTML while the browser is rendering the page.

The scripting code starts with the creation of an ADO Recordset object. It opens the recordset by passing a connect string and an SQL statement to the object. This is exactly the same type of code that can be seen in the other chapters of this redbook.
Next, there is some code that is more interesting. The code first loops through the columns to find the longest column name. Next it loops once more through the column information and generates extra HTML tags. These tags are added to the page, while the browser is displaying the page.

```vbnet
'**********************************************************
' Build the html for viewing the data
'
' determine width for field-name column
',
namewid = 0
for i = 0 to flds.Count - 1
  if len(flds(i).Name) > namewid then namewid = len(flds(i).Name)
next
' write out html for form-fields
',
for i = 0 to flds.Count - 1
  document.write "<pre>
  s = flds(i).Name
  s = s + space( namewid - len(s) + 2 )
  s = s + "<input type=text name=fld_" + cstr(i) + ">
  document.write s
  document.write "</pre>
next
```

In this case, each line of HTML that is generated contains the field name and an edit-box to display the value of this field. Each line is displayed in a fixed width font, so that the edit-boxes can be nicely arranged on the screen.

The third and last piece of code is also quite interesting. This code generates additional VBScript code that is added to the existing VBScript code. It will generate a function called **FillForm**, that contains a line for each field. It will be used to fill up the edit-boxes with the actual data values for a particular record.
After the function is defined, a call to this function is included. This ensures that the first record is displayed as soon as the page is displayed.

```vbscript
'******************************************************************************
' Build the vbscript for copying data to the form

document.writeln "<script language="vbscript">"

document.writeln "sub FillForm"
for i = 0 to flds.Count - 1
    s = "fld_" + cstr(i) + ".value = " + "rs.fields(" + cstr(i) + ").value"
    document.writeln s
next

document.writeln "end sub"

' force 'FillForm' to execute immediately

document.writeln "FillForm"
document.writeln "</vbscript>"

Next there is the code that implements the button click events:

```vbscript
Sub btnNext_OnClick
    if not rs.EOF then
        rs.MoveNext
        if rs.EOF then
            rs.MovePrevious
            MsgBox "At end of file"
        else
            FillForm
        end if
    end if
End Sub

Sub btnPrev_OnClick
    if not rs.BOF then
        rs.MovePrevious
        if rs.BOF then
            rs.MoveNext
            MsgBox "At beginning of file"
        else
            FillForm
        end if
    end if
End Sub

Sub btnFirst_OnClick
    rs.MoveFirst
    FillForm
End Sub
This code is similar to the samples that are discussed in Chapter 2, “AS/400 OLE DB Support” on page 39. It navigates through the result set and takes advantage of the FillForm function to display the information.

Finally, we see the tags to close the scripting section, draw a line and close the page.

```html
</script>
<hr size=4>
</body>
</html>
```

This is all the code that is needed to implement the page in Figure 144 on page 303. However, this code suffers from a major drawback: all processing is done on the client side. It requires that every client PC has the Client Access OLE DB Provider installed. As such, it behaves very much like a conventional Windows client/server application. The only real change is that we are using a browser to take care of the user interface.

This approach may be very useful in intranet situations, where you have control over the client PCs. Even then, this scenario creates additional configuration concerns. These can be avoided by using server side scripting.

A second problem is that every client workstation needs a browser that is capable of executing the scripting language that is included in the page. This might be a reason to select JavaScript as a language, as this engine is supported directly by more browsers than the VBScript engine we used to create this sample.

An advantage of client-side scripting is that there is no restriction on the web server that is being used. When used only in a LAN, the HTML pages could even reside in a directory on a file server.

10.2 Server Side Scripting: an ASP example

ASP is an acronym for Active Server Pages. This is a technique that is used in the Microsoft Internet Information Server (IIS) to create HTML pages using server side scripting.

ASP is different from CGI scripting in that the code in an ASP page is HTML code that is enhanced by the inclusion of scripting code. It looks very much like client side scripting. But the scripting code has a different set of objects that it can talk to. And of course, processing is done on the server only.

An ASP page can generate any kind of HTML page. This makes it much easier to create an application that supports different types of browsers. In this sample, we discuss a simple ASP page that generates a static HTML page. The page looks like the screen capture in Figure 145. It allows us to retrieve a list of customers from the Customer master file using the Client Access OLE DB provider.
If we look at the contents of the ASP file, we see that the page starts just like a regular HTML page. In fact there’s nothing in the header that distinguishes this page from standard HTML.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta name="GENERATOR" content="Microsoft FrontPage Express 2.0">
<title></title>
</head>

<!--#include file="adovbs.inc"-->

<body>
<h1 align="center">List of Customers</h1>
```

The line immediately following this header is a server-side include. This means that the web server will recognize this syntax and replace it by the contents of the `adovbs.inc` file. This file contains all the constants that can be used in ADO code.

In general, server-side includes can be useful to include information in a lot of web pages, without having to retype the HTML tags every time. It also allows to include files recursively. This means that an include file can itself include other files, and so on.
The next few lines in the ASP page deal with creating the HTML table that is used to display the information. There are no surprises here, this is just standard HTML code.

```html
<div align="center"><center>
<table border="2">
<tr>
  <td><strong>Id</strong></td>
  <td><strong>District</strong></td>
  <td><strong>Warehouse</strong></td>
  <td><strong>First Name</strong></td>
  <td><strong>Last Name</strong></td>
</tr>
</table>
</center></div>
```

Then we reach the important part of this page. We see that the server-side code is enclosed in `<%` and `%>` tags. This is opposed to client side scripts, which use the `<script>` and `</script>` tags.

The VBScript code should be familiar to the reader. It is very much like code written for Microsoft Visual Basic, as discussed in Chapter 2, “AS/400 OLE DB Support” on page 39. Please note that there are some special concerns, as the code is running in a server process. Please refer to Chapter 11, “Three-Tier Approach” on page 325 to get an overview of running the Client Access OLE DB Provider in a 3 tier environment.

This piece of code contains one construct that may look strange to programmers who write conventional Windows applications: the code seems to stop suddenly. Then there are some HTML tags and the code continues. In fact, Microsoft Internet Information Server sees this code as one single program. It will execute the code and it will output the HTML tags in the middle of the VBScript code.

The scripting code contains a loop that will cause one set of tags for each record in the recordset. These tags correspond to a row in the HTML table. In the HTML tags you can see statements that look like mini scripts:

```
<%=rs("CID")%>
```

The fact that they start with an equal sign, signals IIS that this is an expression. The web server will use the value when generating the final HTML page. In the sample shown here, IIS will include the value of the CID column in the recordset.
Finally, the last part of the ASP page is quite simple. It closes the definition of the table. And it finishes the HTML page with the appropriate tags.
It can be helpful to run this ASP page and look at the HTML source that the browser receives. You will find only static HTML code there. When IIS executes an ASP script, it will entirely remove all server-side scripting statements.

The advantage of this type of code is that you need no knowledge of the client workstations. You can generate the HTML you want, so it is easy to avoid non-portable constructs. Also, the only system that needs to access the database is the server running IIS. This can reduce the configuration complexity on an intranet and makes it possible to publish information from the database on the Internet.

The disadvantage of this technique is that the ASP technology is implemented only by Microsoft's Internet Information Server.

### 10.3 Using a Custom Business Object

In this section we show how a complete Internet application can be made. This sample includes both client-side and server-side scripting (ASP). It also calls functions that are packaged in a DLL that runs on the server. Many publications call these functions the business logic or the business objects.

The application that we show here is a data-entry screen using the web browser as its user interface. The application enables users to find a customer, based on the customer ID, update the information and add new customers to the file. The application looks like Figure 146.

**Note**

The code for this sample is available for download, see Appendix A.1, “Downloading the Samples” on page 373 for details.

To run the application, the workstation only needs to contact an HTTP server. All communication is done through the TCP/IP protocols. This makes this kind of application suited for use on the Internet.
10.3.1 Creating the Web Page

The first step in creating this application is exactly like developing any other web page. You can use your favorite tool to design the web-page. The sample shown here has a very simple layout. But of course all the HTML cosmetics are available to you when developing your web application.

We will not include the HTML code here for the header or for the basic layout. It is a simple entry form using HTML text boxes and a few HTML pushbuttons. The form makes sure that the entry boxes are nicely arranged by using an HTML table with the line size set to 0. If you want to see this in full detail, please download this example and take a look at the actual files.

After the last buttons and before the scripting code, we include an object that is of particular interest. This is the Microsoft Remote Data Service (RDS) control. In this application we also named this control RDS. The HTML tag for this control is included here:
This RDS control works in many ways like a traditional data control. However it was developed with web applications in mind. This control is able to establish a communication with a middle tier server, call methods of objects on a middle tier server and transfer result sets from the middle tier server back to the workstation. This is the key component to make this application work.

Let's look at a few things we can do with this control;

**Using the RDS.DataControl Object**

One of the uses of the RDS object is as a data control, exactly as the traditional data controls. A small example of this use is shown here:

```html
<object id="RDC1" classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33" align="baseline" border="0" width="14" height="14">
    <param name="SQL" value="SELECT * FROM apilib.cstmr">
    <param name="Connect" value="DSN=tcpasm05;uid=A980502E;pwd=LIGHT01A">
    <param name="Server" value="http://<%=Request.ServerVariables("SERVER_NAME")%>">
</object>

<object id="GRID1" classid="CLSID:AC05DC80-7DF1-11d0-839E-00A024A94B3A" codebase="http://<%=Request.ServerVariables("SERVER_NAME")%>/MSADC/Samples/ssdatb32.cab" align="baseline" border="0" width="500" height="160" datasource="#RDC1">
    <param name="_Version" value="131072">
    <param name="BackColor" value="-2147483643">
    <param name="BackColorOdd" value="-2147483643">
    <param name="ForeColorEven" value="0">
    <param name="AllowAddNew" value="FALSE">
    <param name="AllowDelete" value="FALSE">
    <param name="AllowUpdate" value="FALSE">
</object>
```

This is an extract out of a sample page, called Cust3Tiers.asp. This page is also available for download.
The important changes are printed in a bold font. As compared to the previous use of the RDS object, we see that there are three additional parameters here.

1. The parameter "SQL" provides the data control with an SQL statement
2. The parameter "Connect" is an ODBC Connect string.
3. The parameter "Server" tells the control which server it should contact to get the result set. We used the capabilities of the ASP environment here to fill in the server name.

When this page is loaded, the RDS object will contact the server and forward the SQL statement. The RDS object will cache the result-set on the workstation.

Finally, we include a grid object in the HTML page and set one of its properties:

```html
datasrc="#RDC1"
```

When loading the page, the control will contact the data control and display the result set. This solution doesn’t require a single line of scripting code. However, we will not use this technique or describe it in further detail, since it is based on ODBC. It doesn’t use the OLE DB provider.

**Using the RDSServer.DataFactory Object**

A similar, but slightly more flexible technique is to use the RDSServer.DataFactory object. Here is an extract of the same sample page as in the previous section:

```html
<object id="RDC2"
classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
width="14" height="14">
</object>

<object id="GRID2"
classid="CLSID:AC05DC80-7DF1-11d0-839E-00A024A94B3A"
codebase="http://<%=Request.ServerVariables("SERVER_NAME")%>/MSADC/Samples/ssdatb32.cab"
width="500" height="155"
datasrc="#RDC2">
<param name="_Version" value="131072">
<param name="BackColor" value="-2147483643">
<param name="BackColorOdd" value="-2147483643">
<param name="ForeColorEven" value="0">
<param name="AllowAddNew" value="FALSE">
<param name="AllowDelete" value="FALSE">
<param name="AllowUpdate" value="FALSE">
</object>
```

This looks almost the same. Again we have an RDS Control and a grid. The grid is bound to the data control, just like we did in the previous section. Only, this time we don’t specify any parameters for the RDS control, but instead we define the connection using scripting code. This code is shown below:
This code uses the RDS object, which is local on the workstation, to create the RDSServer.DataFactory object on the server. Next, we call the Query method of this server object. The Query method returns a result set, that is transferred to the (local) RDS object. The data binding of RDS with the grid does the rest of the work for us.

We specify the parameters of the previous section in the scripting code here. In fact, the RDS.DataControl object does exactly the same as the code shown here.

```
<script language="VBScript">
<!--
Option Explicit

Sub Window_OnLoad

    Dim bObj
    Dim rsResult

    Set bObj = RDS.CreateObject("RDSServer.DataFactory",
        "http://<%=Request.ServerVariables("SERVER_NAME")%>")
    Set rsResult = bObj.Query("DSN=tcpasm05;uid=A980502E;pwd=LIGHT01A",
        "SELECT * FROM apilib.cstmr")

    RDC2.SourceRecordset = rsResult

    set bObj = nothing
End Sub
-->
</script>
```

When using the Microsoft Internet Information Server on the Internet, anyone could write a HTML page that instantiates a copy of the RDSServer.DataFactory object on your server. Given a valid data source name, user ID and password, any SQL query can be sent to that data source. This is a potential security risk.

If your NT server is connected to the Internet, you may consider unregistering this object.

Again, we will not elaborate on these topics any further, since they rely on ODBC as the data access mechanism. Instead we want to show how the Client Access OLE DB Provider fits in the picture.

**Using a Business Object**

This third option is in many ways more similar to the previous samples than you might expect at first. The RDSServer.DataFactory object is nothing more than a standard business object, that was implemented by Microsoft.
The declaration of the RDS object was straight forward. Now we look at the initialization and termination code of the web page. After seeing the code for the RDSServer.DataFactory code, there are no surprises here.

In the initialization code (the Window_OnLoad function), we call the CreateObject method of the RDS object and pass it the object name and the server name where we want to have the object created.

The termination code releases the reference to the object. It is not really necessary to include this code. The VBScript engine is responsible to clean up all variables anyway. This cleanup code in included for completeness.

```vbnet
Dim oBO
Sub Window_OnLoad
    ResetPage
    Set oBO = RDS.CreateObject("Redbook.Demo",
        "http://<%=Request.ServerVariables("SERVER_NAME")%>")
End Sub

Sub Window_OnUnload
    Set oBO = Nothing
End Sub
```

To show the business object in use, we include the source of the function that gets executed when the user clicks the Find button. The function starts with a simple check if the key values all have been specified. If so, the function can call the GetCustomer method of our business object, just as if it was an object on the local machine.

The business object actually resides on the middle tier server. What we see here is the COM layer that creates a proxy object. This is an object that imitates the behavior of the real object on the server. This proxy object creates a connection over the network with another proxy object on the middle tier server. All calls are forwarded to the proxy on the other side and passed to the DLL that is running there. Any results come back using the same technique.

After the call to the GetCustomer method, we see code that fills in the values of the edit boxes on the page. All of this is the usual type of scripting code.
Sub btnFind_Click
    Dim FirstName, MiddleName, LastName, Addr1, Addr2, City, State, Zip, Phone

    FirstName = ""
    MiddleName = ""
    LastName = ""
    Addr1 = ""
    Addr2 = ""
    City = ""
    State = ""
    Zip = ""
    Phone = ""

    If (txtCustId.Value = "") Or (txtDistId.Value = "") Or _
        (txtWareHouse.Value = "") Then
        MsgBox "Please specify Customer Id., District Id. and " & _
            "Warehouse Id. first."
    Else
        txtCustId.Value = right ("0000" + trim (txtCustId.Value), 4)
        txtWareHouse.Value = right ("0000" + trim (txtWareHouse.Value), 4)

        If oBO. GetCustomer (txtCustId.Value, txtDistId.Value, _
            txtWareHouse.Value, FirstName, MiddleName, LastName, _
            Addr1, Addr2, City, State, Zip, Phone) Then
            txtFirstName.Value = FirstName
            txtMiddle.Value = MiddleName
            txtLastName.Value = LastName
            txtAddr1.Value = Addr1
            txtAddr2.Value = Addr2
            txtCity.Value = City
            txtState.Value = State
            txtZip.Value = Zip
            txtPhone.Value = Phone
            SetStatus (nStatusUpd)
        Else
            txtFirstName.Value = ""
            txtMiddle.Value = ""
            txtLastName.Value = ""
            txtAddr1.Value = ""
            txtAddr2.Value = ""
            txtCity.Value = ""
            txtState.Value = ""
            txtZip.Value = ""
            txtPhone.Value = ""
            SetStatus (nStatusAdd)
        End If
    End If
End Sub
10.3.2 Creating the Business Object

Next we show how the business object can be created. To implement this, you can use any development environment that allows to develop Automation server code. In this section, we use Microsoft Visual Basic 5.

Note

Also refer to Chapter 11.2, “Client Access OLE DB Provider” on page 337 on three-tier development. Many of the remarks in that chapter apply here as well.

The first step is to create a new project in Visual Basic. The important steps are:

- Select an ActiveX DLL as the type of project. See Figure 147 on page 318
- Set the project name to RedBook. See Figure 147 on page 318
- Add a class module to the project. See Figure 148 on page 319
- Change the name of the class module to Demo and set the Instancing property to MultiUse. See Figure 149 on page 319
- When adding a reference to the ADOR libraries, make sure to select the ADOR 1.5 (or later) type library.

The result of these settings is that you create a DLL that implements an Automation object that is called Demo. The full name of the object, or the PROGID of this object, is RedBook.Demo. This must match the name we specified in the CreateObject method of the RDS object in the ASP page.
In the class modules, we can declare a list of functions. All functions that are marked with the `Public` attribute, are callable through the Automation interface. For our purpose, we need a function `GetCustomer`. This function receives the key fields, and uses record level access via the Client Access OLE DB Provider to access the Customer master file. The function is included here in a simplified version. If you want to see the complete function, please download the sample from the Internet (see appendix A for instructions) and look at the actual source file.

**Tip**

Test the function as complete as possible using a simple client application, before trying to call it from a web client. This makes not only debugging a lot easier, but Microsoft Internet Information Server keeps all objects loaded into memory, once they have been used. This is good for performance, but replacing a DLL with a newer version requires a server reboot every time.
10.3.3 Installing the Business Object

Installing the business object code on the server is an straightforward process. Follow these steps:

1. Copy the DLL to a directory on the web server
2. Register the DLL on the server. This must be done by running the REGSVR32.EXE program on the NT Command Prompt. Specify the name of your DLL as the argument.
3. Run the Registry Editor (REGEDIT.EXE). In the registry tree, navigate to the HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\ADCLaunch key.
4. Add a new key under the ADCLaunch key. Use the PROGID of your business object as the name of the new key. The value of this key can be left to blank.

The result of step 3 and 4 is shown in Figure 150 on page 321. These steps are imperative. RDS will not allow the creation of any object that is not listed in this key.

Finally, keep in mind that IIS will keep a business object in memory when it has been used. To replace a business object with a newer version, it is possible that you will have to reboot the server.
10.3.4 Marking the Business Object as Safe

As a last step, you should mark the business object as "safe for scripting". The development environment that you are using, may provide facilities to mark your objects as safe automatically. Or it may allow the programmer to change the registration logic. In this case, you will probably choose for this option.

Here is how an object can be marked as safe manually. Follow these steps:

1. Start the Registry Editor (REGEDIT.EXE)

2. Navigate to HKEY_CLASSES_ROOT\RedBook.Demo, or the name of your business object (See Figure 151). The hexadecimal number is called the GUID (globally unique identifier) of the business object. This number will be different on your system.

3. Navigate to HKEY_CLASSES_ROOT\CLSID\{C7BEB9CE-F042-11D1-862E-0006296B3C0B}. Replace the hex number with the value on your system.

4. Make sure that the value of this key matches the PROGID of your business object and go to the Implemented Categories key.
5. Add the category \{7DD95801-9882-11CF-9FA9-00AA006C42C4\}
6. Add the category \{7DD95802-9882-11CF-9FA9-00AA006C42C4\}

The registry editor should now look like Figure 152. This completes the manual process of marking the business object as safe.

Figure 152. Adding the New Implemented Categories

You can find a more exact description of these component categories in the registry by going to the registry key `HKEY_CLASSES_ROOT\Component Categories\{7DD95801-9882-11CF-9FA9-00AA006C42C4\}`, as shown in Figure 153.

Figure 153. Category "Safe for Scripting"

Also see the key `HKEY_CLASSES_ROOT\Component Categories\{7DD95802-9882-11CF-9FA9-00AA006C42C4\}`, as shown in Figure 154.
10.3.5 Advantages and Disadvantages

This section shows how an web application can be written in a three-tier fashion. The sample here is fairly simple, but it shows how a complex and functionally rich application can be created using these techniques.

The disadvantage of this approach is that it requires specific products, on both the web server side as well as on the browser side. On the web server side, we need the Microsoft Internet Information Server to implement the object functionality. The browser must be able to handle ActiveX objects and the RDS object must be installed on the client workstation.

This means that this solution is only usable for Windows clients. All these prerequisites are satisfied when using Microsoft Internet Explorer 4 as a browser. If you wish to use another browser, such as Netscape Navigator, you need the appropriate plug-ins to make the ActiveX components work.
Chapter 11. Three-Tier Approach

Developing an application using a three tier approach is basically a way to structure an application into three different building blocks. The first tier (or the presentation tier) deals with the user interface. The second tier implements the business objects. The third tier does the actual processing of the data.

These three tiers can be run on three separate machines on the network. Typically, running the three tiers on separate machines is used as a technique to improve the scalability of an application. However, it is also a good technique to improve the structure of an application that is intended to run on two tier hardware.

This chapter covers the following topics

• Section 11.1 "Microsoft OLE DB Provider for AS/400 and VSAM"
• Section 11.2 "Client Access OLE DB Provider"

11.1 Microsoft OLE DB Provider for AS/400 and VSAM

This section provides an overview of the Microsoft OLE DB driver for AS/400 and VSAM. The purpose is not to describe this product in full detail or serve as a programming guide. Instead the primary target of this section is to give the reader an overview of the feature set product provides and show how it compares to the Client Access OLE DB Provider.

11.1.1 Overview

The Microsoft OLE DB Driver for AS/400 and VSAM is a product that allows Windows applications to access data on hosts in a Systems Network Architecture (SNA) environment. This driver is offered by Microsoft as a part of the Microsoft SNA Server 4.0.

The information in this chapter is based on version 4.0 of this driver and the Microsoft documentation that comes with it. The feature set of this driver may be changed in subsequent releases of the product.

11.1.1.1 Distributed Data Management Requester

The Microsoft OLE DB Driver for AS/400 and VSAM is implemented as a DDM Requester. As such it provides record level access to hosts that implement the DDM Server protocols. Because these protocols are typically integrated in the IBM host operating systems, there is no need to install additional drivers on the host side.

11.1.1.2 Implemented Feature Set

The driver implements record level access to data files. This is done using a typical set of navigational database operations:

• Moving the Record Pointer to the First or Last Record in the File
• Move the Record Pointer to the Next or Previous Record in a File
• Seek a Record using an Index
• Change a Record
• Insert and Delete Records
• Locking Files
• Locking Records
• Set Attributes and Record Description of a Host File

When accessing data on an AS/400 host, it is possible to access both Logical and Physical Files.

The Microsoft OLE DB Provider also allows a client application to call CL Commands on the AS/400 DDM Server. It provides no way to get any output back from this remote command. The only supported way to retrieve results is to create a temporary file and redirect all output to this file. Next, the client application needs to open this file in a new, separate ADODB.RecordSet object to obtain the desired information.

There is no equivalent functionality to start batch processes on the other supported platforms (mainframe systems).

11.1.2 A Comparison with the Client Access OLE DB Provider

This section shows the most important advantages that each OLE DB provider has over the other, competing product.

11.1.2.1 Advantages of the Microsoft OLE DB Provider
The Microsoft Provider offers access to a large range of systems. As it is implemented as a generic DDM Requester application, it could potentially work with any server that implements the SNA protocols and a DDM Server.

In fact, this generic implementation enables the Microsoft Provider to access tables on the AS/400 as well as VSAM tables that are stored on mainframes.

Please make sure to check the Microsoft SNA Server SDK documentation for a complete list of supported platforms.

11.1.2.2 Advantages of the Client Access OLE DB Provider
1. The implementation of the Microsoft Provider as a generic DDM Requester enables the product to support a large variety of systems. But this comes at the cost of not being optimized for the hardware that you are using. The Client Access OLE DB Provider is written in such a way that it allows access to AS/400 hosts only. This makes it possible for the software to optimize the DDM calls towards the AS/400. These optimized calls will provide a better performance when accessing data on an AS/400 system.

2. The Microsoft OLE DB Provider implements only record oriented access to the AS/400 data. Access to data through SQL statements is not possible at all. The Client Access OLE DB Provider implements record oriented access as well as SQL access to data stored on the AS/400.

3. The Microsoft Provider also doesn’t implement the ability to call stored procedures on the AS/400. Calling stored procedures is a technique that can significantly boost application performance. The Client Access Provider allows access to stored procedures.

4. The Client Access Provider also implements a way to call existing AS/400 programs using the standard OLE DB or ADO object model. This includes
programs written in languages such as RPG, COBOL, ILE C, and so on. The Microsoft Provider doesn’t have this capability.

5. Similarly, the Microsoft Provider offers no means to access AS/400 data queues. Again, using the Client Access OLE DB Provider enables full access to data queues.

11.1.3 Software Prerequisites

Next, we give an overview of the products that need to be installed on every system, to make the Microsoft OLE DB Provider work. The complete network can be reduced into this figure:

![Logical Connections on a LAN Environment](image)

**Figure 155. Logical Connections on a LAN Environment**

11.1.3.1 PC-to-Host Gateway

To use the Microsoft OLE DB Provider, you need a server system as a gateway to the host. This server must have installed the following components:

- Microsoft Windows NT Server 4.0 with Service Pack 3, or later
- Microsoft SNA Server version 4.0, or later

11.1.3.2 Windows NT workstations

To use the Microsoft OLE DB provider on a workstation running Windows NT, you need:

- Microsoft Windows NT Client version for SNA Server 4.0
- Microsoft Windows NT Server version 4.0 with Service Pack 3, or Microsoft Windows NT Workstation version 4.0 with Service Pack 3
- OLE DB version 1.1, or later
If you are using an ADO consumer application, you will also need to install the ADO libraries on the workstation. The Microsoft OLE DB Provider for AS/400 and VSAM cannot be used with ADO 1.0. It requires the use of ADO 1.5 or later. In order to run ADO 1.5, you must also have the underlying OLE DB 1.5 installed.

Installing ADO 1.5 shouldn’t break down any applications that are currently using ADO 1.0. It is possible to have multiple versions of OLE DB and ADO installed and running on a single machine.

The provider is available for both Intel and Alpha Windows NT platforms.

11.1.3.3 Windows 95 workstations
To use the Microsoft OLE DB provider on a workstation running Windows 95, you need:

- Microsoft Windows 95 Client version for SNA Server 4.0
- Microsoft Windows 95
- OLE DB version 1.1, or later

When using ADO consumer applications, you need to install ADO 1.5 or later (see Section 11.1.3.2 "Windows NT workstations").

11.1.3.4 Supported Hosts
On the AS/400 system, Microsoft supports the implementation of DDM for OS/400 Version 2 Release 2 or later.

On systems running OS/390 or MVS/ESA, the driver supports version 2 release 1 (or later) of the IBM Distributed File Manager (DFM), which is a component of IBM Data Facility Storage Management Subsystem (DFSMS).

11.1.4 Configuration Issues
This section summarizes only the basic steps to get the Microsoft OLE DB Provider for AS/400 and VSAM up and running. It is possible or even likely that a real life situation will require a more sophisticated configuration than the one shown here. Please refer to the documentation that comes with the Microsoft SNA Server for more information on how configure the product.

11.1.4.1 Creating a Connection
Setting up an OLE DB data source to access AS/400 tables takes two basic steps. First of all, the provider needs a connection to the host system. This connection is provided by the server part of SNA Server. For more information on how to do this, please consult the Microsoft documentation on this topic.

After a successful configuration of the SNA Server, the SNA Server Manager application might look like Figure 156 on page 329.
11.1.4.2 Creating a new Data Source

The second step is declaring the OLE DB data source on the workstation. This process is somewhat similar to declaring ODBC data sources on a Windows system. However, there is a major difference in the way Microsoft provides an application that is used to create data sources for this particular OLE DB Provider only.

Data sources are declared using the SNA OLE DB Management application. This application is developed as a snap-in for the Microsoft Management Console. Figure 157 on page 330 shows how this application could look like when three different data sources have been declared.

The information needed to declare a particular data source is displayed in Figure 158 on page 330. When connecting to an AS/400 system, it basically comes down to specifying:

- the LU names for the host
- the LU name for the gateway system
- the APPC mode for the connection
- the RDB name of the host system (use the DSPRDBDIRE command on the AS/400 system to find this name)
- a default Library for this data source
- the Host Code Page

If you are not sure where to find these values, contact your network administrator.
Optionally, you can specify a Host Column Description file. These are external files that describe the layout of the records, including the names, the types and the sizes of the columns.
AS/400 Physical or Logical files contain all this information by default. So these description files are not required when accessing AS/400 tables. The provider will retrieve the table descriptions from the host by default.

It is possible to skip this step altogether and specify these values in the ADO connect string. Also, it is possible to connect to a predefined OLE DB data source and override some of these values.

11.1.5 Developing with the Microsoft OLE DB Provider

The purpose of this section is to point out some of the differences in programming to the Microsoft OLE DB Provider for AS/400 and VSAM, as compared to the Client Access OLE DB Provider.

Note
All of the example code shown in this chapter is available for download, see Appendix A.1, “Downloading the Samples” on page 373 for more information.

11.1.5.1 The ADO Connect String

Connecting to the Microsoft Provider is very similar to connecting via the Client Access Provider. The only exceptions being the name of the provider (SNAOLEDB) and the fact that a user id. and password are required. While IBM AS/400 Client Access gives the user the convenience of a single logon to the host system.

Creating and opening a connection to an AS/400 system in Visual Basic, would look like:

```
Private Cn400 As New ADODB.Connection
Cn400.Open "Provider=SNAOLEDB;Data Source=TcpAsm05","abc","xyz"
Cn400.CursorLocation = adUseServer
```

The first statement would create an ADO connection object. Then the second statement would open the connection using the definitions of the TcpAsm05 data source, while logging on to the AS/400 system with the user profile abc and password xyz.

11.1.5.2 Opening a Record Set

The Microsoft OLE DB Provider supports access to the host data only via dynamic record-sets and allows only for server-side cursors. The provider will return an error when trying to open a RecordsSet using a different type of cursor or a different location of the cursor. Server-side cursors are set using:

```
Cn400.CursorLocation = adUseServer
```

Opening a table using the Client Access OLE DB Provider using Record Level access is a two step process. You need to open the table, as well as an Index
object (using AD400). You need an index object only if you want to do keyed access. A typical piece of code would look like:

```vbscript
Dim cm As New ADODB.Command
Dim rs As ADODB.Recordset
Dim ix As AD400.Index
Dim nRecords as Long

Set cm.ActiveConnection = Cn400
cm.Properties("Updatability") = 0
cm.CommandText = "/QSYS.LIB/csdb.LIB/cstmr.FILE(*FIRST, *NONE)"

Set rs = cmCustomers.Execute(nRecords, , adCmdTable)
Set ix = Ix400.OpenIndex("/QSYS.LIB/csdb.LIB/CSTMR.FILE(*FIRST,*NONE)", 
"/QSYS.LIB/csdb.LIB/CSTMR.FILE(*FIRST,*NONE)")
```

The Microsoft Provider lets you open just one object, using a DDM Command String. To open a file, you need to use the “EXEC OPEN” command. A small sample of opening a file could look like:

```vbscript
rsCustomers.Open "EXEC OPEN csdb/cstmr", Cn400, _
adOpenDynamic, adLockReadOnly, adCmdText
```

On an AS/400 host, this command can be used to open Physical as well as Logical files. This may look easier at first, but it also has a few implications that you should be aware of. The Microsoft documentation states clearly that you can use the Find method and the Filter property only on Logical files. Also, when opening a Logical file, you can access only those fields that are part of the Logical file definition.

Also, note that adCmdText is the only supported value for the CommandType property that the OLE DB Provider for AS/400 and VSAM currently supports. An error will be raised if you try to specifying other value for the CommandType property.

### 11.1.5.3 Calling CL Commands

Executing a CL command is very similar to opening a Recordset object. Most likely you will create a ADODB.Command object. You then set the CommandText property of this object and use its Execute Method. The actual command text should be the DDM command EXEC COMMAND, followed by the actual CL command.

This provides for an easy way to execute a CL command, but doesn’t provide any way to return any type of output to the calling client program. The solution here is to dump all output that you need to an AS/400 file. Once the processing of the CL command has finished, the client program can open a new Recordset object to read the contents of this output file.

The Microsoft documentation doesn’t mention any possibility of calling AS/400 programs through their provider. However, using the same technique as for CL
commands, it should be possible to call a program, provided that it doesn’t require any parameters.

Finally, calling CL commands is supported on AS/400 host systems only. There is no similar functionality that allows to execute any type of command or executable code on mainframe hosts.

11.1.5.4 Navigating in a RecordSet

Navigating through a RecordSet is done using the same set of commands for any OLE DB provider. However, for our purposes the ADO `Find` method is of particular interest, since its purpose is to locate a record quickly through the index. We will also discuss the behavior of the `Filter` property in the Microsoft OLE DB Provider.

When using the Client Access OLE DB provider, doing a search for a particular index value is implemented by first using the index to get a bookmark. This bookmark is a reference to the record that we want to read. This bookmark can be used in a subsequent `Move` method.

Suppose we have a file with customer data and we have a compound index, based on customer code, district code and a warehouse number. To look up a particular customer, the code would look like:

```
Dim Keys As Variant
Dim BookMark As Variant
Dim nRecords As Long

Keys = Array(CustId, DistId, WareHouse)

BookMark = ix.GetBookmark(Keys, ad400SeekFirstEQ)
rk.Move nRecords, BookMark
```

The Microsoft OLE DB Provider takes a slightly different approach. As has been discussed before, you can open a logical file using the `RecordSet` object. The ADO 1.5 version of this object includes a new method, named `Find`. This method takes 5 parameters. The first of these parameters specifies the condition that must be satisfied by the record to be found. This condition is somewhat similar to an SQL WHERE clause, but the current Microsoft implementation has some strong limitations:

1. The expression is restricted to a single-condition where clause. This means that it can refer to only 1 field, even if the logical file is based on a compound index.

2. the provider supports only the basic comparison operators (<, >, =, <=, >=). If the logical file is based on a compound index, the equality operator is not allowed.

3. the key field can be compared to a literal value only.

If we want to use the `Find` method to implement the previous sample in Visual Basic, we need to write code that looks like:
Since the Microsoft Provider allows only one field in the WHERE clause of the `Find` method, we have to do additional testing and possibly read multiple records to obtain the record that we want.

**Note**

The Client Access OLE DB Provider doesn’t implement the `Find` method, because the provider was implemented at a time when only the ADO 1.0 specifications were available. The `Find` method of the `RecordSet` object was first introduced in the ADO 1.5 specifications.

The Microsoft OLE DB Provider for AS/400 and VSAM has another feature that allows for a more efficient implementation of the previous sample. The `Filter` property of the `RecordSet` object overcomes most of these issues. The provider is able to take advantage of indexes to implement this filtering, because it is always using a dynamic record-set.

The Microsoft documentation mentions a set of rules to have in mind when writing Filter conditions. We won’t discuss these rules in this document. If you need to learn more about them, please refer to the Microsoft SNA Server SDK documentation. You will find a complete description in this document. Following the rules of the Microsoft documentation, we could rewrite the previous sample as:

```vbscript
rs.Filter = "CID = '' & CustId & '' And CDID = '' & DistId & '' And CWID = '' & WareHouse & ''"
```

This version of the code is easy to write and understand, but it also implements a somewhat different behavior. After setting up this filter, you will only be able to move to records that satisfy the filter condition. This may be a concern in some types of programs.

Finally, when testing with this Microsoft OLE DB Driver on AS/400 tables, we discovered two restrictions that are not mentioned in the SNA Server documentation at all.
1. The driver requires all index fields to be specified in every filter condition. If you have a logical file based on three key fields, you have to specify values for each of these fields.

2. We also were unable to set the Filter if the order in which the fields appear in the filter clause didn’t match the order that these fields are defined in the logical file.

We encountered these problems when using the version of the provider that is included with the Microsoft Client for SNA Server 4.0. No updates or service packs of Microsoft SNA Server were installed. Because these limitations are not described in the Microsoft documentation, they are probably caused by some minor problem that will be fixed in a subsequent update of the SNAOLEDB Provider.

11.1.6 Comparing Performance with the Client Access OLE DB Provider

In this section, we discuss a few of the differences between the Microsoft OLE DB Provider for AS/400 and VSAM and the Client Access OLE DB Provider from a performance point of view.

11.1.6.1 Impact of a Dynamic Result Set

One of the things to keep in mind is that the Microsoft OLE DB Provider always works with dynamic result sets and server side cursors. This means that you always get up to date values as you are navigating through the data. It also means that every operation that repositions the cursor, implies a round-trip on the network.

The Client Access OLE DB Provider actually uses the same technique when doing Record-Level access. However when the Client Access Provider is used with SQL statements, it stores at least part of the result set in a buffer on the client system. Moving the record pointer within the records in this buffer won’t generate any additional network traffic.

With this in mind, it is easy to think of situations where it is possible to retrieve multiple records in an SQL statement, where the same action would require multiple seek or move operations using a record-oriented approach. In these cases, SQL code may easily outperform record-level access code simply because of the reduced need for network bandwidth.

11.1.6.2 Use of Indexes

A second significant difference between both OLE DB Providers is the way that both products take advantage of the indexes on the host. The way that the Client Access Provider implements the data access (first get a bookmark, then move to the bookmark), will generate two round trips on the network. This is probably more than what is possible with the implementation in the Microsoft OLE DB Driver. The **Find** method and the **Filter** property can both be implemented using just one round-trip on the network.

11.1.6.3 Use of Network Resources

A third factor that influences performance is that the Client Access OLE DB Provider uses connections that are defined in the IBM AS/400 Client Access product. When setting up a connection in Client Access, you can connect to an AS/400 system directly, for example by specifying its IP address. This will provide efficient routing of network packages to the AS/400 host system.
When using the Microsoft OLE DB Provider for AS/400 and VSAM, you need the Microsoft Client for SNA Server on the workstation. Next, you need a Windows NT server system that will serve as the gateway to the AS/400 host. (See Figure 155 on page 327.) This scheme implies that every round-trip to the AS/400 results into a round-trip to the NT gateway plus a round-trip from the gateway to the AS/400. In other words, there is an additional need for network bandwidth in this scenario.

11.1.6.4 Which provider is the fastest?
After reading these observations on how both products implement record-level access to the host tables, one would expect that both products would show a very similar performance behavior. After all, both have to deal with the overhead associated with dynamic result sets. And while the Client Access OLE DB Provider needs more round-trips to the AS/400, the Microsoft OLE DB Provider has some extra overhead because of the additional Windows NT server to run the SNA Server product.

Surprisingly enough, the speed tests that we ran clearly indicate a lead of the Client Access OLE DB Provider over the Microsoft OLE DB Provider. The reason is that the Microsoft Provider implements access to the host tables using generic DDM calls. The Client Access OLE DB Provider uses DDM calls that are optimized for the AS/400. In this sense, the Client Access OLE DB Provider is a trade-off. It cannot be used to access data on other systems than the AS/400, but this restriction allows for better performance than a generic implementation.

11.1.7 Summary
The feature sets of both the Client Access OLE DB Provider (IBMDA400) and the Microsoft OLE DB Provider for AS/400 (SNAOLEDB) and VSAM are summarized in the table below:

<table>
<thead>
<tr>
<th>Feature</th>
<th>IBMDA400</th>
<th>SNAOLEDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record Level Access</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL Access</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>VSAM Access</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Call CL Commands</td>
<td>Yes</td>
<td>Calling CL commands is supported only on AS/400, not on mainframe hosts.</td>
</tr>
<tr>
<td>Call Stored Procedures</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Use Existing Programs</td>
<td>Yes</td>
<td>No. However, calling an existing AS/400 program that has no parameters at all should work without any problems. Please note that this possibility is not described in the Microsoft documentation.</td>
</tr>
<tr>
<td>Use Data Queues</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
11.2 Client Access OLE DB Provider

This section gives an overview of how to use the Client Access OLE DB Provider in a three tier environment.

11.2.1 Overview

Using the Client Access OLE DB Provider in a middle tier is very similar to using the provider in a two-tier. This shouldn’t be too surprising: a middle tier uses the same ADO API definitions as a two-tier client application. The first topic in this section is on the configuration of Client Access. The next topic is on the changes in programming style. Finally, we show how we can split up an existing application into different tiers.

Warning

The version of the OLE DB Driver that is shipping at the time of this writing (as a part of IBM Client Access AS/400, Version 3 Release 1 Modification level 3) does not support access to the AS/400 database as described in this section. It does not allow the UserId and Password to be passed in.

The information in this chapter was based on our experience with a beta version of Client Access. This update will become generally available with a future version of the IBM Client Access AS/400 product.

The sample code that is discussed in this section, is not currently available on the redbooks web site. When the new version of Client Access is available, the 3 tier examples will be added to the download site.

11.2.2 Configuration Issues

There are a few considerations that Microsoft Windows NT imposes on any software package when running as a middle tier. Technically, this software runs under control of a Windows NT Service. The major change is that a service runs in a different type of environment and has no access to the settings of the user who is using this middle tier.

With these observations in mind, we should also realize that the IBM AS/400 Client Access for Windows NT allows the user to declare connections to AS/400 systems on a per-user basis. The Client Access OLE DB Provider relies on these connections to establish a session with the AS/400 server.

To enable Windows NT Services to use the Client Access OLE DB Provider, we need a way to upgrade the definitions for the connections to make them available for the services. Client Access has an administrative program, called CWBCFG.EXE, to perform all the required settings.

You can find the program in the Client Access directories. Typically, you can find this program in the C:\Program Files\IBM\Client Access folder. This program can be started from a Windows NT Command Prompt. It moves the Client Access Connection information so that it becomes available to service programs. This process is very similar to defining a System Datasource when using the ODBC Data Source Administrator.
Figure 159 shows the typical output of the CWBCFG.EXE command. When using this command, you will add a system definition for a connection, called TCPASM05. The /s option tells the program to define this connection for the system account. Please note that this flag is mandatory when using the Client Access Provider in three tier mode.

In our example in Figure 159, the IP address is not specified. In this case the CWBCFG program looks for either a domain name server (DNS) or a host table entry to obtain the IP address.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/host &lt;name&gt;</td>
<td>Designates the AS/400 connection name.</td>
</tr>
<tr>
<td>/ipaddr &lt;address&gt;</td>
<td>Designates the IP address.</td>
</tr>
<tr>
<td>/view &lt;x&gt;</td>
<td>Designates the Default View in AS/400 Connections. The options are: Connect, 5250, GA. Connect is the default option.</td>
</tr>
<tr>
<td>/uid &lt;user id&gt;</td>
<td>Designates the Default User ID.</td>
</tr>
<tr>
<td>/r</td>
<td>Replace an existing configuration.</td>
</tr>
<tr>
<td>/s</td>
<td>Set the configuration for the NT System Account.</td>
</tr>
<tr>
<td>/?</td>
<td>Displays this list of options.</td>
</tr>
</tbody>
</table>

There are a few things to keep in mind when using the CWBCFG.EXE command:

1. The command works only for connections through the TCP/IP protocols. Other types of connections are not supported.
2. Do not specify the IP address if you have a DNS server in your network.
3. This option modifies the settings for the default user and the system account in the NT Registry. To execute this command successfully, you need sufficient
access rights. Run this command when using a NT account that is a member of the Administrators group.

4. This command works on Windows NT, versions 4.0 and later.

5. This command copies information of configurations in the primary environment only.

6. If you want to use either the Client Access OLE DB Provider for AS/400 or the Client Access ODBC Driver in a Windows NT Service program, you must specify the /s parameter.

7. If there is no managing system defined for your Client Access configuration, the command defines this hostname as the managing system. This setting can be updated through the AS/400 Connections application.

11.2.3 Programming in a Three-Tier Environment

As we already pointed out in the overview of this section, programming for the Client Access OLE DB Provider in a Business object isn’t really any different from programming the provider in a traditional two-tiered client/server application.

11.2.3.1 Logon Information

When using the Client Access OLE DB Provider in a two-tiered approach, the provider allows the user to use Client Access connections. The user doesn’t have to retype a user ID and password. Client Access knows that it can trust the user on this workstation.

When we move on to a three-tier scheme, the code is running on an application server, somewhere on the network. The Client Access OLE DB Provider requires that Client Access be installed on this application server. But now, we don’t want to connect to the AS/400 Host using the logon ID and authorities for the user that is logged on the server. In this scenario, we are required to specify a user id and password for the connection. In fact, that is all we need to change to our code.

According to the Technical Reference in the Client Access AS/400 SDK for ActiveX and OLE DB, creating a connection in a two-tier environment looks like this:

```plaintext
Cn400.Open "Provider=IBMDA400;Data Source=TCPASM05;", ",", ""
```

When writing code for a middle tier object, this code becomes:

```plaintext
Private Const sUserId As String = "A980502E"
Private Const sPassword As String = "light01a"

Cn400.Open "Provider=IBMDA400;Data Source=TCPASM05;", 
            sUserId, _
            sPassword
```
11.2.3.2 Optimize Network Traffic
Developing a program using the three-tier approach doesn’t necessarily mean that you need to implement the three layers on the network.

A three-tier approach can also be used to split up a large complicated programming project into multiple smaller projects. Each of these small projects will be much easier to understand and debug than a big monolithic project. It also becomes easier to manage a group of programmers working together on the same project.

However, in some cases it may make sense to write middle tier objects that run on a dedicated application server. If you plan on writing these kinds of objects, it becomes very important to think about the number of calls that the client tier will make to the middle tier or the business objects. Each call to the middle tier will suffer from a performance hit, because it needs a round-trip on the network.

11.2.4 Changing an Existing Application
In this section, we take a look at the Speed application using Record Level Access, written in Microsoft Visual Basic 5.0. See Chapter 13, “Performance Measurements and Comparisons” on page 361, for a description of the Speed application. We show how this application can be broken up into a middle tier and a client tier. We also show how the middle tier can be moved to a remote machine. We do this while preserving the full functionality of the original application.

11.2.4.1 Stripping Down the Client Tier
When we look at the Project Explorer for the two-tier speed application, we can see that there are two class modules defined. A closer study of the code in this application proves that both modules implement the same set of functions. Actually they provide database access functionality for both the Client Access OLE DB Provider for AS/400 (the LightningLinks module) and for the Microsoft OLE DB Provider (the ThorLinks module).

![Figure 160. Project Explorer for the Two-Tier Application](image)

These modules are set up in a way that they contain every single operation that performs direct database access. They were introduced in this application to
support access through both providers. Selecting a particular provider is easy when using these class modules.

This manner of splitting up the code is now very handy to migrate the project toward a three tiered approach. Actually, this design of the code makes the first step the easiest. In this project, we can just remove these two modules from the project. The Project Explorer now looks like:

![Project Explorer](image)

We need to do some more work on the client tier. But before doing that, we will create the **business** object first.

### 11.2.4.2 Choosing an implementation for the Middle Tier

The first thing to decide is how the client tier and the middle tier can communicate. Until now, this was a tough decision. Since we are concentrating on Windows applications, the natural choice is to take advantage of ActiveX and COM. These technologies provide two major ways to package objects: in an executable file (EXE) or a dynamic-link library (DLL).

The choice between those two options is generally determined by the network topology that you want to implement. If your goal is to write a project using the three-tier paradigm, but run the code on two hardware tiers (PC and AS/400), choose a DLL implementation of your business code. A DLL implementation offers a much more efficient communication between the different tiers. Using an EXE server increases not only the communication overhead, but also requires more system resources. There are some advantages too: for example the client application can catch fatal errors in the server (like GPFs) much easier. And, a 32-bit EXE server can be used in 16-bit client applications.

On the other hand, if you want to off-load the middle tier to an application server, you can write an EXE server module. The exception here is that you may run the **business** object code under control of the Microsoft Transaction Manager (MTS). When using MTS, you may prefer to write a DLL anyway.

These considerations may sound like a serious design issue. But this decision is made easy by the fact that the client code that uses the **ActiveX** object is completely independent from which option you choose. Also, most development environments make it easy to change the implementation from a DLL to a EXE server or the other way around.

In this section, we use the EXE server implementation. This will help us to off-load the **business** object onto a remote machine. We will not include MTS in this picture. The purpose of this section is to show how an application using the
Client Access OLE DB Provider can be split into a three-tier implementation. If you want to run the middle tier under control of MTS, please refer to the Microsoft documentation of the Transaction Server.

11.2.4.3 Creating the Middle Tier
Because we are doing this project in Visual Basic 5.0, we select the File/New option from the menu and select for an ActiveX EXE project. This is shown in Figure 162.

![New Project](Image)

**New Project**

This creates an empty project. We will call this project MiddleTierExe. To set the project name:

1. Go to the Project/Properties option in the menu. You will get a screen like shown in Figure 163.
2. Change the Project Name to "MiddleTierExe". All other options can be left to their default value.

Programmers who do this type of project for the first time may find it strange that the Startup Code is set to (none). This means that only code from the VB run-time is used when the program is run. This code implements the behavior that is required by the COM and ActiveX specifications. So we don’t have to worry about this.

If the program is launched from the command prompt, it returns immediately without any interaction with the user. But under the covers, it checks the registry setting for this application and ensures that everything is properly set up.

If the need arises to undo the setting for an ActiveX EXE server, just launch the program and add the `/UnRegServer` option on the command prompt. Not specifying anything is the same as using the `/RegServer` command option.

If you selected an ActiveX DLL in the previous step, you also need to set the proper settings for the objects. Since you cannot launch a DLL from a command
prompt, you need to use the REGSVR32.EXE program to perform the registration. Setting the proper registry values for an ActiveX DLL called MiddleTier.DLL, is done by typing:

REGSVR32 MiddleTier.DLL

When using Microsoft Visual Basic 5, you don’t need to type these commands when developing the software. The registration will be done for you, each time you build or rebuild the EXE or DLL.

Figure 163. Project Properties of the MiddleTier Object

The next step is to go to the Program Explorer and include the two class modules we deleted from the client tier. To do so, right-click on the project in the Program Explorer and select Add/File from the pop-up menu. After adding the files, the screen looks like:

Figure 164. Project Explorer for the MiddleTier Object
The class modules were written originally to be self-contained units of functionality. Thanks to this design, the code needs almost no change at all. The only exception here is that we need to specify a user ID and password for the Client Access OLE DB Provider, as explained in Section 11.2.3.1 "Logon Information". Double-click on the LightningLinks class module. This opens the editor for the source. On top of the source are constant definitions for the AS/400 connection, the user ID and password. These must be changed into the appropriate values. An example is:

```vbnet
Private Const sSystemName As String = "TCPASM05"
Private Const sUserId As String = "me"
Private Const sPassword As String = "mypassword"
```

Now, we need to indicate that the objects that are implemented in this class module, can be created by external processes, like our own client tier. Again, this is done by a simple setting in the development environment. From the Properties window, change the Instancing property from Private to MultiUse. This enables client processes to create this object directly.

The naming convention for this object is the project name, a dot, and the class name. This object name is sometimes called the PROGID of the ActiveX Object. In our sample project, the project is called MiddleTierExe and the class module for the Client Access OLE DB Provider is called LightningLinks. So, the ActiveX name (or PROGID) of the object is MiddleTierExe.LightningLinks.

The Properties Window should look like Figure 165 for the class modules that uses the IB: OLE DB Provider. The second class modules needs exactly the same value for the Instancing property.

![Figure 165. Properties Window for the LightningLinks Class Module](image)

Before we can create the EXE server, we have one last step to take. We must provide the references to the objects that are being used in the code. This is done on the Visual Basic 5 menu, select the Project/References option. The list of referenced objects should include the ADO 1.5 declarations and library for Record Level access through the Client Access OLE DB Provider.
When these objects are included, the dialog looks like Figure 166:

![Figure 166. Referenced Objects for the MiddleTier Object](image)

Finally, we can create the EXE now. This is done by selecting the File/Make MiddleTier.Exe option on the menu. As explained before, this creates the executable file and sets all necessary registry settings to utilize the objects in the ActiveX server.

**11.2.4.4 Back to the Client Tier**

Now that the middle tier has been finished, we can turn back to the client tier. We need to do just a few additional modifications here.

First, we have to make the client tier aware of the existence of the middle tier. This is done once more by the Project/References option in your project. Here we need to include a reference to our middle tier objects. The project name of the middle tier object should be available in the list. We know that the objects are registered on our system and that is why the Visual Basic Environment is now aware of their existence.

At the same time, we can remove all references to the ADO libraries and to the Client Access OLE DB Provider. We needed these in the two-tier version of the program. But in the our three-tiered version, all data access is done through the middle tier, so there is no reason for these references anymore. Please note that this is not a necessity. It is perfectly possible to create an application that mixes the two-tier and three-tier approaches.

After changing the settings, we see a screen like Figure 167 on page 346. The settings that are left are the default Visual Basic settings, the controls that are being used in the user interface and the middle tier object.
The second change is just a minor code change. This is because the user can select between the Client Access OLE DB Provider and the Microsoft OLE DB Provider while the application is running. The speed test then runs using this provider.

In the code, this is implemented by creating an instance of one of the two class modules. The class module LightningLinks implements access through the Client Access OLE DB Provider. And the class module ThorLinks does the same for the Microsoft OLE DB Provider. Both modules expose exactly the same functionality but have a different implementation.

The code to process the transaction uses a reference to an object and is completely unaware which provider is actually doing the processing. This code is even unaware if it is interacting with an instance of a class module in the project or with an ActiveX object. We will take advantage of this feature to get a three-tier version of the speed application.

In the original two-tier project, an instance of a class module is created using the following code:

```
Set L_Links = New LightningLinks
```

The only change required here is to change the name of the old class module into the ActiveX object name (or PROGID) for the object we want to create. So this line becomes:

```
Set L_Links = New MiddleTierExe.LightningLinks
```
In the project, there are two subroutines that need to be changed. They are called SelectLightning and SelectThor. The complete code of the SelectLightning subroutine is shown here. The changes in this function are highlighted.

```vba
Public Sub SelectLightning()
    Dim L_Links As MiddleTierExe.LightningLinks
    If Trim(LinkName()) <> "Lightning" Then
        ' Create the Connection
        Call SetStatusPanel(ExecStatusConnecting)
        Set G_Links = Nothing
        Set L_Links = New MiddleTierExe.LightningLinks
        Call SetStatusPanel(ExecStatusReady)
        ' Select Lightning as the OLE-DB provider for the test
        Set G_Links = L_Links
        LinkName "Lightning"
    End If
End Sub
```

The function SelectThor needs to be changed in exactly the same way.

After changing these two functions, the speed application is ready to run in a three-tier approach.

11.2.4.5 Running on a Single Machine
Running the application is just a matter of starting up the client tier executable now. The OLE registrations that are made on the client system start up the middle tier.

You can make sure that the middle tier is actually running by using a tool that shows the list of active processes. For example, when using Windows NT Workstation, you can pop up the Task Manager.

A speed comparison of the original two-tier version and this new version shows very little or no difference at all. Basically, we are running the same code, but only packaged in a different format.

11.2.4.6 Running the Middle Tier on a Remote Server
The three-tier project becomes more interesting when we move the EXE to another machine. In this sample we will be using DCOM (Distributed COM) as the transport mechanism between the application and the middle tier.

DCOM is installed as a standard component for Windows NT 4.0 and later. If you wish to run the components on Windows 95, you need to install the DCOM for
Windows 95. If needed, you can download this from the COM section of the Microsoft web-site (http://www.microsoft.com/com).

Step number 1 is to move the executable file to a directory on the application server. This ActiveX components needs to be registered on the server PC as well as on the client. As explained previously, registering is simple: run the executable file on a DOS Command Prompt. The argument /RegServer is optional.

To avoid security issues, we recommend to set security to the lowest level for this test. To set this on the server, start the DCOMCNFG.EXE program. Select the second tab, as shown in Figure 168:

![Distributed COM Configuration Properties](image)

**Figure 168. Set the Default DCOM Security with DCOMCNFG.EXE**

Set the authentication level to none and the impersonation level to Anonymous. This will allow any client workstation to create this object. In this sample we will not go any further into the security features of DCOM.

Now we return to the workstation. On the client machine, we have to run the same utility, DCOMCNFG.EXE. Now we need the first tab, which is a list of object of ActiveX EXE servers that are registered on this workstation. We need to locate the EXE server in this list and edit its Properties. A screen with a typical list of objects is shown in Figure 169.
Three-Tier Approach

The second tab of the properties screen is shown in Figure 170 on page 350. Here we can determine what happens when the object is created. The option "Run Application on the Following Computer" should be checked. The edit box must be filled with the network name of the application server.

When the client program now creates an instance this object, it will get a reference to a system object that imitates the behavior of the real object. The COM documentation calls this a proxy object. This proxy starts a communication over the network and create the object on the application server. Again, the client application is completely unaware of this.

Finally, it is advised to set the security settings on the workstation to the lowest possible level too. The screen is exactly the same as on the application server (see Figure 168 on page 348).

If all settings are done correctly, the speed application runs in full three-tier mode. When you look at the test results, the extra communication overhead of DCOM is noticeable. In the setup we made, the average time needed for a transaction was increased by about 1.5 seconds.
11.2.5 Summary

In this chapter we have shown how the Client Access OLE DB Provider can be used in a three-tier environment. Actually, there is no real difference between programming ADO for two-tier and three-tier environments. Almost all of the issues have to do with system setup or settings.

We also showed how an application can be broken up into components and how a component can be moved to an application server. This sample made clear that a three-tier aware design in the case of the two-tier sample helped tremendously in the migration process.

Any database application can be implemented using a three-tier project structure, even if the project is intended to run in a two-tier setup. This design decision pays off in reduced maintenance costs and provides more flexibility in the future.
Chapter 12. Security Considerations

In this chapter, we deal with some of the security issues that an ADO programmer, or an end-user running the ADO application, is likely to encounter. Note that IBM AS/400 OLE DB support does not introduce any new security risks.

For in-depth reviews on OS/400 Security and implementation, review the following IBM manuals:

- SC41-3740 - OS/400 Server Concept and Administration
- SC41-3302 - AS/400 Security Reference
- SC41-3307 - Distributed Data Management
- GG24-4200 - Implementation Guide for Security and Auditing

12.1 Overview

The Client Access OLE DB Provider makes accessing AS/400 resources such as DB/2 database files, data queues and programs very easy. If proper security measures are not in place, unauthorized individuals may gain access to confidential information residing on your AS/400 system. It is now more important than ever to ensure that your system resources are protected against such threats.

Security can be

- Resource Centric
- Access Method Centric

In a real-life scenario, a combination of these two methods is implemented.

12.1.1 Resource Centric Security

Resource centric security warrants that security is specified at the resource level. Each AS/400 resource is an object. The objects somehow "know" who can access and manipulate them, and who should be denied even the knowledge of their existence. This also known as "Object Level" security.

Object Level security allows you to define who can use objects and how those objects can be used. The ability to access an object is called authority. You can specify detailed authorities, such as adding records or changing records, or you can use the system-defined subsets of authorities: *ALL, *CHANGE, *USE, and *EXCLUDE.

Files, programs, and libraries are the most common objects requiring security protection, but you can specify authority for any object on the system.

You define who can use an object in several ways:

Public Authority:

The public consists of anyone who is authorized to sign on to your system. Public authority is defined for every object on the system, although the public authority for an object may be *EXCLUDE. Public authority to an object is used if no other specific authority is found for the object.
**Private Authority:**
You can define specific authority to use (or not use) an object. You can grant authority to an individual user profile or to a group profile. An object has private authority if any authority other than public authority, object ownership, or primary group authority is defined for the object.

**User Authority:**
Individual user profiles may be given authority to use objects on the system. This is one type of private authority.

**Group Authority:**
Group profiles may be given authority to use objects on the system. A member of the group gets the group’s authority unless an authority is specifically defined for that user. Group authority is also considered private authority.

**Object Ownership:**
Every object on the system has an owner. The owner has *ALL authority to the object by default. However, the owner’s authority to the object can be changed or removed. The owner’s authority to the object is not considered private authority.

**Primary Group Authority:**
You can specify a primary group for an object and the authority the primary group has to the object. Primary group authority is stored with the object and may provide better performance than private authority granted to a group profile. Only a user profile with a group identification number (gid) may be the primary group for an object. Primary group authority is not considered private authority.

**Authority scope**
Anytime an object is created, the operating system defines object-level authority for it. Most of the time, this is transparent to the user that is creating the object. What authority this sets up depends on a number of factors, such as the authority of the user creating the object and the library or IFS directory where the object is being placed. The same level of authority remains on an object as long as it is not explicitly changed or the object is removed from the system.

**Adopted authority**
Adopted authority adds the authority of a program owner to the authority of the user running the program. Adopted authority is a useful tool when a user needs different authority for an object than what is the default.

Sometimes a user needs different authorities to an object or an application, depending on the situation. For example, a user may be allowed to change the information in a customer file when using application programs providing that function. However, the same user should be allowed to view, but not change, customer information when using a decision support tool, such as Microsoft Excel.

A solution to this situation is 1) give the user *USE authority to customer information to allow querying the files and 2) use adopted authority in the customer maintenance programs to allow the user to change the files.

When an object uses the owner’s authority, this is called adopted authority. Objects of type *PGM, *SRVPGM, and *SQLPKG can adopt authority.
When you create a program, you specify a user profile (USRPRF) parameter on the CRTxxxPGM command. This parameter determines whether the program uses the authority of the owner of the program in addition to the authority of the user running the program. For additional information on adopted authority, refer to the AS/400 Security Reference, SC41-3302.

### 12.1.2 Access Centric Security

Access centric security emphasizes securing access to the resources rather than securing the resources themselves. This method implies that you can manipulate a resource (database files, data queue, etc.) if you can access it. If you are implementing access centric security, you should be aware that as new or alternate methods of accessing the resources are available, your system resources becomes vulnerable. Because in this scenario resources or objects themselves do not have any useful knowledge of a user or a class of user, most of the objects show that *PUBLIC has *ALL authority.

The Limit Capabilities (LMTCPB) parameter on the user profile is a prime example of an access centric security implementation. When this option is set, the user profile does not have access to the command line, and the user is not allowed to enter commands.

In this section, we discuss managing access to your AS/400 resources through Client Access. If you enter DSPNETA on the AS/400 command line, you get a display like that shown in Figure 171.

![Figure 171. DSPNETA Command Output](image)

Of all the network attributes displayed above, two, DDM Request Access (DDMACC) and Client Request Access (PCSACC), can be used to control access to your AS/400 resources from Client Access-connected PCs.

The DDMACC network attribute determines how the system processes requests from other systems to access data using the distributed data management (DDM) or the distributed relational database (DRDA) function. Record level access is
one of the functions supported by the AS/400 OLE DB Provider. It uses DDM to access data on the AS/400 system.

Table 49. DDMACC Network Attribute

<table>
<thead>
<tr>
<th>DDMACC Network Attribute Value</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>*REJECT</td>
<td>The system does not allow any DDM or DRDA requests from Client Access PCs. Note: This affects all Client Access PCs, including Windows 95/NT.</td>
</tr>
<tr>
<td>*OBJAUT</td>
<td>Remote requests are controlled by the object authority on the system.</td>
</tr>
</tbody>
</table>

Qualified program name (library/pgm) This user-written exit program is called after normal object authority has been verified. The exit program is called only for DDM files, not for distributed relational database functions. The exit program is passed a parameter list built by the remote system that identifies the local system user and the request. The program evaluates the request and sends a return code, granting or denying the requested access.

The PCSACC network attribute determines how the Client Access licensed program processes requests from attached personal computers to access AS/400 objects. The PCSACC network attribute controls whether personal computer jobs can access objects on the AS/400 system, not whether the personal computer can use workstation emulation. Any function that uses an Optimized Server does not use the PCSACC network attribute to determine if a User Exit program should be invoked. We cover User Exit programs for the Optimized Servers in the next section.

Table 50. PCSACC Network Attribute

<table>
<thead>
<tr>
<th>PCSACC Network Attribute Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*REJECT</td>
<td>Rejects every request from the personal computer running Client Access to access objects on the AS/400 system. An error message is sent to the PC application.</td>
</tr>
<tr>
<td>*OBJAUT</td>
<td>The Client Access programs on the system verify normal object authorities for any object requested by a PC program. For example, if file transfer is requested, authority to copy data from the database file is checked.</td>
</tr>
<tr>
<td>*REGFAC</td>
<td>The system uses the system's registration facility to determine which exit program (if any) to run. If no exit program is defined for an exit point and this value is specified, *OBJAUT is used.</td>
</tr>
</tbody>
</table>

Qualified program name (library/program) Only used by Client Access 16-bit applications that use an "Original server". It does not concern us. Since Client Access for Windows 95/NT is 32-bit and does not use "Original Servers".
**User Exit Program**

User exit programs give you options to secure access at the Host Server level. Most AS/400 client/server jobs interface with a Host Server on the AS/400 system. The exception would be when you are using APPC or sockets interfaces and providing your own host code. After a client request is passed to the Host server, the host server must check the associated exit point before running the request. In a user exit program, you can name a program that provides additional security or processing support. For example, from an ADO application, you can send SQL statements to the Host Database server. You can write exit programs to determine whether these SQL requests should be processed. A user may have object level access to a database file, but by using an user exit program, you can still deny them certain capabilities. For example, you might want to restrict access to the PARTS database in such a way that Update operations are allowed, but not Insert and Delete operations.

For user exit programs, we can categorize all functions supported by the Client Access OLE DB Provider as:

- DDM access
- Non-DDM access

Record level access to AS/400 database files uses DDM to access the AS/400 system. The remainder of the functions (SQL statements, stored procedure calls, data queues, commands, and programs) supported by the Client Access OLE DB Provider do not use DDM.

For DDM exit programs, you must use the DDMACC network attribute. You can modify this network attribute by using the following command:

```
CHGNETA DDMACC(MYLIB/MYPGM)
```

Here, **MYLIB/MYPGM** is the user exit program that manages record level access to your AS/400 database files.

You must use the OS/400 Registration Facility to define exit programs for the non-DDM based Client Access OLE DB functions. The `WRKREGINF` command is used to start the Work With Registration Information window on the AS/400 system. You then choose option 8 to work with exit program followed by option 1 to Add the exit program.

The following table summarizes all the functions supported by the Client Access OLE DB Provider, the exit points where the exit programs can be registered, and when a particular exit program is invoked:

<table>
<thead>
<tr>
<th>OLE DB functions</th>
<th>Exit Point</th>
<th>When Invoked</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Statements/ SP</td>
<td>QIBM_QZDA_INIT</td>
<td>Database server initiation.</td>
</tr>
<tr>
<td></td>
<td>QIBM_QZDA_NDB1</td>
<td>Called for Native database requests.</td>
</tr>
<tr>
<td></td>
<td>QIBM_QZDA_SQL1</td>
<td>Called for SQL requests. SELECT, INSERT, UPDATE, DELETE, requests are passed through this exit point.</td>
</tr>
</tbody>
</table>
Below is a sample RPG user exit program. This program needs to be registered on only these two exit points using the OS/400 Registration Facility:

- QIBM_QZDA_INIT
- QIBM_QZDA_SQL1

<table>
<thead>
<tr>
<th>OLE DB functions</th>
<th>Exit Point</th>
<th>When Invoked</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>QIBM_QZDA_ROI1</td>
<td>Called for retrieving catalog information. Mostly used by the Visual Basic Wizard in the SDK to retrieve information on library, file, field, and so on.</td>
</tr>
<tr>
<td>Data Queue functions</td>
<td>QIBM_QZHQ_DATA_QUEU E</td>
<td>For all data queue operations. This includes data queue open, read, write, and so on.</td>
</tr>
<tr>
<td>Command / Programs</td>
<td>QIBM_QZRC_RMT</td>
<td>For all command and program calls.</td>
</tr>
</tbody>
</table>

```
FREPORT O F 600 DISK
I* HEADER INFORMATION
IPCSDTA DS 4171
I 1 10 USERID
I 11 20 SRVID
I 29 32 FUNCID
I 96 351 SQL1
I 352 607 SQL2
I*--------------------------------------------------------------
I* ADDLIB APILIB C CHQRY
I*I 0 FOUND
C*--------------------------------------------------------------
C* MAIN PROGRAM
* C*--------------------------------------------------------------
C* ENTRY PLIST
C PARM RTNCD 1
C PARM PCSDTA
C* PROCESS BASED ON SERVER ID
C* C TIME TIME6 60
```
*--------------------------------------------------------------
*  DATAQUEUE SERVER
*--------------------------------------------------------------
DQ BEGINSR
  DATA QUEUE SPECIFIC CODE GOES HERE
ENDSR

*--------------------------------------------------------------
*  DATABASE INIT
*  WE ARE GOING TO PREVENT ANY REQUESTS FROM THIS USER
*--------------------------------------------------------------
ZDINIT BEGINSR
  USERID IF EQ 'USER1'
    MOVE '0' RTNCD
  ENDF
ENDSR

*--------------------------------------------------------------
*  SQL PROCESSING
*  WE ARE GOING TO PREVENT ANY UPDATE OR DELETE REQUESTS
*  FROM THIS USER
*--------------------------------------------------------------
SQL BEGINSR
  USERID IF EQ 'USER2'
    MOVE 0 FOUND 10
  SQLUP1 CHECK 'UPDATE' FOUND
    IF EQ 0 MATCH FOUND
      MOVE '0' RTNCD REJECT REQUEST
    ENDF
  ENDF
  EXCEPT
  ENDSR
This exit program generates an output file for every SQL request that it receives. The output file name is Report, and the record length is 600 bytes.

An exit program has two input parameters: a 1-byte return code and a data structure containing the request from the client. The typical call flow of an exit program involves examining the data structure that contains the client request. The data structure contains the user ID of the user sending the request, the AS/400 server the request is intended for, and additional data about the request, such as the SQL statement to be executed. You can examine the data structure and reset the 1-byte return code to allow or deny the request. Setting the return code to "0" denies the request, while setting it to "1" allows the request.
The following table summarizes all exit points and the server name that are used by OLE DB.

Table 52. Summary OLE DB Functions by Exit Point and Server Name

<table>
<thead>
<tr>
<th>OLE DB functions</th>
<th>Exit Point</th>
<th>Server Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL/SP</td>
<td>QIBM_QZDA_INIT</td>
<td>*SQL</td>
</tr>
<tr>
<td>SQL/SP</td>
<td>QIBM_QZDA_NDB1</td>
<td>*NDB</td>
</tr>
<tr>
<td>SQL/SP</td>
<td>QIBM_QZDA_SQL1</td>
<td>*SQLSRV</td>
</tr>
<tr>
<td>SQL/SP</td>
<td>QIBM_QZDA_ROI1</td>
<td>*RTVOBJ</td>
</tr>
<tr>
<td>SQL/SP</td>
<td>QIBM_QZDA_SQL2</td>
<td>*SQLSRV</td>
</tr>
<tr>
<td>Data Queue functions</td>
<td>QIBM_QZHQ_DATA_QUEUE</td>
<td>*DQSRV</td>
</tr>
<tr>
<td>Command / Programs</td>
<td>QIBM_QZRC_RMT</td>
<td>*RMTSRV</td>
</tr>
</tbody>
</table>

In the sample user exit program, we have six subroutines that we use for determining how a request should be handled:

- DQ - For Data Queue functions
- ZDAINIT - For database server initialization requests
- NDB - For native database requests.
- SQL - For SQL requests
- ROI - For catalog lookups
- DPC - For command and program calls

During database server initialization, this program checks if the user ID requesting server initialization is USER1. Any time a user sends a SQL statement or stored procedure, the database server must be initialized. If the requesting user ID is not USER1, we allow the server initialization job to proceed; otherwise the request is denied. This is handled by the ZDAINIT subroutine.

Once the database server job is initialized for any SQL requests, this program checks if user USER2 is making an Update request. If an UPDATE request is found for USER2, that request is denied. This is handled by the SQL subroutine.

Note that in this program we have not implemented any code/logic in the DQ, NDB, ROI, and DPC subroutines. Thus, all of these requests are automatically accepted by the OS/400 Host Server. You can modify this program to add the necessary code in these subroutines. For additional information on this, refer to the OS/400 Server Concept and Administration guide, SC41-3740.

12.1.2.1 Summary

Exit programs place additional processing load on the server. Consider using it only when object level security cannot be implemented. Object level security is fast and cannot be circumvented.

A final point of caution is that there may be other Client Access functions that use the same exit point that is being monitored by a user exit program. For instance, the QIBM_QZDA_INIT exit point is used by Client Access Data Transfer, ODBC, or any program that uses the Client Access for Windows 95/NT Optimized SQL API. Since all these programs use the optimized database server, which checks this exit point before the database server initialization, any client/server
application using these interfaces will be affected by a program registered at this exit point.
Chapter 13. Performance Measurements and Comparisons

In this chapter, we cover the performance considerations when using the Client Access OLE DB Provider (IBMDA400). Use this chapter to help determine the optimum access method for your application based on performance, productivity, maintainability, and other considerations. This chapter also provides tips to optimize your OLE DB code.

In 13.2, “Benchmark Results (Speed Application)” on page 361, a standard IBM client/server benchmark ("Speed") is described and used to show where the Client Access OLE DB Provider method currently compares with other approaches in terms of response time and scalability (AS/400 system loading).

13.3, “Function Timings” on page 368, provides measured timings for specific Client Access OLE DB functional operations (Open, Open Link, Prepare, Move Next, Select/Bookmark/Move to Record) for the purpose of providing actual numbers that you might use to extrapolate for your application. In addition, these timings are compared to the corresponding ODBC API timings.

13.4, “Tips and Techniques” on page 369, is a list of specific do’s and don’ts when using the Client Access OLE DB provider.

Finally, 13.5, “Summary - Usage and Trade-Offs” on page 370, summarizes the performance measurements and provides rationale for your selection of the OLE DB method to access the AS/400 system.

13.1 Measurement Environment

All of the performance results of this chapter were obtained using the following hardware and software configurations:

- IBM AS/400 Model 510, 256 MB
- IBM AS/400 OS V04R02M00
- IBM Client Access V3R1M03
- Token-ring speed 16 MB
- TCP/IP
- IBM PC 266 MHz 128 MB

13.2 Benchmark Results (Speed Application)

This section provides comparison performance data of Client Access OLE DB with two other data access approaches. The access techniques used are:

1. Client Access OLE DB - DDM record level access
2. Client Access OLE DB - SQL
3. Client Access ODBC APIs
4. AS/400 Stored Procedure
13.2.1 Benchmark Overview

In this chapter, we use an AS/400 client/server order entry application (called Speed), which is based on the primary application of the CPW (Commercial Processing Workload) benchmark set of applications. This CPW benchmark includes interactive and batch work and is used by the Rochester Development Lab to determine performance ratings of the various AS/400 processor features.

The CPW benchmark itself is a modified implementation of the TPC-C (Transaction Processing Commercial-benchmark C) workload. Since CPW is not the actual TCP-C benchmark, performance metrics based upon CPW applications are NOT representative of IBM’s or other vendor’s optimized implementations. However, CPW is representative of sophisticated commercial applications and industry standard benchmarks and is more complex than the original AS/400 performance rating RAMP-C benchmark, which was used in AS/400 releases prior to November 1996.

The client/server order entry application used in this benchmark does only the order entry CPW application and uses the CPW database. The primary server functions done on the AS/400 system are database accesses, and the primary client requests use SQL interfaces and OLE DB DDM; however, other client interfaces, such as APPC program to program and Client Access data queue interfaces are alternative choices. In the non-SQL cases, AS/400 “native database operations” (for example, READ record and WRITE record rather than SQL SELECT and SQL INSERT operations) are used.

13.2.2 Application Overview

This section provides an overview of the application and a description of how the application database is used.

13.2.2.1 The Company

The Company is a wholesale supplier with one warehouse and 10 sales districts. Each district serves 3000 customers (30000 total customers for the Company). The warehouse maintains stock for the 100000 items sold by the Company.

13.2.2.2 The Database

The Company runs its business with a database. This database is used in a mission-critical, OLTP (on-line transaction processing) environment. The database represents a company with:

1. 1 warehouse
2. 10 districts
3. 30,000 customers
4. 100,000 items to sell

The database includes tables with the following data:

1. District information (next available order number, tax rate, and so on)
2. Customer information (name, address, telephone number, and so on)
3. Order information (date, time, shipper, and so on)
4. Order-line information (quantity, delivery date, and so on)
5. Item information (name, price, item ID, and so on)
6. Stock information (quantity in stock, warehouse ID, and so on)

### 13.2.2.3 A Customer Transaction

1. Customers telephone one of the 10 district centers to place an order.

2. The district customer service representative answers the telephone, gets the following information, and enters it into the application:
   1. Customer number
   2. Item Numbers of the items the customer wants to order
   3. The quantity required for each item

3. The customer service representative enters the district number into the application.

4. The application then:
   1. From the Customer Table, reads the customer last name, customer discount rate, and customer credit status.
   2. From the Item Table, reads the item names, item prices, and item data for each item ordered by the customer.
   3. Reads the District Table for the district tax and the next available district order number. The next available district order number is incremented by one and updated.
   4. Inserts a new row into both the New Order Table and the Order Table to reflect the creation of the new order.
   5. Checks if the quantity of ordered items is in stock by reading the quantity in the Stock Table. The quantity is reduced by the quantity ordered, and the new quantity is written into Quantity.
   6. A new row is inserted into the Order Line Table to reflect each item in the order.
   7. Writes a shipping record of the order (used to ship order).

### 13.2.3 Summary of Transaction Database Operations

Table 53 on page 363 shows all of the database operations performed for one application transaction of the Speed Benchmark. The response time is defined to include performing all of these operations for the transaction. See 3.1.6, “Database Table Structure” on page 96, for the formats of the Tables used.

<table>
<thead>
<tr>
<th>Database Operation</th>
<th>Table</th>
<th>Table Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read 1 Record</td>
<td>Customer</td>
<td>30,000</td>
<td>Read customer info</td>
</tr>
<tr>
<td>Read 10 Records</td>
<td>Item</td>
<td>100,000</td>
<td>Read item order data</td>
</tr>
<tr>
<td>Read 1 Record</td>
<td>District (DSTRCT)</td>
<td>10</td>
<td>Read district tax data</td>
</tr>
<tr>
<td>Update 1 Record</td>
<td>District (DSTRCT)</td>
<td>10</td>
<td>Update next order number</td>
</tr>
<tr>
<td>Insert 1 Record</td>
<td>New Order (NEWORD)</td>
<td>3500</td>
<td>Insert new order record</td>
</tr>
<tr>
<td>Insert 1 Record</td>
<td>Orders</td>
<td>3500</td>
<td>Insert order record</td>
</tr>
</tbody>
</table>
13.2.4 Speed Benchmark Screens

This section shows the operational screens of the benchmark.

13.2.4.1 Startup Parameters Screen

Figure 172 shows the screen used for entering the data source name. The other fields can optionally be changed. The settings are retained in the registry for other test runs.

<table>
<thead>
<tr>
<th>Database Operation</th>
<th>Table</th>
<th>Table Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read 10 Records</td>
<td>Stock</td>
<td>100,000</td>
<td>Read quantity of items ordered</td>
</tr>
<tr>
<td>Update 10 Records</td>
<td>Stock</td>
<td>100,000</td>
<td>Update stock item quantity based on order</td>
</tr>
<tr>
<td>Insert 10 Records</td>
<td>Order Line (ORDLIN)</td>
<td>35000</td>
<td>Insert detailed order records</td>
</tr>
</tbody>
</table>

![Figure 172. Speed Startup Screen](image)

13.2.4.2 Start Benchmark Screen

Figure 173 on page 365 shows the screen used for initiating the Speed benchmark.
13.2.4.3 Options Screen
Figure 174 on page 365 shows the screen used for controlling the number of transactions performed. Alternatively, the benchmark can be set to run by a time limit for the Speed benchmark. Additionally, the think time (time between transactions) can be set.

13.2.4.4 Application Screen
Figure 175 on page 366 shows the application screen while the benchmark is running.
13.2.4.5 Benchmark Results Screen

Figure 176 on page 367 shows the performance results (average response times) saved in a file and displayed using NotePad after the current run cycle has completed.
13.2.5 Benchmark Response Time Comparison of IBM OLE DB

Table 55 on page 368 shows the average transaction response time as well as the measured host (AS/400 CPU and Disk) load for the four benchmark runs. The stored procedure approach has the fastest response time since all operations take place on the AS/400 system. Although this stored procedure application was not coded with OLE DB (IBMDA400), you would get comparable results using the OLE DB stored procedure method. The ODBC API and OLE DB SQL methods are second and third, respectively, in response time performance. The reason that the OLE DB record level method is about twice the response time of the OLE DB version is that for this particular application the retrieval of the 10 Parts can be done in one SQL statement by using a WHERE PARTID IN (p1, p2, . . . p10) clause. The record level access method has to perform 10 separate operations. For other applications, you might find a closer performance comparison (than the 2 times response time factor) between the OLE DB SQL and the OLE DB record level access methods. See Table 56 on page 368 for a detailed comparison of the performance of specific functions for three of the access methods.

Table 54. Speed Benchmark Response Time and Host Load Factor

<table>
<thead>
<tr>
<th>Data Access Type</th>
<th>Average Response</th>
<th>AS/400 CPU Load Factor (5 transactions)</th>
<th>AS/400 Disk Load Factor (5 transactions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stored Procedure</td>
<td>.75 Seconds</td>
<td>6%</td>
<td>60 sync, 260 async i/o</td>
</tr>
<tr>
<td>ODBC API</td>
<td>1.54</td>
<td>9%</td>
<td>45 sync, 270 async i/o</td>
</tr>
<tr>
<td>IBMDA400 (OLEDB) SQL</td>
<td>2.24</td>
<td>9%</td>
<td>40 sync, 260 async i/o</td>
</tr>
<tr>
<td>IBMDA400 (OLEDB) Record level</td>
<td>4.72</td>
<td>9%</td>
<td>0 sync, 225 async i/o</td>
</tr>
</tbody>
</table>
13.2.6 Benchmark Response Time by Functional Components

Table 55 on page 368 shows the amount of the response time contributed for each component of the application transaction.

<table>
<thead>
<tr>
<th>Operation</th>
<th>OLEDB DDM</th>
<th>OLEDB SQL</th>
<th>ODBC API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read 1 Record</td>
<td>4%</td>
<td>5%</td>
<td>5%</td>
<td>Read customer info</td>
</tr>
<tr>
<td>Read 10 Records</td>
<td>16%</td>
<td>19%</td>
<td>10%</td>
<td>Read item order data</td>
</tr>
<tr>
<td>Read 1 Record</td>
<td>1%</td>
<td>4%</td>
<td>3%</td>
<td>Read district tax data</td>
</tr>
<tr>
<td>Update 1 Record</td>
<td>5%</td>
<td>1%</td>
<td>3%</td>
<td>Update next order number</td>
</tr>
<tr>
<td>Insert 1 Record</td>
<td>2%</td>
<td>2%</td>
<td>4%</td>
<td>Insert new order record</td>
</tr>
<tr>
<td>Insert 1 Record</td>
<td>1%</td>
<td>1%</td>
<td>3%</td>
<td>Insert order record</td>
</tr>
<tr>
<td>Read 10 Records</td>
<td>36%</td>
<td>40%</td>
<td>41%</td>
<td>Read quantity of items ordered</td>
</tr>
<tr>
<td>Update 10 Records</td>
<td>25%</td>
<td>16%</td>
<td>16%</td>
<td>Update stock item quantity based on order</td>
</tr>
<tr>
<td>Insert 10 Records</td>
<td>10%</td>
<td>12%</td>
<td>15%</td>
<td>Insert detailed order records</td>
</tr>
<tr>
<td>Total Time</td>
<td>4720 Ms</td>
<td>2240 Ms</td>
<td>1540 Ms</td>
<td>All Operations Above</td>
</tr>
</tbody>
</table>

13.3 Function Timings

Table 56 on page 368 shows timings (in Milliseconds) for two main types of OLE DB data access compared to the ODBC API Access Method.

Note the result that a keyed read (using a bookmark and a Move to record by bookmark) by the OLE DB record level method had a measured 125 ms response time (40 + 85) compared to a 300-600 ms response for the OLE DB SQL method using a prepared SQL statement with a WHERE clause and parameter markers.

<table>
<thead>
<tr>
<th>Function</th>
<th>Record Level Processing</th>
<th>SQL Processing</th>
<th>ODBC API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Connection</td>
<td>200</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>Open Link</td>
<td>610</td>
<td>na</td>
<td></td>
</tr>
<tr>
<td>Open Index</td>
<td>300</td>
<td>na</td>
<td></td>
</tr>
<tr>
<td>Prepare SQL</td>
<td>na</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Sequential - Next</td>
<td>24</td>
<td>16-27</td>
<td></td>
</tr>
<tr>
<td>Sequential - Last</td>
<td>75</td>
<td>na</td>
<td></td>
</tr>
<tr>
<td>Update Record</td>
<td>210</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insert Record</td>
<td>290</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete Record</td>
<td>450</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
13.4 Tips and Techniques

1. If you are experiencing a performance problem, make sure that error logging and tracing are turned off. Use the cwbzztrc.exe tool to do this. Running the tool with no parameters allows you to see the options and syntax. When you install Client Access, the default is to turn error logging on and tracing off.

2. The IBMDA400 provider does not support blocked reads of tables opened for record level access. Reading multiple records at once will not work. You will need to read the records one at a time from the AS/400 system. For more information on what is not supported, see the CacheSize property in the SDK Reference manual. Consider using SQL SELECT * or a call to an SQL stored procedure to get large Recordset objects back to the PC. Once you have all of your data back, use tables record level access support to do your individual record insert, update, and delete operations.

3. If you are using record level access and running over a TCP/IP connection to a pre-V4R2 AS/400 system, you will notice significantly reduced performance compared to the performance achieved with the new TCP/IP DDM server that is available in OS/400 V4R2. This is because TCP/IP DDM server in V4R2 uses native TCP/IP support, while the server in prior releases is SNA based and interfaces to TCP/IP.

4. If you are executing SQL statements that vary in values from one call to another, consider using parameter markers and prepare the SQL statements before executing them. This will improve your performance.

5. If you are using SQL statements and SQL stored procedures, the IBMDA400 provider should be comparable in performance to the Client Access ODBC driver.

6. Since a large portion of the Client Access OLE DB provider is written to Client Access APIs, you may see slightly improved performance by writing directly to the Client Access APIs.

<table>
<thead>
<tr>
<th>Function</th>
<th>Record Level Processing</th>
<th>SQL Processing</th>
<th>ODBC API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyed - Get</td>
<td>45</td>
<td>na</td>
<td></td>
</tr>
<tr>
<td>Bookmark</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keyed - Move to</td>
<td>40</td>
<td>na</td>
<td></td>
</tr>
<tr>
<td>Bookmark</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keyed Read</td>
<td>85</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Select * Where CID = 1</td>
<td>na</td>
<td>300-600</td>
<td>250-550</td>
</tr>
</tbody>
</table>

30,000 Record Customer Table. Measured on IBM AS/400 Model 510. V4R2 using TCP/IP
13.5 Summary - Usage and Trade-Offs

For the IBM Speed benchmark, the stored procedure method performs best (for this method, whether you use OLE DB to perform the stored procedures or ODBC, the performance will be comparable). The OLE DB SQL method is slower in response time performance compared to the ODBC API method. For this benchmark, the OLE DB record level method does not perform as well as the OLE DB SQL method since the retrieval of the 10 Parts records can be done with one SQL statement while the comparable function in the OLE DB record level method requires 10 keyed reads.

The functional timings show that a simple keyed read of a record is much faster using the OLE DB record level method (125 milliseconds) than using the OLE DB SQL method (300-600 milliseconds) based on the specific measurements taken.

13.5.1 OLE DB Record Level (Using DDM Server) Versus OLE DB SQL

See Chapter 2, “AS/400 OLE DB Support” on page 39, for detailed functional differences between these two methods especially in the areas of updatable Recordsets (which SQL does not support) and dynamic cursors (which SQL does not support). From a performance standpoint, either method may be faster depending upon the application requirements. Generally, you can write SQL to process multiple records faster and with less lines of code than using record level access. Other applications requiring only simple random reads of records would find the record level access method to be faster.

13.5.2 Performance Scalability

Besides the response time of a client/server application, you should also consider the overall resource load on the AS/400 system in terms of CPU and disk processing. These factors determine how many instances of the application can simultaneously be running before the AS/400 CPU and disk utilizations reach saturation levels. Table 55 on page 368 shows the relative host load factor for each of the methods based on measurements of the Speed benchmark.

13.5.3 When and Why You Should Use OLE DB Versus Other Methods

The Client Access OLE DB Provider is a set of application programming interfaces (APIs) which provide access to a wide range of data sources. ODBC provides access only to information in a relational database through SQL, while OLE DB is a method to access data via a standard COM (Component Object Model) Interface. OLE DB provides both the SQL functionality defined in ODBC and interfaces suitable for accessing non-SQL resources.

If performance is the main focus of the application you are developing, OLE DB is not the first option to look at. You should consider using Client Access APIs or writing at a program-to-program level (APPC or Sockets). However, these interfaces require an in-depth knowledge of low-level programming interfaces. You may also consider used AS/400 stored procedures. In this case, you can use either ODBC or the Client Access OLE DB Provider. However, stored procedures require code on the AS/400 system and may cause you to compromise your application portability.
The Client Access OLE DB Provider offers an alternative, a high-level interface that usually provides acceptable performance. This support provides access to a number of AS/400 resources:

- Tables record level access
- Data queues
- Programs
- CL commands
- SQL statements
- SQL stored procedures

You can choose to use one of these interfaces or a combination of them in your application. For example, you may choose to use SQL to retrieve a number of records from the AS/400 system into a Recordset object, but use record level access to read or update an individual record. By combining the use of these interfaces in your application, you can achieve the best possible performance. See Chapter 4, “Application Development Using AS/400 OLE DB Support” on page 103, for an example of developing an application which combines the use of a number of the Client Access OLE DB Provider interfaces.

The bottom line is that you want to develop applications that perform well and are relatively error free. But, you also want to produce these applications in a timely manner and be able to maintain and enhance them. This is the goal of the Client Access OLE DB Provider: to provide an easy to use interface that provides good performance.
Appendix A. Supporting Documentation

In this appendix, we show you how to download the programming samples discussed in this redbook. We also provide information about other documentation and support available.

A.1 Downloading the Samples

The client programs and the AS/400 programs and files used in this redbook are available to be downloaded through the Internet. These examples were developed using the AS/400 OLE DB support. They will work with OS/400 V3R2 or later. The following components are available:

- Client code for the sample programs
- An AS/400 library containing AS/400 programs and database files (V3R2 SAVF)
- Instructions for restoring the AS/400 library and the client code

Note

These example programs have not been subjected to any formal testing. They are provided "AS-IS"; they should be used for reference only. Please refer to the Special Notices section at the back of this document for more information.

To use these files, you must download them to your personal computer from the Internet Web site. A file named README.TXT is included. It contains instructions for restoring the AS/400 libraries, the client example code, and run-time notes.

The URL to access is:

http://www.redbooks.ibm.com

Click on Supporting Materials and then select directory SG245183. In the SG245183 directory, select the files that you want to download.

A.2 Other Documentation and Resources

Other documentation is also available to help you develop applications using the AS/400 OLE DB support. The SDK Technical Reference manual is available as part of the IBM SDK for ActiveX and OLE DB. The SDK also includes the Microsoft ADO online help. The IBM SDK for ActiveX and OLE DB is available as an open beta. For more information, please see:

http://www.as400.ibm.com/ClientAccess/oledb
Appendix B. Listings for AS/400 Programs

This appendix contains source listings for the following AS/400 programs used in the example programs.

- PARTS/PF - the PARTS file that is used with these programs
- SPROC2/SQLRPGLE - stored procedure program
- DPCXRPG/RPGLE - distributed program call program
- DQXRPG/RPGLE - data queue program

B.1 PARTS/PF

```
A          R PARTR
A            PARTNO         5S 0       COLHDG('Part Number')
A            PARTDS        25          COLHDG('Part Description')
A            PARTQY         5  0       COLHDG('Part Qty-on-Hand')
A            PARTPR         6  2       COLHDG('Part Price')
A            PARTDT          L         DATFMT(*ISO)
A                                      COLHDG('Part Shipment Date')
A          K PARTNO
```

B.2 SPROC2/SQLRPGLE

```
H* This program is called from the client via a stored procedure
H* It returns data to the client from the PARTS database file via
H* sql result sets
D* Defines PART ID As a Integer (Binary 4.0)
D*
D  #PRTDS          DS
D  #PART                  1      4B 0
D  #OPTDS          DS
D  #OPT                  1      4B 0
C     *ENTRY        PLIST
C                   PARM                    #OPTDS
C                   PARM                    #PRTDS
C*  Copy PART NUMBER to RPG Native Variable With Same Attributes Of
C*  Field In PARTS Master File (5,0) For Performance Issues
C                   Z-ADD     #PART         PART              5 0
C     #OPT          CASEQ     1             ONEREC
C     #OPT          CASEQ     2             ALLREC
C                   CAS                     BADOPT
C                   ENDCS
C*
C     ONEREC        BEGSR
C/Exec Sql Declare C1 Cursor For
C+   Select
C+   PARTNO,
C+   PARTDS,
C+   PARTQY,
C+   PARTPR,
C+   PARTDT
C+   From PARTS                     -- From PART Master File
```
C+
C+   Where PARTNO = &colon.PART
C+
C+
C+ For Fetch Only     -- Read Only Cursor
C/End-Exec
C*
C/Exec Sql
C+ Open C1
C/End-Exec
C*
C/Exec Sql
C+ Set Result Sets Cursor C1
C/End-Exec
C*
C                   RETURN
C                   ENDSR
C*
C     ALLREC        BEGSR
C/Exec Sql Declare C2 Cursor For
C+ Select
C+   PARTNO,
C+   PARTDS,
C+   PARTQY,
C+   PARTPR,
C+   PARTDT
C+
C+ From PARTS        -- From PART Master File
C+
C+
C+ Order By PARTNO    -- Sort By PARTNO
C+
C+ For Fetch Only     -- Read Only Cursor
C/End-Exec
C*
C/Exec Sql
C+ Open C2
C/End-Exec
C*
C/Exec Sql
C+ Set Result Sets Cursor C2
C/End-Exec
C
C                   RETURN
C                   ENDSR
C-------------------------------------------------------------
C* SUBROUTINE BADOPT
C*
C* AN UNRECOGNIZED OPTION PARAMETER WAS SET - RETURN 'U' FOR
C* UNKNOWN.
C*
C-------------------------------------------------------------
C     BADOPT        BEGSR
C                   MOVE      3             #OPT
C                   RETURN
C                   ENDSR
B.3 DPCXRPG/RPGLE

H* This program is called from the client via the Distributed
H* Program Call Interface. It
H* returns data to the client from the PARTS database file via
H* parameters
H*------------------------------------------------------------------
H
FPARTS IF E K DISK
H*---------------------------------------------------------------
C* MAIN PROGRAM
C*
C* Take action depending on the 'option' parameter&colon;
C* Option    Action
C*          S      Retrieve a single record for supplied key
C*          A      Position to start of file
C*          F      Fetch the next record based on cursor posn.
C*          E      End the program
C*----------------------------------------------------------------
C     *ENTRY        PLIST
C                   PARM                    OPTION            1
C                   PARM                    PARTNO
C                   PARM                    PARTDS
C                   PARM                    PARTQY
C                   PARM                    PARTPR
C                   PARM                    PARTDT
C     OPTION        CASEQ     'S'           ONEREC
C     OPTION        CASEQ     'A'           ALLREC
C     OPTION        CASEQ     'F'           NXTREC
C     OPTION        CASEQ     'E'           ENDPRG
C     CAS                     BADOPT
C     ENDCS
C*----------------------------------------------------------------
C* SUBROUTINE - ONEREC
C* This subroutine attempts to find the requested part in the
C* PARTS file. If the record is found, set the OPTION parameter
C* to 'Y', otherwise to 'X' to indicate record not found, then
C* return.
C*----------------------------------------------------------------
C     ONEREC        BEGSR
C* Return only one record
C     PARTNO        CHAIN     PARTR                              40
C     *IN40         IFEQ      '1'
C                   MOVE      'X'           OPTION
C                   ELSE
C                   MOVE      'Y'           OPTION
C                   ENDIF
C                   RETURN
C     ENDSR
C*----------------------------------------------------------------
C* SUBROUTINE - ALLREC
C* This subroutine re-positions the cursor to the start of the
C* PARTS file anticipating subsequent calls to fetch the records
C* sequentially. If the SETLL operation fails, set the option
C* parameter to 'X', otherwise 'Y'.
C*----------------------------------------------------------------
C     ALLREC        BEGSR
B.4 DQXRPG/RPGLE

H* This is a never-ending-program that runs in the background H* as a batch job. It checks the data queue DQINPT for H* any queue entries received. Once an entry arrives in the H* data queue, the program retrieves and processes it H* and returns output to the DQOUPT data queue.
H* This program should be submitted with the SBMJOB command H* and terminated with ENDJOB OR WRKACTJOB commands, or by H* placing an entry starting with ‘E’ on the DQINPT data queue.
FPARTS IF E K DISK

DATA STRUCTURES

DATAI - input data record 6 bytes
DATAO - output data record 48 bytes

DATAI DS
OPTION 1 1
INPNO 2 6 0
DATAO DS
RESULT 1 1
PARTNO 2 6 0
PARTDS 7 31
PARTQY 32 34P 0
PARTPR 35 38P 2
PARTDT 39 48D

CONSTANTS

DQINPT - data queue used for receiving input records
DQOUPT - data queue used for sending records
APILIB - library name

DQINPT C CONST('DQINPT ')
DQOUPT C CONST('DQOUPT ')
LIBL C CONST('APILIB ')

MAIN PROGRAM

Loop on read to data queue. Action depends on the 'option' flag&colon.

Option Action
S Retrieve a single record for supplied key
A Retrieve all records in file
E End the program

RCVDQ

This subroutine performs the QRCVDTAQ function. Notice that
the wait parameter is set to a negative value to force it
to wait until a queue entry is available.

RCVDQ BEGSR
MOVE DQINPT QUEUEI 10
MOVE LIBL LIBLD 10
Z-ADD 6 FLDDL 5 0
Z-ADD -9 WAIT 5 0
CALL 'QRCVDTAQ'
C                  PARM                    QUEUEI
C                  PARM                    LIBLD
C                  PARM                    FLDDL
C                  PARM                    DATAI
C                  PARM                    WAIT
C                  ENDSR
C*----------------------------------------------------------------
C* SUBROUTINE - READR
C* This subroutine retrieves the part number from the data queue DQINPT, searches the data base file PARTS using the part number just received. If the record is found, send the record to the data queue DQOUPT. If option 'A' is received, send all records to the data queue DQOUPT.
C*
C* The 'result' flag is set as follows
C*   Result    Meaning
C*       Y      Record found and being returned
C*       X      Record not found or eof
C*----------------------------------------------------------------
C     READR         BEGSR
C     OPTION        IFEQ      'A'
C* Return all records in the file
C   *LOVAL        SETLL     PARTS
C   READ      PARTS                                  60
C   *IN60         DOWEQ     '0'
C   MOVE      'Y'           RESULT
C   EXSR      SNDDQ
C   READ      PARTS                                  60
C   ENDDO
C   MOVE      'X'           RESULT
C   EXSR      SNDDQ
C   ELSE
C   INPNO         CHAIN     PARTR                              98
C   *IN98         IFEQ      '1'
C   MOVE      'X'           RESULT
C   ELSE
C   MOVE      'Y'           RESULT
C   ENDF
C   EXSR      SNDDQ
C   ENDF
C   ELSE
C*----------------------------------------------------------------
C* SUBROUTINE SNDDQ
C*
C* This subroutine performs the QSNDDTAQ function.
C*
C   SNDDQ         BEGSR
C   MOVE      DQOUPT     QUEUEO                                  10
C   MOVE      LIBL     LIBLD
C   Z-ADD     48       FLDDL
C   CALL      'QSNDDTAQ'
C   PARM      QUEUEO
C   PARM      LIBLD
C   PARM      FLDDL
C   PARM      DATAI
C   ENDSR
Appendix C. Data Types, ADO Support and Error Messages

The following sections describe the data types used with the OLE DB Provider, the ADO features implemented by the AS/400 OLE DB Provider, and error messages that are sent by the AS/400 OLE DB Provider.

C.1 Data Types

This section describes the data types that are used with the AS/400 OLE DB Provider.

C.1.1 Table Data Types

The following table indicates how the IBMDA400 provider maps AS/400 table data types to OLE DB data types:

<table>
<thead>
<tr>
<th>AS/400 Table</th>
<th>OLE DB</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY2</td>
<td>DBTYPE_I2</td>
<td></td>
</tr>
<tr>
<td>BINARY4</td>
<td>DBTYPE_I4</td>
<td></td>
</tr>
<tr>
<td>CHAR</td>
<td>DBTYPE_STR</td>
<td>CHAR with CCSID 65535 is mapped to DBTYPE_BYTES</td>
</tr>
<tr>
<td>DATE</td>
<td>DBTYPE_STR</td>
<td></td>
</tr>
<tr>
<td>EITHER</td>
<td>DBTYPE_STR</td>
<td></td>
</tr>
<tr>
<td>FLTDBL</td>
<td>DBTYPE_R8</td>
<td></td>
</tr>
<tr>
<td>FLTSNG</td>
<td>DBTYPE_R4</td>
<td></td>
</tr>
<tr>
<td>GRAPHIC</td>
<td>DBTYPE_WSTR</td>
<td></td>
</tr>
<tr>
<td>HEX</td>
<td>DBTYPE_BYTES</td>
<td></td>
</tr>
<tr>
<td>ONLY</td>
<td>DBTYPE_STR</td>
<td></td>
</tr>
<tr>
<td>OPEN</td>
<td>DBTYPE_STR</td>
<td></td>
</tr>
<tr>
<td>PACKED</td>
<td>DBTYPE_DECIMAL</td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>DBTYPE_STR</td>
<td></td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>DBTYPE_STR</td>
<td></td>
</tr>
<tr>
<td>VARCHAR</td>
<td>DBTYPE_STR</td>
<td>VARCHAR with CCSID 65535 is mapped to DBTYPE_BYTES</td>
</tr>
<tr>
<td>VAREITHER</td>
<td>DBTYPE STR</td>
<td></td>
</tr>
<tr>
<td>VARGRAPHIC</td>
<td>DBTYPE_WSTR</td>
<td></td>
</tr>
<tr>
<td>VARHEX</td>
<td>DBTYPE_BYTES</td>
<td></td>
</tr>
<tr>
<td>VARONLY</td>
<td>DBTYPE_STR</td>
<td></td>
</tr>
<tr>
<td>VAROPEN</td>
<td>DBTYPE_STR</td>
<td></td>
</tr>
<tr>
<td>ZONED</td>
<td>DBTYPE_NUMERIC</td>
<td></td>
</tr>
</tbody>
</table>
ADO does not support any of the variable-length data types in the Recordset object. These types are for the parameter object only. See the Type Property topic of the ADO Reference. When a variable-length AS/400 database field is read, the Recordset column length is set to the maximum length of the variable length field. Also, when a variable length AS/400 database field is updated or inserted into, the actual length of the string value for the field is taken, the length of the database field is set, and the data copied.

C.1.2 SQL Data Types

The following table shows how the IBMDA400 provider maps AS/400 SQL data types to OLE DB data types:

<table>
<thead>
<tr>
<th>AS/400 SQL</th>
<th>OLE DB</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>DBTYPE_STR</td>
<td>CHAR with CCSID 65535 is mapped to DBTYPE_BYTES</td>
</tr>
<tr>
<td>Date</td>
<td>DBTYPE_STR</td>
<td></td>
</tr>
<tr>
<td>Decimal</td>
<td>DBTYPE_DECIMAL</td>
<td></td>
</tr>
<tr>
<td>Double</td>
<td>DBTYPE_R8</td>
<td></td>
</tr>
<tr>
<td>Float</td>
<td>DBTYPE_R4</td>
<td>If precision is 1-24 then DBTYPE_R4</td>
</tr>
<tr>
<td>Float</td>
<td>DBTYPE_R8</td>
<td>If precision is 25-53 then DBTYPE_R8</td>
</tr>
<tr>
<td>Graphic</td>
<td>DBTYPE_WSTR</td>
<td></td>
</tr>
<tr>
<td>Integer</td>
<td>DBTYPE_I4</td>
<td></td>
</tr>
<tr>
<td>Numeric</td>
<td>DBTYPE_NUMERIC</td>
<td></td>
</tr>
<tr>
<td>Real</td>
<td>DBTYPE_R4</td>
<td></td>
</tr>
<tr>
<td>SmallInt</td>
<td>DBTYPE_I2</td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td>DBTYPE_STR</td>
<td></td>
</tr>
<tr>
<td>VarChar</td>
<td>DBTYPE_STR</td>
<td>VARCHAR with CCSID 65535 is mapped to DBTYPE_BYTES</td>
</tr>
<tr>
<td>VarGraphic</td>
<td>DBTYPE_WSTR</td>
<td></td>
</tr>
</tbody>
</table>
C.1.3 Data Queue Data Types

When you open a data queue, you have an opportunity to define the entry format. The first column of the following table shows data types that are supported on the open of a data queue. The second column shows how the IBMDA400 provider maps data queue data types to OLE DB data types.

Table 59. Data Queue Data Type to OLE DB Data Type Cross-Reference

<table>
<thead>
<tr>
<th>Data Queue</th>
<th>OLE DB</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR<a href="length">ACTER</a></td>
<td>DBTYPE_STR</td>
<td>CHAR with CCSID 65535 is mapped to DBTYPE_BYTES Use CHAR type for DBCS OPEN, ONLY and EITHER data</td>
</tr>
<tr>
<td>DECIMAL(&lt;precision&gt;,&lt;scale&gt;)</td>
<td>DBTYPE_DECIMAL</td>
<td></td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DBTYPE_R8</td>
<td></td>
</tr>
<tr>
<td>GRAPHIC(&lt;length&gt;)</td>
<td>DBTYPE_WSTR</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>DBTYPE_I4</td>
<td></td>
</tr>
<tr>
<td>NUMERIC(&lt;precision&gt;,&lt;scale&gt;)</td>
<td>DBTYPE_NUMERIC</td>
<td></td>
</tr>
<tr>
<td>SINGLE</td>
<td>DBTYPE_R4</td>
<td></td>
</tr>
<tr>
<td>SMALLINT</td>
<td>DBTYPE_I2</td>
<td></td>
</tr>
</tbody>
</table>

If you define a data queue field as GRAPHIC, the length of the field is the number of graphic characters. The number of bytes allocated for the field is double the length. For example, a GRAPHIC(30) allocates 60 bytes. For more information on data type allocations, see the AS/400 data type descriptions topic.

C.1.4 Program Data Types

The following table shows how the IBMDA400 provider maps ADO data types to OLE DB types and to AS/400 program data types. Check your programming language reference manual for specific data types supported by your language. Since programs do not have their input and output parameters in a common defined catalog entry such as stored procedures, the parameter objects of ADO are used to define input/output parameters and their types to the IBMDA400 provider. Specify the correct ADO data type that will match the program parameter of the AS/400 program that you are calling.

Table 60. Program Data Type to OLE DB Data Type Cross-Reference

<table>
<thead>
<tr>
<th>AS/400 Program</th>
<th>OLE DB</th>
<th>ADO</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY2</td>
<td>DBTYPE_I2</td>
<td>adSmallInt</td>
<td></td>
</tr>
<tr>
<td>BINARY4</td>
<td>DBTYPE_I4</td>
<td>adInteger</td>
<td></td>
</tr>
<tr>
<td>CHAR</td>
<td>DBTYPE_STR</td>
<td>adChar</td>
<td>Use adChar for DBCS OPEN, ONLY and EITHER program parameters</td>
</tr>
<tr>
<td>FLTDBL</td>
<td>DBTYPE_R8</td>
<td>adDouble</td>
<td></td>
</tr>
<tr>
<td>FLTSGN</td>
<td>DBTYPE_R4</td>
<td>adSingle</td>
<td></td>
</tr>
</tbody>
</table>
If you are attempting to call a program on the AS/400 server that has a data structure or a user defined type as a parameter, this type of parameter is not supported. For example, most of the OS/400 system APIs defined in the System API Programming, manual, SC41-5800, require a structure for both input and output parameters.

To get the program call to operate, set the type for these parameters in the ADO parameter object to adBinary, along with the appropriate length of each structure. If you define a program parameter as adBinary, no conversion of the data is done between the PC client and the AS/400 server. This data is not usable, but you may be able to work with some of the other program parameters that are not defined as structures.

As an alternative, define the structure parameter as adChar. The character data in the structure parameter will be usable.

C.1.5 AS/400 Data Type Descriptions

The following table is useful if you want a description or when you are calculating the size in bytes of a particular data type.

<table>
<thead>
<tr>
<th>AS/400 Data Type</th>
<th>Description</th>
<th>Size in Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY1</td>
<td>1-Byte Signed Number</td>
<td>1</td>
</tr>
<tr>
<td>BINARY2</td>
<td>2-Byte Signed Number</td>
<td>2</td>
</tr>
<tr>
<td>BINARY4</td>
<td>4-Byte Signed Number</td>
<td>4</td>
</tr>
<tr>
<td>BINARY8</td>
<td>8-Byte Signed Number</td>
<td>8</td>
</tr>
<tr>
<td>CHAR</td>
<td>Character Value</td>
<td>Declared Length</td>
</tr>
<tr>
<td>DATE</td>
<td>Date Value</td>
<td>10, ISO format</td>
</tr>
<tr>
<td>EITHER</td>
<td>DBCS Either</td>
<td>Declared Length</td>
</tr>
<tr>
<td>FLTDBL</td>
<td>Double Floating Point</td>
<td>8</td>
</tr>
<tr>
<td>FLTSNG</td>
<td>Single Floating Point</td>
<td>4</td>
</tr>
<tr>
<td>GRAPHIC</td>
<td>DBCS Graphic</td>
<td>Declared Length * 2</td>
</tr>
<tr>
<td>HEX</td>
<td>Hexadecimal Value</td>
<td>Declared Length</td>
</tr>
<tr>
<td>ONLY</td>
<td>DBCS Only</td>
<td>Declared Length</td>
</tr>
<tr>
<td>OPEN</td>
<td>DBCS Open</td>
<td>Declared Length</td>
</tr>
<tr>
<td>PACKED</td>
<td>Packed Decimal Value</td>
<td>(Length + 1)/2, round fraction up</td>
</tr>
<tr>
<td>TIME</td>
<td>Time Value</td>
<td>8, ISO format</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>Timestamp Value</td>
<td>26</td>
</tr>
</tbody>
</table>
C.1.6 SQL Data Type Descriptions

The following table is useful if you want a description or when you are calculating the size in bytes of a particular data type.

<table>
<thead>
<tr>
<th>AS/400 Data Type</th>
<th>Description</th>
<th>Size in Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRUCTURE</td>
<td>Pgm defined structure</td>
<td>Declared Length</td>
</tr>
<tr>
<td>UBINARY1</td>
<td>1-Byte Unsigned Number</td>
<td>1</td>
</tr>
<tr>
<td>UBINARY2</td>
<td>2-Byte Unsigned Number</td>
<td>2</td>
</tr>
<tr>
<td>UBINARY4</td>
<td>4-Byte Unsigned Number</td>
<td>4</td>
</tr>
<tr>
<td>UBINARY8</td>
<td>8-Byte Unsigned Number</td>
<td>8</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>Variable Character</td>
<td>Declared Length + 2</td>
</tr>
<tr>
<td>VAREITHER</td>
<td>Variable DBCS Either</td>
<td>Declared Length + 2</td>
</tr>
<tr>
<td>VARGRAPHIC</td>
<td>Variable DBCS Graphic</td>
<td>(Declared Length * 2) + 2</td>
</tr>
<tr>
<td>VARHEX</td>
<td>Variable Hexadecimal</td>
<td>Declared Length + 2</td>
</tr>
<tr>
<td>VARONLY</td>
<td>Variable DBCS Only</td>
<td>Declared Length + 2</td>
</tr>
<tr>
<td>VAROPEN</td>
<td>Variable DBCS Open</td>
<td>Declared Length + 2</td>
</tr>
<tr>
<td>ZONED</td>
<td>Zoned Decimal Value</td>
<td>Length</td>
</tr>
<tr>
<td>Character</td>
<td>Character Value</td>
<td>Declared Length</td>
</tr>
<tr>
<td>Date</td>
<td>Date Value</td>
<td>10, ISO format</td>
</tr>
<tr>
<td>Decimal</td>
<td>Packed Decimal Value</td>
<td>(Length + 1)/2, round up fraction</td>
</tr>
<tr>
<td>Double</td>
<td>Double Floating Point</td>
<td>8</td>
</tr>
<tr>
<td>Float</td>
<td>Double Floating Point</td>
<td>8</td>
</tr>
<tr>
<td>Graphic</td>
<td>DBCS Graphic</td>
<td>Declared Length * 2</td>
</tr>
<tr>
<td>Integer</td>
<td>4-Byte Signed Number</td>
<td>4</td>
</tr>
<tr>
<td>Numeric</td>
<td>Zoned Decimal Value</td>
<td>Length</td>
</tr>
<tr>
<td>Real</td>
<td>Single Floating Point</td>
<td>4</td>
</tr>
<tr>
<td>SmallInt</td>
<td>2-Byte Signed Number</td>
<td>2</td>
</tr>
<tr>
<td>Time</td>
<td>Time Value</td>
<td>8, ISO format</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Timestamp Value</td>
<td>26</td>
</tr>
<tr>
<td>VarChar</td>
<td>Variable Character</td>
<td>Declared Length + 2</td>
</tr>
<tr>
<td>VarGraphic</td>
<td>Variable DBCS Graphic</td>
<td>(Declared Length * 2) + 2</td>
</tr>
</tbody>
</table>
C.1.7 ADO Data Type Descriptions

The ADO variable length types are for Parameter objects only. Fields of a Recordset object cannot have a variable length type. See the Type Property topic of the ADO Reference for a list of all ADO data types.

C.1.8 OLE DB Data Type Descriptions

The following table is useful if you want a list and description of all of the OLE DB data types.

*Table 63. OLE DB Data Type Descriptions*

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBTYPE_EMPTY</td>
<td>No value was specified</td>
</tr>
<tr>
<td>DBTYPE_NULL</td>
<td>A NULL value</td>
</tr>
<tr>
<td>DBTYPE_RESERVED</td>
<td>Reserved for future use by OLE DB</td>
</tr>
<tr>
<td>DBTYPE_I1</td>
<td>A one-byte, signed integer</td>
</tr>
<tr>
<td>DBTYPE_I2</td>
<td>A two-byte, signed integer</td>
</tr>
<tr>
<td>DBTYPE_I4</td>
<td>A four-byte, signed integer</td>
</tr>
<tr>
<td>DBTYPE_I8</td>
<td>An eight-byte, signed integer</td>
</tr>
<tr>
<td>DBTYPE_UI1</td>
<td>A one-byte, unsigned integer</td>
</tr>
<tr>
<td>DBTYPE_UI2</td>
<td>A two-byte, unsigned integer</td>
</tr>
<tr>
<td>DBTYPE_UI4</td>
<td>A four-byte, unsigned integer</td>
</tr>
<tr>
<td>DBTYPE_UI8</td>
<td>An eight-byte, unsigned integer</td>
</tr>
<tr>
<td>DBTYPE_R4</td>
<td>A single-precision floating point value</td>
</tr>
<tr>
<td>DBTYPE_R8</td>
<td>A double-precision floating point value</td>
</tr>
<tr>
<td>DBTYPE_CY</td>
<td>A currency value</td>
</tr>
<tr>
<td>DBTYPE_DECIMAL</td>
<td>A packed decimal numeric value</td>
</tr>
<tr>
<td>DBTYPE_NUMERIC</td>
<td>A zoned decimal numeric value</td>
</tr>
<tr>
<td>DBTYPE_DATE</td>
<td>An OLE Automation date</td>
</tr>
<tr>
<td>DBTYPE_BOOL</td>
<td>An OLE Automation boolean value</td>
</tr>
<tr>
<td>DBTYPE_BYTES</td>
<td>A binary data value</td>
</tr>
<tr>
<td>DBTYPE_BSTR</td>
<td>A pointer to a BSTR, as in OLE Automation</td>
</tr>
<tr>
<td>DBTYPE_STR</td>
<td>A null-terminated ANSI character string</td>
</tr>
<tr>
<td>DBTYPE_WSTR</td>
<td>A null-terminated Unicode character string</td>
</tr>
<tr>
<td>DBTYPE_VARIANT</td>
<td>An OLE Automation VARIANT</td>
</tr>
<tr>
<td>DBTYPE_IDISPATCH</td>
<td>A pointer to OLE object IDispatch interface</td>
</tr>
<tr>
<td>DBTYPE_IUNKNOWN</td>
<td>A pointer to OLE object IUnknown interface</td>
</tr>
<tr>
<td>DBTYPE_GUID</td>
<td>A globally unique identifier (GUID)</td>
</tr>
<tr>
<td>Constant</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>DBTYPE_ERROR</td>
<td>A 32-bit error code</td>
</tr>
<tr>
<td>DBTYPE_BYREF</td>
<td>A pointer to data</td>
</tr>
<tr>
<td>DBTYPE_ARRAY</td>
<td>A pointer to a SAFEARRAY</td>
</tr>
<tr>
<td>DBTYPE_VECTOR</td>
<td>A DBVECTOR structure</td>
</tr>
<tr>
<td>DBTYPE_UDT</td>
<td>A user-defined data type of variable length</td>
</tr>
<tr>
<td>DBTYPE_DBDATE</td>
<td>A date structure</td>
</tr>
<tr>
<td>DBTYPE_DBTIME</td>
<td>A time structure</td>
</tr>
<tr>
<td>DBTYPE_DBTIMESTAMP</td>
<td>A timestamp structure</td>
</tr>
</tbody>
</table>
C.2 ADO Object Summary

This section describes the properties and methods for each ADO object and indicates the support provided with the AS/400 OLE DB Provider.

C.2.1 Connection Object Support Summary

The following tables show the collections, methods, and properties used with the Connection object.

Table 64. Connection Object Collections Support

<table>
<thead>
<tr>
<th>Collection</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Errors</td>
<td>Yes</td>
</tr>
<tr>
<td>Properties</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 65. Connection Object Methods Support

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameter</th>
<th>Value</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeginTrans</td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Close</td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CommitTrans</td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Execute</td>
<td>CommandText</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RecordsAffected</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Options</td>
<td>adCmdText</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adCmdTable</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adCmdStoredProc</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adCmdUnknown</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Open</td>
<td>ConnectionString</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UserID</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Password</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>RollbackTrans</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Table 66. Connection Object Properties Support

<table>
<thead>
<tr>
<th>Property</th>
<th>Option</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>adXactCommitRetaining</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>adXactAbortRetaining</td>
<td>No</td>
</tr>
<tr>
<td>CommandTimeout</td>
<td></td>
<td>No</td>
</tr>
</tbody>
</table>
C.2.2 Error Object, Errors Collection Support Summary

This section provides a summary for the Error object and Errors collection.

C.2.2.1 Error Object

The following table shows the properties used with the Error object.

<table>
<thead>
<tr>
<th>Property</th>
<th>Option</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>HelpContext</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>HelpFile</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>NativeError</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>Number</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Source</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>SQLState</td>
<td></td>
<td>No</td>
</tr>
</tbody>
</table>
C.2.2.2 Errors Collection
The following tables show the methods and properties used with the Errors collection.

Table 68. Errors Collection Methods Support

<table>
<thead>
<tr>
<th>Method</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 69. Errors Collection Properties Support

<table>
<thead>
<tr>
<th>Property</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Yes</td>
</tr>
<tr>
<td>Item</td>
<td>Yes</td>
</tr>
</tbody>
</table>

C.2.3 Command Object Support Summary
The following tables show the collections, methods, and properties used with the Command object.

Table 70. Command Object Collections Support

<table>
<thead>
<tr>
<th>Collection</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>Yes</td>
</tr>
<tr>
<td>Properties</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 71. Command Object Methods Support

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameter</th>
<th>Value</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateParameter</td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Name</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Type</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Direction</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Size</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Execute</td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>RecordsAffected</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Parameters</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Options</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>adCmdText</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adCmdTable</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adCmdStoredProc</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adCmdUnknown</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Table 72. Command Object Properties Support

<table>
<thead>
<tr>
<th>Property</th>
<th>Option</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveConnection</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CommandText</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>
### C.2.4 Parameter Object, Parameters Collection Support Summary

This section provides a summary for the Parameter object and Parameter collection.

#### C.2.4.1 Parameter Object

The following tables show the methods and properties used with the Parameter object.

**Table 73. Parameter Object Methods Support**

<table>
<thead>
<tr>
<th>Method</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppendChunk</td>
<td>No</td>
</tr>
</tbody>
</table>

**Table 74. Parameter Object Properties Support**

<table>
<thead>
<tr>
<th>Property</th>
<th>Option</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>adParamSigned</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>adParamNullable</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>adParamLong</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Direction</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>adParamInput</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>adParamOutput</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>adParamInputOutput</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>adParamReturnValue</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>NumericScale</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Precision</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
C.2.4.2 Parameters Collection
The following tables show the methods and properties used with the Parameters collection.

Table 75. Parameters Collection Methods Support

<table>
<thead>
<tr>
<th>Method</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Append</td>
<td>Yes</td>
</tr>
<tr>
<td>Delete</td>
<td>Yes</td>
</tr>
<tr>
<td>Refresh</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 76. Parameters Collection Properties Support

<table>
<thead>
<tr>
<th>Method</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Yes</td>
</tr>
<tr>
<td>Item</td>
<td>Yes</td>
</tr>
</tbody>
</table>
C.2.5 Recordset Object Support Summary

The following tables show the collections, methods, and properties used with the `Recordset` object.

Table 77. Recordset Object Collections Support

<table>
<thead>
<tr>
<th>Collection</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fields</td>
<td>Yes</td>
</tr>
<tr>
<td>Properties</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 78. Recordset Object Methods Support

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameter</th>
<th>Value</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddNew</td>
<td>Fields</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Values</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CancelBatch</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>CancelUpdate</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Clone</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Close</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Delete</td>
<td>AffectRecords</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adAffectCurrent</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adAffectGroup</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>GetRows</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rows</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Field</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>NumRecords</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>MoveFirst</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>MoveLast</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>MoveNext</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>MovePrevious</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>NextRecordset</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RecordsAffected</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Parameter</td>
<td>Value</td>
<td>Support</td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>Open</td>
<td>Source</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ActiveConnection</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CursorType</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>adOpenForwardOnly</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adOpenKeyset</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adOpenDynamic</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adOpenStatic</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LockType</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>adLockReadOnly</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adLockPessimistic</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adLockOptimistic</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adLockBatchOptimistic</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Options</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adCmdText</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adCmdTable</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adCmdStoredProc</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adUnknown</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Requery</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Resync</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AffectRecords</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adAffectCurrent</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adAffectGroup</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adAffectAll</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Parameter</td>
<td>Value</td>
<td>Support</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------</td>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>Supports</td>
<td>CursorOptions</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adAddNew</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adApproxPosition</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adBookmark</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adDelete</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adHoldRecords</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adMovePrevious</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adResync</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adUpdate</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adUpdateBatch</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Update</td>
<td>Fields</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Values</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>UpdateBatch</td>
<td>AffectRecords</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adAffectCurrent</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adAffectGroup</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adAffectAll</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

Table 79. Recordset Object Properties Support

<table>
<thead>
<tr>
<th>Property</th>
<th>Option</th>
<th>Value</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsolutePage</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>AbsolutePosition</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>ActiveConnection</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>BOF</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Bookmark</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CacheSize</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>CursorType</td>
<td>adOpenForwardOnly</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adOpenKeyset</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adOpenDynamic</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adOpenStatic</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Option</td>
<td>Value</td>
<td>Support</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------</td>
<td>----------------</td>
<td>---------</td>
</tr>
<tr>
<td>EditMode</td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>adEditNone</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>adEditInProgress</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>adEditAdd</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>EOF</td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Filter</td>
<td></td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>CriteriaString</td>
<td></td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>Array of bookmarks</td>
<td></td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>FilterGroup</td>
<td>adFilterNone</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>adFilterPendingRecords</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>adFilterAffectedRecords</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>adFilterFetchedRecords</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>LockType</td>
<td></td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>adLockReadOnly</td>
<td></td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>adLockPessimistic</td>
<td></td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>adLockOptimistic</td>
<td></td>
<td></td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>adLockBatchOptimistic</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>MaxRecords</td>
<td></td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>PageCount</td>
<td></td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>PageSize</td>
<td></td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>RecordCount</td>
<td></td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>Source</td>
<td></td>
<td></td>
<td>No</td>
</tr>
</tbody>
</table>
This section provides a summary for the Field object and Fields collection.

C.2.6.1 Field Object
The following tables show the collections, methods, and properties used with the Field object.

<table>
<thead>
<tr>
<th>Property</th>
<th>Option</th>
<th>Value</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>adRecOK</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>adRecNew</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>adRecModified</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>adRecInvalid</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>adRecMultipleChanges</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>adRecPendingChanges</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>adRecCanceled</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>adRecCantRelease</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>adRecConcurrencyViolation</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>adRecIntegrityViolation</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>adRecMaxChangesExceeded</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>adRecObjectOpen</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>adRecOutOfMemory</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>adRecPermissionDenied</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>adRecSchemaViolation</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>adRecDBDeleted</td>
<td></td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

Table 80. Field Object Collection Support

<table>
<thead>
<tr>
<th>Collection</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 81. Field Object Methods Support

<table>
<thead>
<tr>
<th>Method</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppendChunk</td>
<td>No</td>
</tr>
<tr>
<td>GetChunk</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 82. Field Object Properties Support

<table>
<thead>
<tr>
<th>Property</th>
<th>Option</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActualSize</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>
C.2.7 Fields Collection

The following tables show the methods and properties used with the Fields collection.

Table 83. Fields Collection Methods Support

<table>
<thead>
<tr>
<th>Method</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refresh</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 84. FieldsCollection Properties Support

<table>
<thead>
<tr>
<th>Property</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Yes</td>
</tr>
<tr>
<td>Item</td>
<td>Yes</td>
</tr>
</tbody>
</table>

C.2.8 Property Object, Properties Collection Support Summary

This section provides a summary for the Property object and Properties collection.
C.2.8.1 Property Object
The following table shows the properties used with the Property object.

Table 85. Property Object Properties Support

<table>
<thead>
<tr>
<th>Property</th>
<th>Option</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>adPropNotSupported</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>adPropRequired</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>adPropOptional</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>adPropRead</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>adPropWrite</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Name</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Type</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Value</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

C.2.8.2 Properties Collection
The following tables show the methods and properties used with the Properties collection.

Table 86. Properties Collection Methods Support

<table>
<thead>
<tr>
<th>Method</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refresh</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 87. Properties Collection Properties Support

<table>
<thead>
<tr>
<th>Property</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Yes</td>
</tr>
<tr>
<td>Item</td>
<td>Yes</td>
</tr>
</tbody>
</table>
C.3 Messages Sent from the AS/400 OLE DB Provider

This section lists the messages that are sent from the AS/400 OLE DB Provider.

C.3.1 Connection Messages

The following table shows the messages sent when a connection error is detected.

Table 88. Connection Messages

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Error.Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2147467259</td>
<td>CWBZZ5005 Unable to connect to system %1</td>
</tr>
<tr>
<td>-2147467259</td>
<td>CWBZZ5008 Security error occurred when attempting to connect to system %1</td>
</tr>
<tr>
<td>-2147467259</td>
<td>CWBZZ5009 Configuration error occurred when attempting to connect to system %1</td>
</tr>
<tr>
<td>-2147467259</td>
<td>CWBZZ5010 Communications error occurred with system %1</td>
</tr>
</tbody>
</table>

C.3.2 Table Messages

The following table shows the messages sent when a table error is detected.

Table 89. Table Messages

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Error.Description</th>
<th>Error.Number</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2147467259</td>
<td>CWBZZ5003 Nested transactions are not supported. Commit or roll back the current transaction before starting another transaction. CWBZZ5011 Isolation level %1 not recognized. Use *NONE, *CHG, *CS, *ALL or *YES.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-2147467259</td>
<td>CPF5009: Duplicate record key in member &amp;1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-2147467259</td>
<td>CPF5035: Data mapping error on member &amp;1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-2147467259</td>
<td>CPF3227: Record format &amp;1 in library &amp;2 not found.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-2147467259</td>
<td>CPF9820: Not authorized to use library &amp;1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-2147467259</td>
<td>CPF9822: Not authorized to use file &amp;1 in library &amp;2.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
C.3.3 Data Queue Messages

The following table shows the messages sent when a data queue error is detected.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Error.Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2147467259</td>
<td>CWBZZ5000 Command ‘%1’ not recognized or contains errors.</td>
</tr>
<tr>
<td>-2147467259</td>
<td>CWBZZ5001 Command ‘%1’ has specified key fields and the data queue was not created with a key.</td>
</tr>
<tr>
<td>-2147467259</td>
<td>CWBZZ5002 Maximum key size is %2. Key size of %3 specified in command ‘%1’ is too long.</td>
</tr>
<tr>
<td>-2147467259</td>
<td>CWBZZ5004 Maximum record length is %2. Record length of %3 specified in command ‘%1’ is too long.</td>
</tr>
<tr>
<td>-2147467259</td>
<td>CWBZZ5006 Data queue /QSYS.LIB/%2.LIB/%3.DTAQ on system %1 could not be opened.</td>
</tr>
<tr>
<td>-2147467259</td>
<td>CWBZZ5020 Data queue record format of %1 bytes is shorter than actual data length of %2 bytes for data queue /QSYS.LIB/%3.LIB/%4.DTAQ.</td>
</tr>
<tr>
<td>-2147467259</td>
<td>CWBZZ5021 Data queue record format of %1 bytes is longer than actual data length of %2 bytes for data queue /QSYS.LIB/%3.LIB/%4.DTAQ.</td>
</tr>
</tbody>
</table>

C.3.4 Command Messages

The following table shows the messages sent when a command error is detected.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Error.Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2147467259</td>
<td>CWBZZ5000 Command ‘%1’ not recognized or contains errors.</td>
</tr>
</tbody>
</table>

C.3.5 Program Messages

The following table shows the messages sent when a program error is detected.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Error.Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2147467259</td>
<td>CWBZZ5000 Command ‘%1’ not recognized or contains errors.</td>
</tr>
</tbody>
</table>

C.3.6 SQL Messages

The following table shows the messages sent when an SQL error is detected.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Error.Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2147467259</td>
<td>SQL0104: Token &amp;1 was not valid. Valid tokens: &amp;2.</td>
</tr>
</tbody>
</table>

A single example is provided in the table above. For information on the SQL statement messages, see the *DB2 for OS/400 SQL Programming: SC41-3611* manual. The information in that manual is in the following form:

SQL0104 SQLCODE -104 SQLSTATE 42601
Explanation: Token &1 was not valid. Valid tokens: &2.

The first seven characters of the error description text are the message ID (for example, SQL0104). The IBMDA400 provider does not support the return of the SQLCODE and SQLSTATE values.

C.3.7 Conversion Messages

The following table shows the messages sent when a conversion error is detected.

Table 94. Conversion Messages

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2147467259</td>
<td>CWBZZ5013 Cannot convert from CCSID %1 to CCSID %2.</td>
</tr>
<tr>
<td>-2147467259</td>
<td>CWBZZ5014 Value of parameter %1 could not be converted to the host data type.</td>
</tr>
<tr>
<td>-2147467259</td>
<td>CWBZZ5018 Data conversion error occurred.</td>
</tr>
</tbody>
</table>

Note

Many SQL errors come back as multiple messages to the error object. The message that has the most detail about the error may be the second or third message in the error object.
Appendix D. Special Notices

This publication is intended to help anyone who wants to develop AS/400 client/server applications. The information in this publication is not intended as the specification of any programming interfaces that are provided by the Client Access OLE DB Provider. See the PUBLICATIONS section of the IBM Programming Announcement for Client Access for Windows 95/NT for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM’s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM’s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer’s ability to evaluate and integrate them into the customer’s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AS/400
C/400
IBM
400

Client Access
DB2
OS/400
The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

Java and HotJava are trademarks of Sun Microsystems, Incorporated.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

Pentium, MMX, ProShare, LANDesk, and ActionMedia are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.
Appendix E. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

E.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 407.

- AS/400 Client/Server Performance Using the Windows Clients, SG24-4526
- An Implementation Guide for AS/400 Security, GG24-4200

E.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. Order a subscription and receive updates 2-4 times a year at significant savings.

<table>
<thead>
<tr>
<th>CD-ROM Title</th>
<th>Subscription Number</th>
<th>Collection Kit Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>System/390 Redbooks Collection</td>
<td>SBOF-7201</td>
<td>SK2T-2177</td>
</tr>
<tr>
<td>Networking and Systems Management Redbooks Collection</td>
<td>SBOF-7370</td>
<td>SK2T-6022</td>
</tr>
<tr>
<td>Transaction Processing and Data Management Redbook</td>
<td>SBOF-7240</td>
<td>SK2T-8038</td>
</tr>
<tr>
<td>Lotus Redbooks Collection</td>
<td>SBOF-6899</td>
<td>SK2T-8039</td>
</tr>
<tr>
<td>Tivoli Redbooks Collection</td>
<td>SBOF-6898</td>
<td>SK2T-8044</td>
</tr>
<tr>
<td>AS/400 Redbooks Collection</td>
<td>SBOF-7270</td>
<td>SK2T-2849</td>
</tr>
<tr>
<td>RS/6000 Redbooks Collection (HTML, BkMgr)</td>
<td>SBOF-7230</td>
<td>SK2T-8040</td>
</tr>
<tr>
<td>RS/6000 Redbooks Collection (PostScript)</td>
<td>SBOF-7205</td>
<td>SK2T-8041</td>
</tr>
<tr>
<td>Application Development Redbooks Collection</td>
<td>SBOF-7290</td>
<td>SK2T-8037</td>
</tr>
</tbody>
</table>

E.3 Other Publications

These publications are also relevant as further information sources:

- OS/400 Server Concepts, SC41-3740
- OS/400 Security - Reference, SC41-3302
- OS/400 Distributed Data Management, SC41-3307
How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at http://www.redbooks.ibm.com/.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Redbooks Web Site on the World Wide Web**
  
  http://w3.itso.ibm.com/

- **PUBORDER** – to order hardcopies in the United States

- **Tools Disks**
  To get LIST3820s of redbooks, type one of the following commands:

  TOOLCAT REDPRINT
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)

  To get BookManager BOOKs of redbooks, type the following command:

  TOOLCAT REDBOOKS

  To get lists of redbooks, type the following command:

  TOOLS SENDTO USDIST MKTOOLS MKTOOLS GET ITSOCAT TXT

  To register for information on workshops, residencies, and redbooks, type the following command:

  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1998

- **REDBOOKS Category on INEWS**

- **Online** – send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL

---

**Redpieces**

For information so current it is still in the process of being written, look at “Redpieces” on the Redbooks Web Site (http://www.redbooks.ibm.com/redpieces.html). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.
How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** – send orders to:
  - In United States: usib6fpl at ibmmail
  - In Canada: caibmbkz at ibmmail
  - Outside North America: dkibmbsh at ibmmail

- **Telephone Orders**
  - United States (toll free): 1-800-879-2755
  - Canada (toll free): 1-800-IBM-4YOU
  - Outside North America: (long distance charges apply)
    - (+45) 4810-1320 - Danish
    - (+45) 4810-1420 - Dutch
    - (+45) 4810-1540 - English
    - (+45) 4810-1670 - Finnish
    - (+45) 4810-1220 - French
    - (+45) 4810-1020 - German
    - (+45) 4810-1620 - Italian
    - (+45) 4810-1270 - Norwegian
    - (+45) 4810-1120 - Spanish
    - (+45) 4810-1170 - Swedish

- **Mail Orders** – send orders to:
  - IBM Publications Customer Support
    - In United States: 144-4th Avenue, S.W.
    - In Canada: Calgary, Alberta T2P 3N5
    - Outside North America: DK-3450 Allerød
    - USA

- **Fax** – send orders to:
  - United States (toll free): 1-800-445-9269
  - Canada: 1-800-267-4455
  - Outside North America: (+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States) or (+1) 408 256 5422 (Outside USA)** – ask for:
  - Index # 4421 Abstracts of new redbooks
  - Index # 4422 IBM redbooks
  - Index # 4420 Redbooks for last six months

- **On the World Wide Web**
  - Redbooks Web Site: http://www.redbooks.ibm.com

---

**Redpieces**

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (http://www.redbooks.ibm.com/redpieces.html). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.
IBM Redbook Order Form

Please send me the following:

<table>
<thead>
<tr>
<th>Title</th>
<th>Order Number</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

First name	Last name

Company

Address

City	Postal code	Country

Telephone number	Telefax number	VAT number

☐ Invoice to customer number

☐ Credit card number

Credit card expiration date	Card issued to	Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADO</td>
<td>ActiveX Data Objects</td>
</tr>
<tr>
<td>ADODB</td>
<td>ActiveX Data Objects for Database access</td>
</tr>
<tr>
<td>APPC</td>
<td>Advanced Program to Program</td>
</tr>
<tr>
<td>ASP</td>
<td>Active Server Pages</td>
</tr>
<tr>
<td>CGI</td>
<td>Communication Gateway Interface</td>
</tr>
<tr>
<td>COM</td>
<td>Component Object Model</td>
</tr>
<tr>
<td>DAO</td>
<td>Data Access Objects</td>
</tr>
<tr>
<td>DCOM</td>
<td>Distributed Component Object Model</td>
</tr>
<tr>
<td>DDM</td>
<td>Distributed Data Management</td>
</tr>
<tr>
<td>DPC</td>
<td>Distributed Program Call</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business Machines Corporation</td>
</tr>
<tr>
<td>IIS</td>
<td>Microsoft Internet Information Server</td>
</tr>
<tr>
<td>IPX</td>
<td>Internet Packet Exchange</td>
</tr>
<tr>
<td>ITSO</td>
<td>International Technical Support Organization</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>MDAC</td>
<td>Microsoft Data Access Components</td>
</tr>
<tr>
<td>MSIE</td>
<td>Microsoft Internet Explorer</td>
</tr>
<tr>
<td>MTS</td>
<td>Microsoft Transaction Manager</td>
</tr>
<tr>
<td>OCX</td>
<td>OLE Custom Control</td>
</tr>
<tr>
<td>OLE</td>
<td>Object Linking and Embedding</td>
</tr>
<tr>
<td>OLE DB</td>
<td>OLE for Database</td>
</tr>
<tr>
<td>OOP</td>
<td>Object Oriented Programming</td>
</tr>
<tr>
<td>RDO</td>
<td>Remote Data Objects</td>
</tr>
<tr>
<td>RDS</td>
<td>Remote Data Service</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Developers Kit</td>
</tr>
<tr>
<td>SNA</td>
<td>Systems Network Architecture</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transmission Control Protocol/Internet Protocol</td>
</tr>
<tr>
<td>VB</td>
<td>Visual Basic</td>
</tr>
<tr>
<td>VBX</td>
<td>Visual Basic eXtension</td>
</tr>
</tbody>
</table>
Index

A
abbreviations 411
Access centric security 353
acronyms 411
Active Server Pages 307
ActiveX 13
ADO 13
   Command object 21
   Connection object 20
   Error objects 32
   Errors collection 32
   Field object 30
   Fields collection 30
   Object Reference 45
   Parameter objects 35
   Parameters collection 35
   Recordset object 26
ADO 1.0  115
ADO 1.5  115, 168
ADO Error Report 114
ADO object model 14
ADO objects 18
ASP 307, 309, 311

B
bookmark 219

C
CGI 307
Client Access 39
Client Access APIs 4, 6, 7
Client Access OLE DB Provider 39
client/server Order Entry application 106
Client/Server Order Entry using DPC 105
Client/Server Order Entry using SQL 105
COM 10
commands 80, 193, 230, 259
Commitment Control 69
Commitment control 52, 70, 185, 222
commitment control 216
Company Database
   Customer 88
   District 87
   Item 88
   Order 88
   Order Line 88
   Stock 88
Component Object Model (COM) 11
CPW 362
Customer Table Layout(CSTMR) 98

D
DAO 4
Data Access Objects 4
Data Queues 5
data queues 75, 187, 224, 254
data queues OCX 6
data queues OLE 6
DataWindow 198
DDM 353
DDM/DRDA Server 40
Delphi 199
   Data Queue Application 224
   Distributed Program Call 231
   Record Level Access 214
   remote command 230
   SQL Application 200
   Stored Procedure 208
District Table Layout(Dstrct) 98
DPC 194, 231
DSPF 177
DSPFFD 177
DSPNETA 353

E
Error Handling 171, 207, 278
Error handling 244
error handling 44

F
fat client 105

H
Host Servers 40
   Starting Host Server jobs 41
Host servers 41
HTML 303, 307, 309, 312
HTTP server 311

I
IBMDA400 43, 163
IIS 307, 311
IP 41
IPX 41
Item Table Layout(ITEM) 100

J
journals 71

K
Keyed record search 68

L
Lotus 1-2-3 279, 281
Lotus Notes 262
   Data Queue Application 254
   Distributed Program Call 260
   Record Level Access 249
   Remote Command 259
A Fast Path to AS/400 Client/Server Using AS/400 OLE DB Support

SQL Example 240
Stored Procedure 245
Lotus Notes agent 259
LotusScript 239
Data 239
LotusScript Script Editor 280

M
Member Locking 179, 216
Microsoft Internet Explorer 304
Microsoft Internet Information Server 307
Microsoft Jet database engine 3
MS Access 279, 296
MS Excel 279, 287
MSDASQL 52
multiple result sets 53

N
naming conventions 43, 179
Netscape Navigator 304

O
Object Level Security 351
Object Linking and Embedding 10
OCX 12
ODBC API 3
ODBC Direct 3
OLE DB 8, 9, 13, 39, 42
Optimized Data Queue Server 40
Optimized Database Server 40
ORDENTR 89
Order Line Table Layout(ORDLIN) 100
Order Line Table Layout(ORDLIN) 100
Order Table Layout(ORDERS) 100
OS/400 Registration Facility 356

P
parameter marker 276
Parameter markers 50
parameter markers 167, 204
PARTS database file 46, 164, 200, 272
PCSACC 353
performance
  Function Timings 368
    recommendations 370
  Response Time Comparison 367
  Scalability 370
  Tips and Techniques 369
  Usage and Trade-offs 370
  performance considerations 361
PowerBuilder 163
  Data Queue Application Example 187
  Distributed Program Call 194
  Record Level Access 177
    remote command 193
  SQL statements 164
  Stored Procedure 171
    program call 194, 231, 260
Program Calls 80
program parameters 196
PRTODERR 89

R
RDO 4
RDS 312, 313, 318, 323
  DataControl 315
  DataFactory 315
  RDSServer 314
Record Level Access 41
record level access 63, 177, 214, 249
Record level Access-PRE-V4R2M0 41
Record Locking 69, 180, 216
Remote Command and Distributed Program Call Server 40
Remote Data Objects 4
Restrictions 52, 72, 177, 186, 197, 214, 223, 237, 259
result sets
  multiple result sets 53
RPG
  order entry application
    Customer Transaction Flow 89
    PRTODERR 89
  RPG Application Flow 89
  RPG order Entry application
    ORDENTR 89
  RPG order entry application 87, 104
    SLTCUSTR 89
    SLTPARTR 89
    Tables 98

S
SDK 39
security 351
  Access centric security 353
  Adopted authority 352
  Authority scope 352
  Group Authority 352
  Limit Capabilities (LMTCPB) 353
  Object Level Security 351
  Object Ownership 352
  Primary Group Authority 352
  Private Authority 352
  Public Authority 351
  User Authority 352
  SeekOption 67
  server-side includes 308
    SLTCUSTR 89
    SLTPARTR 89
  SNA 41
  SNA Server 4.0 325
  Speed 363
    Benchmark Results 366
    Speed Application 361
SQL
  prepared statements 50
  Static Cursor 52
  SQL statements 47, 164, 200, 240
sql statements 272
Stock Table Layout(Stock) 100
stored Procedure 52
stored procedure 171, 208, 245
Stored Procedures 52
STRHOSTSVR 41
STRTCPVR 41
System Object Model (SOM) 11

T
TCP/IP 41, 311
The Software Development Toolkit for ActiveX and OLE DB 39
thin client 105
Thor
  Developing with the Microsoft OLE DB Provider for AS/400 and VSAM 331
  Microsoft OLE DB Provider for AS/400 and VSAM 325
  three tier approach 325
tier
  Choosing an implementation for the Middle Tier 341
  Client Access OLE DB Provider 337
  Programming in a Three-Tier Environment 339

U
Updatability Property 69
User exit programs 355

V
VBScript 304, 309
VBXs 12
Visual Basic 107
  commands 80
  CommandText property 49
  Connection Object 48
  CreateParameter 51
data queue 75
  Program Calls 80
  record level acces 63
  SQL statements 47
  Stored Procedures Calls 52
  VBXs 12
Visual Basic Wizard 54, 72
Visual FoxPro 271
  SQL Application 272

W
web server 308
ITSO Redbook Evaluation

A Fast Path to AS/400 Client/Server Using AS/400 OLE DB Support
SG24-5183-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. Please complete this questionnaire and return it using one of the following methods:

- Use the online evaluation form found at http://www.redbooks.ibm.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?

- Customer
- Business Partner
- Solution Developer
- IBM employee
- None of the above

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction __________

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___
If no, please explain:

______________________________________________________________________________

______________________________________________________________________________

______________________________________________________________________________

What other redbooks would you like to see published?
______________________________________________________________________________

______________________________________________________________________________

______________________________________________________________________________

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)
______________________________________________________________________________

______________________________________________________________________________

______________________________________________________________________________

______________________________________________________________________________