AS/400e series

Client Access for Windows 95/NT ODBC User's Guide

Version 3

SC41-3535-01

AS/400e series

Client Access for Windows 95/NT ODBC User's Guide

Version 3

SC41-3535-01

Note

Before using this information and the product it supports, be sure to read the information in Appendix E, “Notices” on page E-1.

Second Edition (February 1998)

| This edition replaces SC41-3535-00.

| This edition applies to version 3, release 1, modification 3 of Client Access for Windows 95/NT (5763-XD1) and to all
| subsequent releases and modifications until otherwise indicated in new editions.

Contents

About Client Access for Windows 95/NT ODBC User's Guide (SC41-3535) . . vii
Who should read this book vii
AS/400 Operations Navigator vii
Prerequisite and related information viii
Information available on the World Wide Web viii
How to send your comments viii

Part 1. Introduction and Setup for ODBC

Chapter 1. Introduction to ODBC 1-1
Why is ODBC needed? 1-1
How does ODBC work? 1-2

Chapter 2. Setting up the Client Access ODBC Driver 2-1
Adding the local AS/400 system to the RDB directory 2-1
Selecting the Client Access ODBC driver 2-1
Setting up the ODBC data source 2-2
Configuring options for the ODBC data source 2-4
Setting Server options 2-4
Setting Package(s) options 2-6
Setting Performance options 2-8
Setting Language options 2-10
Setting Other options 2-11
Setting Translation options 2-13
Setting Format options 2-14
RDB directory 2-15
Data Source Options 2-15
Server options 2-15
Package(s) options 2-17
Performance options 2-18
Translation options 2-18

Part 2. ODBC Performance Considerations

Chapter 3. Performance-Tuning Client Access ODBC 3-1
Introduction to performance 3-1
Introduction to client/server performance 3-1
The performance architecture of the Client Access ODBC driver 3-2
ODBC registry settings 3-3

Chapter 4. Performance Considerations of Common End-User Tools 4-1
Common tool behaviors that hurt performance 4-1
Query Tool A 4-1
Query Tool B 4-2

Query Tool C - Your Worst Possible Nightmare 4-3

Chapter 5. SQL Performance 5-1
General considerations 5-1
Database design 5-2
Normalization 5-2
Table size 5-4
Indexes 5-5
Matching attributes of join fields 5-5
Optimizer 5-6
Cost estimation 5-6
Optimizer decision-making rules 5-8

Chapter 6. Coding Directly to the ODBC APIs 6-1
Calling ODBC Functions 6-1
Basic Application Steps 6-1
Basic Application Flow 6-1
Establishing ODBC Connections 6-2
Executing ODBC Functions 6-3
Executing Prepared Statements 6-4
Converting Strings and Arrays of Byte 6-6
Transaction Control (Commit) 6-7
Retrieving Results 6-8
Calling Stored Procedures 6-9
Calling Stored Procedures using Result Sets 6-10
Extended Fetch 6-13
Block Insert 6-15
Ending ODBC Functions 6-19
Comparison of ODBC Techniques 6-20
VB 4.0: The compromise between Jet and ODBC APIs 6-21

Chapter 7. Stored Procedures 7-1
Examples of stored procedures 7-3
Stored procedure calls from C with return values 7-3
Stored procedure calls from C with result sets 7-4
Host RPG source 7-5
Running CL commands through SQL stored procedures and ODBC 7-6
Stored procedure calls from PowerBuilder with no return values 7-8
Host PL/I source 7-8
Stored procedure calls from PowerBuilder with return values 7-9
Host COBOL source 7-20
Stored procedure calls from PowerBuilder with result sets 7-22
Host COBOL source 7-28
Stored procedure calls from Visual Basic with return values 7-30
Host COBOL source 7-33
Stored procedure calls from Visual Basic with result sets 7-35

Chapter 8. Blocked Inserts 8-1

Examples of blocked insert calls from C 8-2

Chapter 9. Catalog Functions 9-1

Chapter 10. Exit Programs 10-1
Sample user exit programs 10-1
Exit program parameter formats 10-9

Part 3. Appendixes

Appendix A. Open Database Connectivity Additional Information A-1
Advanced use A-1
| Connection Strings A-1
SQL statements, grammar and limitations A-3
ODBC API functions A-4
Implementation issues A-5
Data types A-5
Error messages A-7
Sample ODBC conversation A-8

Appendix B. Enabling Tables to Be Updated B-1
PC application locking B-1
Creating a unique index B-1

Appendix C. ODBC Troubleshooting C-1
Unable to connect to the AS/400 . . . . . C-1
Understanding the database server program C-1
Checking server status (TCP/IP and IPX connections only) C-1
Verify that the proper subsystems are running C-1
Verify that the proper prestart jobs are running C-2
Further TCP/IP and IPX considerations C-2
Verify that the host server program product is installed C-3
Identifying the ODBC job and joblog C-3
How to find your job C-3
How to generate a joblog C-4
How to find the joblog C-4
| Common errors C-5
| Communication errors C-5
| SQL errors C-6
| Delphi and PowerBuilder errors C-8
| Stored procedure errors C-8
| Incorrect output and unpredictable errors C-9
| Gathering information for IBM support C-10
| Recording the OS/400 version and cumulative PTF level C-10
| Recording the version of the Client Access for Windows 95/NT ODBC driver C-10
| Recording the version of the ODBC driver manager C-11
| Recording the type of connection router C-11
| Generating and retrieving the ODBC joblog C-11

Recording additional information C-11
Diagnostic and Performance Tools C-13
Error Message Help C-13
ODBC trace utilities C-13
AS/400 communications trace C-13
AS/400 job traces C-13
AS/400 Performance Tools C-14
AS/400 job log C-14

Appendix D. Security Issues with Client Access and ODBC D-1
Overview D-1
Object Level Security D-1
Program Adopted Authority D-1
Referential Integrity D-2
Other D-2
Client Access Concepts D-3
ODBC D-4
Security Tools D-6
Additional Assistance D-6

Appendix E. Notices E-1
Programming Interface Information E-2
Trademarks E-2

Index X-1

About Client Access for Windows 95/NT ODBC User's Guide (SC41-3535)

This guide contains information that will help you install, configure, and use the Client Access ODBC Driver that enables ODBC-compliant client applications to exchange ODBC data with your AS/400 system.

This guide does not provide comprehensive technical information about the SQL language or database programming concepts.

Who should read this book

The reader of this guide should be an end user or PC administrator who has at least a moderate knowledge of database programming concepts and the SQL programming language in particular.

AS/400 Operations Navigator

AS/400 Operations Navigator is a powerful graphical interface for Windows 95/NT clients. With AS/400 Operations Navigator, you can use your Windows 95/NT skills to manage and administer your AS/400 systems. You can work with database administration, file systems, Internet network administration, users, and user groups. You can even schedule regular system backups and display your hardware and software inventory. Figure 0-1 shows an example of the display.

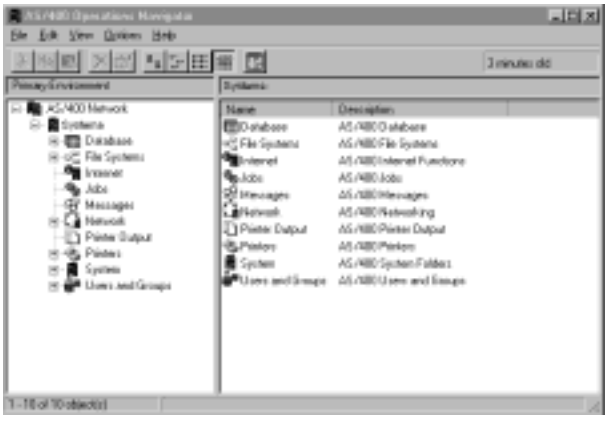


Figure 0-1. AS/400 Operations Navigator Display

IBM recommends that you use this new interface. It is simple to use and has great online information to guide you.

You can access the AS/400 Operations Navigator from the Windows 95 Taskbar or from the Client Access folder. You can also drag this icon to your desktop for even quicker access.

While we develop this interface, you will still need to use the familiar AS/400 “green screens” to do some of your tasks. You can find information to help you in this book and online.

Prerequisite and related information

For information about Advanced 36 publications, see the *Advanced 36 Information Directory*, SC21-8292, in the AS/400 Softcopy Library.

For information about other AS/400 publications (except Advanced 36), see either of the following:

- The *Publications Reference*, SC41-5003, in the AS/400 Softcopy Library.
- The AS/400 online library is available on the World Wide Web at the following uniform resource locator (URL) address:

  `http://as400bks.rochester.ibm.com/`

Information available on the World Wide Web

In addition to the AS/400 online library on the World Wide Web, you can access other information from the AS/400 Technical Studio at the following URL address:

`http://www.as400.ibm.com/techstudio`

Further Client Access ODBC information can be found in the software support knowledge base located at the AS/400 service web site:

`http://www.as400service.ibm.com/as400/service.html`

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book or any other AS/400 documentation, fill out the readers' comment form at the back of this book.

- If you prefer to send comments by mail, use the readers' comment form with the address that is printed on the back. If you are mailing a readers' comment form from a country other than the United States, you can give the form to the local IBM branch office or IBM representative for postage-paid mailing.
- If you prefer to send comments by FAX, use either of the following numbers:
  - United States and Canada: 1-800-937-3430
  - Other countries: 1-507-253-5192
- If you prefer to send comments electronically, use this network ID:
  - IBMMAIL, to IBMMAIL(USIB56RZ)
  - IDCLERK@RCHVMW2.VNET.IBM.COM

Be sure to include the following:

- The name of the book.
- The publication number of the book.
- The page number or topic to which your comment applies.

Part 1. Introduction and Setup for ODBC

Chapter 1. Introduction to ODBC

Why is ODBC needed?

You are a technical assistant to the Chief Executive Officer of your company. You have been given the task of pulling together data from every major department in your organization. To make things worse, you only have six hours to accomplish this three-day task.

You politely remind your boss that gathering this information is normally a three-day process. You remind him that your accounting people use a PC-based product on a local network. Your purchasing people use a UNIX-based server on a different network. Your manufacturing people use an AS/400-based control system. And, your Q&A people use a DB2-based product on the company's mainframe.

Your boss informs you this is your problem, not his. He needs the numbers in six hours for an emergency board meeting. What are you going to do to be able to bring together so much data, from so many different sources?

You go back to your office and start compiling a list of all of the people you need to contact, all of the reports you need generated. And the approximate time you need the results by so you can reenter the data back into your spreadsheet for analysis. As you do this you ask yourself, "Isn't there an easier way than this?"

In 1992 the Microsoft Corporation introduced a new programming interface based on work that was being done by the SQL Access Group on the Call Level Interface (CLI) specification. Their proposal, called Open Database Connectivity (ODBC), stated that instead of having multiple applications that access multiple databases, why not build applications to access one standard database interface. From this standard interface, Microsoft stated that database developers could build whatever back-end code needed to speak to the specific database. This departure from the multiple applications to multiple databases style of thinking has transformed the way users access and use company data.

As you start up your trusty copy of Microsoft Excel, you are still asking yourself, "Isn't there an easier way than this?" You look through the Excel online manuals for some help and notice a reference to something called External Databases. You wonder, "Can this help me?" You do a little reading and learn that through the ODBC interface, Excel can access data from many different databases. You quickly do a little searching and find that there are ODBC drivers for each database from which you need information. You set up different drivers for each of your organization's databases. You fire up Excel and start analyzing data from all over your organization. With time to spare, you walk into your boss's office and present to him the necessary information for his board meeting.

Not only have you pleased your boss, but you have found one of the most powerful business tools available today, ODBC. In today's business world, a three-day turn-

around of information is not good enough. You need instantaneous access to all of your information...no matter where the data resides. ODBC and ODBC-compliant applications provide that instantaneous access to data.

How does ODBC work?

Back in your office, you decide that ODBC worked great. You decide to learn more about ODBC and about how it works.

Your research reveals that ODBC is a standard interface for database connectivity that is defined by the Microsoft Corporation. ODBC sets the standard interface to any database as Structured Query Language (SQL). You also discover that the ODBC architecture consists of four main components:

- ODBC Application
- ODBC Driver Manager
- ODBC Driver
- Data Source

These components are mentioned frequently in your research. So, as you learn about the ODBC components, you begin to build a picture of what each component is and how the components interact.

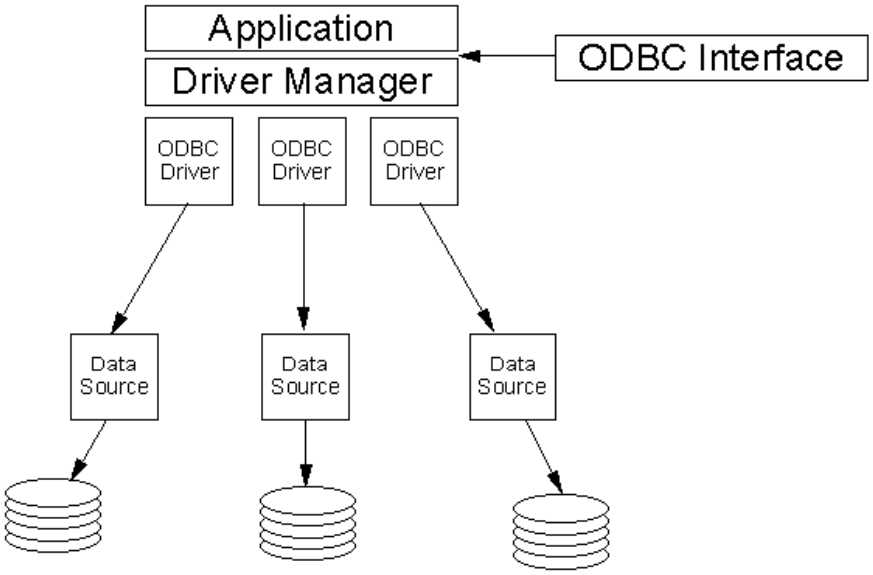


Figure 1-1. ODBC Components

ODBC Application The ODBC application calls ODBC functions to submit SQL requests and retrieve results.

ODBC Driver Manager The ODBC Driver Manager loads ODBC drivers and routes function calls from the applications to the proper ODBC driver.

ODBC Driver The ODBC Driver processes ODBC function calls, submits requests to the database management system, and returns results to the driver manager.

Data Source The Data Source is the component to which applications connect. The Data Source contains the data that the user of the application wants to access, the database management system and its associated operating system, and any network used to access the database management system.

Now that you have an understanding of ODBC and how it works, you are confident that, with the aid of this new tool, you can overcome any future crisis in a far more efficient manner.

Chapter 2. Setting up the Client Access ODBC Driver

Using the Client Access ODBC Driver allows your ODBC client applications to effectively share data with each other and with the AS/400. To set up the Client Access ODBC Driver, you must perform the following tasks:

1. Add the local AS/400 system to the relational database (RDB) directory.
2. Select the Client Access ODBC driver.
3. Set up the ODBC data source.
4. Configure formatting and performance options for the ODBC data source.

Adding the local AS/400 system to the RDB directory

To use ODBC with some applications, the local AS/400 must appear in the RDB directory. To add the name of the local AS/400 machine to the RDB directory, complete the following steps:

1. From the AS/400 command prompt, issue the CL command Add Relational Database Directory Entry (ADDRDBDIRE).
2. When the ADDRDBDIRE screen prompts you for values, enter the name of the AS/400 as the Relational Database parameter.
3. Enter *LOCAL as the Remote Location parameter.

For more information about the RDB directory, see “RDB directory” on page 2-15.

Selecting the Client Access ODBC driver

To access ODBC data, you must select an ODBC driver. To select the Client Access ODBC Driver, complete the following steps:

1. To start ODBC Administration from the Windows 95/NT Taskbar, click Start and choose Programs, IBM AS400 Client Access, ODBC Administration.

   Tip: You can also double-click the IBM AS400 Client Access icon on the desktop, if you have one, and then select ODBC Administration.

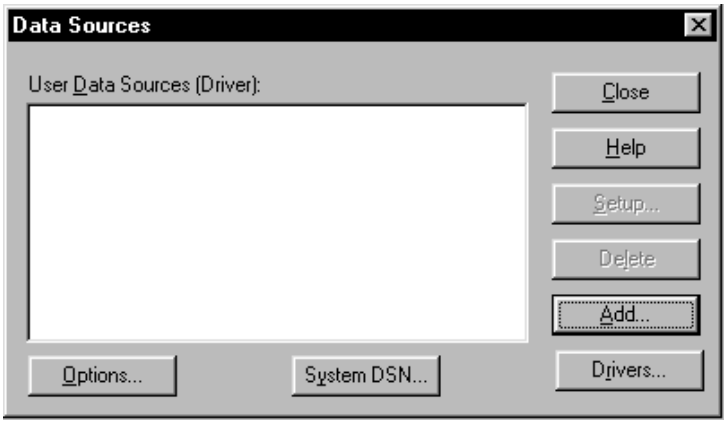


Figure 2-1. ODBC Administration Data Sources dialog

2. In the Data Sources dialog, click Add.

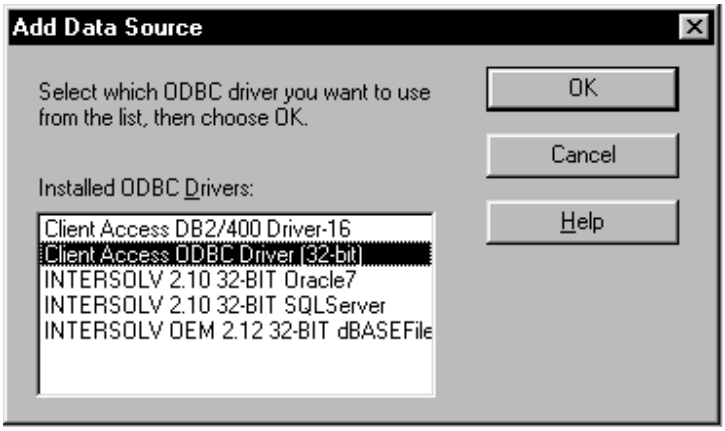


Figure 2-2. ODBC Administration Add Data Source dialog

3. In the Add Data Source dialog, select the Client Access ODBC Driver [32-bit] from the Installed ODBC Drivers selection box.
4. Click OK to finish the selection and activate the Client Access ODBC Setup [32-bit] dialog.

Setting up the ODBC data source

You must set up the data source before ODBC can access and manipulate data. Entering the required information that is in the **General** tab of the Client Access ODBC Driver [32-bit] Setup dialog sets up the ODBC data source.

Detail: If you complete only the General tab, the Client Access ODBC driver supplies default values for the information in the remaining tabs. To configure other

options, see “Configuring options for the ODBC data source” on page 2-4. For information about some of the available options, see “Data Source Options” on page 2-15.

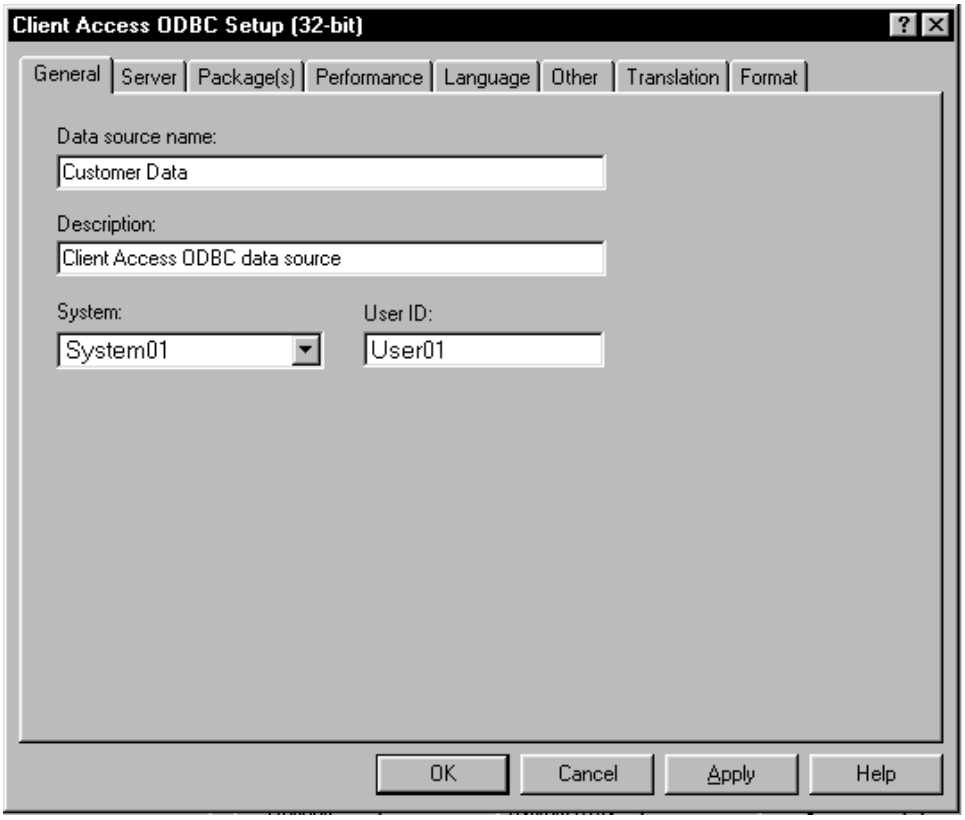


Figure 2-3. Client Access ODBC Setup [32-bit] dialog — General

To set up the Client Access ODBC Driver, complete the following steps:

1. Enter a descriptive **Data source name** in the textbox. This is required information that enables the Client Access ODBC driver to access AS/400 data.

   Detail: The data source name can be up to 32 characters, must start with an alphabetic character and cannot include the following characters: left bracket ([), right bracket (]), left brace ({), right brace (}), left parenthesis (( ), right parenthesis ( )), question mark (?), asterisk (*), equal sign (=), exclamation point (!), atsign (@), or semicolon (;).

2. Enter a **Description** of the data that is found in the data source. This information is optional but useful in identifying the data source when modifying attributes at a later time.

3. Select the configured AS/400 **System** that contains the data source. Click the arrow to select another system. This is required information to enable the Client Access ODBC driver to access AS/400 data.
4. Enter your **User ID** for connecting to the AS/400 system with the SQLDriverConnect prompt. This information is optional.

   Detail: When a Client Access connection is already established, SQLDriverConnect allows ODBC to use that established connection with the default User ID/password. Entering a User ID here requires the user to provide a User ID/password before ODBC can use the established connection.

Configuring options for the ODBC data source

After you set up the data source, the remaining tabs (Server, Package(s), Performance, Language, Other, Translation, and Format) allow you to configure additional data source options. The Client Access ODBC driver supplies default values for those options which are not changed. For additional information about the available data source options, see “Data Source Options” on page 2-15.

Setting Server options

The **Server** options tab allows you to set the default library or libraries and the level of commitment control. For more information about default libraries, see “Default libraries” on page 2-15. For more information about commitment, see “Commitment control” on page 2-16.

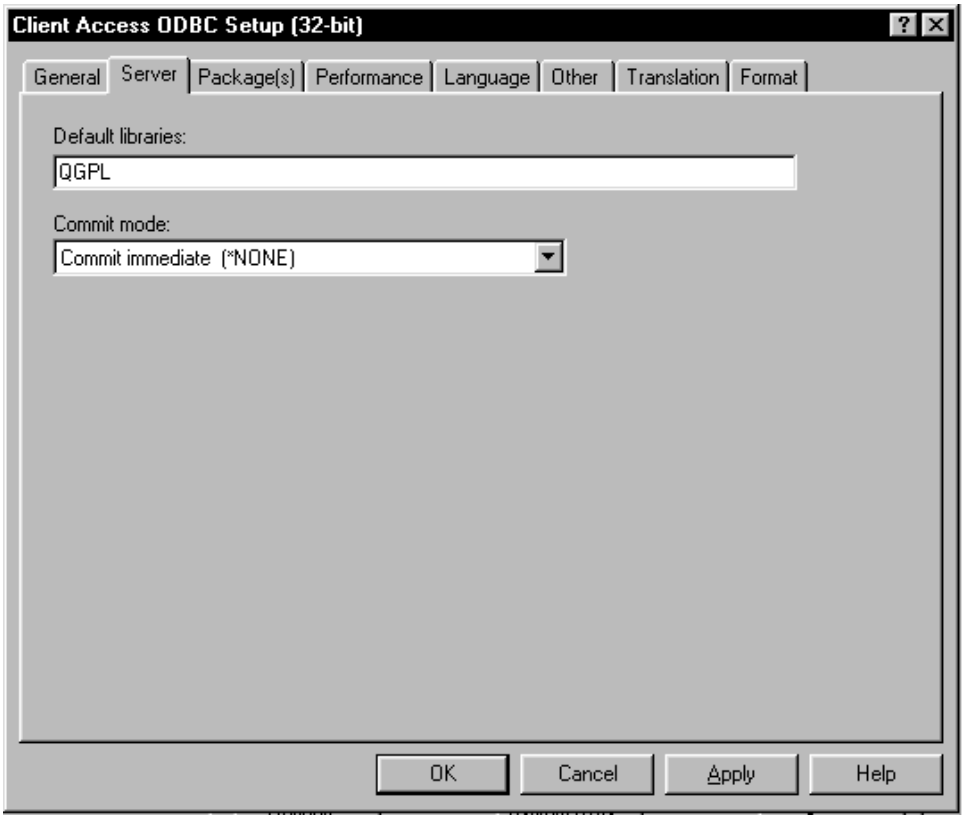


Figure 2-4. Client Access ODBC Setup [32-bit] dialog — Server

To configure the **Server** options, complete the following steps:

1. Enter the AS/400 **Default libraries** that you will use during connections to this data source. Separate names with either commas or spaces.

   Details:

   - To replace the current default library list or to add libraries within an existing list, specify the complete library names.
   - To add a library to the front or end of an existing user library list, use *USRLIBL to specify the existing user library list and add the new libraries in the appropriate order.

Examples:

a. "LIB1, LIB2, LIB3" replaces the user library list with the three specified libraries. The default collection is LIB1.

b. ", LIB1, *USRLIBL, LIB2" adds LIB1 to the front of the user library list, adds LIB2 to the end of the list, and specifies no default collection.

2. Select the **Commit mode** that will be used if the application does not specifically set it. Click the down arrow to select a different commit mode.

Setting Package(s) options

The **Package(s)** options tab allows you to enable and control implementation of extended dynamic support. For more information about packages and extended dynamic support, see “Package(s) options” on page 2-17.

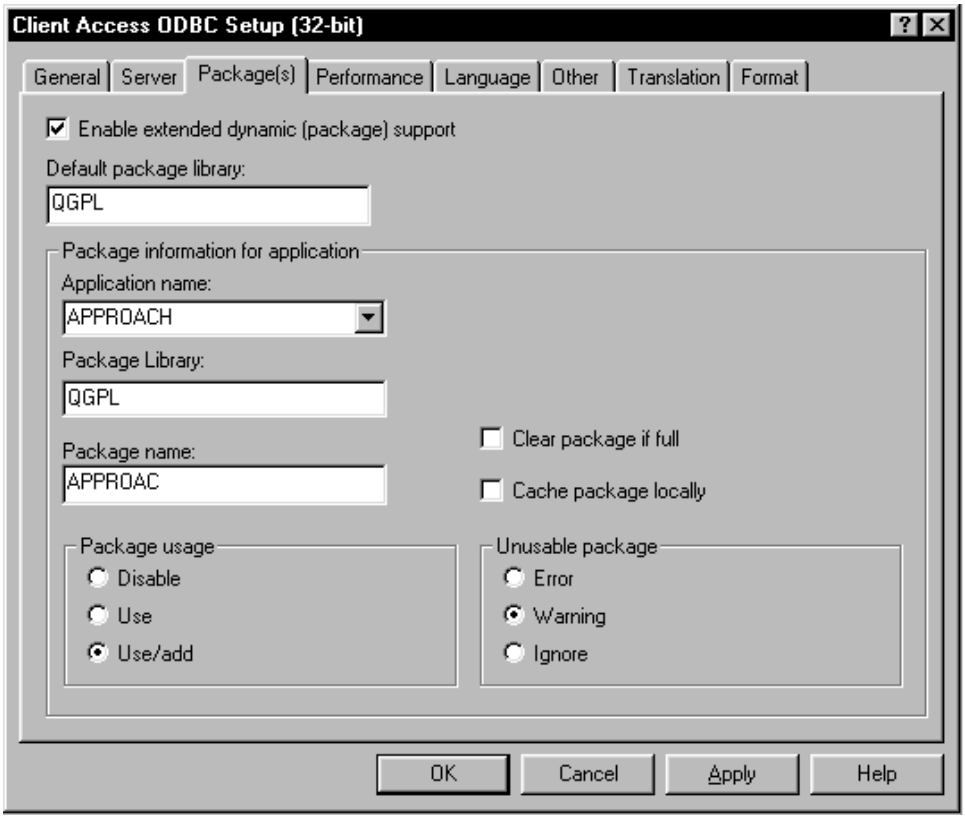


Figure 2-5. Client Access ODBC Setup [32-bit] dialog — Package(s)

To configure the **Package(s)** options, complete the following steps:

1. Click **Enable extended dynamic (package) support** to enable or disable extended dynamic support.

2. Enter the name of your preferred **Default package library**, if you have one.

   Detail: The following options are set initially when a client ODBC application is run for the first time after enabling extended dynamic support. After these options have been set for an application, you may change them at any time by using the Client Access ODBC Setup [32-bit] dialog.

3. Select the **Application name** for which you want to customize the package options. Click the down arrow to select from available applications.

   Detail: Selecting an application name enables the remaining options on the Package(s) options tab.

4. Type the AS/400 library in which the **Package library** for the selected application resides or will be created. This information is optional except when configuring shared packages. See “Private and shared packages” on page 2-17.

   Detail: Selecting a package library for a particular application will override the default package library setting.

5. Enter a **Package name** for the package for the selected application. This information is optional.

   Detail: The package name can be up to seven characters long. The ODBC driver adds a three character suffix to the package name.

6. Click **Clear package if full** to enable or disable deletion of all the data in the package if it becomes full. When the check box is unselected, the package becomes read-only when it is full.

7. Click **Cache package locally** to enable or disable caching a copy of the SQL package on your PC. For more information, see “Local package caching” on page 2-17.

8. Click a **Package usage** option to specify how the ODBC driver uses the package for this application:

   - **Disable** — The ODBC driver will not use a package for this application.
   - **Use** — The ODBC driver uses an existing package in read-only mode.
   - **Use/Add** — The ODBC driver uses an existing package (or creates a package if one does not exist), adds new statements to the package, and reuses existing statements.

9. Click an **Unusable package** option to specify the kind of return code and message that are returned by the ODBC driver when the package is not usable:

   - **Error** — The ODBC driver returns an error when a package is unusable.
   - **Warning** — The ODBC driver returns a warning when a package is unusable.
   - **Ignore** — The ODBC driver does not return a message when a package is unusable.

Setting package options for multiple users

The administrator can set up an application that uses a fixed number of SQL statements to have a single package that all users effectively share without contention. To set package options for multiple users, complete the following steps:

1. Use the Client Access ODBC Administrator to set up and configure the data source. See “Setting up the ODBC data source” on page 2-2 and “Configuring options for the ODBC data source” on page 2-4 for instructions on how to set up and configure a data source.
2. Run the client application. The ODBC driver creates the package, if a package does not already exist.
3. Use the client application to add the desired SQL statements to the package, then exit the application.
4. From the **Package(s)** tab of the Client Access ODBC Setup [32-bit] dialog, click **Use** under **Package Usage**. This will ensure that the ODBC driver uses the existing package in read-only mode.
5. Configure identical**Package(s)** options for each user of the shared package.

   Detail: Be sure to select the same **Application name**, enter the same **Package library** and **Package name** and click **Use** under **Package Usage**.

Setting Performance options

The **Performance** tab allows you to set ODBC options for the AS/400 database server that can enhance the performance of ODBC applications. For more information about performance options, see “Performance options” on page 2-18.

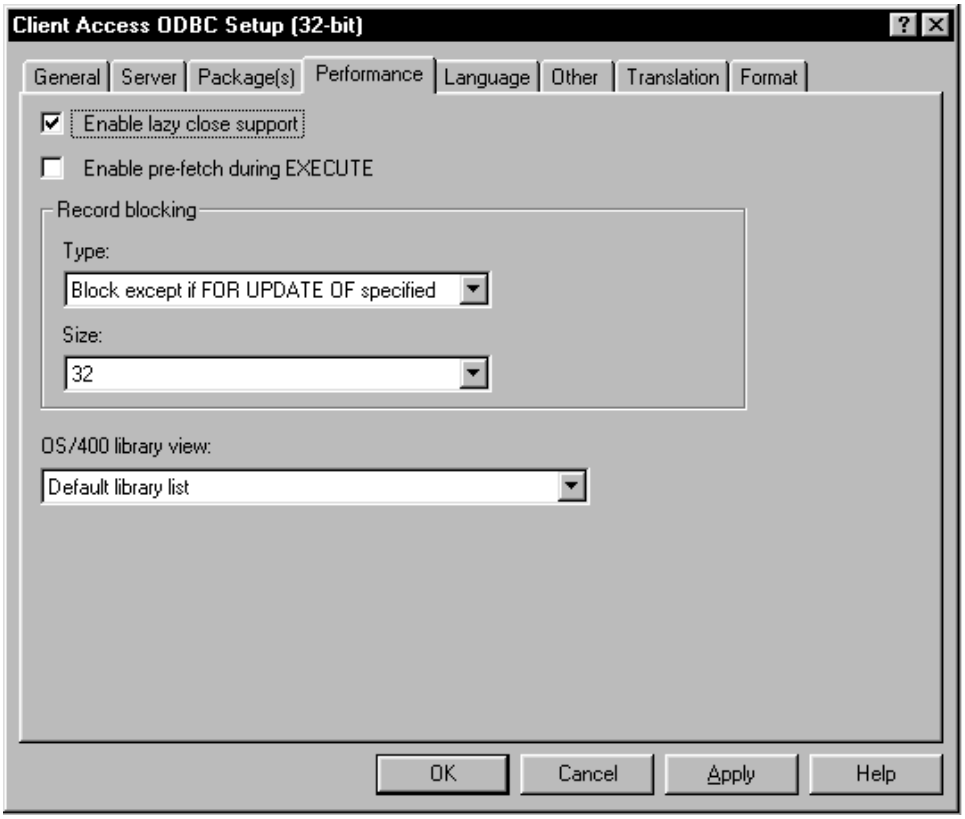


Figure 2-6. Client Access ODBC Setup [32-bit] dialog — Performance

To configure the **Performance** options, complete the following steps:

1. Click **Enable lazy close support** to enable or disable this option. See “Lazy close support” on page 2-18 for more information.
2. Click **Enable pre-fetch during EXECUTE** to enable or disable this option.

   Detail: Leave this option unchecked if your application uses SQL ExtendedFetch or if you are not sure.
3. Select the **Type** of record blocking under which the driver retrieves multiple rows of data from the AS/400 system. Click the down arrow to select a different setting.

   Detail: The types of record blocking available are:

   - Record blocking is disabled.
   - Record blocking enabled only for SELECT statements that end with an explicit "FOR FETCH ONLY" clause.
   - Record blocking enabled for everything except SELECT statements containing a "FOR UPDATE OF" clause.

4. Select the **Size** in kilobytes of the block of rows that is used when record blocking is enabled. Click the down arrow to select a different size.

   Detail: Size is measured in kilobytes (1kb = 1024 bytes). The block size is divided by the size of one row of data to determine the number of rows that are processed in one request.

5. Select a**OS/400 library view** to specify the set of libraries that is searched when the SQLTables function returns a list of table owners.

   Detail: In general, this option should be left at the default setting.

Setting Language options

The Language tab allows you to set options for the ODBC driver that affects certain types of sorting and matching that are based on language.

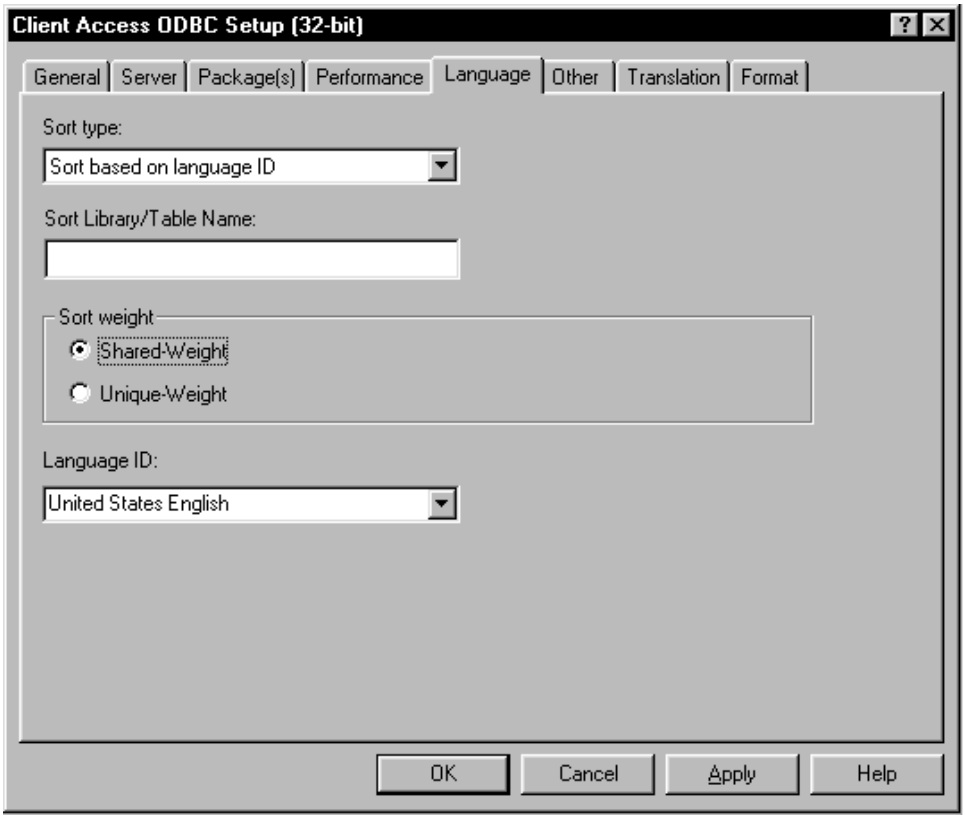


Figure 2-7. Client Access ODBC Setup [32-bit] dialog — Language

To configure the **Language** options, complete the following steps:

1. Select whether the **Sort type** (sort sequence) is based on hexadecimal value, job profile, language ID, or sort sequence table. Click the down arrow to select a new sort type.

Details:

a. Job profile is not supported in the current release and, if chosen, is treated as hexadecimal.

b. Selecting "Sort based on language ID" enables the **Sort weight** and **Language ID** options.

c. Selecting "Sort based on specified table" enables the **Sort Library/Table Name** text box.

2. Enter the **Sort Library/Table Name** (library and file names) of a sort sequence table on an AS/400 to be used as the sort sequence.
3. Select one of the following **Sort Weight** options.
   - **Shared-Weight** — Sort uppercase and lowercase representations of each letter as the same letter.
   - **Unique-Weight** — Sort uppercase and lowercase representations of each letter as unique letters.
4. Select the **Language ID** that is used to assign a sorting sequence. Click the down arrow to select a different language.

Setting Other options

The Other options tab allows you to select the type of connection, type of object description, and cursor preferences.

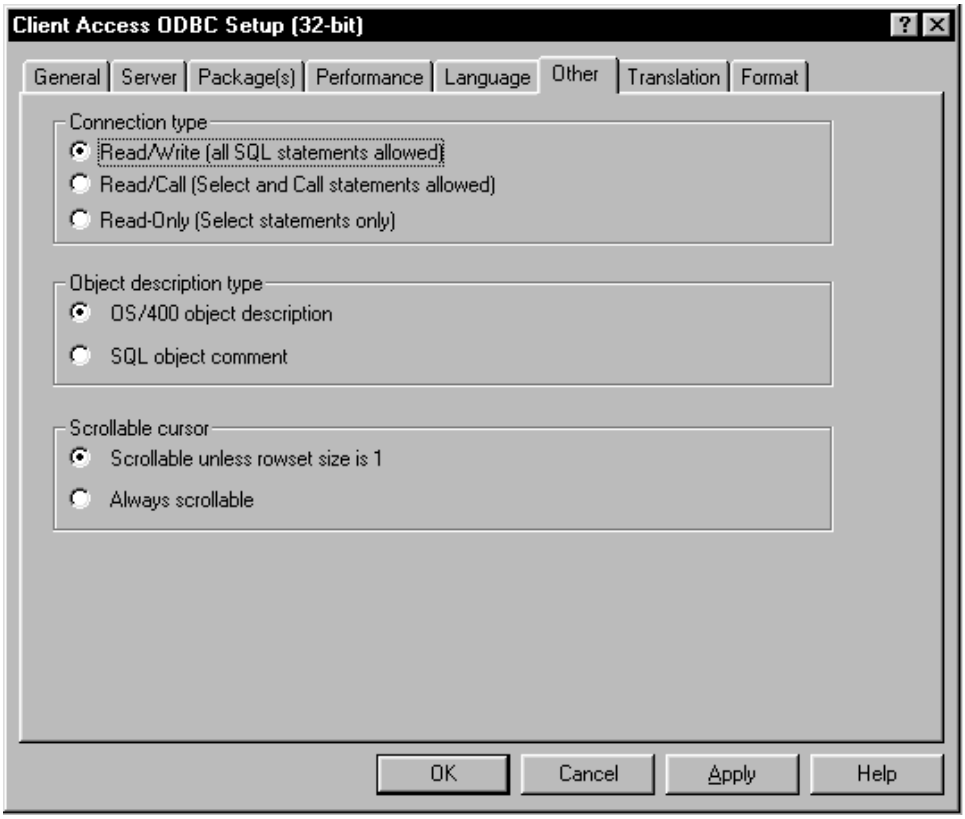


Figure 2-8. Client Access ODBC Setup [32-bit] dialog — Other

To configure the **Other** options, complete the following steps:

1. Click the preferred **Connection type**. The default selection is Read/Write.
2. Click the preferred **Object description type** to determine the type of value you want ODBC catalog APIs to return in the REMARKS column.
3. Click one of the following **Scrollable cursor** options to enable or disable scrollable cursors when the rowset size is 1.
   - **Scrollable unless rowset size is 1** —Cursors with a rowset size of 1 are not scrollable.
   - **Always scrollable** — Cursors are always scrollable.

   Detail: Some client applications expect the cursors they use to be scrollable, but the AS/400 normally treats a cursor with a rowset size of one as non-scrollable. Clicking "Always scrollable" forces all cursors, including those with a rowset size of 1, to be scrollable.

Setting Translation options

The **Translation** options affect how the AS/400 and the ODBC driver translate data. For more information about translation options, see “Translation options” on page 2-18.

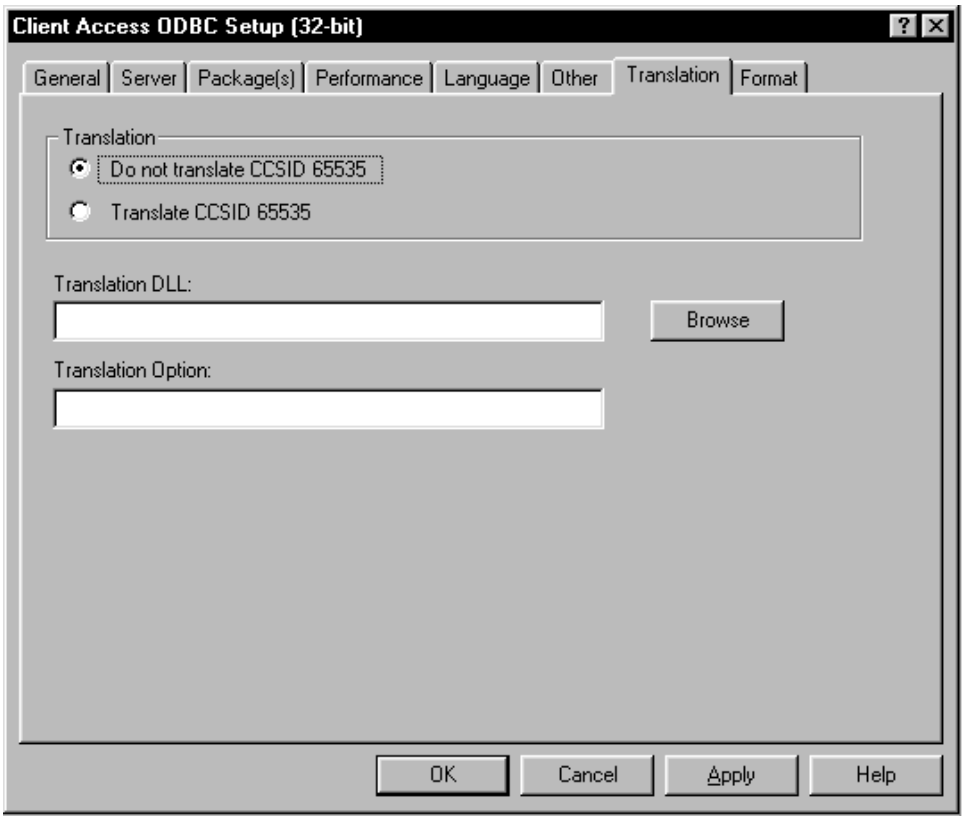


Figure 2-9. Client Access ODBC Setup [32-bit] dialog — Translation

To configure the **Translation** options, complete the following steps:

1. Click a **Translation** option to enable or disable the translation of data stored in columns on the AS/400 system with an explicit coded character set identifier (CCSID) of 65535. The default is "Do not translate CCSID 65535."
2. Enter a **Translation DLL** filename that will be used by the ODBC driver. This information is optional because the ODBC driver handles all normal character and numeric conversions.

   Tip: Click **Browse** to locate an ODBC translation DLL.
3. Specify the **Translation Option** that is passed to the translation DLL. This information is optional and is dependent on the translation DLL.

Setting Format options

The **Format** options determine certain types of formatting that are used for AS/400 database server data.

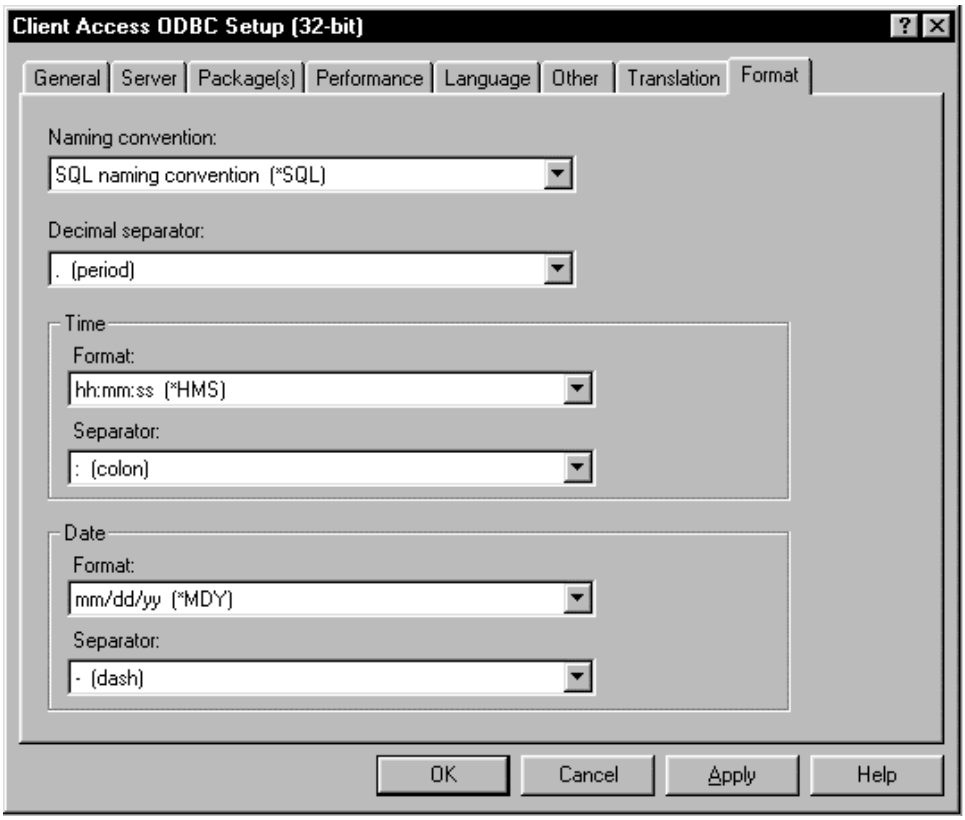


Figure 2-10. Client Access ODBC Setup [32-bit] dialog — Format

To configure the **Format** options, complete the following steps:

1. Select the **Naming convention**. Click the down arrow to select a different convention.

   Details: The different naming conventions are:

   - **SQL** — The SQL naming convention, which uses a period (.) between the collection name and table name. This is the default.
   - **SYS** — The SYS naming convention, which uses a forward slash (/) between the library and file names.

2. Select the **Decimal separator** — period (.) or comma (,) — that your AS/400 system supports. The default is a period. Click the down arrow to select a different separator.

3. Under the **Time** option, select the **Format** that the ODBC driver uses to transfer time data to and from the AS/400 system. The default is hour:minute:second (hh:mm:ss). Click the down arrow to select a different format.

   Detail: This option tells the SQL/400 parser how to interpret time data that is embedded as a constant within aSQL statement.

4. Under the **Time** option, select the **Separator** for the hh:mm:ss (*HMS) time format. This setting is disabled when the time format is anything but *HMS.

5. Under the **Date** option, select the **Format** that the ODBC driver uses to transfer date data to and from the AS/400 system. The default is yyyy-mm-dd. Click the down arrow to select a different setting.

   Detail: This option tells the SQL/400 parser how to interpret date data that is embedded as a constant within aSQL statement.

6. Under the **Date** option, select the **Separator** for the following date formats: mm/dd/yy (*MDY), dd/mm/yy (*DMY), and yy/mm/dd (*YMD). Click the down arrow to select a different setting.

RDB directory

Many ODBC applications use a naming format such as COLLECTION.TABLE. SQL/400 allows the addition of the associated system name to the front of each table or file name (SYSTEM NAME.COLLECTION.TABLE). In order to use an application that provides the system name, the ODBC driver checks the AS/400 RDB directory entry for the local system name. To run the Client Access ODBC driver, you must add the name of your local AS/400 system to the RDB directory.

Data Source Options

The formatting and performance options that are listed below are not comprehensive. The explanations are given only to help you set up the data source.

Server options

Default libraries

It is important to understand that a library list has three portions: a system portion, a current library (designated in the user's AS/400 profile) and a user portion. This option changes only the user portion.

Specify only the libraries that you need to access. Searching all the AS/400 libraries during a query is unnecessary and decreases performance.

The first library that is listed is the default collection. If you do not want a default collection, start the library list with a comma.

When using the *SQL naming convention, the default collection is normally the only collection that is used to satisfy unqualified table names in aSQL collection. For example, `SELECT * FROM MYTABLE` will be treated as `SELECT * FROM`

default_collection.MYTABLE. If no default collection is specified, the library with the same name as the user profile of the ODBC job is used for unqualified table names. The request will fail if the library does not exist.

When using the *SYS naming convention, the unqualified SQL statements will go first to the default collection, if there is one. If there is no default collection, the current library is used; if there is no current library, the library list is used.

If there is no default collection and no current library, the CREATE TABLE request will create the table in library QGPL. Catalog requests for both naming conventions use the default collection, the current library, and the library list.

Two libraries that do not appear on the Server options tab are always added to the default libraries list. ODBC adds the library QIWS to include the ODBC server code in the library path. ODBC also adds the current library (from the AS/400 user profile), which appears as a data source choice in your client application.

Commitment control

Commitment control is a means of processing separate jobs as a single unit of work. A job is an application process that can include the processing of one or more SQL programs. A unit of work is a recoverable sequence of operations.

The ODBC default for SQL_AUTOCOMMIT is SQL_AUTOCOMMIT_ON, effectively disabling all commitment control. Unless the application sets the mode to SQL_AUTOCOMMIT_OFF, this setting will have no affect. Also, the application can override this setting by way of the application program interface (API).

The commit modes available are:

- Commit immediate (*NONE) — Specifies that commitment control is not used. Calls to the SQLTransact API are ignored. Uncommitted changes in other jobs can be seen. If the SQL DROP COLLECTION statement is included in the program, *NONE must be used. When a relational database that is specified on the RDB parameter is on a non-AS/400 system, *NONE cannot be specified.
- Read committed (*CS) — Specifies the objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements. Rows that are updated, deleted and inserted are locked until the end of the transaction. A row that is selected but not updated is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.
- Read uncommitted (*CHG) — Specifies the objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON and REVOKE statements. Rows that are updated, deleted and inserted are locked until the end of the transaction. Uncommitted changes in other jobs can be seen.
- Repeatable read (*ALL) — Specifies the objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements. Rows that are updated, deleted and inserted are locked until the end of the transaction. Uncommitted changes in other jobs cannot be seen.

Package(s) options

Extended dynamic (package) support

Extended dynamic (package) support is a way to store prepared dynamic SQL statements on the AS/400 server. After enabling extended dynamic support, the first time an ODBC application runs an SQL statement, information about the statement is saved in an SQL package object (package).

The package is saved in a default library on the AS/400 server. On subsequent uses of the statement, the database management system (DBMS) recognizes that the statement has been run before. The DBMS then uses the information that has been saved in the package to skip a significant part of the processing.

All statements that contain parameter markers can be stored in a package. The following statements will be stored in the package even when they do not contain parameter markers: positioned UPDATE and DELETE, INSERT with subselect, DECLARE PROCEDURE.

Local package caching

Caching the package locally means that a copy of relevant information that is retrieved from the SQL package on the host is kept locally on the PC. Any application that benefits from extended dynamic support could benefit from local package caching.

In particular, applications that use parameter markers and reexecute statements with SQLExecDirect (instead of SQLPrepare followed by multiple calls to SQLExecute) could see a significant improvement. Local caching allows the possible elimination of the potentially expensive task of preparing an SQL statement by way of SQLPrepare or SQLExecDirect.

Private and shared packages

Users who generate different SQL statements for changing tasks in a particular application should use a private (non-shared) package for that application. Using a private package prevents the package from being locked and ensures that only the desired statements are added to the package. A private package requires that the user's package library name or the first seven characters of the package name be unique.

Multiple users can add to or use a given non-shared package at the same time. However, adding a new statement to the package locks the entire package, which can cause contention between users. This reduces the benefits of using extended dynamic support.

All users who share a package must use the same default library. If the default library for a data source does not match the default library when the package was created, the package cannot be used. In this case, the warning, or error, message "Extended dynamic support disabled" will be returned to the application.

The administrator can set up an application that uses a fixed number of SQL statements to have a single package that all users effectively share without contention. See

| “Setting package options for multiple users” on page 2-7 for instructions on how to set
| up multiple users with a shared package.

Performance options

Lazy close support

Enabling lazy close support delays transmission of an SQLFreeStmt with the SQL_CLOSE option to the AS/400 until the client application sends the next request. This enhances AS/400 performance. When lazy close support is disabled, an SQLFreeStmt with the SQL_CLOSE option causes an explicit flow to the AS/400 to close the statement.

Pre-fetch during EXECUTE

| When pre-fetch during EXECUTE is enabled, open and fetch operations are combined
| when a SELECT statement is processed. Combining the operations allows a fetch of
| the first block of data in advance, before the application requests it. This "pre-fetch"
| reduces the amount of communication between the client and the server.

OS/400 library view

| This option is used only when calling the special case usage of the SQLTables function
| with parameters that request a list of table *owners* (libraries), not tables.

| Selecting "Default library list" returns the libraries in that list. This is the preferred setting
| due to performance reasons. The default library list is set with the **Default libraries**
| option on the **Server** options tab.

| Selecting "All libraries on the system" can potentially return a huge number of libraries
| on a large system, with a corresponding performance penalty.

| It is important to note that this is a special case usage of the SQLTables API. The
| normal use of SQLTables (to get lists of tables) and all other ODBC catalog functions
| are unaffected by this setting.

| Here are the parameters for the special case usage of the SQLTables function:

```
| szTableQualifier  =  "" (empty string)
| cbTableQualifier  =  SQL_NTS
| szTableOwner      =  %
| cbTableOwner      =  SQL_NTS
| szTableName       =  "" (empty string)
| cbTableName       =  SQL_NTS
| szTableType       =  NULL
| cbTableType       =  0
```

Translation options

CCSID 65535 translation

The coded character set identifier (CCSID) is a 2-byte binary (hexadecimal) identifier. The AS/400 identifies the CCSID in which character data is coded for each column in a database.

The CCSID of a field determines which conversion table is required for converting the data, for example, from EBCDIC to ASCII. A CCSID of 65535 often identifies raw data, such as bitmapped graphics.

Clicking "Translate CCSID 65535" allows translation of CCSID 65535 data to the same CCSID of the job. Choosing this option may cause translation of data that is meant to remain untranslated. The administrator should set these options for users.

Translation DLL

This option specifies the ODBC translation DLL that is used to translate data that is passed between the ODBC driver and the data source. The Client Access ODBC driver normally does any necessary data translations, but a translation DLL may be useful for some cases, such as right-to-left or bi-directional languages. The DLL is loaded when a connection is established using this data source.

Translation option

Translation option specifies a 32-bit integer translation option that is passed to the translation DLL. The translation DLL determines the meaning of this option. Refer to the documentation that is provided with the translation DLL for more information. This parameter is optional.

Part 2. ODBC Performance Considerations

Chapter 3. Performance-Tuning Client Access ODBC

In client/server computing, the most important attribute is **performance**. This chapter discusses client-server performance issues in general. And it discusses the performance implications of ODBC with popular query tools and development environments. But first, let's start with a quick review of key performance issues.

Introduction to performance

The performance characteristics of any computing environment can be described in the following ways:

Response time The amount of time required for a request to be processed.

Utilization The percentage of resources consumed when processing requests.

Throughput The volume of requests per unit-of-time being processed.

Capacity The maximum amount of throughput possible.

Response time is typically the critical performance issue for users of a system. Utilization is of concern to the administrators of a system. Maximum throughput is indicative of the performance *bottleneck* and may or may not be a concern. All of these descriptions are interrelated, but for now all you need to remember is the following:

- Every computing system has a bottleneck governing performance - throughput.
- When system utilization goes up, response time degrades.

In many systems, the capacity is wide enough and is hardly noticed by the users. In others, it's the primary performance concern. Response time is critical. And the question usually becomes how far it can be degraded while users are added, or utilization increases, before users start complaining?

Introduction to client/server performance

The performance characteristics of a client/server environment are far different than what the administrators of centralized environments are used to. This is because client/server applications are split between the client and the server. Both pieces communicate with each other in an intelligent fashion by sending and receiving messages. This paradigm is far different than for a program that gets the CPU for some timeslice with a guarantee that memory and the disk drives will be paying full attention.

Instead, when a client requests processing time and data from the server, it places the request on the wire. The request travels to the server and waits in a queue until the server finds time for it. Without going into details, the performance characteristics of this type of architecture degrade exponentially as the number of requests increase. Putting this in English, "Response times increase gradually as more requests are made, but then increase dramatically at some point, also known as the knee of the curve". Actually, dramatically is an understatement, as shown in Figure 3-1 on page 3-2:

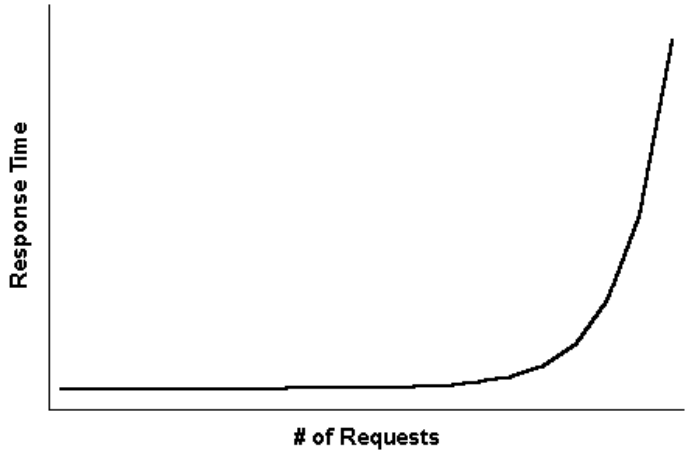


Figure 3-1. Client/Server Performance Curve

The $64 question is, "At what point do things get ugly?". And the answer is simple; it varies with every client-server installation, which is why there are no numbers in Figure 3-1. Worse yet, you won't know where the wall is until you run into it.

With this uncertainty aside, there is a very important rule based upon the mathematics of client/server architectures: Don't go to the server unnecessarily, and when you have to, go in as few trips as possible. It's a simple rule that people understand well in their everyday lives (I like to drive to the grocery store once a week only, and fill my car completely... rather than bringing my milk and cookies home one bag at a time). Unfortunately, this simple rule is too often broken in client/server programming. The old mind set of "open the file and read one record at a time" has gotten many client-server projects and tools into far too much trouble. And it gets very expensive when a business learns this lesson when trying to go from 60 to 90 users, only to find out that the capacity limit is somewhere in between.

The performance architecture of the Client Access ODBC driver

When the Client Access ODBC driver was rewritten, the performance characteristics of client/server architectures significantly affected the design. For example, all of the internal data flows between the client and the server are chained together and make

the trip only when needed. This reduces server utilization because the overhead of the communications layer is only executed once. Response times improve correspondingly.

These types of enhancements are transparent to the user. But other enhancements are externalized as configuration entries in the ODBC registry. See Chapter 2, “Setting up the Client Access ODBC Driver” on page 2-1 for further discussion. This section provides some additional information that relates to performance scalability.

ODBC registry settings

Editing configuration parameters in the ODBC registry configures the Client Access ODBC driver. This registry file is in the directory where Windows 95/NT is installed on your system. Do not edit the registry directly. The preferred method to tune Client Access ODBC performance is through the Client Access ODBC Setup (32-bit) dialogue.

For a description of these parameters and their allowed values, refer to Chapter 2, “Setting up the Client Access ODBC Driver” on page 2-1. This section focuses on the performance implications of some of the more important parameters.

Selecting a stringent level of commitment control

This one's pretty straightforward. Do not use commitment control unless you have to. The overhead of locking not only drives up utilization, but reduces concurrency as well. Of course, if your application isn't read-only, commitment control *might* be required. A common alternative, however, is to use "optimistic" locking. Optimistic locking basically involves issuing explicit UPDATEs using a WHERE clause that uniquely determines a particular record, and also ensures that the record has not changed since being retrieved.

Many third-party tools use this approach, which is why some require a unique index to be defined for updatable tables. A record update can then be made by fully qualifying the entire record contents. Take the following example:

```
UPDATE table SET C1=new_val1, C2=new_val2, C2=new_val3
   WHERE C1=old_val1 AND C2=old_val2 AND C3=old_val3
```

This statement would guarantee that the desired row is accurately updated, if, and only if, the table contained only three columns and each row was unique. A better performing alternative would be:

```
UPDATE table SET C1=new_val1, C2=new_val2, C3=CURRENT_TIMESTAMP
   WHERE C3=old_timestamp
```

This only works, however, if the table has a timestamp column that holds information about when the record was last updated. Always setting the new value for this column to CURRENT_TIMESTAMP guarantees row uniqueness.

If commitment control is required, use the lowest level of record locking possible. For example, use ***CHG:** over ***CS** when possible, and never use ***ALL** when ***CS** gives you what you need. For more information on commitment control, refer to the DB2/400 documentation.

Fine tuning record blocking

Record blocking is a technique that greatly reduces the number of network flows by returning a *block* of rows from the server on the first FETCH request for a cursor. Subsequent FETCH requests are then retrieved from the local block of rows rather then going to the server each time. This technique dramatically increases performance when it is properly used, and the default settings should be sufficient for most situations.

However, from a scaling perspective, a change to one of the record blocking parameters can make a significant difference when the performance of your environment is approaching the "exponential" wall in Figure 30-1. For example, let's assume that your environment has *n* decision support clients doing some amount of work with large queries, typically returning 1 MB of data. When using the default BlockSizeKB parameter of 32 KB, each time a user queried the database, it would take 32 trips between the server and the client to return the data. Changing the BlockSizeKB value to 512 would reduce the number of server requests by a factor of *15*, and might make the difference between being on the left side of the exponential wall or being crowded up against it.

In the opposite extreme, you might have this same scenario where users are consistently asking for large amounts of data, but typically never end up looking at more than a few rows. The overhead of returning 32KB of rows when only a few are needed could hurt performance. Setting the *BlockSizeKB* parameter to a lower value, or even disabling record blocking altogether might actually increase performance.

It is important to note that, as always in client/server, your performance mileage may vary. You might make changes to these parameters and not see any difference, which may indicate that your performance bottleneck is not the client request queue at the server. This parameter does give you one more tool for when your users start complaining.

Using Extended Dynamic SQL

Traditional SQL interfaces used an *embedded* SQL approach. SQL statements were placed directly in an application's source code, along with high-level language statements written in C, COBOL, RPG, and so on. The source code was then *precompiled*, which translated the SQL statements into code that the subsequent compile step could process. One of the advantages of this approach is that the SQL statements are optimized in advance rather than at runtime while the user is waiting, thus improving performance.

ODBC, however, is a *Call Level Interface* (CLI) that uses a different approach. Using a CLI, SQL statements are passed to the DBMS within a parameter of a runtime API. Because the text of the SQL statement is never known until runtime, the optimization step must be performed each time a SQL statement is run. This approach is commonly referred to as **dynamic SQL**, while the previously described method is referred to as **static SQL**.

A good description of the ExtendedDynamic parameter in the Client Access ODBC driver can be found in Chapter 2, “Setting up the Client Access ODBC Driver” on

page 2-1. Using this feature (which is enabled by default) can not only improve response times, but can dramatically improve server utilization. This is because optimizing SQL queries can be very expensive and performing this step only once is always advantageous. This works well with DB2/400 because, unlike other DBMS's, DB2/400 has a unique feature that ensures that statements stored in packages are kept up to date in terms of optimization, without administrator intervention. Even if a statement was prepared weeks or months ago for the first time, DB2/400 automatically regenerates the access plan when it determines that sufficient database changes warrant reoptimization.

Chapter 4. Performance Considerations of Common End-User Tools

Having an ODBC driver that is optimally tuned for performance is only part of the battle. The other part is the tools that are used, whether they are used to simply query the data or to build complex programs using 4GLs. Some of the more common tools include:

- Brio DataPrism
- Crystal Services Crystal Reports Professional
- Cognos Impromptu
- Computer Associates Visual Express
- Gupta SQL Windows
- IBM Visualizer for Windows
- Lotus Approach
- Lotus Notes
- Microsoft Access
- Microsoft Visual Basic
- Powersoft Powerbuilder
- Showcase Vista
- Trinzic Forest and Trees

There are many more available than are on this list, and every tool in the marketplace has its own strengths, weaknesses, and performance characteristics. But most have one thing in common: support for ODBC database servers. However, because ODBC serves as a common denominator for various DBMS's, and because there are subtle differences from one ODBC driver to the next, many tool providers end up writing to the more common ODBC and SQL interfaces, and avoid taking advantage of a particular database server's strengths. This may ease programming efforts, but it often hurts overall performance.

Common tool behaviors that hurt performance

The performance problems incurred by writing SQL and ODBC calls that pay no attention to the particular ODBC driver or the server DBMS are best shown with a few examples.

Query Tool A

Query Tool A makes the following ODBC calls to process SELECT statements:

```
SqlExecDirect("SELECT * FROM table_name")

WHILE there_are_rows_to_fetch DO

   SqlFetch()
   FOR every_column DO
      SqlGetData( COLn )
   END FOR
   ...process the data

END WHILE
```

This tool does not make use of ODBC bound columns, which can help performance. A faster way to process this is as follows:

```
SqlExecDirect("SELECT * FROM table_name")
FOR every_column DO
   SqlBindColumn( COLn )
END FOR

WHILE there_are_rows_to_fetch DO
   SqlFetch()
   ...process the data
END WHILE
```

If a table contained one column, there wouldn't be much difference between the two approaches. But for a table with a 100 columns, you end up with 100 times as many ODBC calls in the first example, *for every row fetched*. We can also optimize the second scenario because the target data types specified by the tool won't change from one FETCH to the next, like they could with each SqlGetData() call.

Query Tool B

Query tool B allows you to update a spreadsheet of rows and then send the updates to the database. It makes the following ODBC calls:

```
FOR every_row_updated DO

   SqlAllocStmt()
   SqlExecDirect("UPDATE...SET COLn='literal'...WHERE COLn='oldval'...")
   SqlFreeStmt( SQL_DROP )

END LOOP
```

The first thing to note is that the tool performs a statement allocation-and-drop for every row. Only one allocate statement is needed, and the free statement call could be changed to `SqlFreeStmt( SQL_CLOSE )` after each SqlExecDirect. This change would save the overhead of creating and destroying a statement handle for every operation. Another performance concern is the use of SQL with literals instead of parameter markers. The SqlExecDirect() call causes a SqlPrepare and SqlExecute every time. Because the UPDATE uses only literals, ExtendedDynamic won't help because the

statement is discarded after the SQL_DROP. A faster way to perform this operation would be as follows:

```
SqlAllocStmt()
SqlPrepare("UPDATE...SET COL1=?...WHERE COL1=?...")
SqlBindParameter( new_column_buffers )
SqlBindParameter( old_column_buffers )
FOR every_row_updated DO

   ...move each rows data into the SqlBindParameter buffers
   SqlExecute()
   SqlFreeStmt( SQL_CLOSE )

END LOOP
```

These sets of ODBC calls will outperform the original set by a large factor when using the Client Access ODBC driver. The server CPU utilization will decrease to 10% of what it was... pushing that scaling wall out a lot further.

Query Tool C - Your Worst Possible Nightmare

Query tool C allows complex decision support type queries to be made by defining complex query criteria with a point and click interface. You might end up with SQL that looks like this for a query:

```
SELECT A.COL1, B.COL2, C.COL3 , etc...
   FROM A, B, C, etc...
      WHERE many complex inner and outer joins are specified
```

The fact that you didn't have to write this complex query yourself sure is nice, but is this statement actually what the tool is processing? Perhaps yes, perhaps no. For example, one tool might pass this statement directly to the ODBC driver, while another would split up the query into many individual queries and process the results at the client, like this:

```
SQLExecDirect("SELECT * FROM A")
SQLFetch() all rows from A
SQLExecDirect("SELECT * FROM B")
SQLFetch() all rows from B

Process the first join at the client

SQLExecDirect("SELECT * FROM C")
SQLFetch() all rows from C

Process the next join at the client
.
.
.
And so on...
```

This approach can lead to tremendous amounts of data being passed to the client, which will adversely affect performance. In one real-world example, a programmer

thought that a 10-way inner, outer join was being passed to ODBC, with 4 rows being returned. What actually was passed, however, was 10 simple SELECT statements and all the FETCHes associated with them. The net result of 4 rows was achieved only after *81,000* ODBC calls were made by the tool! Of course, the programmer was originally blaming ODBC for the slow performance, but not after the ODBC trace was revealed.

Chapter 5. SQL Performance

Good application design includes the efficient use of machine resources. As such, any programmer can write a program. But to execute in a manner that is acceptable to the end user, the program must be elegant in operation and must execute with adequate response time.

This chapter deals with three areas that are critical to SQL performance:

- Database Design

  This section discusses general AS/400 database design and how it affects SQL performance.

- Optimizer

  The Optimizer is the facility that decides how to gather data that should be returned to the program. This section covers some of the techniques and rules used by the Optimizer for performing this task.

- Analyzing Performance Problems

  This section covers defining and resolving performance problems in a client/server environment.

General considerations

Performance of SQL in application programs is important to ALL system users, because inefficient usage of SQL can waste system resources.

The main goal in using SQL is to get the **correct results** for your database request and to get these results in a timely manner.

Before you start designing for performance, you should think about the following considerations:

- When to consider performance:
  - Over 10,000 rows - Performance impact: *"noticeable"*
  - Over 100,000 rows - Performance impact: **"concern"**
  - With complex queries used repetitively
  - Using multiple work stations with high transaction rates
- What resource to optimize:
  - I/O usage
  - CPU usage
  - Effective usage of indexes
  - OPEN/CLOSE performance
  - Concurrency (COMMIT)

- How to design for performance:
  - Database Design
    - Table structure
    - Indexes
    - Table data management
    - Journal management
  - Application design
    - Structure of programs involved
  - Program design
    - Coding practices
    - Performance monitoring

For a fuller and more technical discussion on all of these topics than is discussed in this chapter, please see *DB2 for AS/400 SQL Programming*, SC41-5611 or *DB2 for AS/400 SQL Reference*, SC41-5612.

Database design

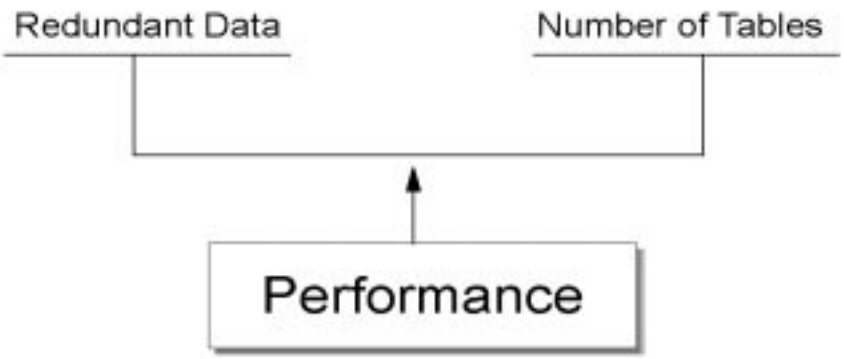
One of the first things you want to do is to determine what tables you require in your database and the relationship between those tables.

Normalization

One of the most critical design decisions that can affect performance is the number of tables that may have to be joined in an SQL statement. The trade-off is between SQL performance and the anomalies that result in tables that have not been designed to be in their fully normalized form.

Numerous design methods are available to enable you to design technically correct databases. Most of these can be used to design good relational database structure. One important part of these techniques is the elimination of the storing of redundant or duplicate data. The main objective is to avoid problems of updating such redundant data.

If this design approach of normalization, for instance 3rd Normal Form (3NF), is taken to its ultimate conclusion, the result is a large number of tables. Many of these tables will be involved in join operations, which can lead to poor OS/400 database performance. You should attempt to design tables that will not create problems caused by redundant data, and, at the same time, not cause significant performance problems. Therefore, you have to find a good balance between:



For instance, try to minimize the use of code tables where very little is gained from their use. For example, an employee table contains a JOBCODE column, with data values 054, 057, and so on. This table must be joined with another table to translate the codes to Programmer, Engineer, and so on. The cost of this join could be quite high compared to the savings in storage and potential update anomalies resulting from redundant data.

For example:

- **Normalized** data form

EMPLOYEE Table

Employee No	Jobcode
00010	057
00020	054
00030	057
...	...

JOBCODE Table

Jobcode	Job Title
054	Programmer
057	Engineer
...	...
...	...

- **Redundant** data form

EMPLOYEE Table

Employee No	Job Title
00010	Engineer
00020	Programmer
00030	Engineer
...	...

The set level (or mass operation) nature of SQL greatly lessens the danger of a certain redundant data form. For example, the ability to update a set of rows with a single SQL statement greatly reduces this risk. In the following example, the job title Engineer must be changed to Technician for all rows that match this condition.

SQL can easily be used to update JOBTITLE

```
UPDATE EMPLOYEE
  SET JOBTITLE = "Technician"
  WHERE JOBTITLE = "Engineer"
```

Table size

The size of the table(s) that your application program is accessing can have a significant impact on the performance of the application program. Consider the following:

- Large row length

  If a table accessed sequentially has a large row length because it has many columns (for example, 100 columns), in some cases you could achieve better performance by splitting it into two or more tables. This assumes that your application is not accessing all of the columns. The main reason for the better performance is that I/O may be reduced because you will get more rows per page. Splitting the table will affect applications that access all of the columns because they will incur the overhead of joining the table back together again. You must decide where to split the table based on the nature of the application and frequency of access to various columns.

- Large number of rows

  If a table has a large number of rows, it is crucial for good performance that your SQL statements are constructed in such a way that the Optimizer will use an index in accessing the table. The use of indexes is very important for achieving the best possible performance.

Indexes

Indexes are probably the most important facility for use in controlling the performance of your applications because the Optimizer uses them for performance optimization. The indexes are created in five different ways:

- CREATE INDEX (in SQL)
- CRTPF, with key
- CRTLF, with key
- CRTLF, as join logical file
- CRTLF, with select/omit specifications, without a key, and without dynamic selection (DYNSLT).

Indexes are used to enable row selection by means of an index versus table scanning, which is usually slower. Table scanning sequentially processes all rows in a table. If a permanent index is available, building a temporary index can be avoided. Indexes are required for:

- Join tables
- ORDER BY
- GROUP BY

and will be built, if no permanent index exists.

The number of indexes has to be managed because of the extra system cost of maintaining the indexes during update types of operations. Below are **rules of thumb** for particular type of tables:

- Primarily read-only tables

  Create indexes over columns as needed. Only consider creating an index if a table is greater than approximately 1,000 rows or is going to be used with ORDER BY, GROUP BY, or join processing. Index maintenance could be more costly than occasionally scanning the entire table.

- Primarily read-only table, with low update rate

  Create indexes over columns as needed. Avoid building indexes over columns that are updated frequently. INSERT, UPDATE, and DELETE will cause maintenance to all indexes related to the table.

- High update rate tables

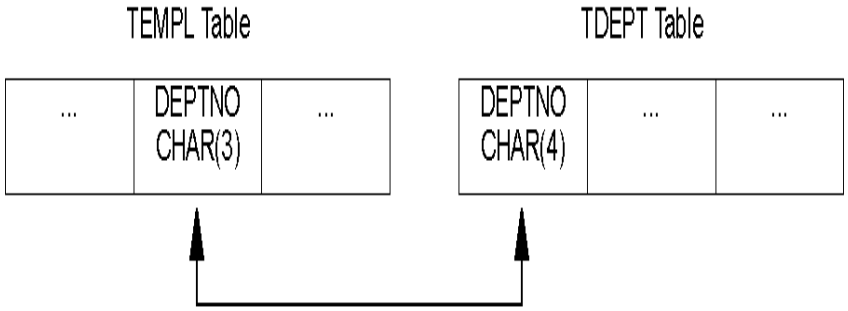
  Avoid creating many indexes. An example of a table that has a high update rate is a logging or a history table.

Matching attributes of join fields

Columns in tables that are joined should have identical attributes, that is, the same column length, same data type (character, numeric, so on). Nonidentical attributes result in temporary indexes being built, even though indexes over corresponding columns may exist already.

In the following example, **join** will build a temporary index and ignore an existing one:

```
SELECT EMPNO, LASTNAME, DEPTNAME
  FROM TEMPL, TDEPT
  WHERE TEMPL.DEPTNO = TDEPT.DEPTNO
```



Optimizer

The Optimizer is an important module of the OS/400 Query component because it makes the key decisions for good database performance. Its main objective is to find the cheapest access path to the data.

Query optimization is a trade-off between the time spent to select a query implementation and the time spent to execute it. Query optimization must handle differing user needs:

- Quick interactive response
- Efficient use of total-machine resources

In deciding *how* to access data, the Optimizer:

- Determines possible implementations.
- Picks the optimal implementation for the OS/400 Query component to execute.

Cost estimation

At runtime, the optimizer chooses an optimal access method for the query by calculating an *implementation cost* based on the current state of the database. The optimizer models the access cost of each of the following:

- Reading rows directly from the table (dataspace scan processing)
- Reading rows through an access path (using either key selection or key positioning)
- Creating an access path directly from the dataspace

- Creating an access path from an existing access path (index-from-index)
- Using the query sort routine (if conditions are satisfied)

The cost of a particular method is the sum of:

- The start-up cost
- The cost associated with the given optimization mode. The OPTIMIZE FOR n ROWS clause indicates to the query optimizer the optimization goal to be achieved. The optimizer can optimize SQL queries with one of two goals:

  1. Minimize the time required to retrieve the first buffer of rows from the table. This goal biases the optimization towards not creating an index.

     Either a data scan or an existing index is preferred. This mode can be specified by:

     The OPTIMIZE FOR n ROWS allowing the users to specify the number of rows they expect to retrieve from the query.

     The optimizer using this value to determine the percentage of rows that will be returned and optimizes accordingly. A small value instructs the optimizer to minimize the time required to retrieve the first n rows.

  2. Minimize the time to process the whole query assuming that all selected rows are returned to the application. This does not bias the optimizer to any particular access method. This mode can be specified by the OPTIMIZE FOR n ROWS allowing the users to specify the number of rows they expect to retrieve from the query.

     The optimizer uses this value to determine the percentage of rows that will be returned and optimizes accordingly. A value greater than or equal to the expected number of resulting rows instructs the optimizer to minimize the time required to run the entire query.

- The cost of any access path creations
- The cost of the expected number of page faults to read the rows and the cost of processing the expected number of rows.

  Page faults and number of rows processed may be predicted by statistics the optimizer obtains from the database objects, including:

  – Table size
  – Row size
  – Index size
  – Key size

  A weighted measure of the expected number of rows to process based on what the relational operators in the row selection predicates, *default filter factors*, are likely to retrieve:

  – 10% for equal
  – 33% for less-than, greater-than, less-than-equal-to, or greater-than-equal-to

- 90% for not equal
- 25% for BETWEEN range
- 10% for each IN list value

Key range estimate is a method that the optimizer uses to gain more accurate estimates of the number of expected rows selected from one or more selection predicates. The optimizer estimates by applying the selection predicates against the left-most keys of an existing index. The default filter factors can then be further refined by the estimate based on the key range. If an index exists whose left-most keys match columns used in row selection predicates, that index can be used to estimate the number of keys that match the selection criteria. The estimate of the number of keys is based on the number of pages and key density of the machine index and is done without actually accessing the keys. Full indexes over columns used in selection predicates can significantly help optimization.

Optimizer decision-making rules

In performing its function, the optimizer uses a general set of guidelines to choose the best method for accessing data. The optimizer:

- Determines the default filter factor for each predicate in the selection clause.
- Extracts attributes of the table from internally stored information.
- Performs an estimate key range to determine the true filter factor of the predicates when the selection predicates match the left-most keys of an index.
- Determines the cost of creating an index over a table if an index is required.
- Determines the cost of using a sort routine if selection conditions apply and an index is required.
- Determines the cost of dataspace scan processing if an index is not required.
- For each index available, in the order of most recently created to oldest, the optimizer does the following until its time limit is exceeded:
  - Extracts attributes of the index from internally stored statistics.
  - Determines if the index meets the selection criteria.
  - Determines the cost of using the index using the estimated page faults and the predicate filter factors to help determine the cost.
  - Compares the cost of using this index with the previous cost (current best).
  - Picks the cheapest one.
  - Continues to search for best index until time out or no more indexes.

The *time limit* factor controls how much time is spent choosing an implementation. It is based on how much time has been spent and the current best implementation cost found. Dynamic SQL queries are subject to the optimizer time restrictions. Static SQL queries optimization time is not limited.

For small tables, the query optimizer spends little time in query optimization. For large tables, the query optimizer considers more indexes. Generally, the optimizer considers five or six indexes (for each table of a join) before running out of optimization time.

Chapter 6. Coding Directly to the ODBC APIs

Many PC applications have the ability to make ODBC calls to allow the user to access data on many different platforms seamlessly. Before examining the samples below, it is important to understand some basics about how an ODBC application connects to and exchanges information with a database server.

Calling ODBC Functions

Now we will examine what a programmer must do to take control of the ODBC calls within his application.

ODBC API functions fall into several categories:

- Setting up the ODBC environment.
- Establishing connections to data sources.
- Executing SQL statements.
- Cleaning up the ODBC environment.

Basic Application Steps

An ODBC application needs to follow a basic set of steps in order to access a database server.

1. Connect to the data source.
2. Place the SQL statement string to be executed in a buffer. This is a text string.
3. Submit the statement in order that it can be prepared or immediately run.
   - Retrieve and process the results.
   - If there are errors, retrieve the error information from the driver.
4. End each transaction with a commit or rollback operation (if necessary).
5. Terminate the connection.

Basic Application Flow

Before an application can use ODBC to connect to a database and its supporting database management system, a data source must be created. The data source defines how to access the database and the database management system. For more information on how to define a data source, see section Chapter 2, “Setting up the Client Access ODBC Driver” on page 2-1.

The ODBC interface defines a set of calls that are used for each of these application steps. For example, SQLAllocEnv allocates the ODBC environment. Documentation for each of these API calls is in the *Microsoft ODBC Software Development Kit and Programmer's Reference*, ISBN 1-57231-516-4. In this section, we examine some of the

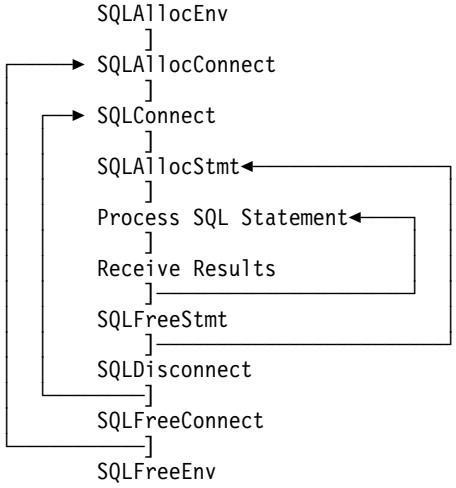


Figure 6-1. Basic ODBC Application Flow

more common ODBC APIs. Coding examples use the "C" language. Also included in this chapter are examples using Microsoft Visual Basic.

Establishing ODBC Connections

- SQLAllocEnv
  - Allocates memory for an environment handle.
    - Identifies storage for global information.
      - Valid connection handles
      - Current active connection handles
      - Variable type HENV
      - Application uses a single environment handle.
  - Initializes the ODBC call level interface for use by an application.
  - Must be called by application prior to calling any other ODBC function.
  - Variable type HENV is defined by ODBC in the SQL.H header file provided by the "C" compiler or ODBC Software Development Kit (SDK).

    The header file contains a type definition for a far pointer:

    ```
    typedef void far * HENV
    ```

  - In "C", this statement is coded:

    ```
    HENV henv;
    SQLAllocEnv(&henv);
    ```

  - In Visual Basic, this statement is coded:

```
Dim henv As long
SQLAllocEnv(henv)
```

- SQLAllocConnect
  - Allocates memory for an connection handle within the environment.
    - Identifies storage for information about a particular connection.
      - Variable type HDBC
      - Application can have multiple connection handles.
  - Application must request a connection handle prior to connecting to the data source.
  - In "C", this statement is coded:

```
HDBC hdbc;
SQLAllocConnect(henv,&hdbc);
```

  - In Visual Basic, this statement is coded:

```
Dim hdbc As long
SQLAllocConnect(henv,hdbc)
```

- SQLConnect
  - Loads driver and establishes a connection.
  - Connection handle references information about the connection.
  - Data source is coded into application.
  - In this example, user ID and password are blank because they are supplied by the Client Access router.

```
 UCHAR source[] = "DBFIL";
 UCHAR uid[] = "";
 UCHAR pwd[] = "";
SQLConnect(hdbc,source,SQL_NTS,uid,SQL_NTS,pwd,SQL_NTS);
```

  Note: SQL_NTS indicates that the statement is ended with a null-terminated string.

- SQLDriverConnect
  - Alternative to SQLConnect.
  - Allows driver manager to obtain login information from the user.
  - Displays dialog boxes (optional).

Executing ODBC Functions

- SQLAllocStmt
  - Allocates memory for information about an SQL statement.
    - Application must request a statement handle prior to submitting SQL statements.

- Variable type HSTMT.

  ```
  HSTMT s_create;
  SQLAllocStmt(hdbc,&s_create);
  ```

- SQLExecDirect
  - Executes a preparable statement.
  - Fastest way to submit an SQL string for one time execution.
  - If rc is not equal to SQL_SUCCESS, a SQLError API call is used to find the cause of the error condition.

    ```
    UCHAR stmt[ ]
    = "CREATE TABLE NAMEID (ID INTEGER,
                  NAME VARCHAR(50))";
    rc = SQLExecDirect(s_create,stmt,SQL_NTS)
    ```

  - Return code
    - SQL_SUCCESS
    - SQL_SUCCESS_WITH_INFO
    - SQL_ERROR
    - SQL_INVALID_HANDLE
- SQLError

  ```
  SQLError(henv,hdbc,s_create,szSqlState,&pfNativeError,
                  szErrorMsg,cbErrorMsg);
  ```

  - szSqlState
    - 5 character string
    - 00000 = success
    - 01004 = data truncated
    - 07001 = wrong number of parameters
  - pfNativeError - specific to data source
  - szErrorMsg - Error Message text

Executing Prepared Statements

If an SQL statement is used more than once, it is best to have the statement prepared and then executed. When a statement is prepared, variable information can be passed as parameter markers which are denoted by question marks ("?"). When the statement is executed, the parameter markers are replaced with the real variable information.

Preparing the statement is done at the server. The SQL statements are compiled and the access plans are built. This allows the statements to be executed much more efficiently. Rather than using dynamic SQL to execute the statements, the result is much closer to static SQL. When the V3R1 database server prepares the statements, it saves some of them in a special AS/400 object called a package (*SQLPKG). This approach

is called extended dynamic SQL. The creation of packages is done automatically by the driver; an option is provided to turn off package support. This is discussed later in the chapter under Performance Tuning IBM's ODBC driver.

- SQLPrepare
  - Prepares an SQL statement for execution:

    ```
    HSTMT s_insert;
    SQLAllocStmt(hdbc,&s_insert);
    UCHAR sqlstr[ ] "INSERT INTO NAMEID VALUES (?,?)";
    SQLPrepare(s_insert,sqlstr,SQL_NTS);
    ```

    Note: SQL_NTS indicates that the string is null-terminated.

- SQLBindParameter
  - Allows application to specify storage, data type, and length associated with a parameter marker in an SQL statement.

    In the example, parameter 1 is found in a signed double word field called **id**, while parameter 2 is found in an unsigned character array called **name**.

    ```
    SDWORD id;
    UCHAR name[51];
    SQLBindParameter(s_insert,1,SQL_PARAM_INPUT,
                     SQL_C_LONG,SQL_INTEGER,
                     0,0,&id,0,NULL);
    SQLBindParameter(s_insert,2,SQL_PARAM_INPUT,
                     SQL_C_CHAR,SQL_VARCHAR,
                     parmLength,0,
                     name,(UDWORD) sizeof(name), NULL);
    ```

- SQLExecute
  - Executes a prepared statement, using current values of parameter markers:

    ```
    id=500;
    strcpy (name,"TEST");
    SQLExecute(s_insert);  // Insert a record with id = 500, name = "Test"
    id=600;
    strcpy (name,"ABCD");
    SQLExecute(s_insert);  // Insert a record with id = 600, name = "ABCD"
    ```

- SQLParamData / SQLPutData

  Visual Basic does not directly support pointers or fixed-location ANSI character null-terminated strings. For this reason, it is best to use another method to bind Character and Binary parameters. One method is to convert Visual Basic String data types to/from an array of Byte data types and bind the array of Byte. This method is demonstrated in “Converting Strings and Arrays of Byte” on page 6-6.

  Another method, that should only be used for input parameters, is to supply the parameters at processing time. This is done using SQLParamData and SQLPutData APIs:

  - They work together to supply parameters.
  - SQLParamData moves the pointer to the next parameter.

– SQLPutData then supplies the data for that parameter.

```
's_parm is a character buffer to hold the parameters
's_parm(1) contains the first parameter
Static s_parm As string
s_parm(1) = "500"
s_parm(2) = "TEST"
cbvalue = SQL_DATA_AT_EXEC     ' the parms will be supplied at run time

SQLBindParameter(s_insert,1,SQL_PARAM_INPUT,
                 SQL_C_CHAR,SQL_CHAR,
                 4,0,0,0&,4,cbvalue)
SQLBindParameter(s_insert,2,SQL_PARAM_INPUT,
                 SQL_C_CHAR,SQL_CHAR,
                 4,0,0,0&,4,cbvalue)

SQLExecute(s_insert)
SQLParamData(s_insert, &token)   '  Param 1
SQLPutData(s_insert, Byval s_parm(1), Len (s_parm(1)))
SQLParamData(s_insert, &token)   '  Param 2
SQLPutData(s_insert, Byval s_parm(2), Len (s_parm(2)))
SQLParamData(s_insert, &token)   '* No more params, it will execute
```

Notes:

1. These two statements operate together to to supply unbound parameter values when the statement is executed.
2. Each call to SQLParamData moves the internal pointer to the next parameter for SQLPutData to supply data to. After the last parameter is filled, SQLParamData must be called again for the statement to be executed.
3. If data for parameter markers is to be supplied using SQLPutData, the parameter must be bound with the cbValue parameter set to a variable whose value is SQL_DATA_AT_EXEC when the statement is executed.

Converting Strings and Arrays of Byte

The following Visual Basic functions can assist in converting strings and arrays of byte.

```
Public Sub Byte2String(InByte() As Byte, OutString As String)
  'Convert array of byte to string
   OutString = StrConv(InByte(), vbUnicode)
End Sub

Public Function String2Byte(InString As String, OutByte() As Byte) As Boolean
    'vb byte-array / string coercion assumes unicode string
    'so must convert String to Byte one character at a time
      'or by direct memory access

    Dim I As Integer
    Dim SizeOutByte As Integer
    Dim SizeInString As Integer

    SizeOutByte = UBound(OutByte)
    SizeInString = Len(InString)
    'Verify sizes if desired

    'Convert the string
    For I = 0 To SizeInString - 1
      OutByte(I) = AscB(Mid(InString, I + 1, 1))
    Next I
   'If size byte array > len of string pad with Nulls for szString
   If SizeOutByte > SizeInString Then           'Pad with Nulls
      For I = SizeInString To SizeOutByte - 1
         OutByte(I) = 0
      Next I
   End If

   String2Byte = True
End Function

Public Sub ViewByteArray(Data() As Byte, Title As String)
   'Display message box showing hex values of byte array

   Dim S As String
   Dim I As Integer
   On Error GoTo VBANext

   S = "Length: " & Str(UBound(Data)) & "  Data (in hex):"
   For I = 0 To UBound(Data) - 1
      If (I Mod 8) = 0 Then
         S = S & "  "         'add extra space every 8th byte
      End If
      S = S & Hex(Data(I)) & " "
   VBANext:
   Next I
   MsgBox S, , Title

End Sub
```

Transaction Control (Commit)

- SQLTransact
  - Requests a commit or rollback for update, insert, or delete transactions:

    ```
    SQLTransact(hdbc, SQL_COMMIT);
    ```

Retrieving Results

Running some SQL statements returns results to the application program. Running an SQL SELECT statement returns the selected rows in a result set. The SQLUsing Fetch API then sequentially retrieves the selected rows from the result set into the application program's internal storage. In order to work with all of the rows in a result set, call the SQLFetch API until no more rows are returned.

We may also issue a Select statement where we do not specify what columns we want returned. For example, "Select * from RWM.DBFIL", selects all columns. We may not know what columns or how many columns will be returned.

- SQLNumResultCols
  - Returns the number of columns in a result set.
    - A storage buffer that receives the information is passed as a parameter.

```
SQLNumResultCols(hstmt, &nresultcols);
```

- SQLDescribeCol
  - Returns the result descriptor for one column in a result set.
    - Column name
    - Column type
    - Column size

      This is used along with SQLNumResultCols to retrieve information about the columns returned. Using this approach, as opposed to hard coding the information in the program, makes for more flexible programs.

      The programmer first uses SQLNumResultCols to find out how many columns were returned in the result set by a select statement. Then a loop is set up to use SQLDescribeCol to retrieve information about each column.

```
SQLDescribeCol(hstmt, i + 1,colname,
  (SWORD)sizeof(colname),
  &colnamelen,&coltype,
  &collen[i];, &scale, &nullable);
```

- SQLBindCol
  - Assigns the storage and data type for a column in a result set:
    - Storage buffer that receives the information.
    - Length of storage buffer.
    - Data type conversion.

```
SQLBindCol(hstmt, 1, SQL_C_LONG, &id, 0 ,Null);
SQLBindCol(hstmt, 2, SQL_C_CHAR, name,
          (SDWORD) sizeof(name),
          &namelen);
```

– If you use this with Visual Basic, we recommend that you use an array of Byte data type in place of String data types.

- SQLFetch
  - Each time SQLFetch is called, the driver fetches the next row. Bound columns are stored in the locations specified. Data for unbound columns may be retrieved using SQLGetData.

    ```
    SQLFetch(hstmt);
    ```

  Visual Basic does not directly support pointers or fixed memory location ANSI character null-terminated strings. For this reason, it is best to use another method to bind Character and Binary parameters. One method is to convert Visual Basic String data types to/from an array of Byte data types and bind the array of Byte. Another method is to use the SQLGetData function instead of SQLBindCol.

- SQLGetData
  - Retrieves data for unbound columns after a fetch. In this example, four columns are returned and SQLGetData is used to move them to the correct storage location.

    ```
    SQLFetch(s_Customer1)

    SQLGetData(s_Customer1, 1, SQL_C_CHAR,
          szName, 16, 0&)
    SQLGetData(s_Customer1, 2, SQL_C_FLOAT,
          &iDiscount, 0, 0&)
    SQLGetData(s_Customer1, 3, SQL_C_CHAR,
               szCredit, 2, 0&)
    SQLGetData(s_Customer1, 4, SQL_C_FLOAT,
               &iTax, 0, 0&)
    ```

Calling Stored Procedures

Stored procedures can be used to improve the performance and function of an ODBC application. Any AS/400 program can act as a stored procedure. AS/400 stored procedures support input, input/output and output parameters. They also support returning result sets, both single and multiple. The stored procedure program can return a result set by specifying a cursor to return (from an embedded SQL statement) or by specifying an array of values. See Chapter 7, “Stored Procedures” on page 7-1 for further information on Stored Procedures.

To call a stored procedure, complete the following steps:

1. Verify that the stored procedure has been declared by using the OS/400 SQL statement CREATE PROCEDURE.

   Detail: CREATE PROCEDURE should only be executed once for the life of the stored procedure. DECLARE PROCEDURE can also be used, but this method has several restrictions. See *DB2 for AS/400 SQL Reference*, SC41-5612for further information about DECLARE PROCEDURE.

2. Prepare the call of the stored procedure using SQLPrepare.

3. Bind the parameters for input and output parameters.
4. Execute the call to the stored procedure.
5. Retrieve the result set (if one is returned)

In this example, we are are calling a COBOL program named NEWORD. We are passing in a value in a field named szCustId and it is returning a value to a field named szName.

```
HSTMT s_StoredProc;
char Query[320];
char szCustId[10];
char szName[30];
ret = SQLAllocStmt(hdbc,&s_StoredProc);

          // Create the stored procedure definition.
          // The create procedure could be moved to the application's
          // install program so that it is only executed once.
strcpy(Query,"CREATE PROCEDURE NEWORD (:CID IN CHAR(10),
         :NAME OUT CHAR(30) )");
strcat(Query," (EXTERNAL NAME NEWORD LANGUAGE
         COBOL GENERAL WITH NULLS)");

          // Create the stored procedure
ret=SQLExecDirect(s_StoredProc,(unsigned char *)Query,SQL_NTS);

strcpy(Query,"CALL NEWORD (?, ?)");

          // Prepare the stored procedure call
ret=SQLPrepare(s_StoredProc,(unsigned char *)Query,SQL_NTS);

          // Bind the parameters
ret=SQLBindParameter(s_StoredProc,1,SQL_PARAM_INPUT,
                       SQL_C_CHAR,SQL_VARCHAR, 10,0,
                       szCustId, 11, NULL);

ret=SQLBindParameter(s_StoredProc,2,SQL_PARAM_OUTPUT,
                       SQL_C_CHAR,SQL_VARCHAR, 30,0,
                       szName, 31, NULL);
strcpy (szCustId,"0000012345");
          // Execute the stored procedure
ret=SQLExecute(s_StoredProc);
```

Calling Stored Procedures using Result Sets

Using result sets allows the client program to easily retrieve large amounts of information returned by the stored procedure. Rather then retrieving data from a large array of values in an output parameter the client can use the exact same methods used to retrieve data from a SELECT statement. This results in a more "SQL-like" interface.

Note that the following examples contain the DROP PROCEDURE and CREATE PROCEDURE statements for illustration only. Once the procedure is created, it can be called repeatedly.

This result sets example calls an RPG program and receives the data as a result set array.

The program on the PC can issue the DROP PROCEDURE and CREATE PROCEDURE statements before issuing a CALL procedure. After returning from the call, do an ODBC SQLBindCol to set the variables to return data into. Then do ODBC SQLFetch to get the data.

The following is a piece of code showing the ODBC client calls:

```
#include "sql.h"
#include "sqlext.h"
#include "string.h"
/* declare the variables  */
#define ROW_COUNT 20
UCHAR           fetch_all[106];
SDWORD          cbcol;
static HENV     henv;
static HDBC     hdbc;
static HSTMT    hstmt;
char            stmt[4000];
int             ii;
RETCODE         rc;
/* connect to a data source  */
rc = SQLAllocEnv(&henv);
rc = SQLAllocConnect(henv, &hdbc);
rc = SQLConnect(hdbc, "SystemA", SQL_NTS,
                "JohnB", SQL_NTS,
                "Sesame", SQL_NTS);

   /* allocate a statement handle  */
SQLAllocStmt(hdbc,&hstmt);

   /* drop the old procedure  */
_fstrcpy(stmt,"DROP PROCEDURE RPGARY");
rc = SQLExecDirect(hstmt, stmt, SQL_NTS);

    /* create the procedure to call  */
_fstrcpy(stmt,"CREATE PROCEDURE RPGARY RESULT SETS 1 EXTERNAL");
_fstrcat(stmt," NAME JMBLIB.RPGARY LANGUAGE RPG GENERAL");
rc = SQLExecDirect(hstmt, stmt, SQL_NTS);

    /* call stored procedure  */
_fstrcpy(stmt,"CALL RPGARY()");
 rc = SQLExecDirect(hstmt, stmt, SQL_NTS);
```

```
    /* bind the columns to return variables   */
SQLBindCol(hstmt, 1, SQL_C_CHAR, fetch_all, 106, &cbcol);

     /* for each row fetch the data  */
     for(ii=0; ii<ROW_COUNT ;ii++)
     {
       rc = SQLFetch(hstmt);
           /* process data */
                    }

     /* get more results  */
rc = SQLMoreResults(hstmt);

     /* release the statement handle  */
SQLFreeStmt(hstmt,SQL_DROP);

     /* release the connection to the data source  */
SQLDisconnect(hdbc);
SQLFreeConnect(hdbc);
SQLFreeEnv(henv);
```

The RPG program that is called by the procedure should declare an array to hold the data. It can then open files and read the data into the array and do whatever other processing needs to be done. Finally it can return the data in the array by using the SQL SET RESULT SETS function.

The following is an example RPG program:

```
*-------------------------------------------------------------*
*                                                             *
* RPG PROGRAM:  RPGARY                                        *
*                                                             *
* PURPOSE: THIS RPG PROGRAM IS CALLED BY AN ODBC STORED       *
*          PROCEDURE.  IT READS DATA INTO AN ARRAY AND        *
*          RETURNS THE RESULTS BY USING THE SET RESULT        *
*          SETS SQL FUNCTION                                  *
*                                                             *
*-------------------------------------------------------------*
* FILE DESCRIPTION SPECIFICATIONS                             *
*-------------------------------------------------------------*
FPERF    IF  E                    DISK
*-------------------------------------------------------------*
* INPUT SPECIFICATIONS                                        *
*-------------------------------------------------------------*
IPERFE     E DSPERF
IARR         DS                        20
I                                      01 105 VAR1
ILCLVAR      DS
I                                    B  01  040L#
*-------------------------------------------------------------*
C*      **  START OF PROGRAM  **                              *
*-------------------------------------------------------------*
C                     OPEN PERF
C                     Z-ADD0         L#
C           LOOP      TAG
C                     READ PERF                   9999
C                     ADD  1         L#
C           L#        OCUR ARR
C                     MOVE PERFE     ARR
C           L#        COMP 20                   99  99
C  N99                GOTO LOOP
*-------------------------------------------------------------*
*** EXECUTE SQL STATEMENT ***
C/EXEC SQL SET RESULT SETS ARRAY :ARR FOR :L# ROWS
C/END-EXEC
*-------------------------------------------------------------*
C                     CLOSEPERF
C                     RETRN
```

Extended Fetch

Extended Fetch and Block Insert can be used to enhance the performance of an ODBC application. They allow you to retrieve or insert rows in blocks rather than individually, This reduces the data flows and line turn around between the client and the server.

- Extended Fetch
  - Returns a block of data (one row set) in the form of an array for each bound column.

– Scrolls through the result set according to the setting of a scroll type argument - forwards, backwards, or by row number.
– Works in conjunction with the SQLSetStmtOption.
– To fetch one row of data at a time in a forward direction, an application should call SQLFetch.

```
HSTMT s_Item1;
char Query[320];
ret = SQLAllocStmt(hdbc,&s_Item1);
char ItemID[15][8]      /*item return array*/
char ItemName[15][30]   /*Name return array*/
float fItemPrice[15]    /*price return array*/
char ItemData[15][60]   /*data return array*/

ret = SQLAllocStmt(hdbc,&s_Item1);
strcpy(Query,"Select IID, INAME, IPRICE, IDATA from ITEM ");
strcat(Query,"where IID in ( ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
                                  ?, ?, ?, ?)");
// For extended fetch
 ret=SQLSetStmtOption(s_Item1,SQL_CONCURRRENCY,SQL_CONCUR_READ_ONLY);

Note: SQL_CONCUR_READ_ONLY is the default, so this statement could be
left out. However, if update was required then you would use
SQL_CONCUR_LOCK value as the last parameter.

   ret=SQLSetStmtOption(s_Item1,SQL_CURSOR_TYPE,SQL_CURSOR_FORWARD_ONLY);

Note: SQL_CURSOR_FORWARD_ONLY is the default, so this statement could be
left out. However, if you wanted to scroll other then forward, you
could use this statement to set the option that you require.

 ret=SQLSetStmtOption(s_Item1,SQL_ROWSET_SIZE,15);

Note: We will retrieve 15 rows.

// Prepares s_Item1 for use.
 ret=SQLPrepare(s_Item1,(unsigned char *)Query,SQL_NTS);

// Sets the input parameters (the items to fetch).
// note s_parm is an array to hold items to be fetched

 ret=SQLBindParameter(s_Item1, 1,SQL_PARAM_INPUT,SQL_C_CHAR,
                      SQL_CHAR,6,0,s_parm[ 1],6,NULL);
 ret=SQLBindParameter(s_Item1, 2,SQL_PARAM_INPUT,SQL_C_CHAR,
                      SQL_CHAR,6,0,s_parm[ 2],6,NULL);
          .
          .
 ret=SQLBindParameter(s_Item1,15,SQL_PARAM_INPUT,SQL_C_CHAR,
                      SQL_CHAR,6,0,s_parm[15],6,NULL);

// Bind the columns.
// Bind dependant of EFetch support.

 ret=SQLBindCol(s_Item1, 1, SQL_C_CHAR, ItemID, 8, CheckID);
 ret=SQLBindCol(s_Item1, 2, SQL_C_CHAR, ItemName, 30, CheckName);
 ret=SQLBindCol(s_Item1, 3, SQL_C_FLOAT,fItemPrice, sizeof(float),
                                 CheckPrice);
 ret=SQLBindCol(s_Item1, 4, SQL_C_CHAR, ItemData, 60, CheckData);

//crow will show number of rows actually fetched
ret=SQLExtendedFetch(s_Item1,SQL_FETCH_FIRST,Ordercount,&crow,
                                          fgrRowStatus);
//                          SQL_FETCH_NEXT
//                          SQL_FETCH_PREV
//                          SQL_FETCH_LAST
```

Block Insert

- Insert blocks of records with one SQL call.
- Reduces the flows between the client and server.

```
// Make ready for ORDLIN insert
    strcpy(tmpbfr,"Insert into ORDLIN (OLOID, OLDID, OLWID, OLNBR,
              OLSPWH, OLIID, OLQTY, OLAMNT, OLDLVD, OLDLVT, OLDSTI)
              VALUES (?,?,?,?,?,?,?,?,?,?,?)");

 ret= SQLPrepare(s_Ordlin1,(unsigned char far *)tmpbfr,SQL_NTS);

// Set the parameters.
//the storage areas that we are binding are arrays (lOrderNum,uiDistrict)
//we will fill these arrays with values before we execute
ret=SQLBindParameter(s_Ordlin1, 1,SQL_PARAM_INPUT,SQL_C_LONG ,
         SQL_DECIMAL,9,0,&(lOrderNum);,9,NULL);
ret=SQLBindParameter(s_Ordlin1, 2,SQL_PARAM_INPUT,SQL_C_SHORT,
         SQL_DECIMAL,3,0,&(uiDistrict);,3,NULL);
ret=SQLBindParameter(s_Ordlin1, 3,SQL_PARAM_INPUT,SQL_C_CHAR,
         SQL_CHAR,4,0,szWid[0],4,NULL);

 //  Insert into ORDLIN
 //  m_OrderCount will contain the number of rows to insert
 //  rowcnt is a pointer to keep track of current row number
  ret=SQLParamOptions(s_Ordlin1, m_OrderCount, &rowcnt);

 //  a loop is used to fill in the arrays with contain the data
 //  for the rows to be inserted
  for(ol_ctr=1;ol_ctr<=NewOrd_IO->m_OrderCount;ol_ctr++){

 // Fill parameters for block insert
    lOrderNum[ol_ctr-1] =  NewOrd_IO->m_OrderNum;
    uiDistrict[ol_ctr-1] = NewOrd_IO->m_District;
    strcpy(szWid[ol_ctr-1],s_parm[3]);
       .
       .
       point to next order
    }

  ret=SQLExecute(s_Ordlin1);         // Execute Order Line insert.
```

The next example is a Visual Basic block insert that is significantly faster than a parameterized insert.

```
Dim hStmt As Long
   Dim cbNTS(BLKSIZE - 1) As Long           'NTS array

   Dim lCustnum(BLKSIZE - 1) As Long

   '2nd parm passed by actual length for demo purposes
   Dim szLstNam(7, BLKSIZE - 1) As Byte      'NOT USING NULL ON THIS PARM
   Dim cbLenLstNam(BLKSIZE - 1) As Long      'Actual length of string to pass
   Dim cbMaxLenLstNam As Long                  'Size of one array element

   'These will be passed as sz string so size must include room for null
   Dim szInit(3, BLKSIZE - 1) As Byte        'Size for field length + null
   Dim szStreet(13, BLKSIZE - 1) As Byte     'Size for field length + null
   Dim szCity(6, BLKSIZE - 1) As Byte        'Size for field length + null
   Dim szState(2, BLKSIZE - 1) As Byte       'Size for field length + null
   Dim szZipCod(5, BLKSIZE - 1) As Byte      'Size for field length + null

   Dim fCdtLmt(BLKSIZE - 1) As Single
   Dim fChgCod(BLKSIZE - 1) As Single
   Dim fBalDue(BLKSIZE - 1) As Single
   Dim fCdtDue(BLKSIZE - 1) As Single


   Dim irow As Long               '//row counter for block errors
   Dim lTotalRows As Long         '// ************ Total rows to send *************
   Dim lNumRows  As Long          '// Rows to send in one block
```

```
Dim lRowsLeft As Long           '// Number of rows left to send

Dim I As Long, j As Long
Dim S As String
Dim addr As Long                'check address of array element...

'************** Start *****************************************************
S = "Number of records to insert into QCUSTCDT.  "
S = S & "Use menu option Table Mgmt, Create QCUSTCDT to "
S = S & "create the table.  Use Misc, AS/400 Cmd and CLRPFM "
S = S & "command if you wish to clear it"
S = InputBox(S, gAppName, "500")
If Len(S) = 0 Then Exit Sub

lTotalRows = Val(S)             'Total number to insert

rc = SQLAllocStmt(ghDbc, hStmt)
If (Not (rc = SQL_SUCCESS Or rc = SQL_SUCCESS_WITH_INFO)) Then GoTo errBlockInsert

rc = SQLPrepare(hStmt, _
         "INSERT INTO QCUSTCDT ? ROWS VALUES (?,?,?,?,?,?,?,?,?,?,?)", _
        SQL_NTS)
If (Not (rc = SQL_SUCCESS Or rc = SQL_SUCCESS_WITH_INFO)) Then GoTo errBlockInsert


rc = SQLBindParameter(hStmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, _
                        10, 0, lCustnum(0), 0, ByVal 0)
If (rc = SQL_ERROR) Then Call PrintError(ghDbc, hStmt) '//report and continue

'Pass first parm w/o using a null
cbMaxLenLstNam = UBound(szLstNam, 1) - LBound(szLstNam, 1) + 1
rc = SQLBindParameter(hStmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, _
                        8, _
                        0, _
                        szLstNam(0, 0), _
                        cbMaxLenLstNam, _
                        cbLenLstNam(0))

rc = SQLBindParameter(hStmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, _
                      3, 0, szInit(0, 0), _
                      UBound(szInit, 1) - LBound(szInit, 1) + 1, _
                      cbNTS(0))
If (rc = SQL_ERROR) Then Call PrintError(ghDbc, hStmt)

rc = SQLBindParameter(hStmt, 4, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, _
                      13, 0, szStreet(0, 0), _
                      UBound(szStreet, 1) - LBound(szStreet, 1) + 1, _
                      cbNTS(0))
If (rc = SQL_ERROR) Then Call PrintError(ghDbc, hStmt)

rc = SQLBindParameter(hStmt, 5, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, _
                      6, 0, szCity(0, 0), _
                      UBound(szCity, 1) - LBound(szCity, 1) + 1, _
                      cbNTS(0))
If (rc = SQL_ERROR) Then Call PrintError(ghDbc, hStmt)  '//report and continue

rc = SQLBindParameter(hStmt, 6, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, _
                      2, 0, szState(0, 0), _
                      UBound(szState, 1) - LBound(szState, 1) + 1, _
                      cbNTS(0))
If (rc = SQL_ERROR) Then Call PrintError(ghDbc, hStmt)  '//report and continue

rc = SQLBindParameter(hStmt, 7, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_NUMERIC, _
                        5, 0, szZipCod(0, 0), _
                        UBound(szZipCod, 1) - LBound(szZipCod, 1) + 1, _
                        cbNTS(0))
If (rc = SQL_ERROR) Then Call PrintError(ghDbc, hStmt)  '//report and continue
```

```
rc = SQLBindParameter(hStmt, 8, SQL_PARAM_INPUT, SQL_C_FLOAT, SQL_NUMERIC, _
                      4, 0, fCdtLmt(0), 0, ByVal 0)
If (rc = SQL_ERROR) Then Call PrintError(ghDbc, hStmt)  '//report and continue
rc = SQLBindParameter(hStmt, 9, SQL_PARAM_INPUT, SQL_C_FLOAT, SQL_NUMERIC, _
                      1, 0, fChgCod(0), 0, ByVal 0)
If (rc = SQL_ERROR) Then Call PrintError(ghDbc, hStmt)  '//report and continue
rc = SQLBindParameter(hStmt, 10, SQL_PARAM_INPUT, SQL_C_FLOAT, SQL_NUMERIC, _
                      6, 2, fBalDue(0), 0, ByVal 0)
If (rc = SQL_ERROR) Then Call PrintError(ghDbc, hStmt)  '//report and continue
rc = SQLBindParameter(hStmt, 11, SQL_PARAM_INPUT, SQL_C_FLOAT, SQL_NUMERIC, _
                      6, 2, fCdtDue(0), 0, ByVal 0)
If (rc = SQL_ERROR) Then Call PrintError(ghDbc, hStmt)  '//report and continue


lRowsLeft = lTotalRows        'Initialize row counter
'Number of inserts to run = (Numrows -1)/BLKSIZE +1
For j = 0 To ((lTotalRows - 1) \ BLKSIZE)
   '// set the values...
   For I = 0 To BLKSIZE - 1
      cbNTS(I) = SQL_NTS                        '// init array to NTS
      lCustnum(I) = I + (j * BLKSIZE)           'Customer number = row number
      S = "Nam" & Str(lCustnum(I))              'Last Name
      cbLenLstNam(I) = Len(S)
      rc = String2Byte2D(S, szLstNam(), I)
      'Debug info: Watch address to see layout
      addr = VarPtr(szLstNam())
      addr = CharNext(szLstNam(0, I))           'address of 1,I
      addr = CharPrev(szLstNam(0, I), szLstNam(1, I))     'address of 0, I)
      addr = CharNext(szLstNam(1, I))
      addr = CharNext(szLstNam(6, I))           'should point to null (if used)
      addr = CharNext(szLstNam(7, I))           'should also point to next row

      rc = String2Byte2D("DXD", szInit, I)
      'Vary the length of the street
      S = Mid("1234567890123", 1, ((I Mod 13) + 1))
      rc = String2Byte2D(S, szStreet, I)

      rc = String2Byte2D("Roches", szCity, I)
      rc = String2Byte2D("MN", szState, I)
      rc = String2Byte2D("55902", szZipCod, I)
      fCdtLmt(I) = I
      fChgCod(I) = 1
      fBalDue(I) = 2 * I
      fCdtDue(I) = I / 2
   Next I

   lNumRows = lTotalRows Mod BLKSIZE      '//Number of rows to send in this block
   If (lRowsLeft >= BLKSIZE) Then _
               lNumRows = BLKSIZE         '//send remainder or full block
   irow = 0
   lRowsLeft = lRowsLeft - lNumRows
   rc = SQLParamOptions(hStmt, lNumRows, irow)
   If (rc = SQL_ERROR) Then GoTo errBlockInsert

   rc = SQLExecute(hStmt)
   If (rc = SQL_ERROR) Then
      S = "Error on Row: " & Str(irow) & Chr(13) & Chr(10)
      MsgBox S, , gAppName
      GoTo errBlockInsert
   End If
   '// Only commit every 50000 records?
   '//if ((lNumRows % 50000) == 0) {
   'rc = SQLTransact(ghEnv, ghDbc, SQL_COMMIT )
   '// if (Not (rc = SQL_SUCCESS OR rc = SQL_SUCCESS_WITH_INFO))  then goto
   '// errBlockInsert;
   '// S="Committed 50,000:"
```

```
      '// S = S & "     lCustnum = : " & Str(lCustnum)
      '// MsgBox S, , gAppName
      '//}
      '//left over from win 3.1
      'DoEvents
   Next j                          '}  // j loop (number of blocks)


   rc = SQLTransact(ghEnv, ghDbc, SQL_COMMIT)
   If (Not (rc = SQL_SUCCESS Or rc = SQL_SUCCESS_WITH_INFO)) Then GoTo errBlockInsert
   rc = SQLFreeStmt(hStmt, SQL_DROP)
   Screen.MousePointer = 0
   StatusBar1.Panels.Item(1).Text = "Block Insert Successfully Inserted " _
                                    & Str(lTotalRows) & " Records"
   Exit Sub



errBlockInsert:
   Call PrintError(ghDbc, hStmt)
   rc = SQLTransact(ghEnv, ghDbc, SQL_ROLLBACK)
   rc = SQLFreeStmt(hStmt, SQL_DROP)
   Screen.MousePointer = 0
   StatusBar1.Panels.Item(1).Text = "Block Insert Failed"

Public Function String2Byte2D(InString As String, OutByte() As Byte, RowIdx As Long)
As Boolean
  'VB byte arrays are layed out in memory opposite of C.  The string would
  'be by column instead of by row so must flip flop the string.
  'ASSUMPTIONS:
  '   Byte array is sized before being passed
  '   Byte array is padded with nulls if > size of string

   Dim I As Integer
   Dim SizeOutByte As Integer
   Dim SizeInString As Integer

   SizeInString = Len(InString)
   SizeOutByte = UBound(OutByte, 1)

   'Convert the string
   For I = 0 To SizeInString - 1
      OutByte(I, RowIdx) = AscB(Mid(InString, I + 1, 1))
   Next I
   'If byte array > len of string pad
   If SizeOutByte > SizeInString Then             'Pad with Nulls
      For I = SizeInString To SizeOutByte - 1
         OutByte(I, RowIdx) = 0
      Next I
   End If
   'ViewByteArray OutByte, "String2Byte"
   String2Byte2D = True
End Function
```

Ending ODBC Functions

The last function that must be done before ending an ODBC application is to free the resources and memory allocated by the application. This must be done so that they are available when the application is run the next time.

- SQLFreeStmt
  - Stops processing associated with a specific statement handle.

```
SQLFreeStmt(hstmt,option); // option can be SQL_CLOSE
                                             SQL_DROP
                                             SQL_UNBIND
```

- SQLDisconnect
  - Closes the connection associated with a specific connection handle.

    ```
    SQLDisconnect(hdbc);
    ```
- SQLFreeConnect
  - Releases connection handle and frees all memory associated with a connection handle.

    ```
    SQLFreeConnect(hdbc);
    ```
- SQLFreeEnv
  - Frees environment handle and releases all memory associated with the environment handle.

    ```
    SQLFreeEnv(henv);
    ```

Comparison of ODBC Techniques

Using the different techniques discussed can result in very different performances for your application.

The following figure shows the number of communication I/O operations and some response times for a “sample” order entry style operation. The identical application was written using three different methods. All of the examples were done using Visual Basic. The first method uses Visual Basic database objects, the second method uses ODBC APIs, and the third method uses a combination of ODBC APIs and stored procedures. The following table summarizes performance information for these applications. As is shown in the table, reducing I/O requests between the client and server can dramatically affect response time.

Figure 6-2. ODBC I/O and Response Times

	I/O Count	Response time (secs)
Visual Basic Database Objects	660	39.0
ODBC APIs	85	5.0
ODBC APIs calling a stored procedure	3	1.5

Note: This system configuration for this was:

- i486® DX/2 '66
- 16 Meg
- 4 Meg Token Ring
- Visual Basic 3.0 Professional

VB 4.0: The compromise between Jet and ODBC APIs

While the database objects are easy to code, Figure 6-2 on page 6-20 shows just how adversely they affect performance. While coding to the APIs and to stored procedures can be a frustrating adventure. What's a developer to do?

Luckily if you are using Visual Basic 4.0 Enterprise Edition in the Windows 95 environment, there are additional options. These options are a good compromise between the usability of database objects and the high performance of APIs: Remote Data Objects (RDO) and Remote Data Control (RDC).

The RDO is a thin layer over the ODBC APIs. It provides a simple interface to advanced ODBC functionality without requiring programming to the API level. It contrasts from the normal Jet controlled Data Access Object (DAO) in that the RDO doesn't have all of the overhead of the Jet Engine or it's SQL optimizer, while maintaining a nearly identical programming interface as the DAO's. If you understand programming to the DAO, then switching over to the RDO is relatively simple compared to trying to switch over to API calls.

The following are differences between DAO and RDO:

- The DAO model is used for ISAM, Access and ODBC databases. The RDO model is designed for ODBC databases only, and it has been optimized for Microsoft SQL Server 6.0 and Oracle.
- The RDO model can have better performance, with the processing being done by the server and not the local machine. Some processing is done locally with the DAO model, so performance may not be as good.
- The DAO model uses the Jet Engine. The RDO model does not use Jet Engine, it uses the ODBC backend engine.
- The RDO model has the capability to perform synchronous or asynchronous queries. The DAO model has limitations in performing these type of queries.
- The RDO model can perform complex cursors, which are limited in the DAO model.

The RDC is a data control similar to the standard data control. This means that where ever you might have used a data control, and the Jet engine, you can now use the RDC. So you can drag a "data aware" control on your form and it can be bound to a RDC just like it could be bound to a regular data control.

Some of the advanced ODBC functionality the RDO allows is prepared SQL statements, multiple result sets, and stored procedures. When Jet executes a SQL statement dynamically it is a two-step process on the AS/400. In the first step, the AS/400 looks at the statement and determines the best plan to retrieve the data requested based upon the current database schema. In the second step, that plan is used to actually retrieve the data. Creating that plan can be expensive in terms of time because the AS/400 has to evaluate many alternatives and determine the best way to access the data. Rather than forcing the AS/400 to re-create the access plan every time a SQL statement is run the CreatePreparedStatement method of the rdoConnection object

allows you to compile a data access plan on the AS/400 for an SQL statement without executing it. You can even include parameters in prepared statements, so you can pass new selection criteria every time you run the select statement.

The following sample VB code will show how to prepare a SQL statement with a parameter marker and run it multiple times with different values.

```
Private Sub Command1_Click()

 Dim rdoEnv As rdoEnvironment
 Dim rdoConn As rdoConnection
 Dim rdoPS As rdoPreparedStatement
 Dim rdoRS As rdoResultset
 Dim strSQL As String

A→ strSQL = "Select * from Customer where CUSTNUM=?"
 Set rdoEnv = rdoCreateEnvironment("TestEnv", "GUEST", "GUEST")
 Set rdoConn = rdoEnv.OpenConnection("Customer Data", rdDriverComplete)
 Set rdoPS = rdoConn.CreatePreparedStatement("MyFirstPS", strSQL)

B→ rdoPS.rdoParameters(0).Value = "17"
 Set rdoRS = rdoPS.OpenResultset()
 Debug.Print rdoRS("CUSTNAME"), rdoRS.RowCount
C→ rdoRS.MoreResults

 rdoPS.rdoParameters(0).Value = "13"
 rdoRS.Requery
 Debug.Print rdoRS("CUSTNAME"), rdoRS.RowCount

 Debug.Print "Done"

End Sub
```

Figure 6-3. VB 4.0 RDO sample code

Label A in Figure 6-3 shows where the SQL statement is defined. Notice that the statement does not include a specific for the CUSTNUM, but has a question mark for the value. The question mark signifies that this value is a parameter of the prepared statement. Before you can create a result set with the prepared statement, you must set the value of any parameters in the statement.

Label B in Figure 6-3 shows where the value for the parameter is defined. Notice that the first parameter is defined as 0 not as 1. Once the value for the parameter is set you can run the OpenResultSet method of the rdoPreapredStatement to return the requested data.

Before you can requery a prepared statement on the AS/400, you have to make sure that the cursor has been completely processed and closed. Label C in Figure 6-3 shows the MoreResults method of the rdoResultSet being used to do this. The MoreResults method will query the database and determine if there is any more data in

the result set to be processed or if the result set has been processed completely. Once the cursor has been fully processed you can reset the parameter value and run the ReQuery method of the rdoResultSet to open a new result set.

Chapter 7. Stored Procedures

This chapter describes how you can take advantage of stored procedures when developing client/server applications that use ODBC. Stored procedures provide a standard way to call a program on the AS/400 from within an application by using aSQL statement. It is important to note that on the AS/400 a stored procedure is a program object.

The invocation of a stored procedure is treated as a regular external call: the application will wait for the stored procedure to terminate. Stored procedures allow for input, output, and input/output parameters, as well as result sets. Stored procedures can be called locally (on the same system that is the applications database server) and remotely (on a different system). However, you may find stored procedures particularly useful in client/server environments because they may considerably improve performance by reducing the traffic of information across the communication network.

For example, if an application needs to perform several database operations on the server, you can choose between issuing many different database requests from the client or calling a stored procedure. In the first case, each request results in a communication flow to the server; however, if you call a stored procedure, only the call request and the parameters will flow to the server. In addition, the server will execute some of the logic of your application with potential performance benefits for the client.

Stored procedures can be used for many different application purposes, for example:

- Distributing the logic between the client and the server
- Performing a sequence of query functions on the server
- Combining results of query functions on the server
- Generating statistical data
- Performing non-database functions

This chapter shows typical examples in which stored procedures can be very effective. In the first example, a company runs an order entry client/server application. Entering an order causes the following steps to occur on the server:

- Updating the invoice master file reflects the new invoice that is generated.
- Updating the invoice detail file reflects the detailed information for the new invoice.
- Updating the accounts receivable file reflects the amount of the new invoice.
- Updating the inventory file reflects the items that are ordered.

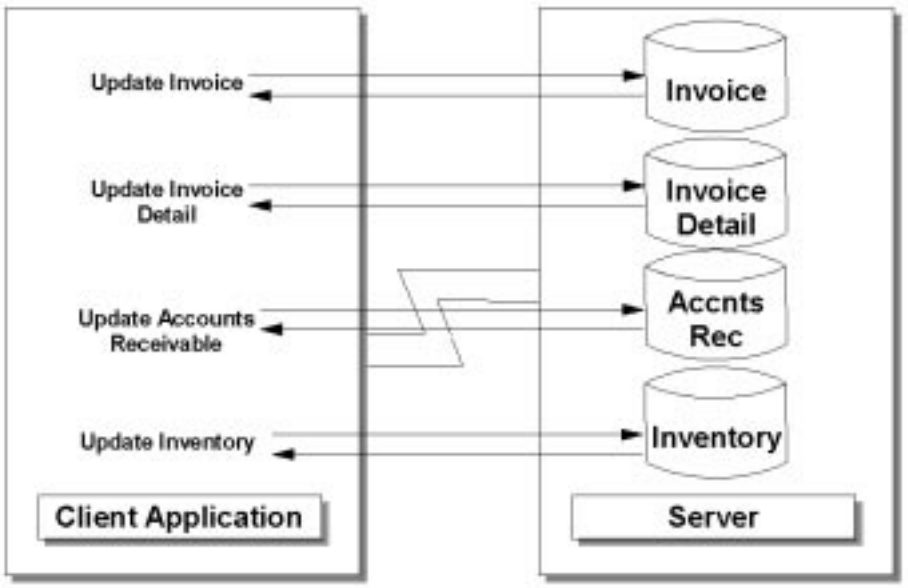


Figure 7-1. Client/server application without stored procedures

The diagram in Figure 7-1 shows a client/server application for order entry processing that was created without stored procedures. The client system has to access the server several times for every part of the transaction, sending and receiving data across the communications line for every request. In addition, the client carries out all the application logic.

In Figure 7-2 on page 7-3, you can see how to take advantage of stored procedures in implementing the following application.

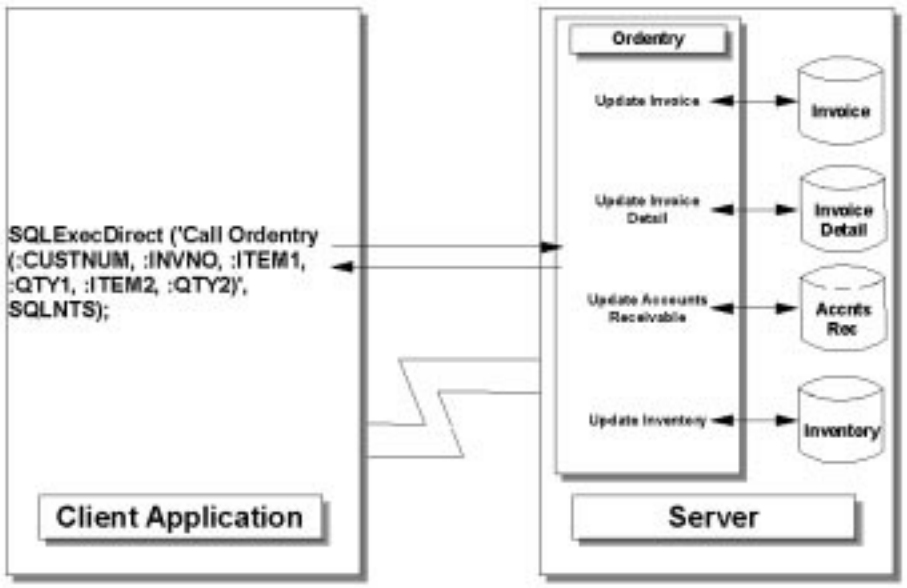


Figure 7-2. Client/server application with stored procedures

Calling a single stored procedure, which runs on the server, carries out the same application transaction. This reduces the communication flows and splits the application logic, which better balances the computational resources of the network.

For further references or information concerning stored procedures please see *DB2 for AS/400 SQL Programming*, SC41-5611.

Examples of stored procedures

For information regarding specific SQL commands that are used in these examples of stored procedures, please see *DB2 for AS/400 SQL Reference*, SC41-5612

Stored procedure calls from C with return values

```
#define CHAR_COL_LEN 10         /* Character column length       */

UCHAR col1[CHAR_COL_LEN+1];

/* Execute the CREATE PROCEDURE statement. This step creates an  */
/* entry for the procedure in the system catalog. This Create    */
/* only needs to be executed one time.                           */

   _fstrcpy(stmt,"CREATE PROCEDURE P1 (:HV1 INOUT CHAR(10)) ");
   _fstrcat(stmt,"(EXTERNAL NAME JMBLIB.P1 LANGUAGE PLI SIMPLE CALL)");
   returncode = SQLExecuteDirect(hstmt,stmt,SQL_NTS);"
   if (returncode != SQL_SUCCESS)
      {
       /* Handle error case here.                                  */
      }
      _fstrcpy(stmt,"CALL P1 (?)");
      returncode = SQLPrepare(hstmt,stmt,SQL_NTS);
      if(returncode != SQL_SUCCESS)
        {
         /* Handle error here  */
        }
    /*************************************/
    /* Perform set param for argument and */
    /* initialize argument                */
    /*************************************/
   SQLBindParameter(hstmt,1,SQL_PARAM_INPUT_OUTPUT,SQL_C_CHAR,
   SQL_CHAR,CHAR_COL_LEN,0,col1,sizeof(col1),NULL);
     _fstrcpy(col1,CHAR_CONST1);

   /*************************************/
   /* Execute the CALL statement        */
   /*************************************/

   returncode = SQLExecute(hstmt);
   if(returncode != SQL_SUCCESS)
      {
       /* Handle error here   */
      }
   else
       {
        /* Process returned value here */
       }
```

Stored procedure calls from C with result sets

This result sets example calls an RPG program and receives the data as a result set array. The program on the PC can issue the DROP PROCEDURE and CREATE PROCEDURE statements before issuing a CALL procedure. After returning from the call, do an ODBC SQLBindCol to set the variables to return data into. Then do ODBC SQLFetch to get the data.

The following is an example of code showing the ODBC calls:

```
#include "sql.h"
#include "sqlext.h"
#include "string.h"
/* declare the variables  */
#define ROW_COUNT 20
UCHAR           fetch_all[106];
SDWORD          cbcol;
static HENV     henv;
static HDBC     hdbc;
static HSTMT    hstmt;
char            stmt[4000];
int             ii;
RETCODE         rc;
/* connect to a data source  */
rc = SQLAllocEnv(&henv);
rc = SQLAllocConnect(henv, &hdbc);
rc = SQLConnect(hdbc, "SystemA", SQL_NTS,
                "JohnB", SQL_NTS,
                "Sesame", SQL_NTS);
/* allocate a statement handle  */
SQLAllocStmt(hdbc,&hstmt);
/* drop the old procedure  */
_fstrcpy(stmt,"DROP PROCEDURE RPGARY");
rc = SQLExecDirect(hstmt, stmt, SQL_NTS);
/* create the procedure to call  */
_fstrcpy(stmt,"CREATE PROCEDURE RPGARY RESULT SETS 1 EXTERNAL");
_fstrcat(stmt," NAME JMBLIB.RPGARY LANGUAGE RPG GENERAL");
rc = SQLExecDirect(hstmt, stmt, SQL_NTS);
/* call stored procedure  */
_fstrcpy(stmt,"CALL RPGARY()");


rc = SQLExecDirect(hstmt, stmt, SQL_NTS);
/* bind the columns to return variables   */
SQLBindCol(hstmt, 1, SQL_C_CHAR, fetch_all, 106, &cbcol);
/* for each row fetch the data  */
for(ii=0; ii<ROW_COUNT ;ii++)
{
  rc = SQLFetch(hstmt);
  /* process data */
}
/* get more results  */
rc = SQLMoreResults(hstmt);
/* release the statement handle  */
SQLFreeStmt(hstmt,SQL_DROP);
/* release the connection to the data source  */
SQLDisconnect(hdbc);
SQLFreeConnect(hdbc);
SQLFreeEnv(henv);
```

Host RPG source

The RPG program called by the procedure should declare an array to hold the data. It can then open files and read the data in to the array and do whatever other processing needs to be done. The SQL SET RESULT SETS statement identifies the number of rows and attributes of the array result set.

```
 *-------------------------------------------------------------*
 *                                                             *
 * RPG PROGRAM:  RPGARY                                        *
 *                                                             *
 * PURPOSE: THIS RPG PROGRAM IS CALLED BY AN ODBC STORED       *
 *          PROCEDURE.  IT READS DATA INTO AN ARRAY AND        *
 *          RETURNS THE RESULTS BY USING THE SET RESULT        *
 *          SETS SQL FUNCTION                                  *
 *                                                             *
 *-------------------------------------------------------------*
 * FILE DESCRIPTION SPECIFICATIONS                             *
 *-------------------------------------------------------------*
FPERF    IF  E                    DISK
 *-------------------------------------------------------------*
 * INPUT SPECIFICATIONS                                        *
 *-------------------------------------------------------------*
IPERFE     E DSPERF
IARR         DS                        20
I                                      01 105 VAR1
ILCLVAR      DS
I                                    B  01  040L#
 *-------------------------------------------------------------*
C*      **  START OF PROGRAM  **                                *


 *-------------------------------------------------------------*
C                    OPEN PERF
C                    Z-ADD0         L#
C           LOOP     TAG
C                    READ PERF                   9999
C                    ADD  1         L#
C           L#       OCUR ARR
C                    MOVE PERFE     ARR
C           L#       COMP 20                   99  99
C  N99               GOTO LOOP
 *-------------------------------------------------------------*
 *    *** EXECUTE SQL STATEMENT ***
C/EXEC SQL SET RESULT SETS ARRAY :ARR FOR :L# ROWS
C/END-EXEC
 *-------------------------------------------------------------*
C                    CLOSEPERF
C                    RETRN
=============================================================
```

Running CL commands through SQL stored procedures and ODBC

Stored procedure support provides a means to run AS/400 Control Language (CL) commands using the SQL CALL statement. Cases where running CL commands are beneficial include performing an override for files, initiating debug, and other commands including those that can affect the performance of subsequent SQL statements.

The following examples show cases where a CL command is run on the AS/400 by means of the CALL statement by calling the program that processes CL commands. That program (QCMDEXC in library QSYS) expects two parameters: the first is a string containing the command text to execute and the second is decimal(15,5) field containing the length of the command text. The parameters need to have exactly these attributes for the command to be interpreted properly, so the second parameter on the CALL statement needs to have characters explicitly specified for all places of the decimal (15,5) field.

In the following example, a C program on the PC is going to run an OVRDBF command that is 65 characters long (including embedded blanks). The text of the OVRDBF command is as follows:

```
OVRDBF FILE(TESTER) TOFILE(JMBLIB/TESTER) MBR(NO2) OVRSCOPE(*JOB)
```

The code for performing this command by means of ODBC APIs is as follows:

```
SQLAllocStmt(hdbc,&hstmt);
strcpy(stmt,"call qsys.qcmdexc('OVRDBF FILE(TESTER) TOFILE(JMBLIB/");
strcat(stmt,"TESTER) MBR(NO2) OVRSCOPE(*JOB)',0000000065.00000)");
SQLExecDirect(hstmt,stmt,SQL_NTS);
```

Statements now run against file jmblib/tester will reference member no2 rather than the first member.

Another CL command that you might find useful to run against a database server job is the STRDBG command. This command forces SQL to generate completion messages for all SQL statements run. In addition, the query optimizer generates messages indicating how the queries run in the job were optimized. This information can be used to troubleshoot queries that are performing poorly. All of the messages are written to the joblog of the server job running on the AS/400. This joblog can then be studied to identify problem queries or operations.

In the example below, a C program on the PC is going to run a STRDBG command that is 20 characters long (including embedded blanks). The text of the STRDBG command is as follows:

```
STRDBG UPDPROD(*YES)
```

The code for performing this command by way of ODBC APIs is as follows:

```
SQLAllocStmt(hdbc,&hstmt);
strcpy(stmt,"call qsys.qcmdexc('STRDBG UPDPROD(*YES)',");
strcat(stmt,"0000000020.00000)");
SQLExecDirect(hstmt,stmt,SQL_NTS);
```

SQL statements run in the job will generate completion messages and optimizer debug messages.

Stored procedure calls from PowerBuilder with no return values

PowerBuilder has the ability to call stored procedures on the server that it is connected to. If the stored procedure has only input variables, and no output variables or result sets, then the definition is done through a combination of two powerscript commands:

DECLARE PROCEDURE Declares the procedure name, an alias for the procedure used by PowerBuilder, and the variables to be passed to the stored procedure at execution time to PowerBuilder, but not the AS/400.

EXECUTE Calls the stored procedure with any defined variables.

The final procedure code that defines the stored procedure to PowerBuilder, calls the stored procedure with input variables, and checks the SQL return codes from the AS/400 would look like the following:

```
integer custnumber

declare store_proc1 procedure for storedp
:custnumber;

custnumber = integer(sle_customer_number.text)

execute store_proc1;

if sqlca.sqldbcode <> 0 then
   messagebox ("Stored Procedure Error",
   "The stored procedure failed to find the customer&n&r;"&
   +sqlca.sqlerrtext, Exclamation!, OK!)
else
   messagebox ("Stored Procedure complete",
   "The stored procedure completed normally", Exclamation!, OK!)
end if

close(parent)
```

Host PL/I source

Following you will find the PL/I source code to the stored procedures used in this exercise.

```
0001.00 storedp:  proc(incustnum);
0002.00 exec sql include sqlca;
0003.00 dcl sysprint file print external stream;
0004.00 dcl incustnum fixed binary(31);
0005.00 /*****************************************************************/
0006.00 /* Receive the parameter and then generate the file for the query */
0007.00 /* If the input value is out of range, print an error and return  */
0008.00 /* an escape message so that SQLCODE -443 is returned to the app. */
0009.00 /* If an error occurs on an SQL statement, the message will be in */
0010.00 /* the joblog.                                                    */
0011.00 /*                                                               */
0012.00 /*****************************************************************/
0013.00 exec sql whenever sqlerror continue;
0014.00 exec sql create table odbcsamp/outfile (custnum integer, custname
0015.00          char(35), address char(35), city char(25), state char(2),
0016.00          phone char(15));
0017.00 exec sql whenever sqlerror goto err;
0018.00 exec sql delete from odbcsamp/outfile;
0019.00 if (incustnum < 1) || (incustnum >  100) then
0020.00    do;
0021.00      open file(sysprint);
0022.00      put file(sysprint) edit('Invalid input number entered',
0023.00          incustnum) (a,f(5,0));
0024.00      goto err;
0025.00    end;
0026.00 else;
0027.00 exec sql insert into odbcsamp/outfile select custnum,custname,
0028.00          address,city,state,phone from odbcsamp/customer where
0029.00           custnum = :incustnum;
0030.00 goto ok;
0031.00 err:
0032.00  signal error;
0033.00  ok:
0034.00 return;
0035.00 end storedp;
```

Stored procedure calls from PowerBuilder with return values

PowerBuilder has the ability to call stored procedures on the server that it is connected to. One way in which this is done is through a remote procedure call (RPC). An RPC is used when you want to use a stored procedure which has return values, not result sets. The following steps will define a RPC:

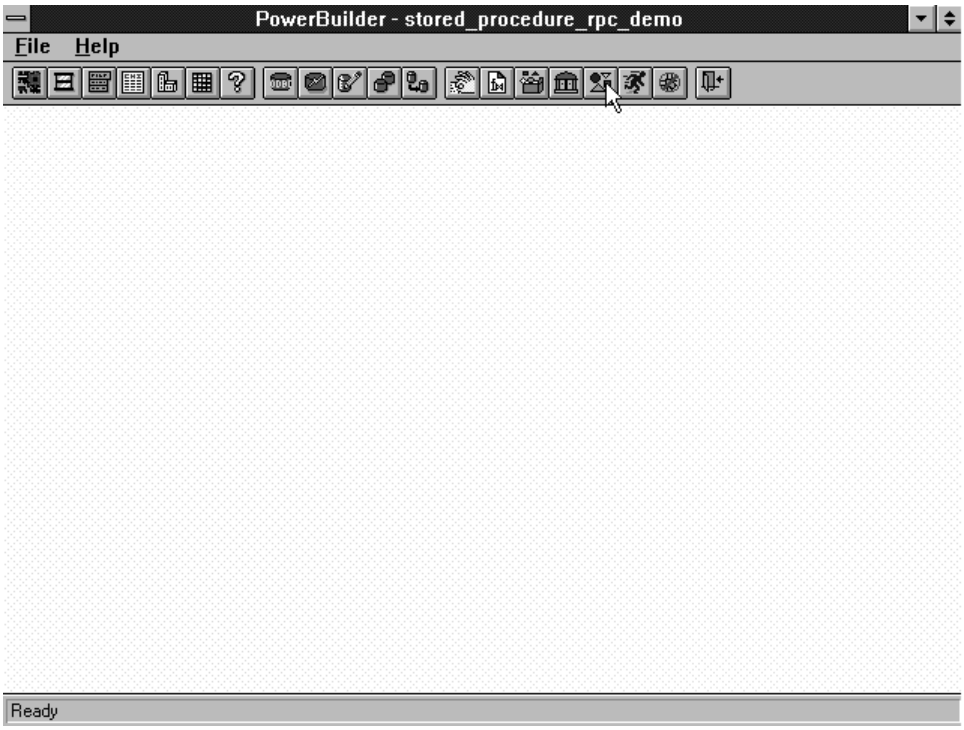


Figure 7-3. PowerSoft PowerBuilder screen

1. Press the User Object button on the toolbar. It is the 4th button from the right.

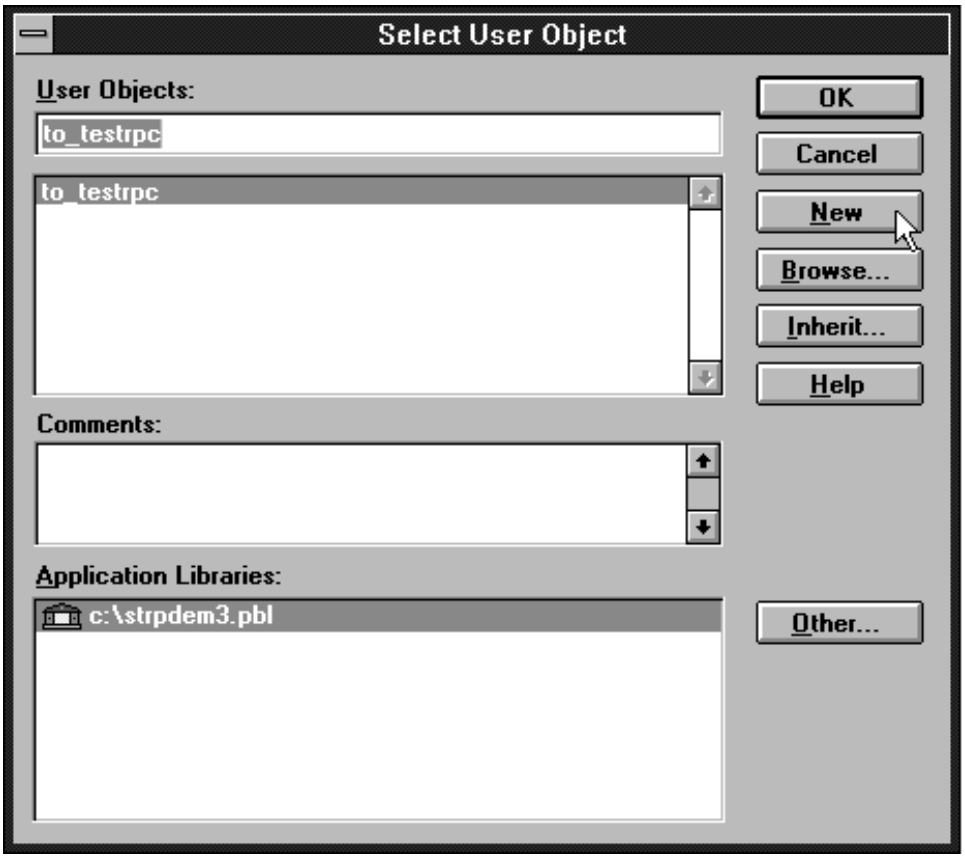


Figure 7-4. Select User Object screen

2. Press the New button.

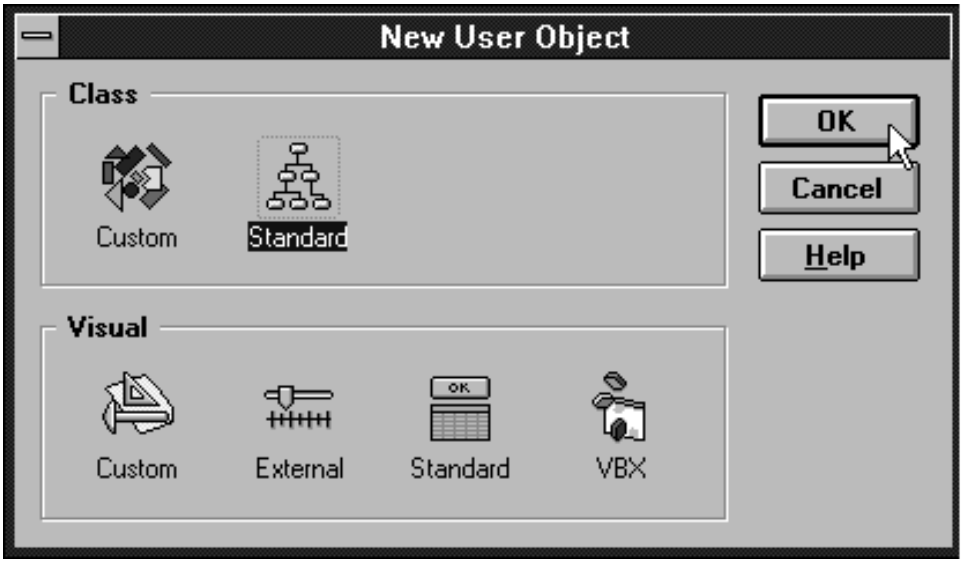


Figure 7-5. New User Object screen

3. Select ***Standard*** from the Class selection area
4. Press the OK button.

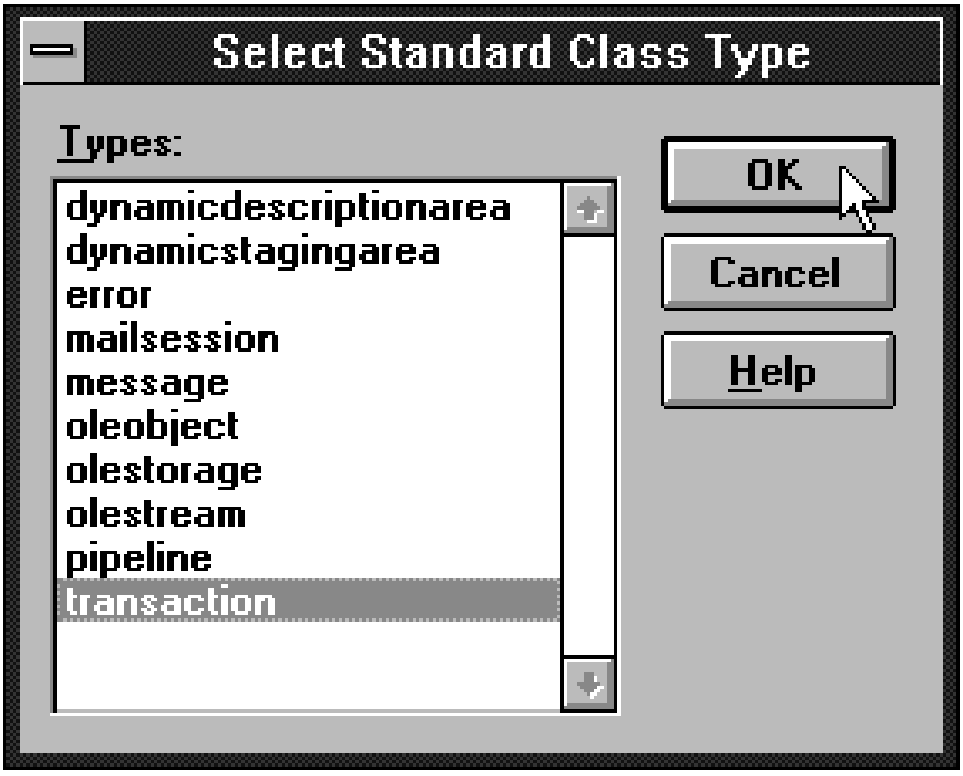


Figure 7-6. Select Standard Class Type screen

5. Select ***transaction*** from the Types selection box
6. Press the OK button.

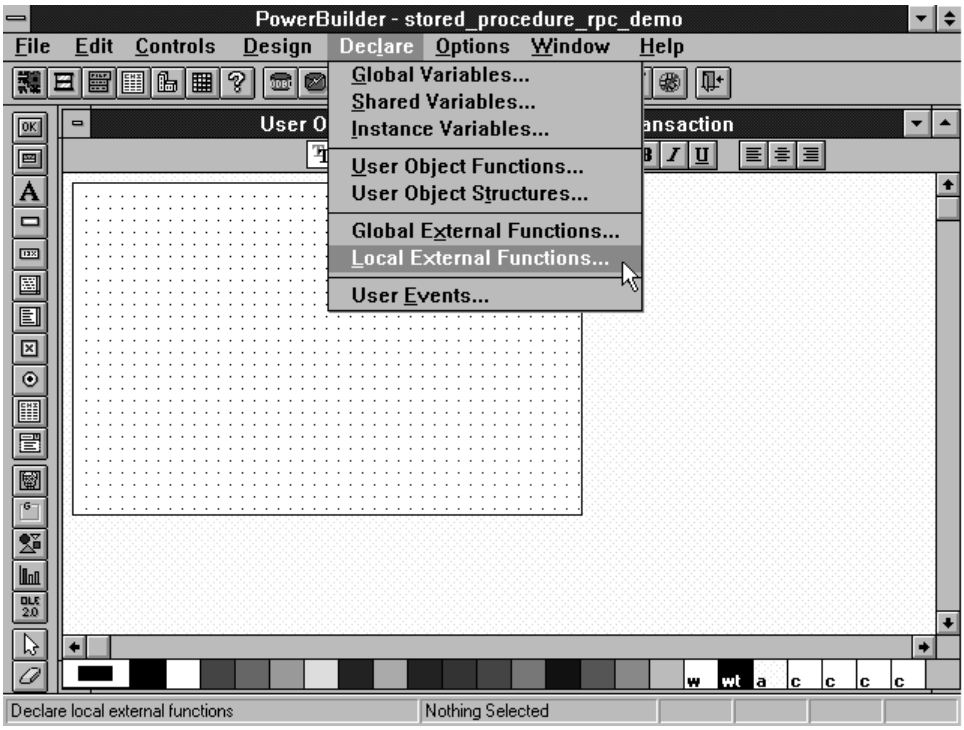


Figure 7-7. PowerSoft PowerBuilder screen

7. From the menu select Declare - Local External Functions

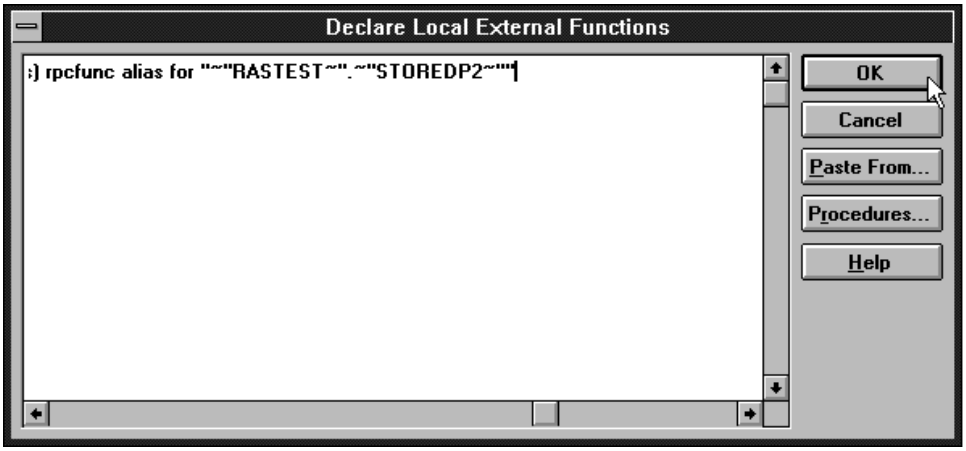


Figure 7-8. Declare Local External Function screen

8. Declare the function to be run
9. Our example RPC function declaration is the following:

- subroutine sp_storedp (integer custnumber, ref string custname, ref string address, ref string city, ref string state, ref string phone, ref string status) rpcfunc alias for ""RASTEST"."STOREDP2""
- Where sp_storedp is the name used to call the procedure from within PowerBuilder
- Where output parameters are pass by reference
- This declaration should be all on one line

10. Press the OK button

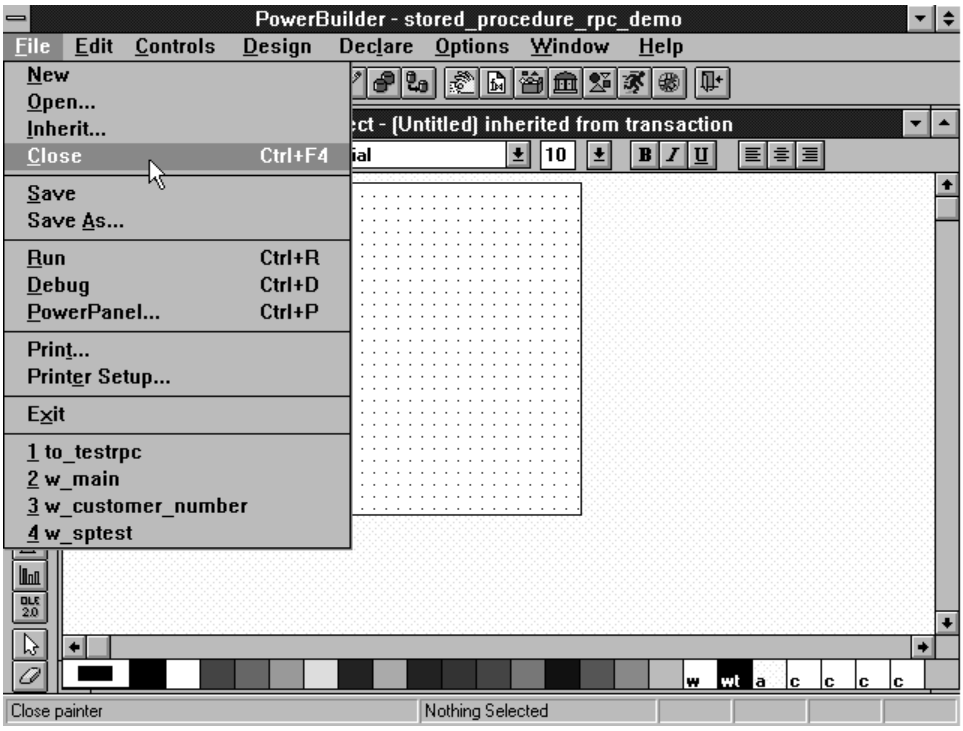


Figure 7-9. User Object screen

11. From the menu select File - Close

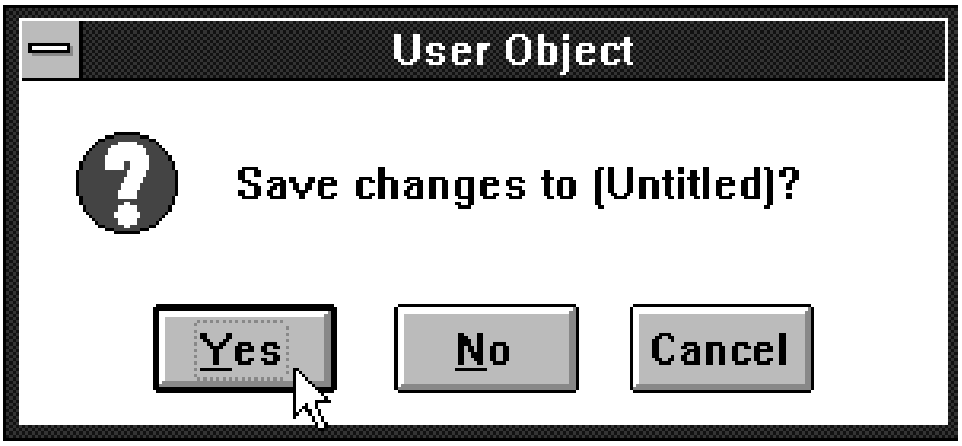


Figure 7-10. User Object confirmation screen

12. Press the Yes button to save changes.

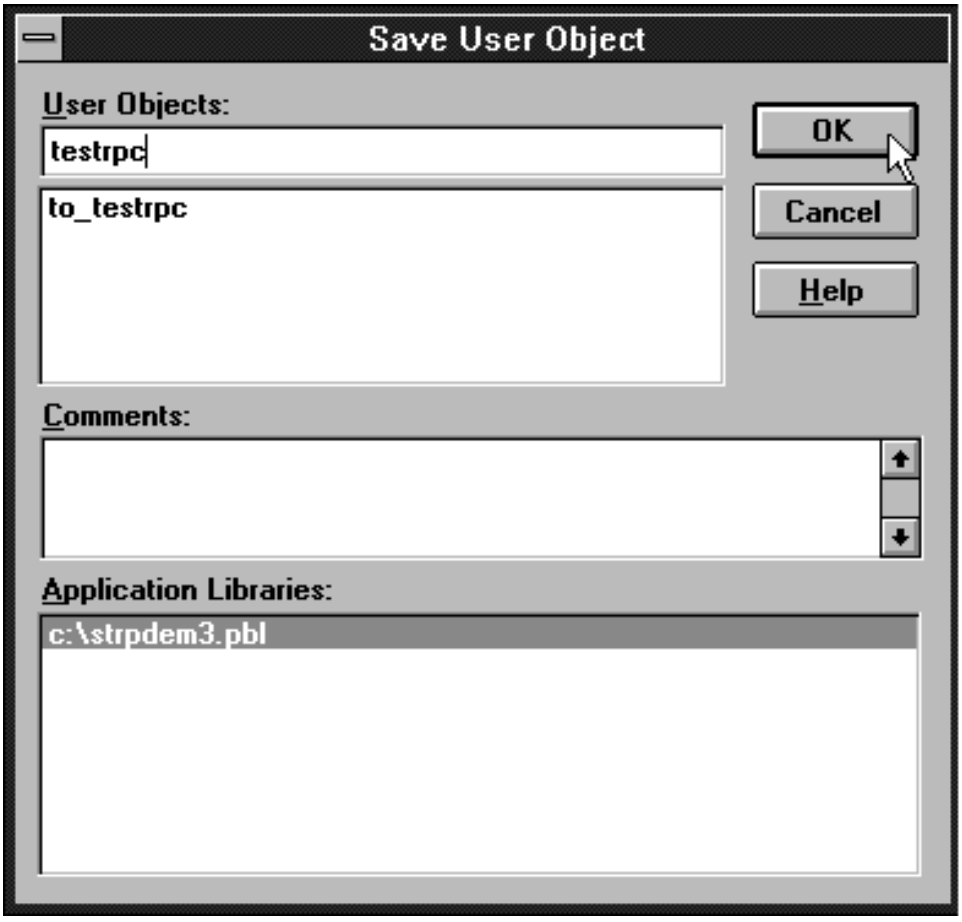


Figure 7-11. Save User Object screen

13. Enter a name for you user object into the User Objects text entry box
14. Press the OK button.

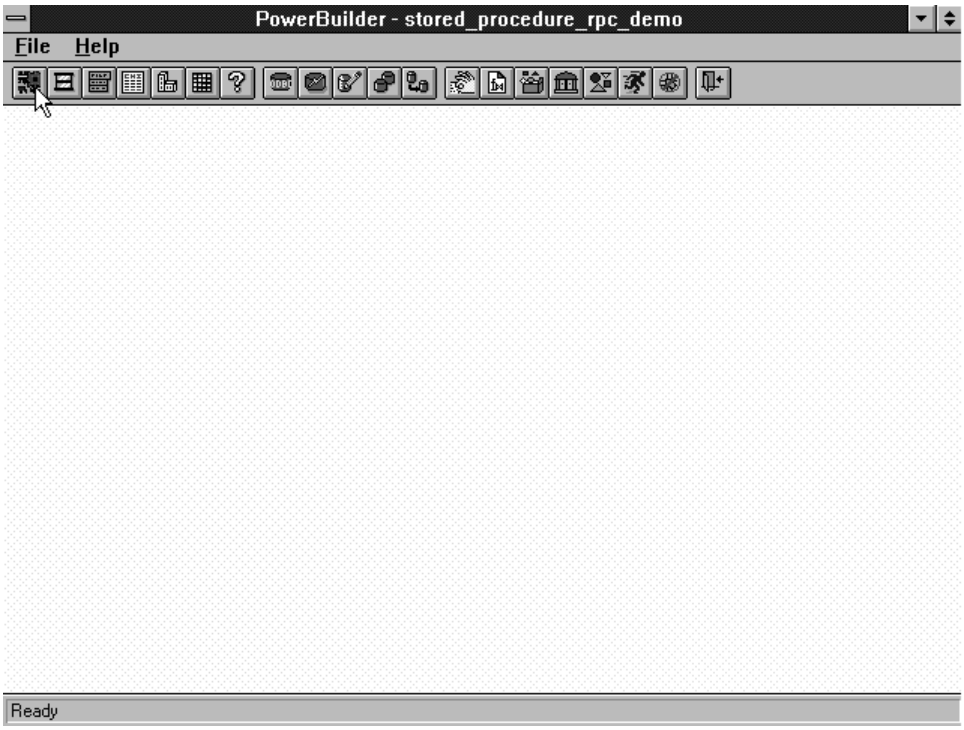


Figure 7-12. PowerBuilder screen

15. Press the Application button from the toolbar, it is the 1st button on the left.

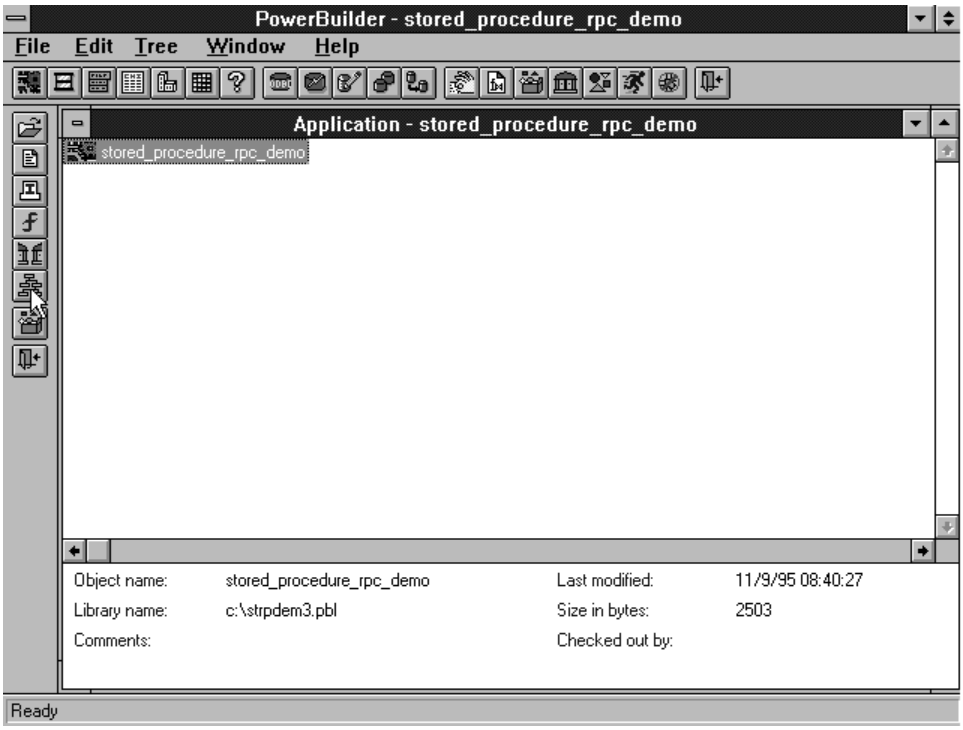


Figure 7-13. PowerBuilder screen

16. Press the Default Global Variable button, it is the 3rd button up from the bottom.

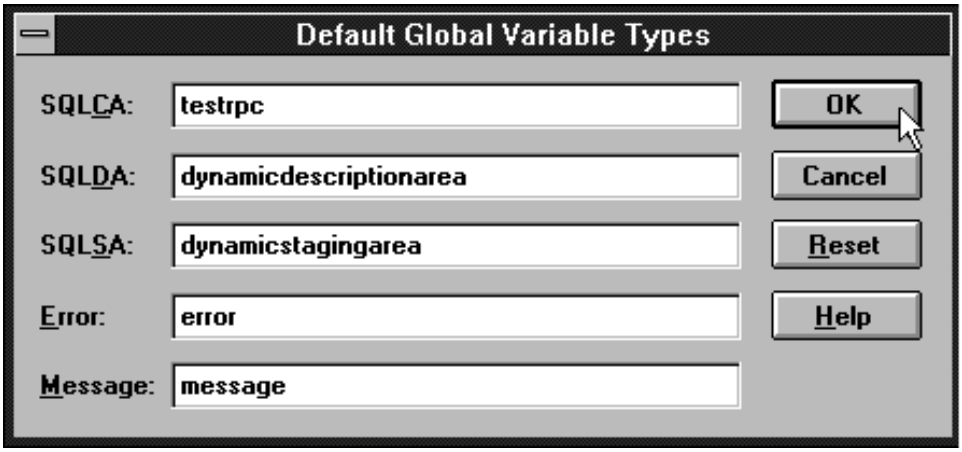


Figure 7-14. Global Variable Types screen

17. Set the SQLCA variable to the name you saved your user object under.
18. Press the OK button

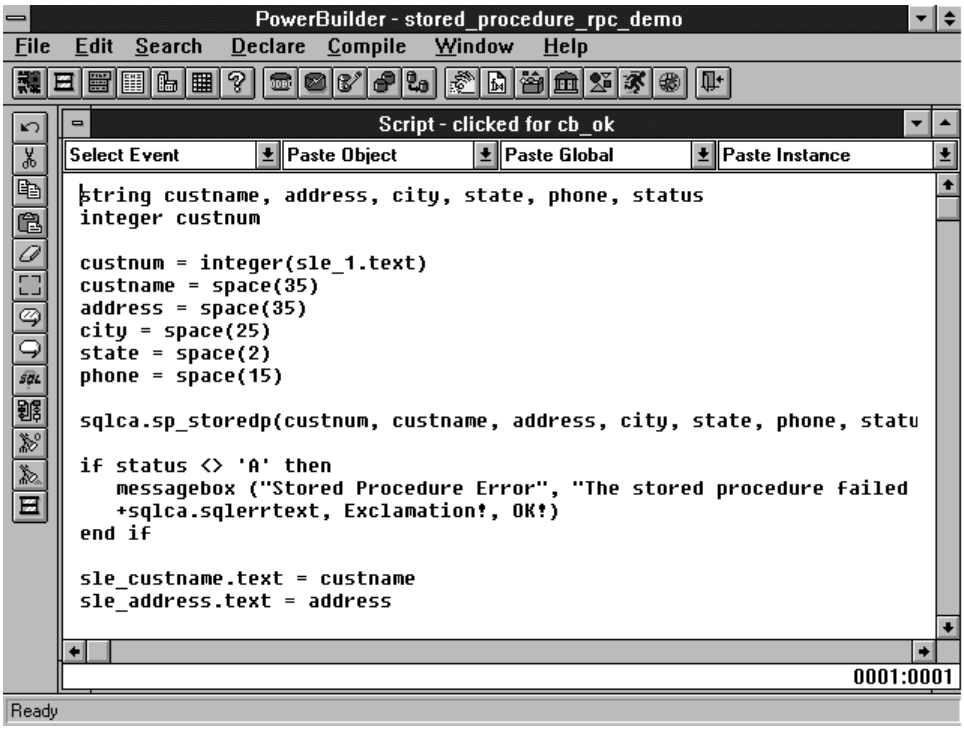


Figure 7-15. Script Painter

19. To call the externally defined procedure use the following command:
    - SQLCA.PROCEDURENAME(PARAMETER1, PARAMETER2, ... PARAMETERn)
    - In my sample code the call is sqlca.sp_storedp(custnum, custname, address, city, state, phone, status)
    - If you are passing character strings by reference make sure to initialize them to their maximum size. This will allow PowerBuilder to properly pass these strings back to your applications. See in my example above the call to sqlca.sp_storedp for example.

Host COBOL source

Following you will find the COBOL source code to the stored procedures used in this exercise.

```
0031.00
0032.00        IDENTIFICATION DIVISION.
0033.00
0034.00        PROGRAM-ID. STOREDP2.
0035.00        ENVIRONMENT DIVISION.
0036.00        CONFIGURATION SECTION.
0037.00        SOURCE-COMPUTER. IBM-AS400.
0038.00        OBJECT-COMPUTER. IBM-AS400.
0039.00
0040.00        INPUT-OUTPUT SECTION.
0041.00
0042.00        FILE-CONTROL.
0043.00
0044.00        DATA DIVISION.
0045.00
0046.00        FILE SECTION.
0047.00
0048.00 ****************************************************************
0049.00 ****************************************************************
0050.00
0051.00        WORKING-STORAGE SECTION.
0052.00
SQL                 EXEC SQL INCLUDE SQLCA
SQL                 END-EXEC.
0055.00
0055.01        01 INDIC1 PIC S9(4) COMP-4.
0055.02        01 INDIC2 PIC S9(4) COMP-4.
0055.03        01 INDIC3 PIC S9(4) COMP-4.
0055.04        01 INDIC4 PIC S9(4) COMP-4.
0055.05        01 INDIC5 PIC S9(4) COMP-4.
0055.06        01 INDIC6 PIC S9(4) COMP-4.
0056.00
0088.00
0089.00 ****************************************************************
0090.00 ****************************************************************
0091.00
0092.00        LINKAGE SECTION.
0093.00
0094.00        01 CUSTNUM PIC S9(9) COMP-4.
0095.00        01 CUSTNAME PIC X(35).
0096.00        01 CUSTADDR PIC X(35).
0097.00        01 CITY   PIC X(25).
0098.00        01 STATE  PIC X(2).
0099.00        01 PHONE  PIC X(15).
0099.01        01 STATF  PIC X.
0104.00
0105.00 ****************************************************************
0106.00 ****************************************************************
```

```
0107.00 ************************************************************
0108.00 ************************************************************
0109.00
0110.00        PROCEDURE DIVISION
0111.00             USING CUSTNUM
0112.00                   CUSTNAME
0113.00                   CUSTADDR
0114.00                   CITY
0115.00                   STATE
0116.00                   PHONE
0117.00                   STATF.
0121.00
SQL                 EXEC SQL WHENEVER SQLERROR GO TO 60-SQLERRTAG
SQL                 END-EXEC.
SQL                 EXEC SQL WHENEVER SQLWARNING GO TO 70-SQLWARNTAG
SQL                 END-EXEC.
SQL                 EXEC SQL WHENEVER NOT FOUND GO TO 80-SQLNOTFND
SQL                 END-EXEC.
0123.05
0124.00             PERFORM 10-RUN-SELECT.
0127.00             GO TO 999-END-PGM.
0128.00
0129.00        10-RUN-SELECT.
0129.01             MOVE "A" TO STATF.
SQL                 EXEC SQL SELECT CUSTNUM,CUSTNAME,ADDRESS,CITY,STATE,PHONE
SQL                     INTO :CUSTNUM :INDIC1, :CUSTNAME
SQL                       :INDIC2, :CUSTADDR :INDIC3,
SQL                       :CITY :INDIC4, :STATE
SQL                       :INDIC5, :PHONE :INDIC6
SQL                      FROM RASTEST.CUSTOMER WHERE CUSTNUM = :CUSTNUM
SQL                   END-EXEC.
0362.00
0371.00
0372.00        60-SQLERRTAG.
0374.00             MOVE "B" TO STATF.
0374.01             GO TO 999-END-PGM.
0374.03        70-SQLWARNTAG.
0374.04             MOVE "C" TO STATF.
0374.05             GO TO 999-END-PGM.
0374.06        80-SQLNOTFND.
0374.07             MOVE "D" TO STATF.
0379.00
0380.00        999-END-PGM.
0381.00             EXIT.
```

Stored procedure calls from PowerBuilder with result sets

PowerBuilder has the ability to call stored procedures on the server that it is connected to. One way in which this is done is through a datawindow. A datawindow is used when you want to use a stored procedure which is returning a result set. The following steps will show setting up the datawindow:

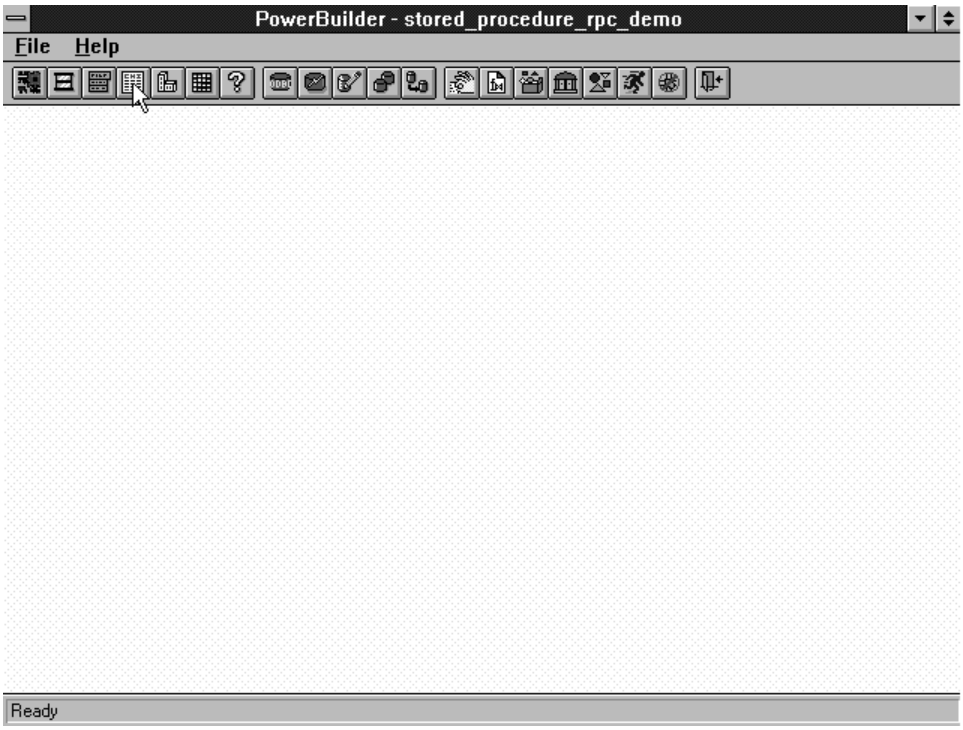


Figure 7-16. PowerSoft PowerBuilder screen

1. Press the DataWindow button on the toolbar. It is the 4th button from the left.

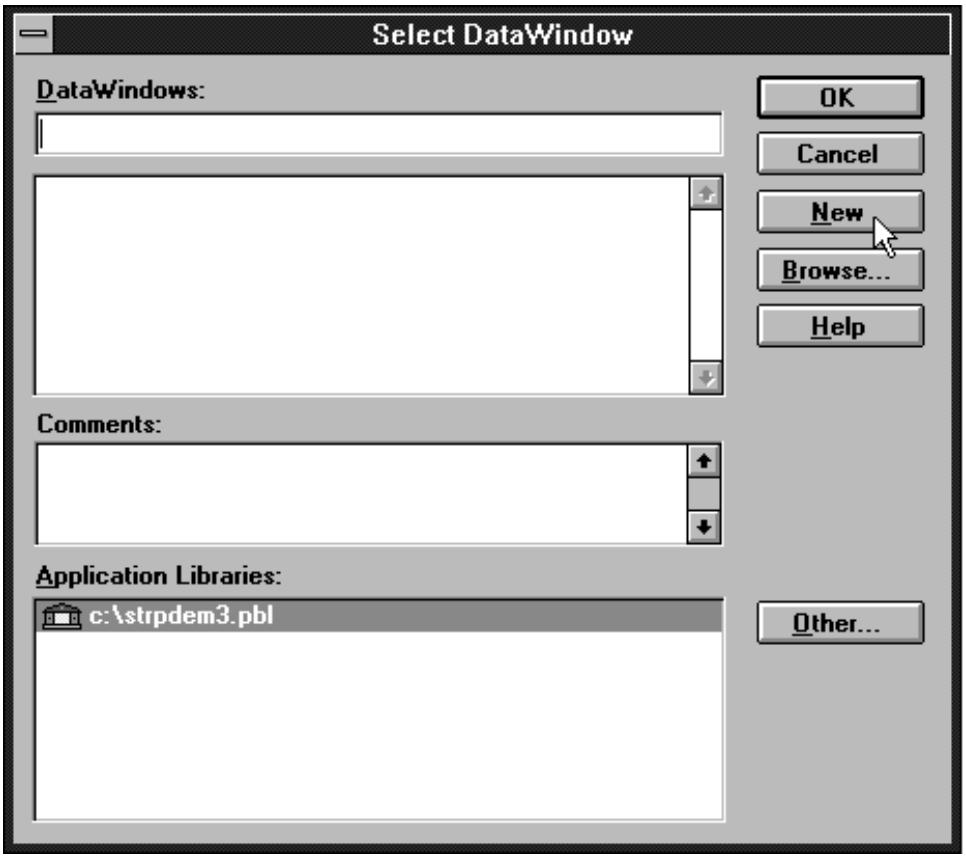


Figure 7-17. Select DataWindow screen

2. Press the New button

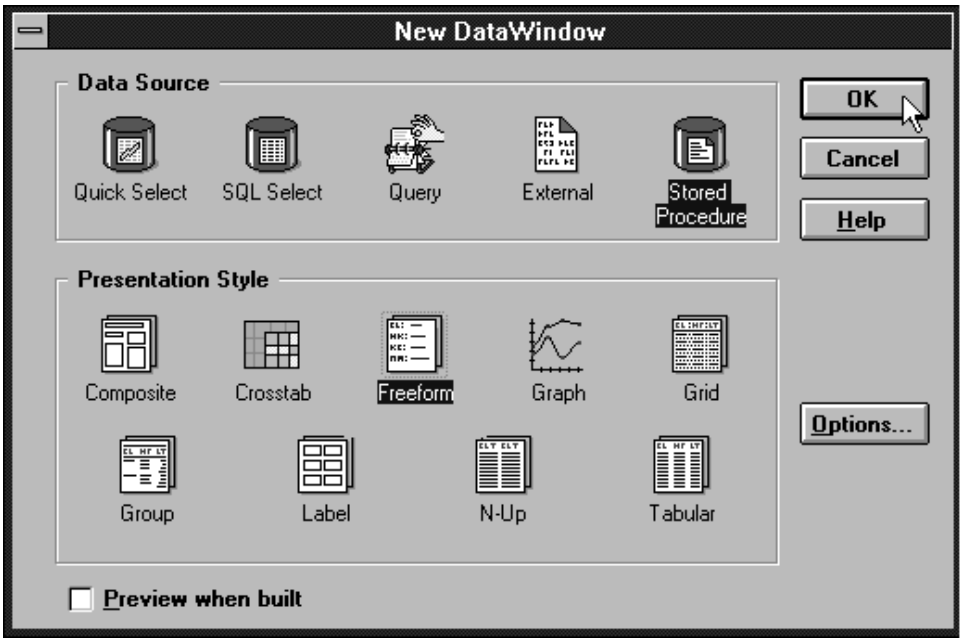


Figure 7-18. New DataWindow screen

3. From the Data Source selection box select Stored Procedure
4. From the Presentation Style selection box select the style of datawindow you want to create. My example will build a freeform datawindow.

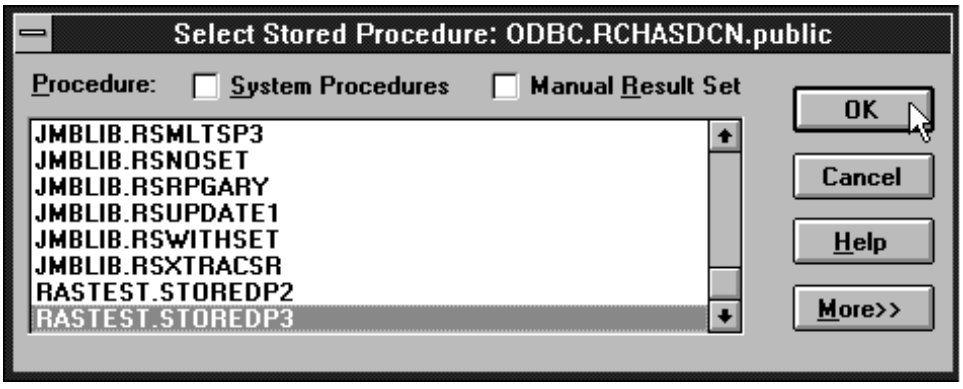


Figure 7-19. Select Stored Procedure screen

5. Select the stored procedure you want to use to generate the datawindow. The selected stored procedure has to have a result set or else PowerBuilder will not be able to generate the datawindow correctly.

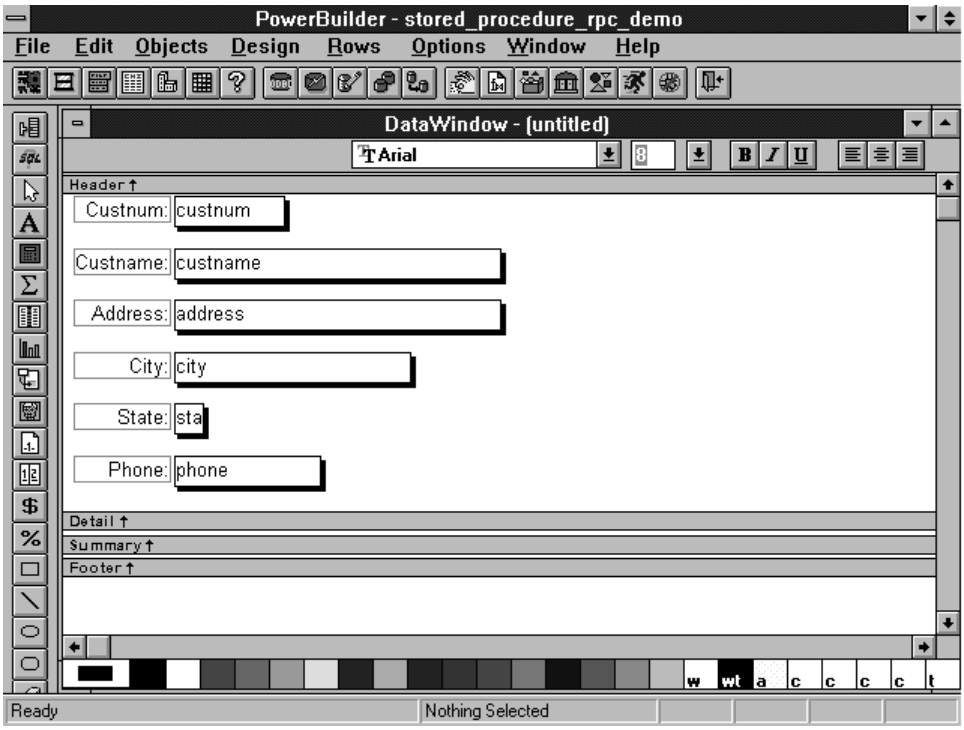


Figure 7-20. DataWindow Painter screen

6. PowerBuilder will then ask the AS/400 for the record format of the result set expected to be returned. PowerBuilder will then generate a datawindow which reflects the structure of the result set.

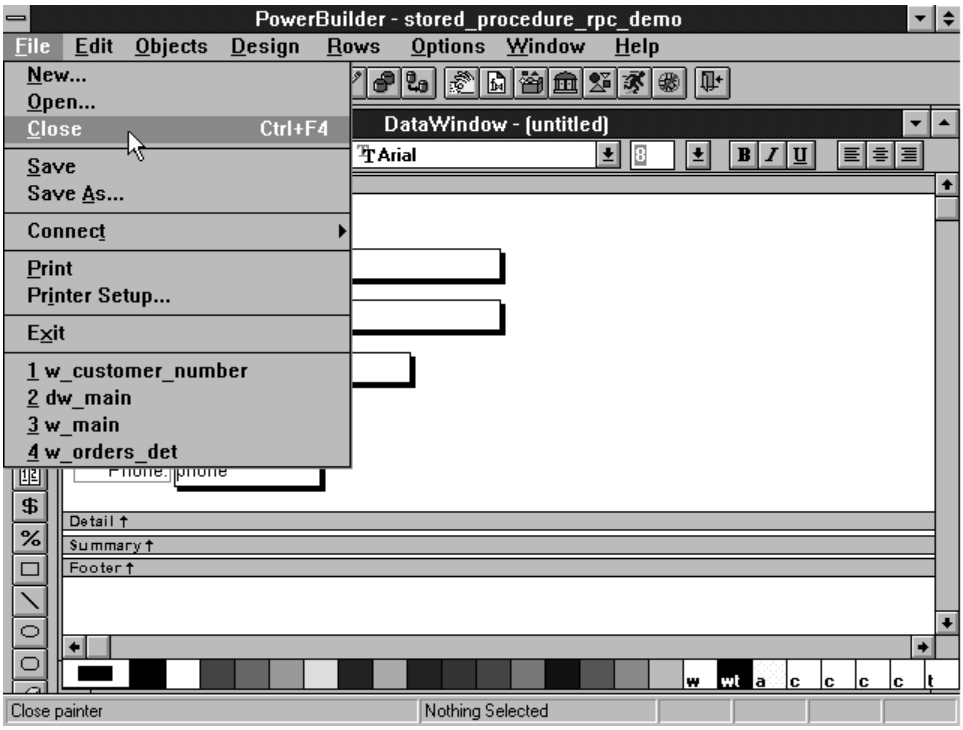


Figure 7-21. DataWindow Painter screen

7. From the menu select File - Close

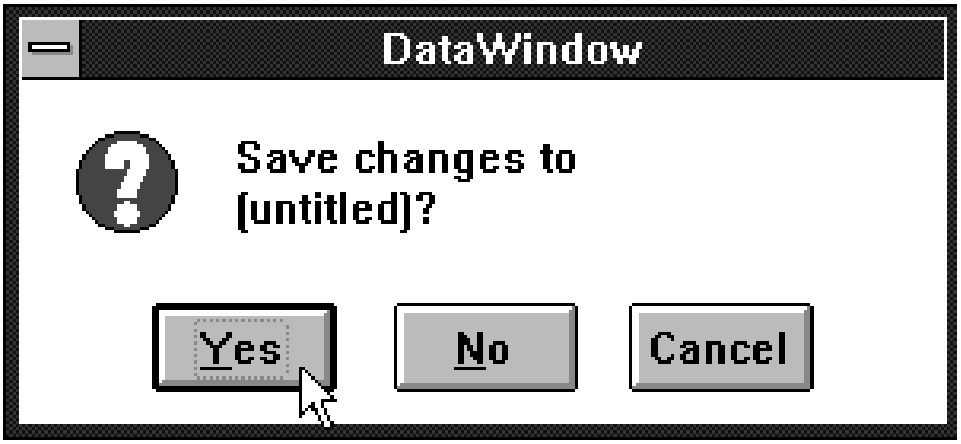


Figure 7-22. Verify Save Changes screen

8. Press the Yes button

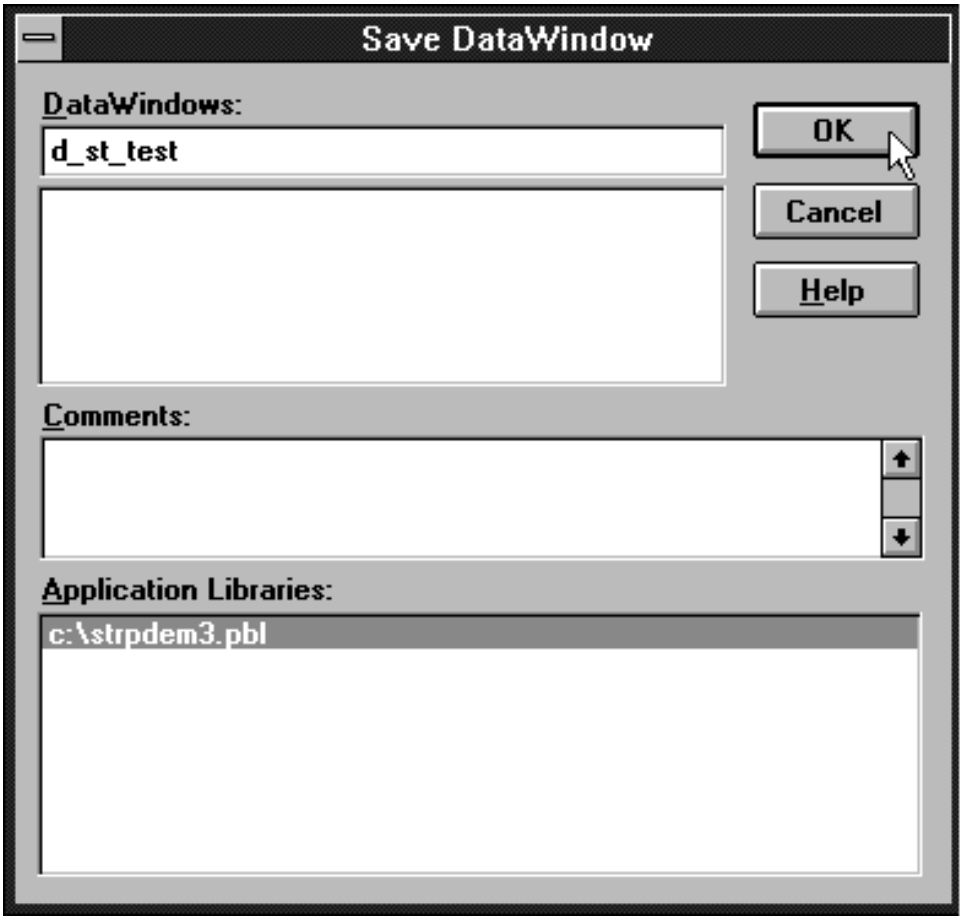


Figure 7-23. Save DataWindow screen

9. Name the datawindow
10. Press the OK button

Host COBOL source

Following you will find the COBOL source code to the stored procedures used in this exercise.

```
0031.00
0032.00        IDENTIFICATION DIVISION.
0033.00
0034.00        PROGRAM-ID. STOREDP3.
0035.00        ENVIRONMENT DIVISION.
0036.00        CONFIGURATION SECTION.
0037.00        SOURCE-COMPUTER. IBM-AS400.
0038.00        OBJECT-COMPUTER. IBM-AS400.
0039.00
0040.00        INPUT-OUTPUT SECTION.
0041.00
0042.00        FILE-CONTROL.
0043.00
0044.00        DATA DIVISION.
0045.00
0046.00        FILE SECTION.
0047.00
0048.00 ****************************************************************
0049.00 ****************************************************************
0050.00
0051.00        WORKING-STORAGE SECTION.
0052.00
SQL                 EXEC SQL INCLUDE SQLCA
SQL                 END-EXEC.
0055.00
0055.01        01 INDIC1 PIC S9(4) COMP-4.
0055.02        01 INDIC2 PIC S9(4) COMP-4.
0055.03        01 INDIC3 PIC S9(4) COMP-4.
0055.04        01 INDIC4 PIC S9(4) COMP-4.
0055.05        01 INDIC5 PIC S9(4) COMP-4.
0055.06        01 INDIC6 PIC S9(4) COMP-4.
0056.00
0088.00
0089.00 ****************************************************************
0090.00 ****************************************************************
0091.00
0092.00        LINKAGE SECTION.
0093.00
0094.00        01 CUSTNUM PIC S9(9) COMP-4.
0104.00
0105.00 ****************************************************************
0106.00 ****************************************************************
```

```
0107.00 ****************************************************************
0108.00 ****************************************************************
0109.00
0110.00        PROCEDURE DIVISION
0111.00             USING CUSTNUM.
0121.00
SQL                 EXEC SQL WHENEVER SQLERROR GO TO 60-SQLERRTAG
SQL                 END-EXEC.
SQL                 EXEC SQL WHENEVER SQLWARNING GO TO 70-SQLWARNTAG
SQL                 END-EXEC.
SQL                 EXEC SQL WHENEVER NOT FOUND GO TO 80-SQLNOTFND
SQL                 END-EXEC.
0123.05
0124.00             PERFORM 10-RUN-SELECT.
0127.00             GO TO 999-END-PGM.
0128.00
0129.00        10-RUN-SELECT.
SQL                 EXEC SQL
SQL                 DECLARE C1 CURSOR FOR SELECT CUSTNUM, CUSTNAME, ADDRESS,
SQL                  CITY,STATE,PHONE FROM RASTEST.CUSTOMER WHERE CUSTNUM =
SQL                    :CUSTNUM
SQL                    END-EXEC.
SQL                 EXEC SQL OPEN C1 END-EXEC.
0371.00
0372.00        60-SQLERRTAG.
0374.01             GO TO 999-END-PGM.
0374.03        70-SQLWARNTAG.
0374.05             GO TO 999-END-PGM.
0374.06        80-SQLNOTFND.
0379.00
0380.00        999-END-PGM.
0381.00             EXIT.
```

Stored procedure calls from Visual Basic with return values

Visual Basic has the ability to call external functions found in a DLL. Since all ODBC drivers are DLLs, Visual Basic can be used to code directly to the ODBC APIs. By coding directly to the ODBC APIs a Visual Basic application can call an AS/400 stored procedure and return result values.

The following example of Visual Basic code will show how to call an AS/400 stored procedure and then retrieve the returned values into Visual Basic variables.

```
'**********************************************************************
'*                                                                    *
'* Because of the way Visual Basic stores and manages the String data  *
'* type, it is recommended that you use an array of Byte data type     *
'* instead of a String variable on the SQLBindParameter API.           *
'*                                                                    *
'**********************************************************************

Dim sTemp As String
Custnum As Integer
```

```
Dim abCustname(34) As Byte
Dim abAddress(34) As Byte
Dim abCity(24) As Byte
Dim abState(1) As Byte
Dim abPhone(14) As Byte
Dim abStatus As Byte
Dim RC As Integer
Dim nullx As Long        'Used to pass null pointer, not pointer to null
Dim lpSQL_NTS As Long    'Used to pass far pointer to SQL_NTS
Static link(7) As Long   'Used as an array of long pointers to the size
                         'each parameter which will be bound


 '************************************************************************
 '*                                                                      *
 '* Initialize the variables needed on the API calls                     *
 '*                                                                      *
 '************************************************************************

link(1) = 6
link(2) = Ubound(abCustname) +1
link(3) = Ubound(abAddress) +1
link(4) = Ubound(abCity) +1
link(5) = Ubound(abState) +1
link(6) = Ubound(abPhone) +1
link(7) = 1

RC = 0
nullx = 0
lpSQL_NTS = SQL_NTS      ' -3 means passed as sz string

'************************************************************************
'*                                                                      *
'* Create the procedure on the AS/400. This will define the             *
'* procedure's name, parameters, and how each parameter is passed.      *
'* Note: This information is stored in the system catalog tables and    *
'* and only needs to be executed one time for the life of the stored    *
'* procedure.  It normally would not be run in the client application. *
'*                                                                      *
'************************************************************************

sTemp = "Create Procedure Storedp2 (:Custnum in integer, "
sTemp = sTemp & ":Custname out char(35), :Address out char(35),"
sTemp = sTemp & ":City out char(25), :State out char(2),"
sTemp = sTemp & ":Phone out char(15), :Status out char(1))
sTemp = sTemp & "(External name rastest.storedp2 language cobol General)"

RC = SQLExecDirect(Connection.hstmt, sTemp, Len(sTemp))

'Ignore error assuming that any error would be from procedure already
'created.
```

```
'*********************************************************************
'*                                                                   *
'* Prepare the call of the procedure to the AS/400.                  *
'* For best performance, prepare the statement only one time and
'* execute many times.
'*                                                                   *
'*********************************************************************

sTemp = "Call storedp2(?, ?, ?, ?, ?, ?, ?)"
RC = SQLPrepare(Connection.hstmt, sTemp, Len(sTemp))

If (RC <> SQL_SUCCESS) Then
   DescribeError Connection.hdbc, Connection.hstmt
   frmMain.Status.Caption = "Error on SQL_Prepare " & RTrim$(Tag)
End If

'*********************************************************************
'*                                                                   *
'* Bind all of the columns passed to the stored procedure. This will   *
'* set up the variable's data type, input/output characteristics,      *
'* length, and initial value.                                        *
'* The SQLDescribeParam API can optionally be used to retrieve the
'* parameter types.
'*                                                                   *
'* To properly pass an array of byte to a stored procedure and receive *
'* an output value back, you must pass the first byte ByRef.         *
'*                                                                   *
'*********************************************************************

RC = SQLBindParameter(Connection.hstmt, 1, SQL_PARAM_INPUT, SQL_C_SHORT, _
      SQL_NUMERIC, 6, 0, Custnum, 6, link(1))

RC = SQLBindParameter(Connection.hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_CHAR, _
      SQL_CHAR, 35, 0, abCustname(0), UBound(abCustname)+1, link(2))
RC = SQLBindParameter(Connection.hstmt, 3, SQL_PARAM_OUTPUT, SQL_C_CHAR, _
      SQL_CHAR, 35, 0, abAddress(0), UBound(abAddress)+1, link(3))
RC = SQLBindParameter(Connection.hstmt, 4, SQL_PARAM_OUTPUT, SQL_C_CHAR, _
      SQL_CHAR, 25, 0, abCity(0), UBound(abCity)+1, link(4))
RC = SQLBindParameter(Connection.hstmt, 5, SQL_PARAM_OUTPUT, SQL_C_CHAR, _
      SQL_CHAR, 2, 0, abState(0), UBound(abState)+1, link(5))
RC = SQLBindParameter(Connection.hstmt, 6, SQL_PARAM_OUTPUT, SQL_C_CHAR, _
      SQL_CHAR, 15, 0, abPhone(0), UBound(abPhone)+1, link(6))
RC = SQLBindParameter(Connection.hstmt, 7, SQL_PARAM_OUTPUT, SQL_C_CHAR, _
      SQL_CHAR, 1, 0, abStatus, 1, link(7))


'*********************************************************************
'*                                                                   *
'* The Prepare and Bind only needs to be execute once.  The Stored
'* procedure can now be called multiple times by just changing the data
```

```
'*                                                                      *
'************************************************************************
Do While

'************************************************************************
'* Read in a customer number                                           *
'*                                                                      *
'************************************************************************

Custnum = Val(input.text)


'************************************************************************
'*                                                                      *
'* Execute the call of the procedure to the AS/400.                    *
'*                                                                      *
'************************************************************************

RC = SQLExecute(Connection.hstmt)
frmMain.Status.Caption = "Ran Stored Proc" & RTrim$(Tag)

If (RC <> SQL_SUCCESS) Then
   DescribeError Connection.hdbc, Connection.hstmt
   frmMain.Status.Caption = "Error on Stored Proc Execute " & RTrim$(Tag
End If

'************************************************************************
'*                                                                      *
'* Set text labels to display the output data                          *
'* You must convert the array of Byte back to a String
'*                                                                      *
'************************************************************************


lblCustname = StrConv(abCustname(), vbUnicode)
lblAddress = StrConv(abAddress(), vbUnicode)
lblCity = StrConv(abCity(), vbUnicode)
lblState = StrConv(abState(), vbUnicode)
lblPhone = StrConv(abPhone(), vbUnicode)
lblStatus = StrConv(abStatus(), vbUnicode)

Loop
```

Host COBOL source

Following you will find the COBOL source code to the stored procedures used in this exercise.

```
0031.00
0032.00        IDENTIFICATION DIVISION.
0033.00
0034.00        PROGRAM-ID. STOREDP2.
0035.00        ENVIRONMENT DIVISION.
0036.00        CONFIGURATION SECTION.
0037.00        SOURCE-COMPUTER. IBM-AS400.
0038.00        OBJECT-COMPUTER. IBM-AS400.
0039.00
0040.00        INPUT-OUTPUT SECTION.
0041.00
0042.00        FILE-CONTROL.
0043.00
0044.00        DATA DIVISION.
0045.00
0046.00        FILE SECTION.
0047.00
0048.00 ****************************************************************
0049.00 ****************************************************************
0050.00
0051.00        WORKING-STORAGE SECTION.
0052.00
SQL                 EXEC SQL INCLUDE SQLCA
SQL                 END-EXEC.
0055.00
0055.01        01 INDIC1 PIC S9(4) COMP-4.
0055.02        01 INDIC2 PIC S9(4) COMP-4.
0055.03        01 INDIC3 PIC S9(4) COMP-4.
0055.04        01 INDIC4 PIC S9(4) COMP-4.
0055.05        01 INDIC5 PIC S9(4) COMP-4.
0055.06        01 INDIC6 PIC S9(4) COMP-4.
0056.00
0088.00
0089.00 ****************************************************************
0090.00 ****************************************************************
0091.00
0092.00        LINKAGE SECTION.
0093.00
0094.00        01 CUSTNUM PIC S9(9) COMP-4.
0095.00        01 CUSTNAME PIC X(35).
0096.00        01 CUSTADDR PIC X(35).
0097.00        01 CITY   PIC X(25).
0098.00        01 STATE  PIC X(2).
0099.00        01 PHONE  PIC X(15).
0099.01        01 STATF  PIC X.
0104.00
0105.00 ****************************************************************
0106.00 ****************************************************************
```

```
0107.00 ************************************************************
0108.00 ************************************************************
0109.00
0110.00        PROCEDURE DIVISION
0111.00             USING CUSTNUM
0112.00                   CUSTNAME
0113.00                   CUSTADDR
0114.00                   CITY
0115.00                   STATE
0116.00                   PHONE
0117.00                   STATF.
0121.00
SQL                EXEC SQL WHENEVER SQLERROR GO TO 60-SQLERRTAG
SQL                END-EXEC.
SQL                EXEC SQL WHENEVER SQLWARNING GO TO 70-SQLWARNTAG
SQL                END-EXEC.
SQL                EXEC SQL WHENEVER NOT FOUND GO TO 80-SQLNOTFND
SQL                END-EXEC.
0123.05
0124.00            PERFORM 10-RUN-SELECT.
0127.00            GO TO 999-END-PGM.
0128.00
0129.00        10-RUN-SELECT.
0129.01            MOVE "A" TO STATF.
SQL                EXEC SQL SELECT CUSTNUM,CUSTNAME,ADDRESS,CITY,STATE,PHONE
SQL                    INTO :CUSTNUM :INDIC1, :CUSTNAME
SQL                      :INDIC2, :CUSTADDR :INDIC3,
SQL                      :CITY :INDIC4, :STATE
SQL                      :INDIC5, :PHONE :INDIC6
SQL                     FROM RASTEST.CUSTOMER WHERE CUSTNUM = :CUSTNUM
SQL                  END-EXEC.
0362.00
0371.00
0372.00        60-SQLERRTAG.
0374.00            MOVE "B" TO STATF.
0374.01            GO TO 999-END-PGM.
0374.03        70-SQLWARNTAG.
0374.04            MOVE "C" TO STATF.
0374.05            GO TO 999-END-PGM.
0374.06        80-SQLNOTFND.
0374.07            MOVE "D" TO STATF.
0379.00
0380.00        999-END-PGM.
0381.00            EXIT.
```

Stored procedure calls from Visual Basic with result sets

Visual Basic's Data Control has the ability to call a stored procedure which returns a result set. This result set can then be bound to data control, like a data grid.

The following example will show how to code the data control and bind the datagrid to the data to return the result set from the stored procedure.

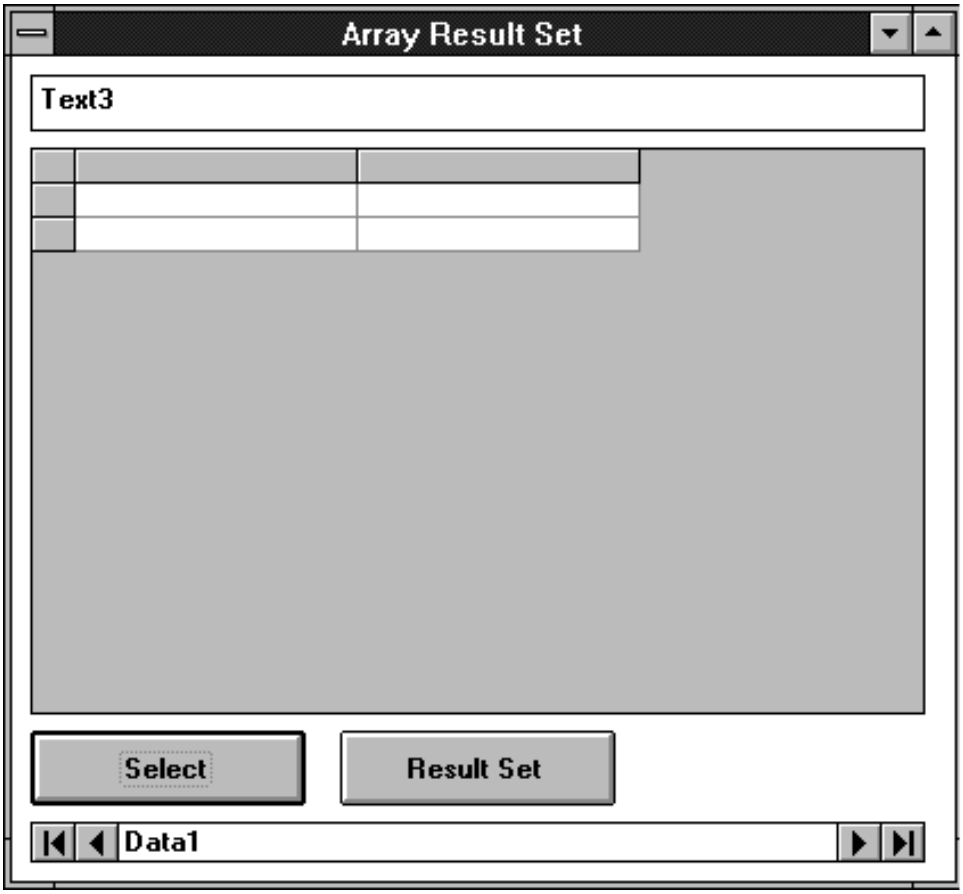


Figure 7-24. Visual basic sample Form

1. Build the form which contains the data control (DATA1) and the bound datagrid. In this form I have also added two command buttons which will show the difference between using a normal select statement in a data control and using a stored procedure.

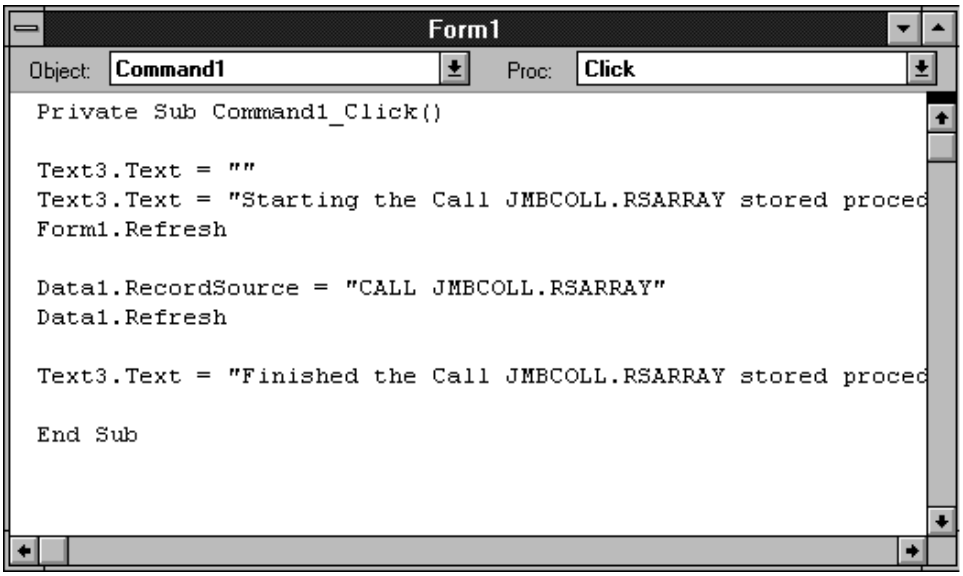


Figure 7-25. Visual Basic sample module

2. In the Click event of Command1 I have setup to show how a stored procedure would be entered into the Data1.RecordSource. Then a Refresh is done of Data1 to run the stored procedure and the result set used to populate the bound datagrid.

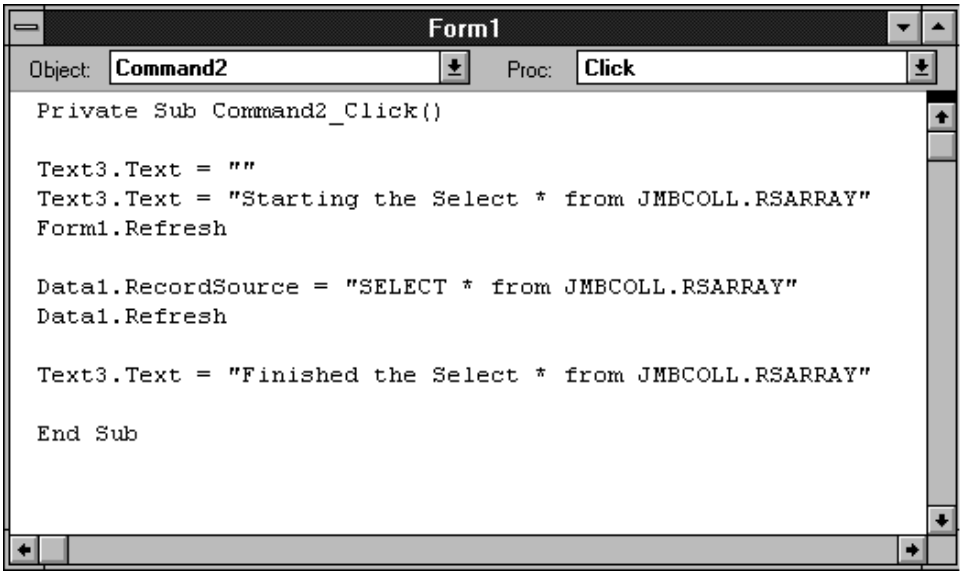


Figure 7-26. Visual Basic sample module

3. In the Click event of Command2 I have setup to show how a normal select statement would be entered into the Data1.RecordSource. Then a Refresh is done of Data1 to populate the bound datagrid.

Chapter 8. Blocked Inserts

The blocked INSERT statement provides a means to insert multiple rows with a single SQLExecute request. For performance, it provides the one of the best ways to populate a table; at times providing a 10 times performance improvement over the next best method.

The three forms of INSERT statements that can be executed from ODBC are:

1. INSERT with VALUES using constants
2. INSERT with VALUES using parameter markers
3. blocked INSERT

The INSERT with VALUES using constants statement is the least efficient method of performing inserts. For each request, a single INSERT statement is sent to the server where it is prepared, the underlying table is opened, and the record is written.

Example:

```
INSERT INTO TEST.TABLE1 VALUES('ENGINEERING',10,'JONES','BOB')
```

The INSERT with VALUES using parameter markers statement performs better than the statement that uses constants. This form of the INSERT statement allows for the statement to be prepared only once and then reused on subsequent executions of the statement. It also allows the table on the server to remain open, thus removing the overhead of opening and closing the file for each insert.

Example:

```
INSERT INTO TEST.TABLE1 VALUES (?, ?, ?, ?)
```

The blocked INSERT statement is the most efficient way to perform inserts into a table at times when multiple records can be cached on the client and sent at once. The advantages with blocked INSERT are:

1. The data for multiple rows is sent in one communication request rather than one request per row.
2. The server has an optimized path built into the database support for blocked INSERT statements.

Example:

```
INSERT INTO TEST.TABLE1 ? ROWS VALUES (?, ?, ?, ?)
```

The INSERT statement has additional syntax that identifies it as a blocked INSERT. The "? ROWS" clause indicates that an additional parameter will be specified for this

INSERT statement. It also indicates that the parameter will contain a row count that determines how many rows will be sent for that execution of the statement. The number of rows must be specified by means of the SQLParamOptions API (see the following examples).

Examples of blocked insert calls from C

```
//     Block insert variables and statements
            // variable declares
    HSTMT s_Ordlin1;                  // Order Line Insert.
    UDWORD rowcnt;                    // for Blocked insert.
    long   lOrderNum[15];
    // array of order numbers for blocked insert
    UINT   uiDistrict[15];
    // District Number (from TxnLeader).
    char   szWid[15][5];       // Warehouse id.
    UINT   uiOl_ctr[15];       // Order Line Counter.
    char   szSplyWid[15][5];   // Supply warehouse id.
    char   szIid[15][7];       // Item id
    char   szOlAmt[15][10];
    // $ amount for ordered item (7,2)
    UINT   uiZero [15];        // delivery date & time
    UINT   uiOlQty[15];        // Quantity Ordered - for non-numeric.
    char   szDistrctInfo[15][25];  //District info.

    UINT    m_Warehouse;         // Member copy of the warehouse #.
    long    m_OrderNum;          // Order Number.
    UINT    m_District;          // District Number (from TxnLeader).
    Order_Line  m_OrdLines[16];
     // Array for interface

   // Prepare the block insert statement
   strcpy(tmpbfr,"Insert into library/ORDLIN ? ");
   strcat(tmpbfr,"ROWS  VALUES (?,?,?,?,?,?,?,?,?,?,?)");

  ret= SQLPrepare(s_Ordlin1,(unsigned char far *)tmpbfr,SQL_NTS);
  // Set the parameters. This table has 11 columns.

  ret=SQLBindParameter(s_Ordlin1, 1,SQL_PARAM_INPUT,SQL_C_LONG,SQL_DECIMAL,
  9,0,lOrderNum,0,NULL);         //order id
```

```
ret=SQLBindParameter(s_Ordlin1, 2,SQL_PARAM_INPUT,SQL_C_SHORT,SQL_DECIMAL,
3,0,uiDistrict,0,NULL);        //district id

ret=SQLBindParameter(s_Ordlin1, 3,SQL_PARAM_INPUT,SQL_C_CHAR,SQL_CHAR,4,0,
szWid[0],5,NULL);   //warehouse id

ret=SQLBindParameter(s_Ordlin1, 4,SQL_PARAM_INPUT,SQL_C_SHORT,SQL_DECIMAL,
3,0,&uiOI_ctr,0,NULL);     //number of order lines

ret=SQLBindParameter(s_Ordlin1, 5,SQL_PARAM_INPUT,SQL_C_CHAR,SQL_CHAR,4,0,
szSplyWid,5,NULL);                //supplying wrhs id

ret=SQLBindParameter(s_Ordlin1, 6,SQL_PARAM_INPUT,SQL_C_CHAR,SQL_CHAR,6,0,
szIid,7,NULL);                    //item id

ret=SQLBindParameter(s_Ordlin1, 7,SQL_PARAM_INPUT,SQL_C_SHORT,SQL_NUMERIC,3,0,
uiOlQty,0,NULL);                  //quantity

ret=SQLBindParameter(s_Ordlin1, 8,SQL_PARAM_INPUT,SQL_C_CHAR,SQL_NUMERIC,7,2,
szOlAmt,10,NULL);                    //A float for $ amount or purchase

ret=SQLBindParameter(s_Ordlin1, 9,SQL_PARAM_INPUT,SQL_C_SHORT,SQL_NUMERIC,8,0,
uiZero,0,NULL);                   //Delivery date  (set to zero)

ret=SQLBindParameter(s_Ordlin1,10,SQL_PARAM_INPUT,SQL_C_SHORT,SQL_NUMERIC,6,0,
uiZero,0,NULL);                   //Delivery time  (set to zero)

ret=SQLBindParameter(s_Ordlin1,11,SQL_PARAM_INPUT,SQL_C_CHAR,SQL_CHAR,24,0,
szDistrctInfo,25,NULL);           //District data waste space

//%%%%%%%%%%%%%%%%%%%%%%%%%%  Insert into ORDLIN  %%%%%%%%%%%%%%%%%%%%%%%%%%
 rowcnt = 0;

 // Use SQLParamOptions to set the row count for the Blocked Insert.
 // The row count field is passed in a separate variable.

 ret=SQLParamOptions(s_Ordlin1, NewOrd_IO->m_OrderCount, &rowcnt);

 // Fill the parameters for the Blocked Insert by moving the data
 // from the input array to the parameters.
```

```
 for(ol_ctr=1;ol_ctr<=NewOrd_IO->m_OrderCount;ol_ctr++)
{
  lOrderNum[ol_ctr-1] =  NewOrd_IO->m_OrderNum;
  uiDistrict[ol_ctr-1] = NewOrd_IO->m_District;
  sprintf(s_parm[3],"%04u",NewOrd_IO->m_Warehouse);
  strcpy(szWid[ol_ctr-1],s_parm[3]);
  uiOl_ctr[ol_ctr-1] = ol_ctr;
  sprintf(s_parm[5],"%04u",*NewOrd_IO->m_OrdLines
  [ol_ctr].Supp_W);
  strcpy(szSplyWid[ol_ctr-1], s_parm[5]);
  sprintf(s_parm[6],"%06ld",*NewOrd_IO->m_OrdLines
  [ol_ctr].Item_ID);
  strcpy(szIid[ol_ctr-1], s_parm[6]);
  uiOlQty[ol_ctr-1] = *NewOrd_IO->m_OrdLines
  [ol_ctr].Qty;
  sprintf(s_parm[8],"%+09.2f",*NewOrd_IO->m_OrdLines
  [ol_ctr].Amount);
  strcpy(szOlAmt[ol_ctr-1], s_parm[8]);
  sprintf(s_parm[13],"%s",txttemp[
  NewOrd_IO->m_District]);
  strcpy(szDistrctInfo[ol_ctr-1], s_parm[13]);
 }   // end of for loop

ret=SQLExecute(s_Ordlin1);         // Execute Order Line insert.
```

Chapter 9. Catalog Functions

Catalog functions return information about a data source's catalog.

To process ODBC SQLTable requests, logical files are built over the system cross reference file QADBXREF in library QSYS. QADBXREF is a database file for database-maintained cross-reference information that is part of the dictionary function for the system.

The following are the actions for SQLTable when Table_Type is set to the following:

NULL Selects all LOGICAL and PHYSICAL files and SQL TABLES and VIEWS.

TABLE Selects all PHYSICAL files, and SQL TABLES that are not system files (cross reference or data dictionary).

VIEW Selects all LOGICAL files and SQL VIEWS that are not system files (cross reference or data dictionary).

SYSTEM TABLE Selects all PHYSICAL and LOGICAL files and SQL views that are either system files or data dictionary files.

TABLE, VIEW Selects all LOGICAL and PHYSICAL files and all SQL TABLES and VIEWS that are not system files or data dictionary files.

Non-relational files (files with more than one format) are not selected. Also not selected are indexes and flat files.

The result sets returned by the catalog functions are ordered by table type. In addition to the TABLE and VIEW types, the AS/400 has the data source-specific type identifiers of PHYSICAL and LOGICAL files. The PHYSICAL type is handled as a TABLE, and the LOGICAL type is handled as a VIEW. Though they appear as type TABLE or VIEW, PHYSICAL and LOGICAL are sorted on their true type. The sort order is Logical files, Physical files, Tables, and Views.

The following system naming conventions are used:

System cross reference file names all start with QADB.....

System data dictionary file names all start with QIDCT.....

System view names all start with SYS.....

To process ODBC SQLColumn requests, a logical file is built over the system cross reference file QADBIFLD in the QSYS library. This logical file selects all relational database files except for indexes. QADBIFLD is a database file for database-maintained cross-reference information that is part of the dictionary function for the system. Specifically, this includes database file column and field information.

Chapter 10. Exit Programs

When you specify an exit program, the servers pass the following two parameters to the exit program before running your request.

- A 1-byte return code value.
- A structure containing information about your request. This structure is different for each of the exit points.

These two parameters allow the exit program to determine whether your request is allowed. If the exit program sets the return code to X'F0', the server rejects the request. If the return code is set to anything else, the server allows the request.

The same program can be used for multiple exit points. The program can determine what function is being called by looking at the data in the second parameter structure.

Use the Work with Registration Information (WRKREGINF) command to add your exit programs to the database exit points.

The database server has four different exit points defined:

QIBM_QZDA_INIT - called at server initiation.

QIBM_QZDA_NDB1 - called for native database requests.

QIBM_QZDA_SQL1 - called for SQL requests.

QIBM_QZDA_ROI1 - called for retrieving object information requests and SQL catalog functions.

Sample user exit programs

These examples do not show all of the programming considerations or techniques. Review the examples before you begin application design and coding.

See the *AS/400 Client Access Host Servers*, SC41-5740 manual for additional exit program examples.

```
/*--------------------------------------------------------------------------
 *                OS/400 Servers - Sample Exit Program
 *
 *    Exit Point Name         : QIBM_QZDA_INIT
 *
 *    Description             : The following ILE C/400 program handles
 *                              ODBC security by rejecting requests from
 *                              certain users.
 *                              It can be used as a shell for developing
 *                              exit programs tailored for your
 *                              operating environment.
 *
 *    Input                   : A 1-byte return code value
 *                                  X'F0' server rejects the request
 *                                  anything else server allows the request
 *                              Structure containing information about the
 *                                request.  The format used by this program
 *                                is ZDAI0100.
 *-------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------
 *     Includes
 *-------------------------------------------------------------------------*/
#include <string.h>                /* string functions                     */
/*--------------------------------------------------------------------------
 *    User Types
 *-------------------------------------------------------------------------*/
typedef struct {             /* Exit Point QIBM_QZDA_INIT format ZDAI0100 */
    char User_profile_name[10];     /* Name of user profile calling server*/
    char Server_identifier[10];     /* database server value (*SQL)       */
    char Exit_format_name[8];       /* User exit format name (ZDAI0100)   */
    long Requested_function;        /* function being preformed (0)       */
} ZDAI0100_fmt_t;
```

Figure 10-1. (Part 1 of 2) Example of an ILE C/400 User Exit Program for Exit Point QIBM_QZDA_INIT

```
/*====================================================================
 *    Start of mainline executable code
 *====================================================================*/
int main (int argc, char *argv[])
{
    ZDAI0100_fmt_t input;           /* input format record               */

    /* copy input parm into structure                                   */
    memcpy(&input, (ZDAI0100_fmt_t *)argv[2], 32);

    if /* if user name is GUEST                                         */
        ( memcmp(input.User_profile_name, "GUEST     ", 10)==0 )
    {
        /* set return code to reject the request.                       */
        memcpy( argv[1], "0", 1);
    }
    else /* else user is someone else                                  */
    {
        /* set return code to allow the request.                        */
        memcpy( argv[1], "1", 1);
    }
}  /* End of mainline executable code                                  */
```

Figure 10-2. (Part 2 of 2) Example of an ILE C/400 User Exit Program for Exit Point QIBM_QZDA_INIT

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*               OS/400 Servers - Sample Exit Program                */
/*                                                                   */
/*    Exit Point Name         : QIBM_QZDA_INIT                       */
/*                                                                   */
/*    Description             : The following Control Language program   */
/*                              handles ODBC security by rejecting       */
/*                              requests from certain users.             */
/*                              It can be used as a shell for developing */
/*                              exit programs tailored for your          */
/*                              operating environment.                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
PGM PARM(&STATUS &REQUEST)

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*  Program call parameter declarations                              */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
  DCL VAR(&STATUS) TYPE(*CHAR) LEN(1)  /* Accept/Reject indicator       */
  DCL VAR(&REQUEST) TYPE(*CHAR) LEN(34) /* Parameter structure          */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*  Parameter declares                                               */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
  DCL VAR(&USER) TYPE(*CHAR) LEN(10)  /* User profile name calling server*/
  DCL VAR(&SRVID) TYPE(*CHAR) LEN(10) /* database server value (*SQL)    */
  DCL VAR(&FORMAT) TYPE(*CHAR) LEN(8) /* Format name (ZDAI0100)          */
  DCL VAR(&FUNC) TYPE(*CHAR) LEN(4)   /* function being preformed (0)    */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*  Extract the various parameters from the structure                */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
  CHGVAR VAR(&USER)   VALUE(%SST(&REQUEST  1 10))
  CHGVAR VAR(&SRVID)  VALUE(%SST(&REQUEST 11 10))
  CHGVAR VAR(&FORMAT) VALUE(%SST(&REQUEST 21  8))
  CHGVAR VAR(&FUNC)   VALUE(%SST(&REQUEST 28  4))
```

Figure 10-3. (Part 1 of 2) Example of a CL User Exit Program for Exit Point QIBM_QZDA_INIT

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*  Begin main program                                               */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  /* set return code to allow the request.                            */
  CHGVAR VAR(&STATUS) VALUE('1')

  /* if user name is GUEST set return code to reject the request.     */
  IF (&USER *EQ 'GUEST') THEN( +
      CHGVAR VAR(&STATUS) VALUE('0') )

EXIT:
ENDPGM
```

Figure 10-4. (Part 2 of 2) Example of a CL User Exit Program for Exit Point QIBM_QZDA_INIT

```
/*--------------------------------------------------------------------------
 *                OS/400 Servers - Sample Exit Program
 *
 *    Exit Point Name         : QIBM_QZDA_SQL1
 *
 *    Description             : The following ILE C/400 program will
 *                              reject any UPDATE request for user GUEST.
 *                              It can be used as a shell for developing
 *                              exit programs tailored for your
 *                              operating environment.
 *
 *    Input                   : A 1-byte return code value
 *                                 X'F0' server rejects the request
 *                                 anything else server allows the request
 *                              Structure containing information about the
 *                                request.  The format used by this program
 *                                is ZDAQ0100.
 *--------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------
 *     Includes
 *--------------------------------------------------------------------------*/
#include <string.h>                /* string functions                    */
#include <stdio.h>                 /* standard IO functions               */
#include <ctype.h>                 /* type conversion functions           */
/*==========================================================================
 *    Start of mainline executable code
 *==========================================================================*/
main(int argc, char *argv[])
{
    long i;
    _Packed struct zdaq0100 {
        char name[10];
        char servid[10];
        char fmtid[8];
        long funcid;
        char stmtname[18];
        char cursname[18];
        char prepopt[2];
        char opnattr[2];
        char pkgname[10];
        char pkglib[10];
        short drdaind;
        char commitf;
        char stmttxt[512];
    } *sptr, stx;
```

Figure 10-5. (Part 1 of 2) Example of an ILE C/400 Program for Exit Point QIBM_QZDA_SQL1

```
    /* initialize return variable to indicate ok status                  */
    strncpy(argv[1],"1",1);

    /*********************************************************************/
    /*  Address parameter structure for SQL exit program and move local  */
    /*  parameters into local variables.                                 */
    /* (note : this isn't necessary to evaluate the arguments passed in). */
    /*********************************************************************/
    sptr = (_Packed struct zdaq0100 *) argv[2];

    strncpy(stx.name, sptr->name, 10);
    strncpy(stx.servid, sptr->servid, 10);
    strncpy(stx.fmtid, sptr->fmtid, 8);
    stx.funcid = sptr->funcid;
    strncpy(stx.stmtname, sptr->stmtname, 18);
    strncpy(stx.cursname, sptr->cursname, 18);
    strncpy(stx.opnattr, sptr->opnattr, 2);
    strncpy(stx.prepopt, sptr->prepopt, 2);
    strncpy(stx.pkglib, sptr->pkglib, 10);
    strncpy(stx.pkgname, sptr->pkgname, 10);
    stx.drdaind = sptr->drdaind;
    stx.commitf = sptr->commitf;
    strncpy(stx.stmttxt, sptr->stmttxt, 512);

    /*********************************************************************/
    /* check for user GUEST and an UPDATE statement                      */
    /* if found return an error                                          */
    /*********************************************************************/
    if (! (strncmp(stx.name, "GUEST     ", 10)) )
    {
        for (i=0; i<6; i++)
            stx.stmttxt[i] = toupper(stx.stmttxt[i]);

        if (! strncmp(stx.stmttxt, "UPDATE", 6) )
            /* Force error out of  SQL user exit pgm                     */
            strncpy(argv[1], "0", 1);
        else;
    }
    return;
}  /* End of mainline executable code                                    */
```

Figure 10-6. (Part 2 of 2) Example of an ILE C/400 Program for Exit Point QIBM_QZDA_SQL1

```
/*--------------------------------------------------------------------------
 *                OS/400 Servers - Sample Exit Program
 *
 *    Exit Point Name         : QIBM_QZDA_ROI1
 *
 *    Description             : The following ILE C/400 program logs all
 *                              requests for catalog functions to the
 *                              ZDALOG file in QGPL.
 *                              It can be used as a shell for developing
 *                              exit programs tailored for your
 *                              operating environment.
 *
 *    Input                   : A 1-byte return code value
 *                                 X'F0' server rejects the request
 *                                 anything else server allows the request
 *                              Structure containing information about the
 *                                request.  The format used by this program
 *                                is ZDAR0100.
 *
 *    Dependencies            : The log file must be created using the
 *                              following command:
 *                              CRTPF FILE(QGPL/ZDALOG) RCDLEN(132)
 *--------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------
 *     Includes
 *--------------------------------------------------------------------------*/
#include <recio.h>                  /* record IO functions               */
#include <string.h>                 /* string functions                  */
/*--------------------------------------------------------------------------
 *    User Types
 *--------------------------------------------------------------------------*/
typedef struct {             /* Exit Point QIBM_QZDA_ROI1 format ZDAR0100 */
    char User_profile_name[10];     /* Name of user profile calling server*/
    char Server_identifier[10];     /* database server value (*RTVOBJINF) */
    char Exit_format_name[8];       /* User exit format name (ZDAR0100)   */
    long Requested_function;        /* function being preformed           */
    char Library_name[20];          /* Name of library                    */
    char Database_name[36];         /* Name of relational database        */
    char Package_name[20];          /* Name of package                    */
    char File_name[256];            /* Name of file                       */
    char Member_name[20];           /* Name of member                     */
    char Format_name[20];           /* Name of format                     */
} ZDAR0100_fmt_t;
```

Figure 10-7. (Part 1 of 3) Example of an ILE C/400 Program for Exit Point QIBM_QZDA_ROI1

```
/*====================================================================
 *    Start of mainline executable code
 *====================================================================*/
int main (int argc, char *argv[])
{
    _RFILE *file_ptr;               /* pointer to log file               */
    char output_record[132];        /* output log file record            */
    ZDAR0100_fmt_t input;           /* input format record               */
    /* set return code to allow the request.                             */
    memcpy( argv[1], "1", 1);

    /* open the log file for writing to the end of the file              */
    if (( file_ptr = _Ropen("QGPL/ZDALOG", "ar") ) == NULL)
    {
        /* open failed                                                   */
        return;
    }

    /* copy input parm into structure                                    */
    memcpy(&input, (ZDAR0100_fmt_t *)argv[2], 404);

    switch /* Create the output record based on requested function       */
        (input.Requested_function)
    {
        case 0X1800:  /* Retrieve library information                    */
            sprintf(output_record,
              "%10.10s retrieved library %20.20s",
              input.User_profile_name, input.Library_name);
            break;
        case 0X1801:  /* Retrieve relational database information        */
            sprintf(output_record,
              "%10.10s retrieved database %36.36s",
              input.User_profile_name, input.Database_name);
            break;
        case 0X1802:  /* Retrieve SQL package information                */
            sprintf(output_record,
              "%10.10s retrieved library %20.20s package %20.20s",
              input.User_profile_name, input.Library_name,
              input.Package_name);
            break;
        case 0X1803:  /* Retrieve SQL package statement information      */
            sprintf(output_record,
         "%10.10s retrieved library %20.20s package %20.20s statement info",
              input.User_profile_name, input.Library_name,
              input.Package_name);
            break;
```

Figure 10-8. (Part 2 of 3) Example of an ILE C/400 Program for Exit Point QIBM_QZDA_ROI1

```
        case 0X1804:  /* Retrieve file information                        */
            sprintf(output_record,
              "%10.10s retrieved library %20.20s file %40.40s",
              input.User_profile_name, input.Library_name, input.File_name);
            break;
        case 0X1805:  /* Retrieve file member information                 */
            sprintf(output_record,
            "%10.10s retrieved library %20.20s member %20.20s file %40.40s",
              input.User_profile_name, input.Library_name,
              input.Member_name, input.File_name);
            break;
        case 0X1806:  /* Retrieve record format information               */
            sprintf(output_record,
            "%10.10s retrieved library %20.20s format %20.20s file %40.40s",
              input.User_profile_name, input.Library_name,
              input.Format_name, input.File_name);
            break;
        case 0X1807:  /* Retrieve field information                       */
            sprintf(output_record,
              "%10.10s retrieved field info library %20.20s file %40.40s",
              input.User_profile_name, input.Library_name, input.File_name);
            break;
        case 0X1808:  /* Retrieve index information                       */
            sprintf(output_record,
              "%10.10s retrieved index info library %20.20s file %40.40s",
              input.User_profile_name, input.Library_name, input.File_name);
            break;
        case 0X180B:  /* Retrieve special column information              */
            sprintf(output_record,
              "%10.10s retrieved column info library %20.20s file %40.40s",
              input.User_profile_name, input.Library_name, input.File_name);
            break;
        default : /* Unknown requested function                           */
            sprintf(output_record, "Unknown requested function");
            break;
    }  /* end switch statement                                           */

    /* write the output record to the file                              */
    _Rwrite(file_ptr, &output_record, 132);

    /* close the log file                                               */
    _Rclose ( file_ptr );

}  /* End of mainline executable code                                  */
```

Figure 10-9. (Part 3 of 3) Example of an ILE C/400 Program for Exit Point QIBM_QZDA_ROI1

Exit program parameter formats

The exit points for native database and retrieving object information have two formats defined. Depending on the type of function requested, a different format will be used.

The QIBM_QZDA_INIT exit point is defined to run an exit program at server initiation. If a program is defined for this exit point, it is called each time the database server is initiated.

Figure 10-10 on page 10-10 shows parameter fields and their descriptions for the exit program called at exit point QIBM_QZDA_INIT using the ZDAI0100 format.

Figure 10-10. Exit Point QIBM_QZDA_INIT format ZDAI0100

Offset				
Dec	**Hex**	**Type**	**Field**	**Description**
0	0	CHAR(10)	User profile name	The name of the user profile that is calling the server.
10	A	CHAR(10)	Server identifier	The value is *SQL for this exit point.
20	14	CHAR(8)	Format name	The user exit format name being used. For QIBM_QZDA_INIT the format name is ZDAI0100.
28	1C	BINARY(4)	Requested function	The function being performed. The only valid value for this exit point is 0.

Note: This format is defined by member EZDAEP in files H, QRPGSRC, QRPGLESRC, QCBLSRC and QCBLLESRC in library QSYSINC.

The QIBM_QZDA_NDB1 exit point is defined to run an exit program for native database requests for the database server. Two formats are defined for this exit point. Format ZDAD0100 is used for the following functions:

- Create source physical file
- Create database file, based on existing file
- Add, clear, delete database file member
- Override database file
- Delete database file override
- Delete file

Format ZDAD0200 is used when a request is received to add libraries to the library list.

Figure 10-11 shows parameter fields and their descriptions for the exit program called at exit point QIBM_QZDA_NDB1 using the ZDAD0100 format.

Figure 10-11. Exit Point QIBM_QZDA_NDB1 format ZDAD0100

Offset				
Dec	**Hex**	**Type**	**Field**	**Description**
0	0	CHAR(10)	User profile name	The name of the user profile that is calling the server.
10	A	CHAR(10)	Server identifier	For this exit point the value is *NDB.
20	14	CHAR(8)	Format name	The user exit format name being used. For the following functions, the format name is ZDAD0100.
28	1C	BINARY(4)	Requested function	The function being performed. This field contains one of the following: **X'1800'** - Create source physical file **X'1801'** - Create database file, based on existing file **X'1802'** - Add database file member **X'1803'** - Clear database file member **X'1804'** - Delete database file member **X'1805'** - Override database file **X'1806'** - Delete database file override **X'1807'** - Create save file **X'1808'** - Clear save file **X'1809'** - Delete file
32	20	CHAR(128)	File name	Name of the file used for the requested function.
160	A0	CHAR(10)	Library name	Name of the library that contains the file.
170	AA	CHAR(10)	Member name	Name of the member to be added, cleared, or deleted.
180	B4	CHAR(10)	Authority	Authority to the created file
190	BE	CHAR(128)	Based on file name	Name of the file to use when creating a file based on an existing file.
318	13E	CHAR(10)	Based on library name	Name of the library containing the based on file
328	148	CHAR(10)	Override file name	Name of the file to be overridden
338	152	CHAR(10)	Override library name	Name of the library that contains the file to be overridden
348	15C	CHAR(10)	Override member name	Name of the member to be overridden

Note: This format is defined by member EZDAEP in files H, QRPGSRC, QRPGLESRC, QCBLSRC and QCBLLESRC in library QSYSINC.

Figure 10-12 on page 10-12 shows parameter fields and their descriptions for the exit program called at exit point QIBM_QZDA_NDB1 using the ZDAD0200 format.

Figure 10-12. Exit Point QIBM_QZDA_NDB1 format ZDAD0200

Offset				
Dec	**Hex**	**Type**	**Field**	**Description**
0	0	CHAR(10)	User profile name	The name of the user profile that is calling the server.
10	A	CHAR(10)	Server identifier	For this exit point the value is *NDB.
20	14	CHAR(8)	Format name	The user exit format name being used. For the add to library list function the format name is ZDAD0200.
28	1C	BINARY(4)	Requested function	The function being performed. **X'180C'** - Add library list
32	20	BINARY(4)	Number of libraries	The number of libraries (the next field)
36	24	CHAR(10)	Library name	The library names for each library

Note: This format is defined by member EZDAEP in files H, QRPGSRC, QRPGLESRC, QCBLSRC and QCBLLESRC in library QSYSINC.

The QIBM_QZDA_SQL1 exit point is defined to run an exit point for certain SQL requests that are received for the database server. Only one format is defined for this exit point. The following are the functions that cause the exit program to be called:

- Prepare
- Open
- Execute
- Connect
- Create package
- Clear package
- Delete package
- Execute immediate
- Prepare and describe
- Prepare and execute or prepare and open
- Open and fetch
- Execute or open

Figure 10-13 shows parameter fields and their descriptions for the exit program called at exit point QIBM_QZDA_SQL1 using the ZDAQ0100 format.

Figure 10-13 (Page 1 of 2). Exit Point QIBM_QZDA_SQL1 format ZDAQ0100

Offset				
Dec	**Hex**	**Type**	**Field**	**Description**
0	0	CHAR(10)	User profile name	The name of the user profile that is calling the server.
10	A	CHAR(10)	Server identifier	For this exit point the value is *SQLSRV.
20	14	CHAR(8)	Format name	The user exit format name being used. For QIBM_QZDA_SQL1 the format name is ZDAQ0100.
28	1C	BINARY(4)	Requested function	The function being performed. This field contains one of the following: **X'1800'** - Prepare **X'1803'** - Prepare and describe **X'1804'** - Open/Describe **X'1805'** - Execute **X'1806'** - Execute immediate **X'1809'** - Connect **X'180D'** - Prepare and execute or prepare and open **X'180E'** - Open and fetch **X'180F'** - Create package **X'1810'** - Clear package **X'1811'** - Delete package **X'1812'** - Execute or open
32	20	CHAR(18)	Statement name	Name of the statement used for the prepare or execute functions.
50	32	CHAR(18)	Cursor name	Name of the cursor used for the open function.
68	44	CHAR(2)	Prepare option	Option used for the prepare function.
70	46	CHAR(2)	Open attributes	Option used for the open function.
72	48	CHAR(10)	Extended dynamic package name	Name of the extended dynamic SQL package.
82	52	CHAR(10)	Package library name	Name of the library for extended dynamic SQL package.
92	5C	BINARY(2)	DRDA indicator	**0** - Connected to local RDB **1** - Connected to remote RDB

Figure 10-13 (Page 2 of 2). Exit Point QIBM_QZDA_SQL1 format ZDAQ0100

Offset				
Dec	**Hex**	**Type**	**Field**	**Description**
94	5E	CHAR(1)	Commitment control level	**'A'** - Commit *ALL **'C'** - Commit *CHANGE **'N'** - Commit *NONE **'S'** - Commit *CS (cursor stability)
95	5F	CHAR(512)	First 512 bytes of the SQL statement text	First 512 bytes of the SQL statement

Note: This format is defined by member EZDAEP in files H, QRPGSRC, QRPGLESRC, QCBLSRC and QCBLLESRC in library QSYSINC.

The QIBM_QZDA_ROI1 exit point is defined to run an exit program for the requests that retrieve information about certain objects for the database server. It is also used for SQL catalog functions.

This exit point has two formats defined.

Format ZDAR0100 is used for requests to retrieve information for the following objects:

- Field (or column)
- File (or table)
- File member
- Index
- Library (or collection)
- Record format
- Relational database (or RDB)
- Special columns
- SQL package
- SQL package statement

Format ZDAR0200 is used for requests to retrieve information for the following objects:

- Foreign keys
- Primary keys

Figure 10-14 shows parameter fields and their descriptions for the exit program called at exit point QIBM_QZDA_ROI1 using the ZDAR0100 format.

Figure 10-14 (Page 1 of 2). Exit Point QIBM_QZDA_ROI1 format ZDAR0100

Offset				
Dec	**Hex**	**Type**	**Field**	**Description**
0	0	CHAR(10)	User profile name	The name of the user profile that is calling the server.
10	A	CHAR(10)	Server identifier	For the database server the value is *RTVOBJINF.
20	14	CHAR(8)	Format name	The user exit format name being used. For the following functions, the format name is ZDAR0100.
28	1C	BINARY(4)	Requested function	The function being performed. This field contains one of the following: **X'1800'** - Retrieve library information **X'1801'** - Retrieve relational database information **X'1802'** - Retrieve SQL package information **X'1803'** - Retrieve SQL package statement information **X'1804'** - Retrieve file information **X'1805'** - Retrieve file member information **X'1806'** - Retrieve record format information **X'1807'** - Retrieve field information **X'1808'** - Retrieve index information **X'180B'** - Retrieve special column information
32	20	CHAR(20)	Library name	The library or search pattern used when retrieving information about libraries, packages, package statements, files, members, record formats, fields, indexes, and special columns.
52	34	CHAR(36)	Relational database name	The relational database name or search pattern used to retrieve RDB information.
88	58	CHAR(20)	Package name	The package name or search pattern used to retrieve package or package statement information.
108	6C	CHAR(256)	File name (SQL alias name)	The file name or search pattern used to retrieve file, member, record format, field, index, or special column information.
364	16C	CHAR(20)	Member name	The member name or search pattern used to retrieve file member information.

Figure 10-14 (Page 2 of 2). Exit Point QIBM_QZDA_ROI1 format ZDAR0100

Offset				
Dec	**Hex**	**Type**	**Field**	**Description**
384	180	CHAR(20)	Format name	The format name or search pattern used to retrieve record format information.

Note: This format is defined by member EZDAEP in files H, QRPGSRC, QRPGLESRC, QCBLSRC and QCBLLESRC in library QSYSINC.

Figure 10-15 shows parameter fields and their descriptions for the exit program called at exit point QIBM_QZDA_ROI1 using the ZDAR0200 format.

Figure 10-15. Exit Point QIBM_QZDA_ROI1 format ZDAR0200

Offset				
Dec	**Hex**	**Type**	**Field**	**Description**
0	0	CHAR(10)	User profile name	The name of the user profile that is calling the server.
10	A	CHAR(10)	Server identifier	For the database server the value is *RTVOBJINF.
20	14	CHAR(8)	Format name	The user exit format name being used. For the following functions, the format name is ZDAR0200.
28	1C	BINARY(4)	Requested function	The function being performed. This field contains one of the following: **X'1809'** - Retrieve foreign key information **X'180A'** - Retrieve primary key information
32	20	CHAR(10)	Primary key table library name	The name of the library that contains the primary key table used when retrieving primary and foreign key information.
42	2A	CHAR(128)	Primary key table name (alias name)	The name of the table that contains the primary key used when retrieving primary or foreign key information.
170	AA	CHAR(10)	Foreign key table library name	The name of the library that contains the foreign key table used when retrieving foreign key information.
180	64	CHAR(128)	Foreign key table name (alias name)	The name of the table that contains the foreign key used when retrieving foreign key information.

Note: This format is defined by member EZDAEP in files H, QRPGSRC, QRPGLESRC, QCBLSRC and QCBLLESRC in library QSYSINC.

Part 3. Appendixes

Appendix A. Open Database Connectivity Additional Information

This appendix contains additional advanced information about using the ODBC driver provided with Client Access for Windows 95/NT.

Advanced use

The following section describes some useful considerations for advanced users or programmers.

Connection Strings

Connection strings can be used with SQLDriverConnect to provide additional information. The following keywords can be used with the Client Access ODBC driver:

DSN Data source name

DRIVER Name of the driver

SYSTEM System name as configured for Client Access for Windows 95/NT

UID User ID for AS/400

PWD User-specified password

DBQ User-specified default library

CMT Commit mode

```
0 - Commit Immediate
1 - Read Committed
2 - Read Uncommitted
3 - Repeatable Read
```

XDYNAMIC Enable extended dynamic support

```
0 - No
1 - Yes
```

DFTPKGLIB Default package library

LAZYCLOSE Enable lazy close support

```
0 - Disable lazy close
1 - Enable lazy close
```

PREFETCH Enable pre-fetch during EXECUTE

```
0 - No
1 - Yes
```

RECBLOCK Record blocking

```
0 - Disable record blocking
1 - Block if FOR FETCH ONLY specified
2 - Block except FOR UPDATE OF specified
```

BLOCKSIZE Block size

```
Any number between 0 - 512, inclusive
```

LIBVIEW OS/400 library view

```
0 - Use default library list
1 - All libraries on the system
```

SORTTYPE Sort type

```
0 - Sort based on hex values
1 - Sort based on job profile
2 - Sort based on language ID
3 - Sort based on specified table
```

SORTTABLE Library/table name of sort table

SORTWEIGHT Sort weight

```
0 - Shared-weight
1 - Unique-weight
```

LANGUAGEID Language ID for sort

```
ENU - English US
JPN - Japanese Katakana
(etc.)
```

CONNTYPE Connection type

```
0 - Read/write (all SQL statements allowed)
1 - Read/Call (Select and Call statements allowed)
2 - Read-Only (Select statements only)
```

REMARKS Object description type

```
0 - OS/400 object description
1 - SQL object comment
```

SCROLLABLE Scrollable cursor

```
0 - Scrollable unless row size is 1
1 - Always scrollable
```

TRANSLATE Translate CCSID 65535

```
0 - Do not translate CCSID 65535
1 - Translate CCSID 65535
```

XLATEDLL Translation DLL

XLATEOPT Translation option

NAM Naming convention

```
0 - SQL naming convention
1 - System naming convention
```

DEC Decimal separator

```
0 - . (period)
1 - , (comma)
```

TFT Time format

```
0 - hh:mm:ss (*HMS)
1 - hh.mm AM/PM (*USA)
2 - hh.mm.ss (*ISO)
3 - hh.mm.ss (*EUR)
4 - hh:mm:ss (*JIS)
```

TSP Time separator

```
0 - : (colon)
1 - . (period)
2 - , (comma)
3 -   (blank)
```

DFT Date format

```
0 - mm/dd/yy
1 - dd/mm/yy
2 - yy/mm/dd
3 - mm/dd/yyyy
4 - yyyy-mm-dd
5 - dd.mm.yyy
6 - yyyy-mm-dd
```

DSP Date separator

```
0 - / (forward slash)
1 - - (dash)
2 - . (period)
3 - , (comma)
4 -   (blank)
```

Below is an example of a connection string:

```
DSN=Travel;UID=Bruce;PWD=Parrot;DBQ=Testlib;RECBLOCK=1;BLOCKSIZE=256;
NAM=1;DFT=4;DSP=0;TFT=1;TSP=0;DEC=0
```

Connection strings can be used to set defaults for the connection to a data source through SQLDriverConnect. As an example, the user id and password for connection to the data source supplied here will override those in the Client Access logon procedure, providing greater or less authority as required.

SQL statements, grammar and limitations

In accordance with the design of ODBC, the Client Access ODBC Driver passes native SQL grammar to the AS/400 DBMS. The Client Access ODBC Driver supports core SQL grammar, as well as SQL statements that are supported by AS/400 SQL and most extended grammar SQL statements.

The same limitations that exist with AS/400 DB2/400 are imposed on the ODBC SQL grammar. Refer to Appendix A of the publication *DB2 for AS/400 SQL Reference*, SC41-5612.

Unsupported ODBC SQL grammar

The Client Access ODBC Driver supports SQL statements and clauses in both the core and extended ODBC grammars, except for the following statements.

Integrity Enhancement Facility (IEF) The CASCADE and RESTRICT clauses in the DROP TABLE, DROP VIEW, and REVOKE statements are not supported.

The DEFAULT, REFERENCES CHECK, UNIQUE, PRIMARY KEY and FOREIGN KEY clauses in the CREATE TABLE statement are not supported. The REFERENCES clause in the GRANT statement is not supported.

Implementation of the ODBC SQL grammar

Whenever the table name is not qualified, the Client Access ODBC Driver uses the library list. The default library can be named either by specifying the “DBQ” parameter if SQLDriverConnect API or SQLBrowseConnect APIs are called, or by setting it in the Client Access ODBC Driver Setup dialog box.

Libraries can either be added to the current server job's library list or replace the list entirely. To replace the list specify a list of library names. To add to the existing list you will need to add a special entry, *USRLIBL, to the list of libraries. All libraries listed before *USRLIBL will be added to the front of the library list and all libraries listed after *USRLIBL will be added to the end of the list. The first library in the list will be used as the default collection. If you do not want a default collection, start the list with a comma.

For example,

```
DefaultLibraries=LIB1 LIB2 LIB3 ;
```

will replace the server job's library list with the three libraries specified. The default collection will be LIB1. However,

```
DefaultLibraries=LIB1, *USRLIBL, LIB2 ;
```

will add LIB1 to the front of the server job's library list and add LIB2 to the end of the list. The default collection will be LIB1.

ODBC API functions

Figure A-1 documents how the Client Access ODBC Driver implements specific API functions.

Figure A-1. Implementation of ODBC API Functions

Function	Description
SQLConnect	The data source name is required. Other values are optional.
SQLBrowseConnect	Uses all keywords. See “Connection Strings” on page A-1. Only DSN is required. Other values are optional.
SQLDriverConnect	Uses all keywords. See “Connection Strings” on page A-1. Only DSN is required. Other values are optional.
SQLSpecialColumns	If called with the SQL_BEST_ROWID option, returns all indexed columns of that table.

Limitations to ODBC API functions

The way in which some functions are implemented in the Client Access ODBC Driver does not meet the specifications in the *Microsoft ODBC Software Development Kit Programmer's Reference*. See Figure A-2 for a complete list.

Figure A-2. Limitations of ODBC API Functions

Function	Description
SQLCancel	The Client Access ODBC driver does not support asynchronous processing. SQLCancel is equivalent to SQLFreeStmt (*htstmt*,SQL_CLOSE).
SQLGetStmtOption and SQLSetStmtOption	SQLSetStmtOption with SQL_USE_BOOKMARKS is not supported. SQL_RETRIEVE_DATA is always set to SQL_RD_ON (default) SQL_SIMULATE_CURSOR is not supported.
SQLExtendedFetch	Open cursor for update then call SQLExtendedFetch is not supported. You cannot use SQLExtendedFetch in combination with SQLSetPos and SQLGetdata. SQL_FETCH_BOOKMARKS is not supported. You cannot use SQLExtendedFetch to retrieve result set for cataloging functions (SQLTables, SQLSpecialColumns, SQLStatistics,SQLColumns, SQLForeignKeys, SQLPrimaryKeys) SQL_NOSCAN options.
SQLSetPos	SQL_UPDATE, SQL_DELETE, SQL_ADD are not supported. SQL_LOCK_EXCLUSIVE, SQL_LOCK_UNLOCK are not supported.
SQLSetScrollOption	SQL_CONCUR_ROWVER, SQL_CONCUR_VALUES are not supported. SQL_SCROLL_KEYSET_DRIVEN will be changed to SQL_SCROLL_DYNAMIC
SQLColumnPrivileges, and SQLTablePrivileges	Returns SQL_SUCCESS, then fetch returns SQL_NO_DATA_FOUND

Implementation issues

Usually, the ODBC Administrator calls the function ConfigDSN when users configure data sources. In the Client Access ODBC Driver this function is in a setup DLL: CWBSTP.DLL.

Data types

The Client Access ODBC Driver maps AS/400 DBMS data types to ODBC SQL data types. Figure A-3 and Figure A-4 on page A-6 show how data types are mapped between the AS/400 DBMS and ODBC SQL and vice versa.

Figure A-3. ODBC to AS/400 Data Mapping

SQL Data Type	AS/400 DBMS SQL Data Type
SQL_BINARY	CHAR FOR BIT DATA
SQL_CHAR	CHAR
SQL_DATE	DATE
SQL_DECIMAL	PACKED DECIMAL
SQL_DOUBLE	FLOAT (8 byte)
SQL_FLOAT	FLOAT (8 byte)
SQL_INTEGER	LARGE INTEGER
SQL_LONGVARBINARY	VARCHAR FOR BIT DATA
SQL_LONGVARCHAR	VARCHAR
SQL_NUMERIC	ZONED DECIMAL
SQL_REAL	FLOAT (4 byte)
SQL_SMALLINT	SMALL INTEGER
SQL_TIME	TIME
SQL_TIMESTAMP	TIMESTAMP
SQL_VARBINARY	VARCHAR FOR BIT DATA
SQL_VARCHAR	VARCHAR

Figure A-4. AS/400 to ODBC Data Mapping

AS/400 DBMS SQL Data Type	SQL Data Type
CHAR	SQL_CHAR
CHAR FOR BIT DATA	SQL_BINARY
DATE	SQL_DATE
FLOAT	SQL_DOUBLE (8 byte) or SQL_REAL (4 byte)
LARGE INTEGER	SQL_INTEGER
PACKED DECIMAL	SQL_DECIMAL
SMALL INTEGER	SQL_SMALLINT
TIME	SQL_TIME
TIMESTAMP	SQL_TIMESTAMP
VARCHAR	SQL_VARCHAR
VARCHAR FOR BIT DATA	SQL_LONGVARBINARY SQL_VARBINARY
ZONED DECIMAL	SQL_NUMERIC

Notice:

All conversions in the *Microsoft ODBC Software Development Kit Programmer's Reference* are supported for these ODBC SQL data types.

For more information on AS/400 data types, please refer to *DB2 for AS/400 SQL Reference*, SC41-5612.

Some limitations are imposed by the Client Access ODBC Driver on data types. See Figure A-5 for a detailed list.

Figure A-5. ODBC Data Types Limitations

Data Type	Limitation
SQL_LONGVARCHAR	32,740 bytes.

Error messages

When an error occurs, the Client Access ODBC Driver returns the SQLSTATE (an ODBC error code) and an error message. The driver obtains this information both from errors detected by the driver and errors returned by the AS/400 DBMS.

For errors that occur in the data source, the Client Access ODBC Driver maps the returned native error to the appropriate SQLSTATE. When both the Client Access ODBC Driver and the Microsoft Driver Manager detect an error, they generate the appropriate SQLSTATE, and the Client Access ODBC Driver returns an error message based on the message returned by the AS/400 DBMS.

For errors that occur in the Client Access 400 ODBC Driver or the Microsoft Driver Manager, the Client Access ODBC Driver returns an error message based on the text associated with the SQLSTATE.

Error messages have the following format:

```
[vendor][ODBC-component][data-source]
error-message
```

The prefixes in brackets ([]) identify the source of the error. Figure A-6 shows the values of these prefixes returned by the Client Access ODBC Driver.

When the error occurs in the data source, the [vendor] and [ODBC-component] prefixes identify the vendor and name of the ODBC component that received the error from the data source.

Figure A-6. ODBC Error Messages

Error Source	Prefix	Value
Driver Manager	[vendor] [ODBC-component] [data-source]	[Microsoft] [ODBC DLL] [N/A]
Client Access ODBC Driver (32bit)	[vendor] [ODBC-component] [data-source]	[IBM] [Client Access ODBC Driver (32-bit)] N/A
AS/400 DBMS	[vendor] [ODBC-component] [data-source]	[IBM] [Client Access ODBC Driver (32-bit)] [DB2/400][Host message:]

Sample ODBC conversation

The following is the record of an ODBC connection to an AS/400 data source. This record was taken using the ODBCSPY tool provided Microsoft Developer Network - Development Platform. The ODBC compliant product used was Microsoft Query.

You can see from the log that there is a large amount of handshaking activity needed to initiate a conversation between the ODBC enabled application and the ODBC driver before any data is retrieved from the AS/400. The application's ODBC engine needs to determine the level of service the driver provides before submitting requests for data.

The following SQL conversation was conducted between Microsoft Query and the Client Access/400 for Windows 3.1 ODBC driver.

The comments to the right of the log details have been added to clarify some of the details to the left.

```
SQLAllocEnv                    Request for an environment handle
        0x00010000
        SQL_SUCCESS
SQLAllocConnect                Request for a connection handle
        0x00010000
        0x00010000
        SQL_SUCCESS
SQLGetInfo                     Request specific driver information
        0x00010000
        SQL_DRIVER_ODBC_VER
        [1]
        6
        14
        SQL_ERROR
SQLError                       Status information
        NULL
        0x00010000
        NULL
        NULL
        NULL
        NULL
        0
        NULL
        SQL_SUCCESS
SQLError                       Status information
        NULL
        0x00010000
        NULL
        NULL
        NULL
        NULL
        0
        NULL
        SQL_NO_DATA_FOUND
```

```
SQLDriverConnect                Connects to the driver by its dialog box
        0x00010000
        0xF053
        [12]DSN=RCHASD18
        SQL_NTS
        [79]DSN=RCHASD18;DBQ=HOACOL, TPCD, DBITRK;CMT=3;NAM=1;
              DFT=4;DSP=0;TFT=1;TSP=0;DEC=0
        256
        79
        SQL_DRIVER_COMPLETE
        SQL_SUCCESS
SQLGetFunctions                 Request info on supported functions
        0x00010000
        0
        20335
        SQL_ERROR
SQLGetInfo                      Request specific driver information
        0x00010000
        SQL_CURSOR_COMMIT_BEHAVIOR
        SQL_CC_PRESERVE
        2
        NULL
        SQL_SUCCESS
SQLGetInfo
        0x00010000
        SQL_CURSOR_ROLLBACK_BEHAVIOR
        SQL_CR_PRESERVE
        2
        NULL
        SQL_SUCCESS
SQLGetInfo
        0x00010000
        SQL_DATA_SOURCE_NAME
        [8]RCHASD18
        256
        8
        SQL_SUCCESS
SQLAllocStmt                    Request an instruction handle
        0x00010000
        0x00010001
        SQL_SUCCESS
```

```
SQLGetInfo                      Request specific driver information
        0x00010000
        SQL_ACTIVE_STATEMENTS
        0
        2
        NULL
        SQL_SUCCESS
SQLGetInfo
        0x00010000
        SQL_DATA_SOURCE_READ_ONLY
        [1]N
        256
        1
SQLGetInfo
        SQL_SUCCESS
        0x00010000
        SQL_DRIVER_NAME
        [12]EHNODBC3.DLL
        256
        12
        SQL_SUCCESS
SQLGetInfo
        0x00010000
        SQL_SEARCH_PATTERN_ESCAPE
        [1]\
        256
        1
        SQL_SUCCESS
SQLGetInfo
        0x00010000
        65000
        SQL_ERROR
SQLError                        Status Information
        NULL
        0x00010000
        NULL
        NULL
        NULL
        NULL
        0
        NULL
        SQL_SUCCESS
```

```
SQLError                        Status Information
        NULL
        0x00010000
        NULL
        NULL
        NULL
        NULL
        0
        NULL
        SQL_NO_DATA_FOUND
SQLGetInfo                      Request specific driver information
        0x00010000
        65001
        SQL_ERROR
SQLError                        Status Information
        NULL
        0x00010000
        NULL
        NULL
        NULL
        NULL
        0
        NULL
        SQL_SUCCESS
SQLError
        NULL
        0x00010000
        NULL
        NULL
        NULL
        NULL
        0
        NULL
        SQL_NO_DATA_FOUND
SQLGetInfo                      Request specific driver information
        0x00010000
        SQL_QUALIFIER_NAME_SEPARATOR
        [1]/
        256
        1
        SQL_SUCCESS
SQLGetInfo
        0x00010000
        65002
        SQL_ERROR
```

```
SQLError                        Status Information
        NULL
        0x00010000
        NULL
        NULL
        NULL
        NULL
        0
        NULL
        SQL_SUCCESS
SQLError
        NULL
        0x00010000
        NULL
        NULL
        NULL
        NULL
        0
        NULL
        SQL_NO_DATA_FOUND
SQLGetInfo
        0x00010000
        SQL_SEARCH_PATTERN_ESCAPE
        [1]\
        65
        1
        SQL_SUCCESS
SQLGetInfo                      Request specific driver information
        0x00010000
        SQL_QUALIFIER_TERM
        [11]SYSTEM NAME
        65
        11
        SQL_SUCCESS
SQLGetInfo                      Request specific driver information
        0x00010000
        SQL_DATABASE_NAME
        [11]SYSTEM NAME
        65
        11
        SQL_SUCCESS
```

```
SQLTables
        0x00010001
        [1]%
        SQL_NTS
        [0]
        0
        [0]
        0
        [0]
        0
        SQL_SUCCESS
SQLBindCol                      Allocate memory for a retrieved column
        0x00010001
        1
        SQL_C_CHAR
        0x6E9C673B
        65
        0xDA9E673B
        SQL_SUCCESS
SQLFetch                        Gives you back a row, result from a query
        0x00010001
        SQL_NO_DATA_FOUND
SQLError
        NULL
        NULL
        0x00010001
        NULL
        NULL
        NULL
        0
        NULL
        SQL_NO_DATA_FOUND
SQLFreeStmt                     End current instr. clearing environment
        0x00010001
        SQL_CLOSE
        SQL_SUCCESS
SQLFreeStmt
        0x00010001
        SQL_UNBIND
        SQL_SUCCESS
```

```
SQLGetInfo
        0x00010000
        SQL_MAX_OWNER_NAME_LEN
        10
        2
        2
        SQL_SUCCESS
SQLTables
        0x00010001
        [0[
        0
        [1]%
        SQL_NTS
        [0[
        0
        [0[
        0
        SQL_SUCCESS
SQLBindCol
        0x00010001
        2
        SQL_C_CHAR
        0x6E9C673B
        65
        0xD69E673B
        SQL_SUCCESS
SQLFetch                        Gives you back a row, result from a query
        0x00010001
        SQL_SUCCESS
<BOUND>
        2
        SQL_C_CHAR
        [7]#CGULIB
        65
        7
```

```
SQLFetch
        0x00010001
        SQL_SUCCESS
<BOUND>
        2
        SQL_C_CHAR
        [7]#COBLIB
        65
        7
SQLFetch
        0x00010001
        SQL_SUCCESS
<BOUND>
        2
        SQL_C_CHAR
        [7]#DFULIB
        65
        7
SQLFetch
        0x00010001
        SQL_SUCCESS
<BOUND>
        2
        SQL_C_CHAR
        [7]#DSULIB
        65
        7
        .
        .
        .
        .
        .  (Similar entries for each library on
        .   the AS/400 would be included here)
        .
        .
        .
        .
```

```
SQLFetch
        0x00010001
        SQL_SUCCESS
<BOUND>
        2
        SQL_C_CHAR
        [5]QIWSF
        65
        5
SQLFetch
        0x00010001
        SQL_NO_DATA_FOUND
SQLError                        Status Information
        NULL
        NULL
        0x00010001
        NULL
        NULL
        NULL
        0
        NULL
        SQL_NO_DATA_FOUND
SQLFreeStmt                     Close Conversation
        0x00010001
        SQL_CLOSE
        SQL_SUCCESS
SQLFreeStmt                     Connection released
        0x00010001
        SQL_UNBIND
        SQL_SUCCESS
```

Appendix B. Enabling Tables to Be Updated

ODBC works for queries and reports, but eventually you are going to want to update some of your data. Because the Client Access ODBC driver is a read/write driver, updating tables is not a problem. All you need is a little setup work. Once you understand a few things about how PC applications work, you will see why this setup work is required to enable these tables to be updated.

PC application locking

Most major PC applications read data from the file that the user wants to use as read-only data. The application then queries the database to see if the file has a unique index defined over it. If there is a unique index on the table, the application will allow editing of data because optimistic locking will work. Optimistic locking states that to lock a record so that it can be edited or deleted later, the original record has to be refound in the table and then locked for update. This means that to update a record, a program will issue the following command:

```
UPDATE RASTEST.CUSTOMER SET CITY = 'Cartegena                '
WHERE CUSTNUM = 20 AND CUSTNAME = 'Jaun''s Coffee                        '
AND ADDRESS = '1 Mountainside Rd                  '
AND CITY = 'Bogata                    '
AND STATE = '' AND PHONE = '37-44-COFFEE   '
```

If there was not a unique index on this table, this style of locking would not work. It would not be able to go out and expect to update just one record.

Another byproduct of this style of locking is the problem of data versioning. Data versioning is:

- You read in the data you need to review.
- While you review the data, another user updates a couple records in the table.
- You then realize that you need to make a couple corrections to the data, and it just happens to be the same data just changed by the other user.

The update you just tried to do would fail with a record-not-found message. You would not be informed that another user has updated that record and you would not be asked if you wanted to see the new version of the record as updated by the other user. You would have to reread in the table to see what might have changed to cause the error.

Creating a unique index

To create the unique index on the AS/400, you can either use interactive SQL or the RUNSQLSTM command.

If you are using interactive SQL:

- Type in STRSQL to start interactive SQL.

- Type in the command `CREATE UNIQUE INDEX indexname ON libraryname/tablename (fieldname)`.

If you are using the RUNSQLSTM command

- Create a source file.
- Create a member which contains the command `CREATE UNIQUE INDEX indexname ON libraryname/tablename (fieldname)`
- Run the RUNSQLSTM command against the source file and member that you defined in the previous steps.

Once either one of these methods is used to create a unique index, or if you create a logical file with a unique key, you will be able to see and use the index from any ODBC application. Once this index is visible to ODBC applications, optimistic locking can occur, and your data is able to be edited.

Appendix C. ODBC Troubleshooting

Unable to connect to the AS/400

Understanding the database server program

Each ODBC connection communicates with one Database Server program that runs on the AS/400. This program is referred to as the 'host server' program. The name of the Database Server program differs depending on the type of connectivity. For example:

- SNA (or 802.2) connection: QIWS/QZDAINIT
- TCP/IP and IPX socket connection: QIWS/QZDASOINIT

When troubleshooting the database server program, you must be aware of the type of connectivity.

Under normal conditions these programs are evoked transparently; i.e. the user needs to take no action except to verify the proper subsystems and communication protocols are running. See *AS/400 Client Access Host Servers*, SC41-5740 for details on administration of host server jobs.

Checking server status (TCP/IP and IPX connections only)

The Client Access for Windows 95/NT product has a special command to verify status of host servers:

```
CWBPING systemname protocol
```

Note: where systemname is the name of the AS/400 system and protocol is IP or IPX, the default is IP.

The command should return the following:

```
pinging server Port Mapper successful
pinging server as-central successful
pinging server as-database successful
pinging server as-dtaq successful
pinging server as-file successful
pinging server as-netprt successful
pinging server as-rmtcmd successful
pinging server as-signon successful
```

If the servers are not active or you are using a SNA connection then continue with the next section.

Verify that the proper subsystems are running

TCP/IP and IPX connected ODBC jobs (QZDASOINIT) will run in the QSERVER subsystem. SNA connected ODBC jobs (QZDAINIT) will normally run in QSERVER. Verify that this subsystem is running. The Client Access for Extended DOS router and many

OEM SNA connections (such as NetWare for SAA) can only use the mode QPCSUPP and will therefore run the ODBC job in QCMN.

Because of changes made to subsystem descriptions in V3R1 and later, the QSERVER subsystem may need to be manually started. To do this, simply issue the following command:

```
STRSBS QSERVER
```

To have the subsystem start automatically at IPL, then modify the AS/400 IPL Start up procedure (the default is QSYS/QSTRUP) to include the STRSBS QSERVER command.

In addition to subsystem QSERVER, subsystem QSYSWRK must be running for TCP/IP and IPX connections.

Verify that the proper prestart jobs are running

IBM ships the QSERVER subsystem configured to use prestart jobs to improve performance at job initialization/start up. When prestart jobs are configured in the subsystem, the job MUST be active to connect. The prestart job used depends on the type of connection:

- SNA (or 802.2) connection
  - QZDAINIT - Server program
- TCP/IP and IPX socket connection
  - QZDASOINIT - Server program
  - QZDASRVSD - Server Daemon program

To verify a prestart job is running:

```
WRKSBSJOB QSERVER
```

The appropriate prestart jobs should be active:

```
Job          User      Type     -----Status-----
QZDAINIT     QUSER     PJ       ACTIVE           (SNA connection)
QZDASOINIT   QUSER     PJ       ACTIVE           (socket connection)
QZDASRVSD    QUSER     PJ       ACTIVE           (socket connection)
```

Prestart jobs do not display in WRKACTJOB unless a connection is already active. You must use F14 - Include from the WRKACTJOB panel for prestart jobs to display.

Further TCP/IP and IPX considerations

Verify that TCP/IP or IPX is started with the following command:

```
NETSTAT *CNN
```

Use the commands STRTCP or STRIPX to start the desired protocol if it is not running.

Verify the necessary daemons are running by browsing the information returned from the NETSTAT *CNN command:

```
Remote           Remote     Local
Address          Port       Port       Idle Time  State
*                *          as-cent >  000:09:31  Listen
*                *          as-signon  000:09:41  Listen
*                *          as-svrmap  002:57:45  Listen
*                *          as-data >  002:57:45  Listen
```

Use the command STRHOSTSVR SERVER(*ALL) to start them if necessary.

- Verify QZDASRVSD, the ODBC socket daemon, is running.
  - as-database should be in the Listen State
  - WRKJOB QZDASRVSD should be used to check the joblog of the daemon for any error messages.
- Verify that socket daemon QZSOMAPD is running in QSYSWRK subsystem.
  - as-svrmap should be in the Listen State as shown by NETSTAT *CNN.
  - WRKJOB QZSOMAPD should be used to check the joblog of the daemon for any error messages.

The PC locates the socket used by the database server by connecting to the server mapper socket. It retrieves the socket used by as-database. It then connects to the proper socket which is being monitored by the file server daemon, QZDASRVSD. The server daemon will attach the client's connection to a QZDASOINIT prestart job in QSERVER. After validating the user profile and password and swapping the user profile into the prestart job, the job will run similar to the QZDAINIT jobs of an SNA connection. If this is the first connection made to the AS/400 for this PC, then two other servers are used: Central server for licensing and Signon server for userid/password validation.

Verify that the host server program product is installed

The Database Server is installed as part of the operating system. If all PCs are still unable to connect to the server or the QSERVER subsystem is not on the system, then verify that product xxxxSS1, option Host Servers is installed where xxxx is the OS/400 lpp (license program product) for the version current version of OS/400.

Identifying the ODBC job and joblog

How to find your job

This can sometimes be challenging as WRKACTJOB shows the prestart jobs as all running under QUSER. You must display the joblog to find the user profile the job is actively running under.

For a SNA connection, there is an easier method. Use Work with Configuration status on the PCs APPC device. All jobs allocated for the device will appear there. To do this, type the following command:

```
WRKCFGSTS CFGTYPE(*DEV) CFGD(LOCNAME)
```

Where LOCNAME is the Local Location name of your PC. To find this value, look up the LOCALLUNAME parameter. This parameter is part of the SNA router configuration. For the Client Access for Windows 3.1 router it is contained in the WINDOWS\NSD.INI file. Keep in mind that the AS/400 may add a '0x' to the end of the name if it detects a damaged controller.

The following will be displayed:

```
QSERVER     ACTIVE/TARGET         QZDAINIT    QUSER       123456

(Where 123456 is your Job ID.)
```

or if another mode description is used:

```
QPCSUPP     ACTIVE/TARGET         LOCNAME     USERID      123456

(Where LOCNAME is your Local Location name, USERID, is the
user id that you used to sign-on to the router with, and
123456 is your Job ID.)
```

For a TCP/IP or IPX connection, the following command may help locate the ODBC job:

```
WRKOBJLCK OBJ(userid) OBJTYPE(*USRPRF)

"userid" is your user profile used for the ODBC connection.
```

How to generate a joblog

Some internal errors cause the job to immediately end. To isolate these types of errors, use the CHGJOB command to modify the existing database server job and generate a joblog. Modify the job description to change all new jobs. To modify the default job description, enter the following command prior to recreating the error:

```
CHGJOBD JOBD(QDFTJOBD) LOG(4 00 *SECLVL)
```

When finished, execute this command to change it back:

```
CHGJOBD JOBD(QDFTJOBD) LOG(4 00 *NOLIST)
```

If the job description is not changed back, other jobs on the system will be forcing joblogs to be written and system storage will fill up rapidly.

How to find the joblog

Type the following command and prompt for parameters using F4:

```
CHGPRTF FILE(QPJOBLOG)
```

Press F10 and Page down until to a field called Spooled Output Queue. This is the OUTQ that the joblog will be written to. By default it is QEZJOBLOG in library QUSRSYS. Record the file and library name of the output queue, and then run the following command to work with all of the entries in the output queue:

```
WRKOUTQ QEZJOBLOG
```

Press F16 to move to the bottom of the list.

An alternative way of finding the joblog is to use the WRKSPLF command. If running under QSERVER, then the USERID is always QUSER. If running under another subsystem, then the USERID will be the one used for the router sign on. The following command can be used:

```
WRKSPLF USERID
```

Where USERID is QUSER or your USERID.

Common errors

Communication errors

Communication link failure. COMM RC=0x4

The Client Access configuration uses an SNA connection (Netsoft or WinAPPC compatible router), but the QZDAINIT job can not be started. Not starting the QZDAINIT prestart job or not starting the QSERVER subsystem usually causes this error. Use the checklist that is described in “Unable to connect to the AS/400” on page C-1.

Communication link failure. COMM RC=0xc

Client Access is configured to use a Transmission Control Protocol/Internet Protocol (TCP/IP) or IPX connection, but the as-database daemon was not running. The QZDASOINIT prestart job not running or the QSERVER subsystem not running can also cause this error. Review the checklist that is described in “Unable to connect to the AS/400” on page C-1. This error will also occur if a configuration of 16-bit ODBC application uses the incorrect thunk layer. For more information, see APAR II09863 available at the AS/400 Service web site: http://www.as400service.ibm.com.

Communication link failure. COMM RC=0x3

A connection that uses any of the three supported protocols can generate this return code. The error occurs when Client Access is unable to verify a userid, a user password, or licensing information. The error usually occurs in TCP/IP and IPX connections when the signon or central server daemon is not active. Use the AS/400 CL command STRHOSTSVR *ALL to correct it or review the checklist that is described in “Unable to connect to the AS/400” on page C-1. Other common causes:

- Attempting to run Client Access ODBC from a Windows NT service that is using Client Access for Windows 95/NT R312 or earlier. This is not supported. You must be at R313 or later.

- Attempting to run a 16-bit ODBC application over SNA without configuring the EHNAPPC thunk layer. For more information, see APAR II09863 available at the AS/400 Service web site: http://www.as400service.ibm.com.

SQL errors

MSGSQL0113 - Name &1 not allowed.

See “MSGSQL0114 - Relational database &1 not the same as current &2 server.”

MSGSQL0114 - Relational database &1 not the same as current &2 server

It is likely that the system name is not in the Remote Database Directory. Run the Add Relational Database Directory Entry command:

```
ADDRDBDIRE RDB(SYSNAME) RMTLOCNAME(*LOCAL)
```

Where SYSNAME is the name of your system's Default Local Location name (as specified in the DSPNETA command).

Another common cause for this error is a period (.) in a table or library name. Although valid in AS/400 naming conventions, in order to use it within an SQL statement, enclose the name within double quotes. A short term circumvention may be to build a logical file over the desired physical file, using the SQL naming syntax.

MSGSQL0122 - Column &1 or function specified in SELECT...

When using a GROUP BY clause, all of the columns in the SELECT list must be in the GROUP BY clause or within a column function. This use is different than in a scalar function. *DB2 for AS/400 SQL Reference* contains an explanation:

"If GROUP BY or HAVING is used:

- Each column-name in the select list must either identify a grouping column or be specified within a column function.
- The RRN, PARTITION, NODENAME, and NODENUMBER functions cannot be specified in the select list.
- The select list is applied to each group of R. The result contains as many rows as there are groups in R. When the select list is applied to a group of R, that group is the source of the arguments of the column functions in the select list."

For example:

"SELECT workdept, salary from employee GROUP BY workdept" files because salary is not in the GROUP BY.

"SELECT field1, DATE(field2) from T1 GROUP BY field1" fails because field2 is not in a column function or the GROUP BY.

"SELECT workdept, AVG(salary) from employee GROUP BY workdept order by 2" completes because workdept is a grouping column and salary is specified in a column function (AVG).

MSGSQL0204 - MYSYSCONF not found

Usually only joblogs for jobs using the Microsoft Jet Engine (MS ACCESS or MS Visual Basic applications) contain this message. The MS Jet Engine always checks for an optional table on the server that is called MSYSCONF. The applications ignore this warning. For further information see the MS Jet Database Engine Connectivity white paper or contact Microsoft.

MSGSQL0900 - Application process not in a connected state

See “MSGSQL5016 - Object name &1 not valid for naming convention” and “MSGSQL0114 - Relational database &1 not the same as current &2 server” on page C-6.

MSGSQL5001 - Column qualifier or table &2 undefined.

See “MSGSQL5016 - Object name &1 not valid for naming convention.”

MSGSQL5016 - Object name &1 not valid for naming convention

Your ODBC Data Source Name (DSN) configuration uses the wrong naming convention. Use the ODBC Administrator to change your DSN to use the proper (*SQL or *SYS) naming convention. Always use "*SQL" unless your application design specifically expects *SYS.

MSGSQL0104 - Token &1 was not valid. Valid tokens: &2

- The application generated an SQL statement with incorrect syntax.
- See “MSGSQL0114 - Relational database &1 not the same as current &2 server” on page C-6 if "*" is the token.
- The SQL statement exceeded the 32K size limitation of the AS/400. If this error occurs because of a very large literal in the SQL statement, such as a large CHAR or VARCHAR field, consider using a parameter marker instead. This reduces the size of the SQL statement while allowing you to pass the maximum field size allowed.
- The application is using incorrect syntax for left outer join. Some applications default to a proprietary left outer join syntax of *= in the WHERE clause (Powerbuilder 3.0 & 4.0, Crystal Reports). Check with your application vendor. Most provide an ini setting or a configuration value to use ODBC left outer join syntax.
- Your ODBC Data Source Name (DSN) configuration uses the wrong decimal separator character. Some users have set the decimal separator parameter of the ODBC connection to a comma instead of a period.

MSGSQL7008 &1 in &2 not valid for operation. The reason code is 3

The AS/400 caarries out commitment control by journaling. Any ODBC application that takes advantage of commitment control will require journaling the files that are used.

Delphi and PowerBuilder errors

Incorrect number of rows (or no rows) returned on SELECT

Your ODBC Data Source Name (DSN) may be configured to use prefetch. Prefetch can not be used with applications that use extended fetch and change the rowset size after issuing the EXECUTE.

Stored procedure errors

Internal driver error (DB2/400 SQL)

Preparing or executing a stored procedure when the server is unable to find a valid declaration of the stored procedure generates this error. Stored procedures should be defined to DB2/400 by using either the DECLARE PROCEDURE or CREATE PROCEDURE SQL statement. The internal driver error occurs during the prepare when the SQL statement contains parameter markers and the database server is unable to locate the definition of the parameters.

When using CREATE PROCEDURE, run a query over QSYS2/SYSPROCS to verify that the CREATE PROCEDURE was run correctly. SPECIFIC_SCHEMA and ROUTINE_SCHEMA must match the library that is used on the call in the PC application. EXTERNAL_NAME must resolve to the actual program name. The parameter descriptions must also be correct. You can also use the SQLProcedures and SQLProcedureColumns ODBC APIs to retrieve the catalog information.

When using the older DECLARE PROCEDURE, you must use extended dynamic support to return output or input/output parameters. You must also store the call of the stored procedure in the active package. Use PRTSQLINF on the package to verify the package contents. Delete the SQL package after making any change to DECLARE PROCEDURE. Parameter descriptions are stored in the package as part of the call entry, not the declare entry.

DECLARE PROCEDURE takes precedence over CREATE PROCEDURE. Delete the package if the problem persists.

SQL0444 - External program &A in &B not found (DB2/400 SQL)

The SQL0444 is generated on an execute or execute direct when the database server is unable to locate the program. Remember that SQL does not perform a library list search. The stored procedure name (the procedure name parameter that is used on the CREATE PROCEDURE statement) must be in the default collection.

No data returned on OUTPUT and INPUT_OUTPUT parameters

An ODBC spy utility is useful for trouble-shooting problems that involve parameter markers. Most spys will show the value of the parameter when it was entered and returned.

- The ODBC SQLBindParameter API incorrectly specified fParamType as SQL_PARAM_INPUT.

- DECLARE PROCEDURE was used instead of CREATE PROCEDURE, and extended dynamic support is disabled.
- The programmer incorrectly declared a parameter as IN on the CREATE or DECLARE PROCEDURE.
- The stored procedure program incorrectly returned the parameter.
- An application error damageed the output. This is a common error with Visual Basic (all versions) and Powerbuilder (3.0 and 4.0).

Powerbuilder (3.0 and 4.0) only supports input parameters on "store procedures." You must use the Powerbuilder Remote Program Call (RPC) to work with input/output or output parameters. See the ODBC User's Guide for an example. When using Visual Basic to make direct ODBC API calls, verify that Visual Basic has not changed the address of the parameter that was set on the SQLBindParameter API. This usually occurs when binding a String data type and results in damageed data or an irrecoverable exception.

MSGSQL0501 - Cursor CRSR000x not open

To return data when using embedded SQL in ILE programs, you must specify the compile option ACTGRP(*CALLER) and not the default of *NEW.

Verify that the program executes a "return" instead of an exit.

When the stored procedure program executes an exit instead of a return, you must set the Close SQL Cursor option to *ENDACTGRP. If the Close SQL Cursor option is set to *ENDMOD, the cursor will be closed before data is retrieved.

Also verify that the CREATE PROCEDURE specifies the correct number of result sets. This is especially important when using array result sets.

No data returned on result set

See “MSGSQL0501 - Cursor CRSR000x not open.”

Incorrect output and unpredictable errors

Ensure that the Client Access ODBC driver and the database server program are at matching code levels. Check for PTF corequisite requirements on any PTF that you order or in the readme.txt file of the Client Access Service pack. If problems continue, verify that you have disabled the prefetch option in the ODBC Data Source. The pre-fetch option should be used only with applications that use the Fetch API, not the Extended Fetch API.

Note that Stored procedure result set cursors are forward only, read only.

Binary or hexadecimal data instead of ASCII characters

The default value of the Translation parameter is "Do Not Translate CCSID 65535." The AS/400 attaches a character set identifier (CCSID) to files, tables and even fields (columns). This CCSIC determines which conversion table will be used to convert data, for example from EBCDIC to ASCII. A CCSID of 65535 often identifies raw data (binary

or hexadecimal), such as bitmapped graphics, that is language independent. A Translation setting of "Do Not Translate CCSID 65535" ensures that the raw data is not damageed.

Setting the Translation parameter to "Translate CCSID 65535" updates the CCSID that is attached to the data to the CCSID of the job. This parameter setting can cause damage to the data, if the data is truly binary.

Gathering information for IBM support

So the IBM Support staff can offer you the best service, please have certain information available when you open a problem record to IBM Support. To gather this information, complete the following tasks:

1. Record the OS/400 version and cumulative PTF level.
2. Record the version of the Client Access ODBC driver.
3. Record the version of the ODBC driver manager.
4. Record the type of connection router.
5. Generate and retrive the ODBC joblog.

Please record additional information about other components, such as the communication adaptor, PC application, error description, and ODBC log. See “Recording additional information” on page C-11.

Recording the OS/400 version and cumulative PTF level

To receive optimal support, you will need to record the OS/400 version and cumulative PTF level for your AS/400 system. Record this information by completing the following steps:

1. Issue the display PTF command on an AS/400 terminal emulation command line:

   ```
   DSPPTF
   ```

2. Record the OS/400 release information that has the format VxRxMx.
3. Verify that the IPL source is ##MACH#B.
4. Press F5 to display the PTF details.
5. Record the first PTF ID in the list. It will have the format Tzxxyyy where xx is the year, yyy the Julian date and z is either L or C.

Recording the version of the Client Access for Windows 95/NT ODBC driver

To receive optimal support, you will need to record the version for your Client Access for Windows 95/NT ODBC driver. Record this information by completing the following steps:

1. From the Task bar select Start, Settings, and Control Panel.
2. Double click the Client Access Properties icon.

3. The General tab of the Client Access Properties dialog will display by default. Record the Client Access version and service pack levels.

Recording the version of the ODBC driver manager

Client Access for Windows 95/NT 32-bit applications

If your problem concerns a 32-bit application, you will need to record the version for your 32-bit ODBC driver manager. Record this information by completing the following steps:

1. From the Task bar, select Start, Find, and Files or Folders.
2. Enter ODBC32.DLL as the file name and click Find.
3. Note the size and date of the file.
4. Right click on the file name and select Properties.
5. Select the Version tab and record the file version.

Client Access for Windows 95/NT 16-bit applications

If your problem concerns a 16-bit application, you will need to record the version for your 16-bit ODBC driver manager. Record this information by completing the steps that are described in “Client Access for Windows 95/NT 32-bit applications.”

Recording the type of connection router

To receive optimal support, you will need to record information regarding the Client Access for Windows 95/NT connection router. Record this information by completing the following steps:

1. On the Windows 95 Task bar, click Start and select Programs/IBM AS400 Client Access/AS400 Connections.
2. Note the connection type for the connection being used.

Generating and retrieving the ODBC joblog

To receive optimal support, you will need to generate and retrive QZDAINIT (the ODBC joblog). See “Identifying the ODBC job and joblog” on page C-3 for instructions.

Recording additional information

To receive optimal support, you will need to record additional information that is listed below. Consult your hardware or software documentation for instructions on how to obtain the needed information.

Manufacturer and model of the communication adaptor

The following information is required if you experience connection failures or application hangs, particularly if the problem occurs when transferring large amounts of data. Verify that you have the latest device drivers for your communication adaptor. Record the following information for your particular network interface card or modem:

Token-Ring (Manufacturer and Model)

Ethernet (Manufacturer and Model)

Twinaxial (TDLC) (Manufacturer and Model)

Synchronous Data Link Control (SDLC)

Asynchronous (ASYNC) (Manufacturer and Model of modem)

Name and version of the PC application

Record the name and version of the PC application that was used when the error was generated.

Error messages

To receive optimal support, you will need to record the PC application error message, the Client Access ODBC error message, and the SQLState. It may also be helpful to have a description of the action you were attempting when the failure occurred. This information will better enable the IBM Support staff to accurately re-create the problem.

The PC application will provide an error message. Include the exact text and error numbers in the error message. When a printer is unavailable, the image of the error message can be captured by pressing the print screen key. Print screen copies a bitmap of the current screen to the clipboard. Use your preferred application to access and store the error bitmap in a folder for future reference.

ODBC defines a standard, layered error handling protocol. Include the return code, the identity of the error source, the actual error text, and the ODBC SQLSTATE in the ODBC error message. Client Access ODBC uses the following format to identify the source of the error:

```
[IBM][Client Access ODBC Driver (32-bit)][DB2/400 SQL]
```

Entry	Definition
[IBM]	The vendor identifier. If this value is anything except IBM, the failure did not occur in IBM code. An error starting with `[Microsoft] [ODBCxxx DLL]` implies the failure was at the ODBC driver manager.
[Client Access ODBC Driver 32-bit)]	The component that is reporting the error. When the Client Access ODBC Driver reports the error and the [DB2/400 SQL] field does not follow, then the error occured on the PC. The request was never sent to the AS/400.
[DB2/400 SQL]	The AS/400. This field, when included, means that the failure originated from the AS/400. The AS/400 joblog may contain additional information for these types of errors.

Some programming languages encapsulate the ODBC APIs within their own object structure. Examples include Microsoft DAO, RDO, ADO. These objects usually return their own error or return code. It is up to the programmer to retrieve the actual ODBC error message. The ODBC error message may be held in an "Error object."

Diagnostic and Performance Tools

Error Message Help

All of the Client Access error messages can be found either on the AS/400 or in a Client Access help. For errors that begin with SQL, use the following OS/400 command to view the message text:

```
DSPMSGD RANGE(SQLxxxx) MSGF(QSQLMSG)
```

For error messages that begin with IWS or PWS, use the following OS/400 command:

```
DSPMSGD RANGE(ZZZxxxx) MSGF(QIWS/QIWSMSG)
where ZZZ is IWS or PWS.
```

ODBC trace utilities

| ODBC 3.0 includes its own trace utility. Available retail utilities can be more robust, providing detailed entry and exit point tracing of ODBC API calls. These tracing utilities include Trace Tools (Dr. DeeBee) and SST Trace Plus (Systems Software Technology).

AS/400 communications trace

| The OS/400 communications trace facility will trace and format any communications type that has a line description (Token-Ring, Ethernet, and SDLC).

| This is a great tool for isolating many problems. It is also a useful aid for diagnosing where a performance delay is occurring. Use the timestamp and eye-catcher fields to measure how long the AS/400 takes to process a request.

AS/400 job traces

The OS/400 job trace can help isolate most host problems and many performance issues. A service job must first be started on the job to be traced. Locate the fully qualified job name of the ODBC job. See “How to find your job” on page C-3 for instructions. From any 5250 emulation session, start a service job on this ODBC job by using the STRSRVJOB command. Then choose one of two traces, depending on the information needed:

Trace job Traces the internal calls made by the host server. Run the TRCJOB *ON command.

Debug trace Used to review the performance of your application and to determine the cause of a particular problem.

The STRDBG command can be run against an active service job. This command logs the decisions made by the query optimizer to the joblob of the debug session. It records estimated query times, access paths used, cursor errors, etc. Use the following command:

```
STRDBG UPDPROD(*YES)
```

AS/400 Performance Tools

AS/400 performance toolkit provides reports and utilities that can be used to create an in—depth analysis of your application performance. The toolkit provides information about CPU utilization, disk arm utilization, memory paging and much more. Although the base operating system includes the ability to collect performance data, you will need the separately licensed program Performance Tools/400 to analyze the results.

AS/400 job log

The ODBC joblog can record all errors that occur on the AS/400. When the job is in debug mode (see “AS/400 job traces” on page C-13), the joblog will also contain performance-related information.

Appendix D. Security Issues with Client Access and ODBC

Overview

The following information is not intended to be a comprehensive guide to security issues on the AS/400 or with Client Access. It's intent is to give an overview of security issues that impact Client Access and ODBC users.

ODBC does not introduce any new security exposures. The same security issues have existed since Version 1 of OS/400. What ODBC did introduce was a plethora of desktop applications that offer easy access to data on the AS/400 by way of a few mouse clicks.

Object Level Security

Object level security allows you to define who can use an object and how that object can be used. Object level security cannot be 'circumvented'. OS/400 checks authority every time the object is accessed. Here's an example of the object security on a file:

```
 Object secured by authorization list  . . . . . .   ORDENTRY

                          Object    ----------Object-----------
 User        Group       Authority  Opr  Mgt  Exist  Alter  Ref
 ADMIN1                  *ALL        X    X     X      X     X
 *PUBLIC                 *USE        X

             Object    ---------------Data---------------
 User        Authority  Read  Add  Update  Delete  Execute
 ADMIN1      *ALL        X     X     X       X        X
 *PUBLIC     *USE        X                            X
```

In this example an authorization list named ORDENTRY is used to ease administration. User profile ADMIN1 has full access to the file, while *PUBLIC can read but not update the file. In this example, user profiles running the order entry application would be entered into the ORDENTRY authorization list with *CHANGE authority. For further information reference the manual: *Security - Reference*, SC41-5302.

Program Adopted Authority

The system administrator may want to limit the 'methods' by which a user profile can access data. In the example above, they might decide that the order entry user profiles should have NO access to the files except when running the order entry application. OS/400 supports this concept using 'program adopted authority'. With program adopted authority, the user profile running the application program 'adopts' an authorization level attached to the program itself. This technique was commonly used before R310 to ensure the integrity of data.

Using the order entry example, it might be changed such that the order entry PROGRAM has *CHANGE authority to the files but the order entry user profiles have

*NONE access to the files. The order entry user profiles have *USE authority to the PROGRAM. When running the program, the order entry profiles 'adopt' the authority of the program, thereby gaining access to the file. For further information, reference the manual: *Security - Reference*.

Referential Integrity

In R310, DB2/400 introduced database support for referential integrity. Prior to R310, many customers used 'program adopted authority' to protect the integrity of their data base. Using our order entry example, we might want to ensure that no order detail entry could be made w/o matching an existing order header. Using program adopted authority, every program that updates the order entry files must contain this check. Referential Integrity allows the database 'rules' to be coded directly in the database. We no longer have to restrict file updates to certain programs to ensure integrity.

For further information, reference the manuals: *DB2 for AS/400 Database Programming*, SC41-5701 and *Database 2/400 Advanced Database Functions, GG24-4249*.

Other

Some system administrators attempt to secure access to the data rather then the data itself. This is extremely risky as they have to know ALL the methods by which data can be accessed. Some common techniques are listed below:

Limit Capabilities - User Profile Parameter Useful for 'green screen' or 5250 emulation based applications, this method assumes that if you prevent users from typing in commands on a 5250 screen, they can only access data via the programs and menus that the system administrator provides them. THIS METHOD IS ONLY SECURE IF ALL 'BACKDOORS' ARE ACCOUNTED FOR. 'Backdoors' include TCP/IP, APPC communications, Client Access and a multitude of OEM products. For further information, reference the manual: *Security - Reference*.

User Exit Programs A user exit program allows the system administrator to secure an IBM-supplied host server program. System administrators often overlook the DDM user exit, which is documented in OS/400 Communications. The DDM server is used by some Client Access functions such as the OLE DB provider and the older Remote Command, as well as many APPC programs. See "Client Access Concepts - Host Server" below for an explanation of a host server. For detailed information, reference the manuals: *AS/400 Client Access Host Servers*, SC41-5740 and *Distributed Data Management*, SC41-5307.

AUDIT LOGS (monitoring security) OS/400 has several logs that can be used to monitor security. QHST, the history log, contains messages that relate to security changes made to the system. For detailed monitoring of security related functions, QAUDJRN can be enabled. Using the *SECURITY value logs the following functions:

- Changes to object authority
- Create, change, delete, display, and restore operations of user profiles

- Changes to object ownership
- Changes to programs (CHGPGM) that will now adopt the owners profile
- Changes to system values and network attributes
- Changes to subsystem routing
- When the QSECOFR password is reset to the shipped value by DST
- When the DST security officer password is requested to be defaulted
- Changes to the auditing attribute of an object

For further information, reference the manual: *Security - Reference*.

JOURNALS Journaling is often used with client/server applications to provide commitment control. The journals will contain detailed information on every update made to a file being journaled. The journal information can be formatted and queried to sift for specific information, such as the user profiles that updated the file, the records that were updated, the type of update, etc. OS/400 also allows user-defined journal entries. Used with a user exit program or trigger, this offers a relatively low overhead method of maintaining user-defined audits. For further information, reference the manuals: *Backup and Recovery*, SC41-5304 and *Security - Reference*.

Client Access Concepts

HOST SERVER Each client application on a PC is a program that communicates with a corresponding server on the AS/400. This server processes a request and returns data. OS/400 has several servers included in the base operating system. These are documented in the manual *AS/400 Client Access Host Servers*. Examples of host server programs: File Server - used by shared folders; Database Server - used by ODBC; Data Queue Server- used by the data queue APIs and the Windows network driver. A system administrator must watch for changes or additions to these servers.

Any communications program can act as a server. The system administrator must also be aware of ANY OEM server or user-written server that may reside on their system. Any of these can be used to access AS/400 data.

SERVER PROGRAM SECURITY IBM server programs authenticate the user ID and password sent by the client on the program 'allocate' or connect. By default, this is the userID and password of the Client Access 'router,' but many servers, including ODBC, allow you to override this and pass a different userID and password. After verifying the security information, the server program uses the authority of this userid when attempting to access data. For further information reference the manual: *APPC Programming*, SC41-5443.

USER EXIT PROGRAMS Each server program has an associated user exit. A user exit program allows an administrator to control what activities a client is allowed to do for each of the specific servers. Whenever a request is sent to the

server, the server first checks if a user exit program is registered. If so, it will pass to the user exit a block of data with the request information. The user exit program can process this data and return a flag allowing or preventing the server from processing the request.

User exit programs can have a significant impact on performance.

COMMON BACKDOORS Several servers offer methods to submit AS/400 commands via the client. Restricting command line usage does not block this. Remote Command uses either the DDM server or the Remote Command and Distributed Program Call Server depending on the client. Both remote SQL and ODBC support a method of executing AS/400 commands and programs by a call to QCMDEXC.

ODBC

Restricting Program access to the database System administrators often need to limit access to particular files to a certain program or set of programs. A 'green screen' programmer would implement restriction by using program adopted authority. A similar method can be used with ODBC.

Stored procedures allow ODBC programmers to implement program adopted authority. The programmer may not want users to be able to manipulate database files using desktop applications such as Microsoft Access or Lotus 1-2-3. Instead, the programmer may want to limit database updates to only their application. To implement this, user access to the database must be restricted with object level security or user exit programs. The application must be written to send data requests to the stored procedure and have the stored procedure update the database.

Another advantage to this approach is that it 'hides' the details of your code from others. Anyone can use ODBC spy to monitor SQL calls and reverse engineer an application or database. Stored procedures hide many of the details from a 'spy'.

For further information on stored procedures reference the manuals: *DB2 for AS/400 Database Programming*and *AS/400 Client/Server Performance Using the Windows Clients, SG24-4526*.

Restricting CPU utilization by user ODBC has greatly eased accessing AS/400 data. One negative impact has been that users may accidentally create very CPU intensive queries without realizing it. ODBC runs at an interactive job priority and this can severely impact system performance. The AS/400 supports a 'query governor'. The query governor can be willingly invoked by ODBC (for example, by the PC application) in a stored procedure call or by ODBC APIs by way of the query timeout parameter (R312 or later of Client Access). Also, a user exit program can 'force' the query governor on the ODBC job. The time limit is specified on the QRYTIMLMT parameter of the CHGQRYA CL command.

For further information reference the manuals: *DB2 for AS/400 Database Programming*, *AS/400 Client Access Host Servers* and *AS/400 Client/Server Performance Using the Windows Clients*.

ODBC user exit (Database server user exit) Client Access ODBC requests use one of four user exit points in the IBM-supplied database server. To allow read only access to data, QIBM_QZDA_NB1 and QIBM_QZDA_SQL1 must be monitored.

QIBM_QZDA_INIT This is called once at server initialization, such as an ODBC connect. Obviously, this is the ideal exit to use for a query governor or to completely reject incoming requests.

QIBM_QZDA_NB1 This exit is called for 'native' database requests. An exit program should monitor this exit point to block or restrict database file functions such as Delete file, Clear file (erase all data), certain create file commands, and so forth.

QIBM_QZDA_SQL1 This exit point processes the data related to SQL requests. The first 512 bytes of the SQL statement are passed to the exit program. NO PARSING OR PREPROCESSING OF THE STATEMENT IS DONE. The programmer writing the exit program must parse the statement and interpret its content. For example, to limit certain users to read only access, the exit program would have to scan each string for any SQL function that can update, delete, or insert data into a file. There is significant overhead associated with using this exit point because it is called for each SQL request. Object level security should be considered first.

QIBM_QZDA_ROI1 This exit point is called for requests that retrieve information about objects on the database server (catalog requests). Information can only be retrieved.

Caution:

1. Different ODBC drivers use different servers. You may secure the IBM Database Server by way of a user exit, but other ODBC drivers may not use this. Some of the AS/400 ODBC drivers use the remote SQL server. Many OEM ODBC drivers supply their own server program on the AS/400 or use DRDA or DDM.
2. DRDA (a type of DDM request) uses the SQL Client Integration exit point. This exit point was not documented in R310. See an R360 or later *System API Reference*, SC41-5801, and see the chapter regarding File APIs for further information.
3. DDM user exits are documented in the DDM manual. See Distributed Data Management under the references.

For detailed information, reference *AS/400 Client Access Host Servers* or your OEM ODBC driver documentation.

ODBC INI file restrictions ODBC supports a query timeout that can be set by way of an ODBC API call. Release R311 or later of Client Access for Windows 95/NT implements this function.

Some ODBC drivers support a 'read only' ini file setting. Although not secure, this setting can assist in preventing 'accidental' delete or update operations. The Client Access for Windows 95/NT ODBC driver supports this function.

Security Tools

IBM has a Security Toolkit available for V2R3 systems and above that helps analyze system security. At the time of this printing, the Security Toolkit was available for free from IBM DIRECT. Call 1-800-426-2255 and request THE SECURITY TOOLKIT FOR OS/400.

PRPQ's:

R230 and R305	PRPQ P84277, LP 5799-XDH
R310	PRPQ P84280, LP 5799-XJD
R360	PRPQ P84281, LP 5799-XDK

Additional Assistance

In-depth security reviews and assistance to implement the strategies above is available through IBM Consultline (1-800-274-0015). Please consult the following manuals for in-depth information on specific topics:

AS/400 Client Access Host Servers

Security - Reference

DB2 for AS/400 Database Programming

Database 2/400 Advanced Database Functions

APPC Programming

Distributed Data Management

Backup and Recovery

AS/400 Client/Server Performance Using the Windows Clients

Appendix E. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described 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:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

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
Software Interoperability Coordinator
3605 Highway 52N
Rochester, MN 55901-9986
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Programming Interface Information

This publication is intended to help you to understand and use the ODBC interface as implemented in Client Access for Windows 95/NT.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

Advanced 36
Application System/400
AS/400
AS/400e
AT
C/400
Client Access
DRDA
IBM
Operating System/400

OS/2
OS/400
SAA
SQL/400
400

Approach and Notes are trademarks of Lotus Development Corporation in the United States, or other countries, or both.

Microsoft, Windows, Windows NT, and the Windows 95 logo are registered trademarks of Microsoft Corporation.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

i486 is the trademark or registered trademark of Intel Corporation in the U.S. and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Index

A

ALWCPYDTA parameter
  effect on query optimizer 5-7
application locking, PC B-1

B

blocked insert calls from C, examples of 8-2
blocked inserts 8-1

C

C, blocked insert calls from 8-2
C, stored procedure calls from 7-4
catalog functions 9-1
CL commands through SQL stored procedures and ODBC, running 7-6
Client Access ODBC performance tuning
  fine tuning record blocking 3-4
  introduction to client/server performance 3-1
  introduction to performance 3-1
  registry 3-3
  selecting a stringent level of commitment control 3-3
  the performance architecture 3-2
  using extended dynamic SQL 3-4
COBOL source, host 7-20, 7-28, 7-33
Coding to ODBC APIs
  basic application flow 6-1
  basic application step 6-1
  block insert 6-15
  Calling ODBC Functions 6-1
  calling stored procedure 6-9
  calling stored procedures with result sets 6-10
  ending 6-19
  establishing ODBC connection 6-2
  executing ODBC function 6-3
  executing prepared statement 6-4
  extended fetch 6-13
  retrieving results 6-8
commitment control 2-16
creating a unique index B-1

D

data source options 2-15
default filter factors 5-7
Definition
  default filter factors 5-7

E

end-user tools, performance considerations of common 4-1
examples of blocked insert calls from C 8-2
examples of stored procedures 7-3
exit program parameter formats 10-9
Exit programs 10-1
exit programs, sample user 10-1
extended dynamic support 2-17

F

filter factors, default
  in query optimization 5-7
formats, exit program parameter 10-9
functions, catalog 9-1

H

host COBOL source 7-20, 7-28, 7-33
host PL/I source 7-8
host RPG source 7-5

I

index, creating a unique B-1

K

key range estimate 5-8

L

lazy close support 2-18
library
  default libraries 2-15
  OS/400 library view 2-18

M

making tables able to be updated B-1

N

Notices E-1

O

ODBC
- basic application flow 6-1
- block insert 6-15
- calling stored procedure 6-9
- calling stored procedures with result sets 6-10
- Coding to ODBC APIs 6-1
- comparison of ODBC techniques 6-20
- compromise between Jet and ODBC APIs 6-21
- ending 6-19
- establishing ODBC connection 6-2
- executing ODBC function 6-3
- executing prepared statement 6-4
- extended fetch 6-13
- ODBC and Visual Basic 6-5
- retrieving results 6-8

ODBC driver
- CCSID 65535 translation 2-19
- commitment control 2-16
- default libraries 2-15
- extended dynamic (package) support 2-17
- lazy close support 2-18
- local package caching 2-17
- OS/400 library view 2-18
- pre-fetch during EXECUTE 2-18
- private and shared packages 2-17
- translation DLL 2-19
- translation option 2-19

ODBC driver setup 2-1
- adding the local AS/400 system to RDB directory 2-1
- configuring the ODBC data source 2-4
  - Format options - procedure 2-14
  - Language options - procedure 2-10
  - Other options 2-11
  - Package(s) options - explanation 2-17
  - Package(s) options - procedure 2-6
  - Package(s) options for multiple users - procedure 2-8
  - Performance options - explanation 2-18
  - Performance options - procedure 2-8
  - Server options - explanation 2-15

ODBC driver setup *(continued)*
- configuring the ODBC data source *(continued)*
  - Server options - procedure 2-4
  - Translation options - explanation 2-18
  - Translation options - procedure 2-13
- selecting the Client Access ODBC driver 2-1
- setting up the ODBC data source 2-2

ODBC performance tuning 3-1
ODBC troubleshooting C-1
ODBC, running CL commands through SQL stored procedures and 7-6
OPTIMIZE FOR n ROWS clause
- effect on query optimizer 5-7

P

packages
- extended dynamic support 2-17
- local package caching 2-17
- private and shared packages 2-17

parameter formats, exit program 10-9
PC application locking B-1
performance
- lazy close support 2-18
- OS/400 library view 2-18
- pre-fetch during EXECUTE 2-18

performance considerations of common end-user tools 4-1
performance tuning 3-1
performance, SQL 5-1
performance, tool behaviors that degrade 4-1
PL/I source, host 7-8
PowerBuilder, remote procedure calls 7-9
PowerBuilder, result sets 7-22
PowerBuilder, stored procedure calls from 7-8, 7-9, 7-22
procedures and ODBC, running CL commands through SQL stored 7-6
procedures, examples of stored 7-3
procedures, stored 7-1
program parameter formats, exit 10-9
Programs, exit 10-1
programs, sample user exit 10-1

Q

query optimizer
- default filter factors 5-7
- optimization goals 5-7

R

RDB directory 2-15
relational database directory 2-15
Result Sets, stored procedure calls from C 7-4
RPG source, host 7-5

S

sample user exit programs 10-1
setup 2-1
  adding the local AS/400 system to RDB directory 2-1
  configuring the ODBC data source 2-4
    Format options - procedure 2-14
    Language options - procedure 2-10
    Other options 2-11
    Package(s) options - explanation 2-17
    Package(s) options - procedure 2-6
    Package(s) options for multiple users - procedure 2-8
    Performance options - explanation 2-18
    Performance options - procedure 2-8
    Server options - explanation 2-15
    Server options - procedure 2-4
    Translation options - explanation 2-18
    Translation options - procedure 2-13
  selecting the Client Access ODBC driver 2-1
  setting up the ODBC data source 2-2
SQL performance
  cost estimation 5-6
  database design 5-2
  general considerations 5-1
  indexes 5-5
  normalization 5-2
  optimizer 5-6
  optimizer decision-making rules 5-8
  table size 5-4
SQL stored procedures and ODBC, running CL commands through 7-6
stored procedure 6-9
stored procedure calls from C with result sets 7-4
stored procedure calls from C with return values 7-4
stored procedure calls from PowerBuilder with no return values 7-8
stored procedure calls from PowerBuilder with result sets 7-22
stored procedure calls from PowerBuilder with return values 7-9
stored procedure calls from Visual Basic with result sets 7-35
stored procedure calls from Visual Basic with return values 7-30
stored procedures 7-1
stored procedures and ODBC, running CL commands through SQL 7-6
stored procedures with result sets 6-10
stored procedures, examples of 7-3

T

tables updating B-1
tool behaviors that hurt performance 4-1
translation
  CCSID 65535 translation 2-19
  translation DLL 2-19
  translation option 2-19
troubleshooting, ODBC C-1

U

unique index, creating a B-1
user exit programs, sample 10-1

V

Visual Basic
  binding character and binary parameters 6-9
  compromise between Jet and ODBC APIs 6-21
  converting string and arrays of byte 6-6
  stored procedure calls with result sets 7-35
  stored procedure calls with return values 7-30

Communicating Your Comments to IBM

AS/400e series
Client Access for Windows 95/NT ODBC User's Guide
Version 3

Publication No. SC41-3535-01

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
  - United States and Canada: 1-800-937-3430
  - Other countries: 1-507-253-5192
- If you prefer to send comments electronically, use this network ID:
  - IBMMAIL(USIB56RZ)
  - IDCLERK@RCHVMW2.VNET.IBM.COM

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies.

Readers' Comments — We'd Like to Hear from You

AS/400e series
Client Access for Windows 95/NT ODBC User's Guide
Version 3

Publication No. SC41-3535-01

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	□	□	□	□	□

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	□	□	□	□	□
Complete	□	□	□	□	□
Easy to find	□	□	□	□	□
Easy to understand	□	□	□	□	□
Well organized	□	□	□	□	□
Applicable to your tasks	□	□	□	□	□

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? □ Yes □ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

Company or Organization _____ _____

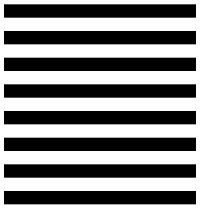
Phone No. _____ _____

Readers' Comments — We'd Like to Hear from You
SC41-3535-01

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM CORPORATION
ATTN DEPT 542 IDCLERK
3605 HWY 52 N
ROCHESTER MN 55901-7829

Fold and Tape | **Please do not staple** | Fold and Tape

SC41-3535-01

Cut or Fold
Along Line

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC41-3535-01

Spine information:

AS/400e series **Client Access for Windows 95/NT ODBC User's Guide** *Version 3*