User's Guide

Acu4GL®

Version 8.1.3

Micro Focus (IP) Ltd.
9920 Pacific Heights Blvd, Suite 150
San Diego, CA 92121
858.795.1900

E-01-UG-100501-Acu4GL-8.1.3

Contents

Chapter 4: Using Directives

Chapter 5: Invalid Data

Chapter 6: Working with COBOL

Chapter 7: New and Existing Databases

Appendix C: Acu4GL for Oracle Information

Appendix D: Acu4GL for ODBC Information

Appendix E: Acu4GL for Sybase Information

Appendix F: Acu4GL for DB2 Information

1 Acu4GL Overview

Key Topics

1.1 Welcome to Acu4GL

Welcome to the Acu4GL® family of products! This guide describes how the ACUCOBOL-GT® programming language uses Acu4GL interfaces to access information stored in Oracle®, Informix®, Sybase®, Microsoft® SQL Server, DB2®, and ODBC-compliant relational database management systems (RDBMSs). It explains how to describe data in your COBOL programs, and how to handle differences in the field names and data types that are passed between COBOL and the database engine. Acu4GL is part of the ***extend***® family of solutions.

Note: Unless otherwise indicated, the references to "Windows" in this manual denote the following versions of the Windows operating systems: Windows XP, Windows Vista, Windows 7, Windows 2003, Windows 2007, Windows 2008 R2. In those instances where it is necessary to make a distinction among the individual versions of those operating systems, we refer to them by their specific version numbers ("WindowsXP," "Windows Vista," etc.).

The Micro Focus ***extend*** unit tested the Acu4GL Version 8.1.1 products with the following RDBMSs:

- Oracle
- Microsoft SQL Server
- DB2
- Sybase
- Informix
- ODBC

As a result of our testing and according to general information received from the makers of the RDBMS regarding upward compatibility of their product, we believe the following to be accurate. Please check the Micro Focus Web site for the most recent versions of this table.

RDBMS	**Acu4GL Version 8 tested with RDBMS version**	**Probable compatibility with Acu4GL Version 8, based on vendor information** (Client/Server may be used for connecting to other versions)
Informix	9, 10 (32- and 64-bit)	Versions greater than 7.3
Microsoft SQL Server	NT, 2000, and 2005	Versions greater than 6.5
ODBC Specification	3.0	Versions greater than 3.0
Oracle	9i for UNIX® (32- and 64-bit) and Windows, 10g for UNIX (32- and 64-bit) and Windows	Versions greater than 9i
Sybase	11.*x*, 12.5	Versions greater than 11.*x*
DB2	8.1, 8.2, and 9	No others

The essence of an Acu4GL product is that *standard COBOL I/O statements are used to access databases*. Acu4GL handles the translation between standard COBOL and the language that the RDBMS understands.

There are special instructions and status messages that apply to each individual RDBMS. These are included in the appendices of this manual: each RDBMS has an appendix dedicated to its own specific information.

1.2 Document Overview

The *Acu4GL User's Guide* is organized in the following manner:

Chapter 1: Acu4GL Overview

This chapter offers an overview of this manual, an explanation of accessing data, an explanation of database concepts, and a lesson on how Acu4GL works.

Chapter 2: **Chapter 2: Getting Started**

After pointing you in the right direction for installing and setting up Acu4GL, the "Getting Started" chapter shows you how to use the **sql.acu** program, as well as instructs you on using the Acu4GL demo program. Technical support information is also found in this chapter.

Chapter 3: **Chapter 3: Data Dictionaries**

"Data Dictionaries" details all you should know about XFD files, which map COBOL records to database fields, and how to work with them.

Chapter 4: **Chapter 4: Using Directives**

This chapter tutors you on the use of directives, the optional comments that control how things look on the database side.

Chapter 5: **Chapter 5: Invalid Data**

"Invalid Data" tells you about illegal COBOL data items and what happens when they are stored in the database, as well as an explanation of invalid database items and how they are translated into COBOL.

Chapter 6: **Chapter 6: Working with COBOL**

This chapter gives you an opportunity to work with COBOL by preparing and compiling a COBOL program.

Chapter 7: **Chapter 7: New and Existing Databases**

The "New and Existing Database Files" chapter discusses default behavior, along with matching existing COBOL data to database data.

Chapter 8: **Chapter 8: Compiler and Runtime Options**

This chapter details compiler options and runtime configuration variables that can be used with all of the Acu4GL products.

Chapter 9: **Chapter 9: Performance and Troubleshooting**

The “Performance and Troubleshooting” chapter provides guidelines for improving system performance and an alphabetical listing of compile-time error messages and recommended recovery procedures for each error.

Chapter 10: **Chapter 10: General Questions and Answers**

This chapter lists some frequently asked questions and answers that pertain across the Acu4GL family of interfaces. A section in each appendix contains FAQs specific to that RDBMS.

Appendix A: **Appendix A: Acu4GL for Informix Information**

The Informix appendix instructs you on installing and setting up the Informix Acu4GL product, discusses Informix technical specifications, and provides troubleshooting options.

Appendix B: **Appendix B: Acu4GL for Microsoft SQL Server Information**

This appendix provides you with an overview of Microsoft SQL Server concepts, instructs you on installing and setting up the Microsoft SQL Server Acu4GL product, details the RDBMS’s technical specifications, and also tells you about troubleshooting options.

Appendix C: **Appendix C: Acu4GL for Oracle Information**

This appendix provides you with an overview of Oracle concepts, instructs you on installing and setting up the Oracle Acu4GL product, gives you the RDBMS’s technical specifications, and also lists troubleshooting options.

Appendix D: **Appendix D: Acu4GL for ODBC Information**

The ODBC appendix provides you with an overview of ODBC concepts, instructs you on installing and setting up the ODBC Acu4GL product, informs you of the RDBMS's technical specifications, and discusses troubleshooting options.

Appendix E: **Appendix E: Acu4GL for Sybase Information**

This appendix provides you with an overview of Sybase concepts, instructs you on installing and setting up the Sybase Acu4GL product, and lists the RDBMS's technical specifications, as well as troubleshooting options.

Appendix F: **Appendix F: Acu4GL for DB2 Information**

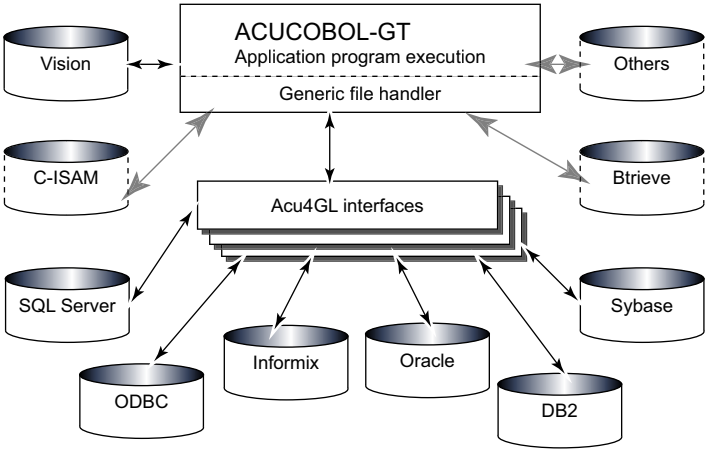
This appendix provides you with an overview of DB2 concepts, instructs you on installing and setting up the DB2 Acu4GL product, and lists the RDBMS's technical specifications, as well as troubleshooting options.

1.3 Accessing Data

The standard file system supplied with ACUCOBOL-GT is the Vision indexed file system. (On VAX VMS systems, RMS is used instead of Vision.) Vision offers a wide variety of features, including variable-length records, data compression, and data encryption.

At your option, Vision can be replaced by (or used in conjunction with) other indexed file systems such as C-ISAM, and Btrieve®; database management systems such as Oracle, Microsoft SQL Server, Informix, Sybase, and DB2;

and data sources accessible through ODBC. This is possible because all of ACUCOBOL-GT's I/O passes through a *generic file handler* that can accommodate a wide variety of protocols.

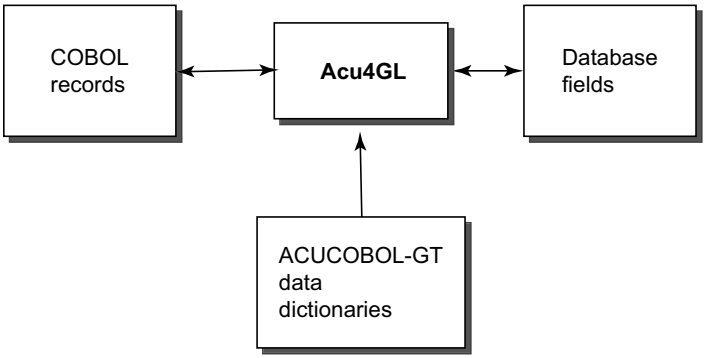


1.3.1 Interface Routines

All file systems communicate with the ACUCOBOL-GT generic file handler via *interface routines*. The interfaces to external file systems such as C-ISAM and the various RDBMSs are available from Micro Focus as add-on modules. The user is free to add any combination of traditional ISAM interfaces and RDBMS interfaces that will meet a site's particular needs. The ACUCOBOL-GT runtime system communicates with relational databases via a special family of add-on products called Acu4GL.

1.3.2 Data Dictionaries and Acu4GL

An Acu4GL product uses *data dictionaries* created by the ACUCOBOL-GT compiler. These dictionaries map COBOL records into database fields and map the database fields back into records. Data dictionaries are also known as XFDs (e**X**tended **F**ile **D**escriptors).



The following sections describe the Acu4GL products and explain how they interface to database systems by referencing data dictionaries.

Interfaces to indexed file systems such as Vision and C-ISAM do not require data dictionaries, because they are record-oriented systems, but they can be used by the indexed file editor **alfred** to view records in these alternate file systems.

1.3.3 The ACUCOBOL-GT Web Runtime and Acu4GL

The ACUCOBOL-GT Web Runtime is enabled for Acu4GL. In essence, if an Acu4GL ".dll" file is in the path and the Acu4GL license with the same base name is in the Web runtime directory, and the appropriate configuration file options are set, the Web runtime can take advantage of the acu4GL interface.

The particular .dll files for supported 32-bit runtimes are as follows:

a4ora32.dll — Acu4GL for Oracle
a4sql32.dll — Acu4GL for Microsoft SQL Server

a4syb32.dll — Acu4GL for Sybase
a4db232.dll — Acu4GL for DB2
a4odbc32.dll — Acu4GL for ODBC

Note: Microsoft Internet Explorer Version 5.5 Service Pack 2 and above supports the ACUCOBOL-GT Web Runtime. For more information regarding the ACUCOBOL-GT Web Runtime, please consult the *Programmer's Guide to the Internet* user manual.

1.4 Database Concepts

Relational databases differ from indexed file systems in several significant ways. These are the logical associations between database concepts and COBOL indexed file concepts:

Indexed File Concept	Database Concept
Directory	Database
File	Table
Record	Row
Field	Column

Or, put another way:

- Database operations are performed on columns. Indexed file operations are performed on records.
- A COBOL indexed file is logically represented in database table format. Within a table, each column represents one COBOL field; each row represents one COBOL record.
- Database table columns are strictly associated with a particular data type, such as date, integer, or character. In COBOL, data can have multiple type definitions.

For example, for Oracle, a COBOL record that looks like this:

```
01  terms-record.
    03  terms-code        pic 999.
    03  terms-rate        pic s9v999.
    03  terms-days        pic 9(2).
    03  terms-descript    pic x(15).
```

would be represented in the database as a table with a format similar to this:

Name	Null	Type
TERMS_CODE	NOT NULL	NUMBER (3)
TERMS_RATE		NUMBER (4, 3)
TERMS_DAYS		NUMBER (2)
TERMS_DESCRIPT		CHAR (15)

1.5 How Acu4GL Works

Acu4GL implements a direct, transparent interface between COBOL and RDBMSs.

Previously, accessing a relational database from a COBOL program involved writing SQL code and embedding that code in your program. You had to know SQL, and you had to write SQL statements appropriate for the specific database you wanted to access. Because your queries were tailored to suit one database management system, you had to recode your COBOL application if you wanted to access a different RDBMS, or an indexed file system, or even to migrate a file from one system to another.

As an alternative, some interface products now translate COBOL I/O statements into direct reads and writes on the database files, without going through the driver, or engine, supplied by the database manufacturer. This means that the COBOL programmer must provide for enforcement of database rules that the engine already knows about and is designed to handle. Bypassing the database engine also means that new constraints or changes in the database structure will require reprogramming of the COBOL application.

Acu4GL implements a more suitable approach by dynamically generating industry standard SQL from COBOL I/O statements. As the ACUCOBOL-GT runtime module is executing your COBOL application, Acu4GL is running behind the scenes to match up the requirements and rules of both COBOL and the RDBMS to accomplish the task set by your application. This means that Acu4GL uses the full power designed into the database engine. The engine enforces database rules and constraints; any violations are returned to the COBOL program as I/O error conditions.

1.5.1 What Is a Transparent Interface?

Acu4GL products provide a transparent, efficient interface between your program and the relational database. The interface is categorized as transparent because the communication between your COBOL program and the database is smooth, with no special query coding on your part and no interruptions in the execution of your program. You need not change your COBOL code if you later want to access a different database or to access an alternate indexed file system.

The information exchange operations between the database and the COBOL program are invisible to the end user. For example, if your program specifies a READ, a database query is automatically generated by the interface. Then the data that is read from the database is translated into a COBOL record. This exchange occurs in fractions of a second, and the application proceeds without interruption.

1.5.2 Data Dictionaries and Mapping

Because relational databases handle I/O at the *column* (field) level, and COBOL handles I/O at the *record* level, some mapping is necessary to associate records with their fields. One function of the Acu4GL product is to map COBOL records into database fields, and to map the database fields back into records. Acu4GL does this by consulting data dictionaries generated by the compiler. The detailed structure of data dictionaries is discussed in **section 3.2, "XFD Files."** The next section describes how and when these data dictionaries are built.

1.5.3 Steps to Follow

The Acu4GL product builds its own database queries dynamically whenever an input or output request is received. These are the steps that you take to compile your program and execute it using Acu4GL:

Compiling with "-Fx"

You compile your standard COBOL application with ACUCOBOL-GT. See the appendix specific to your RDBMS for details to determine which version of the compiler is required. When you compile, you specify via a compile-time option that you want the compiler to generate data dictionaries, in addition to an object code file.

Creating dictionaries

An ACUCOBOL-GT data dictionary is created by the compiler for each indexed file in your program. These data dictionaries map COBOL records in an indexed file to rows in a database. Like ACUCOBOL-GT object files, these data dictionaries are portable across platforms.

Setting the files host

In your configuration file, you may specify which RDBMS or ISAM file system you are using by setting the **DEFAULT_HOST** variable (sets a default for all files), or the **filename_HOST** variable (sets a file system for individual files), or both. For example, you might say DEFAULT_HOST VISION and EMPFILE_HOST ORACLE. This would direct EMPFILE input and output functions to the Oracle RDBMS via Acu4GL, and direct I/O for all other indexed files to the Vision system. These are runtime settings that allow you to change hosts without recompilation, and enable you to tailor your application for the specific needs of a particular end-user site.

Note: The *filename*_HOST variable also enables you to do load balancing and migration of your application. You can move Vision files into the database in an incremental fashion, allowing you to test scaling and performance of the new configuration.

Setting database-specific variables

At this point, you set variables in the runtime configuration file that apply to the specific database system you are using. See the appendix specific to your RDBMS for details.

Passing I/O requests to the interface

You use the Acu4GL-enabled ACUCOBOL-GT runtime system to execute your application. Whenever the runtime system encounters an input or output instruction (such as READ or WRITE) on a file that is directed to an RDBMS, it passes the request to the Acu4GL product.

Automatically building SQL statements

Acu4GL automatically builds SQL instructions that your database management system can understand. As it builds these SQL instructions, it looks at the ACUCOBOL-GT data dictionary, which maps the COBOL record and its fields to the table rows and its columns.

Accessing a database

The database management system uses its own dictionary as a pointer into its own data files, performs the requested I/O operation, and passes the results back to the Acu4GL product.

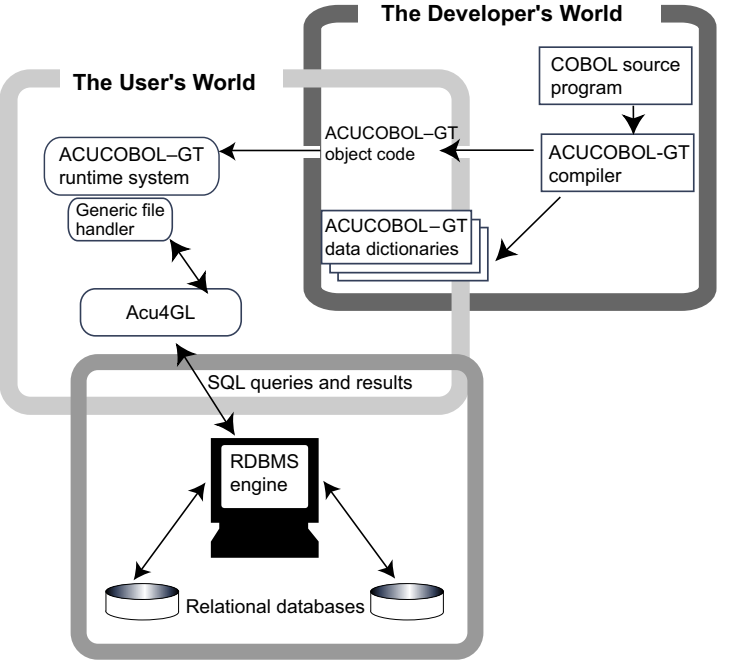
Forming COBOL records

Acu4GL translates the data fields into COBOL records or status codes, which are then passed back to the runtime system via the generic file handler.

All of this communication is automatic, and all database queries and translations are performed behind the scenes, so that the end user experiences no interruption in program execution.

1.5.4 Summary

The Acu4GL product ensures that all changes to your database are immediately available to your COBOL program. Also, it ensures that all data updates introduced by your COBOL program are immediately reflected in the database.



After your ACUCOBOL-GT data dictionaries have been generated, you can switch to a different RDBMS simply by linking in a different Acu4GL module and setting **DEFAULT_HOST** to point to the new RDBMS. No recompiling is necessary.

Because Acu4GL accesses the database through its native engine, the full relational integrity of the database is maintained. The COBOL program need not be concerned about enforcing relationships between keys and foreign keys on tables, constraints on field relationships and contents, and so forth.

If you have specified that a file is to reside on a RDBMS by setting either DEFAULT_HOST or **filename_HOST**, an OPEN OUTPUT *filename* statement in your COBOL program will automatically generate a CREATE TABLE *filename* function on the specified host database. Appropriate permissions will be granted based on your login and password and on access parameters you have set. See the "Contents" page to find the appendix specific to your RDBMS for details. If a table will eventually be large, you may wish to have your systems database administrator create the table and indexes for you. Databases offer many methods of creating tables that may improve performance for large tables, such as spreading the table across multiple hard drives or storing the data and the indexes on different drives.

Note: Acu4GL uses a specific naming convention for indexes it creates. Indexes created with Acu4GL use the i<TABLENAME>_<key value> convention. If you are using indexes created outside of the Acu4GL application, you must ensure that the index names match the Acu4GL naming convention.

Changes to the way a database functions should be carefully considered and tested before being fully implemented. If a database violation occurs, the engine detects it, and Acu4GL returns an I/O error condition to the COBOL program. The program can, if desired, call a standard ACUCOBOL-GT library routine for an extended error description.

It is possible that there are ACUCOBOL-GT library routines that do not work with or do not make sense to use with some of the Acu4GL products. To find out if that is the case with your Acu4GL product, look in the "Common Questions and Answers" section of the appendix specific to your product.

2 Getting Started

Key Topics

2.1 Getting Started

The “Getting Started” chapter points you in the right direction for installing and setting up the Acu4GL® interface for your particular RDBMS. Once Acu4GL is installed and ready, you should return to this chapter to find the information you need for operating the **sql.acu** utility, preparing and compiling a COBOL program, and running the demonstration program.

2.2 Technical Services

For the latest information on contacting customer care support services go to:

http://www.microfocus.com/about/contact

For worldwide technical support information, please visit:

http://supportline.microfocus.com/xmlloader.asp?type=home.

2.3 Installation

The installation for your particular RDBMS can be found in the appendices of this manual:

Appendix A: **Appendix A: Acu4GL for Informix Information**
Appendix B: **Appendix B: Acu4GL for Microsoft SQL Server Information**
Appendix C: **Appendix C: Acu4GL for Oracle Information**
Appendix D: **Appendix D: Acu4GL for ODBC Information**
Appendix E: **Appendix E: Acu4GL for Sybase Information**
Appendix F: **Appendix F: Acu4GL for DB2 Information**

Each of the appendices contains an overview of the Acu4GL functions, installation instructions, setup instructions, technical specifications, and a troubleshooting guide.

Note: Before going any further in this manual, go to the appendix listed above that pertains to your RDBMS and begin the Acu4GL installation and setup processes. Once the installation is complete and you are ready to run, return to this chapter and begin **section 2.4, "Using the "sql.acu" Program."**

2.4 Using the "sql.acu" Program

Acucorp provides a utility program called **sql.acu**. This program gives you access to some of the standard SQL commands. It can be called from a COBOL program or executed from the command line.

As a general rule, **sql.acu** may be used to issue all SQL commands except those that perform data retrieval. The **sql.acu** program *cannot* perform statements that return data, such as the SELECT statement. This category of statements returns an error when passed to the **sql.acu** program.

The global variable *return-code* is 0 when a command has completed successfully. If a command is not successful, *return-code* contains an error code.

2.4.1 Running "sql.acu" From the Command Line

To use the **sql.acu** utility to create an empty table and grant access privileges to other users

1. Enter

   ```
   runcbl sql.acu
   ```

   The program will pause to accept an SQL command.

2. Enter the following (italics indicate variable names, non-italics are SQL reserved words):

   ```
   CREATE TABLE newtab (col1 CHAR(30), col2 CHAR(11))
   ```

   The program will pause to accept an SQL command.

3. Now enter the following command:

   ```
   GRANT ALL PRIVILEGES ON newtab TO PUBLIC
   ```

   The program will pause to accept an SQL command.

4. Now enter:

   ```
   CREATE UNIQUE INDEX newtab_index ON newtab (col1)
   ```

5. Next, press **Enter** again to exit the program.

Note: When naming columns and tables, make sure to use underscores ("_"), not hyphens ("-"), or a syntax error will result.

The above entries will create a new table and grant access permission to everyone. If you want to delete the table you created, enter the following command:

```
DROP TABLE newtab
```

2.4.2 To Call "sql.acu" From a Program

You can also call **sql.acu** from within a COBOL program. The syntax is:

```
call "sql.acu" using mysql-command
```

The SQL command may be up to 50,000 characters and may be a variable or a quoted command string in the CALL statement.

Note: The **sql.acu** utility may be used only with SQL commands that are data definition statements (those that do not return data). Be sure to end each SQL command with a semicolon (";").

Example

```
IDENTIFICATION DIVISION.
program-id.  command.
* the sql.acu command to issue data
* definition commands to the RDBMS.
DATA DIVISION.
working-storage section.
```

```
01  sql-command                        pic x(75).
01  error-status.
    03 primary-error                   pic x(2).
    03 secondary-error                 pic x(40).
01  error-text                         pic x(40).
01  error-window                       pic x(10).
PROCEDURE DIVISION.
main-logic.  display window erase.
     display window line 20, column 2
            size 75, lines 3, boxed,
            top title "SQL COMMAND",
            bottom right title "Return to exit".
     perform do-sql-command, with test after,
            until sql-command = spaces.
stop run.
do-sql-command.
     accept sql-command, line 1, column 1,
        erase to end of line.
if sql-command not = spaces
call "sql.acu" using sql-command
if return-code not zero
perform show-error.
show-error.
     display window line 2, column 2,
        size 75 lines 6, boxed, erase,
        pop-up area is error-window.
     call "C$RERR" using error-status,
        error-text.
     display "DATABASE ERROR:",
        secondary-error.
display error-text.
accept omitted.
close window error-window.
     display window line 20, column 2,
        size 75, lines 3.
```

2.5 The Demonstration Program

As soon as your installation and setup are complete, you may run the demonstration program (“demo.cbl”) included on your distribution medium. This program illustrates many of the Acu4GL product capabilities, including:

- how to map RDBMS data types to traditional COBOL data types
- how to compile a COBOL program in preparation for interfacing with an RDBMS
- how to modify records within the RDBMS database

The demo program simulates what might be used in a distributor's shipping department. You'll be creating the acuorders table, which contains information about the orders placed by the distributor's customers. The acuorders table has the following columns of information (the database data type and COBOL data type are shown for each):

Note: The data type may be slightly different depending on the RDBMS.

Column Name	Data Type	COBOL PIC
order_num	varchar(4)	pic 9(4)
order_date	int	pic 9(6)
customer_num	varchar(3)	pic 9(3)
ship_instruct	varchar(40)	pic x(40)
backlog	varchar(1)	pic x
po_num	varchar(10)	pic x(10)
ship_date	int	pic 9(6)
ship_weight	decimal(8,2)	pic 9(6)v99
ship_charge	decimal(6,2)	pic 9(4)v99
paid_date	int	pic 9(6)

Here is the COBOL FD (file descriptor) that matches the acuorders table. Note that the FD entries must match the names of the RDBMS fields and must match their data types:

```
fd  orders.
01  order-record.
    03  order-num              pic x(4).
    03  order-fields.
        05  order-date         pic 9(6).
        05  customer-num       pic 9(3).
```

```
05  ship-instruct      pic x(40).
05  backlog            pic x.
05  po-num             pic x(10).
05  ship-date          pic 9(6).
05  ship-weight        pic 9(6)v99.
05  ship-charge        pic 9(4)v99.
05  paid-date          pic 9(6).
```

First we'll compile the demo program. Then we'll prepare the data and run the demo.

Compiling the demo program

To compile the demo program for use with the RDBMS data source, enter one of the following commands.

For ACUCOBOL-GT® Version 4.2 or later, 32-bit version:

```
ccbl32 -Fx -o demo demo.cbl
```

For ACUCOBOL-GT Version 4.2 or later, UNIX version:

```
ccbl -Fx -o demo demo.cbl
```

This compiles the program and generates the ACUCOBOL-GT data dictionary "acuorders.xfd". The runtime system will use this data dictionary to map RDBMS data types to COBOL data types.

Verifying the runtime

Before starting the demo program, you need to verify the runtime. To do this, enter the following:

```
runcbl -vv  or wrun32 -vv
```

Be sure that the version information you receive references your Acu4GL product.

Starting the demo program

Be sure the RDBMS data source is available and that all the non-Acu4GL environment variables required for your data source are set. To run the demo program, you'll need to notify the ACUCOBOL-GT runtime system that you

will be interfacing to an RDBMS driver. You accomplish this by adding the following line to your runtime configuration file or to the machine's environment:

```
DEFAULT_HOST RDBMS_NAME
```

If you're running "demo.cbl" from the same directory where you placed the ACUCOBOL-GT data dictionary, you are ready to run.

If the data dictionary ("acuorders.xfd", created when you compiled the demo program) is in another directory, you will need to add the following line to the "cblconfig" configuration file:

```
XFD_PREFIX dir-containing-dictionary
```

Set any environment variables required for your specific Acu4GL product. See the "Designating the Host Data Source" or "Designating the Host File System" section in the appendix specific to your particular RDBMS.

At this point, you can run the demo program by entering one of the following commands.

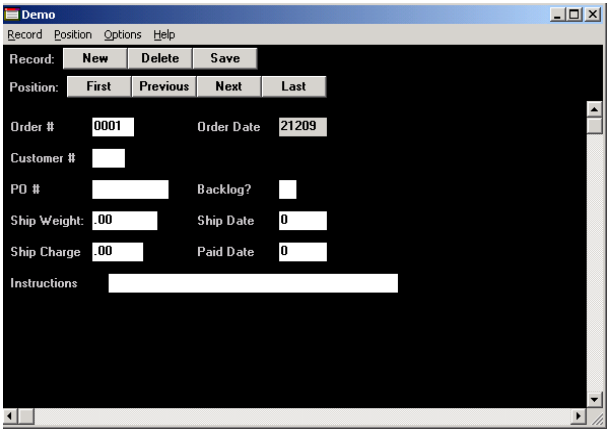
For ACUCOBOL-GT Version 5.0 or later, 32-bit Windows runtime:

```
wrun32 -c config demo
```

For ACUCOBOL-GT Version 5.0 or later, UNIX runtime:

```
runcbl -c config demo
```

For UNIX versions, you will see a character version of the following screen. For Window-based versions, this is what you'll see:



The demo program enables you to view or change any of the rows (records) in the orders table. You can also view the source code for the demo program.

Record Menu

The choices under the *Record* menu item offer the following functionality:

New

Clear all fields.

Delete

Remove the order that is being displayed from the database.

Save

Write (or rewrite) the displayed record.

Exit

Exit the demo program and return to the operating system prompt.

Position Menu

The choices under the *Position* menu item offer the following functionality:

First

View (or modify) the first record in the file.

Next

View (or modify) the next record in the sequence.

Previous

View (or modify) the previous record in the sequence.

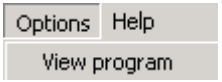
Last

View (or modify) the last record in the file.

Search

Enter the order number of the order to be viewed.

Options Menu



The choices under the *Options* menu item offer the following functionality:

View program

Start the debugger and display the source code. Be sure you compiled the program with "-Zd" and execute it with "-d".

Help

Help brings up a window containing a short description of the demo program.

3 Data Dictionaries

Key Topics

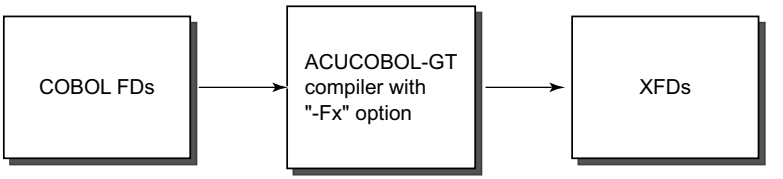
3.1 Data Dictionaries or XFDs

At the heart of the Acu4GL® database product are the data dictionaries that map COBOL records to database fields. These dictionaries are also called e**X**tended **F**ile **D**escriptors (XFDs), because they're based on standard COBOL file descriptors (FDs).

XFDs are used by Acu4GL to interface to database management systems. They are also used by the **alfred** record editor, and European character mapping in AcuConnect and AcuServer™ file server software.

3.2 XFD Files

The compiler creates an XFD for each indexed file in the compiled program when you specify the "-Fx" compile-time option. XFDs are fully portable, so no recompilation is necessary if you change hardware. The compiler option "-Fa" causes the generation of XFDs for indexed, relative, and sequential files for use with international character mapping only. The compiler option "-F4" generates version 4 XFD files for backwards compatibility with older versions of the runtime. See **section 8.1, "Compiler Options,"** for additional information.



See Chapter 5 in Book 1, *User's Guide,* of the ACUCOBOL-GT® compiler documentation for details on how the XFD files are generated from the COBOL program's FDs.

Creating XFD files at compile time offers two significant advantages:

- Any changes made to the file definitions are automatically included in the data dictionaries when the program is recompiled.

- The effects of all compile-time options, COPY REPLACING, source-code control lines, and data layout compiler flags ("-d"...) are reflected correctly in the XFDs.

3.2.1 Understanding How the Database Table Is Formed

ACUCOBOL-GT data dictionaries (XFDs) enable the Acu4GL product to create a table (or access an existing one) in the database for each indexed file. Each *column* in the table contains the values for one *field*. The column names are essentially the field names.

The table that is built is *based on the largest record in the COBOL file* and contains the fields from that record plus any key fields (key fields are those that are named in KEY IS} phrases of SELECT verbs in the FILE CONTROL section). This ensures that data from all of the COBOL records will fit within the table and simplifies the storage and retrieval process. If you were to examine the database columns, only the fields from the largest record, and the key fields, would appear. See **section 3.2.2, "Defaults Used in XFD Files,"** for additional information.

Note: If the field named in the KEY IS phrase is a group item, it will not become a column in the database table. Instead, each of the elementary items subordinate to the named group item will become a column.

You can force a group item to be a column by using the **USE GROUP** directive, described in Chapter 4.

With multiple record formats (level 01), not all COBOL fields are represented in the database by name, but all fields *are* storable and retrievable. The data dictionary maps fields from all records of a file to the corresponding locations in the master (largest) record of the file, and thus to the database table. Since Acu4GL has access to the data dictionary, it knows where the data from a given COBOL record fits in the database tables. This activity is invisible to the COBOL application.

For example, if your program has one file with the three records shown below, the underlined fields will be included in the database table by default (this example assumes that **ar-codes-key** is named in a KEY IS phrase).

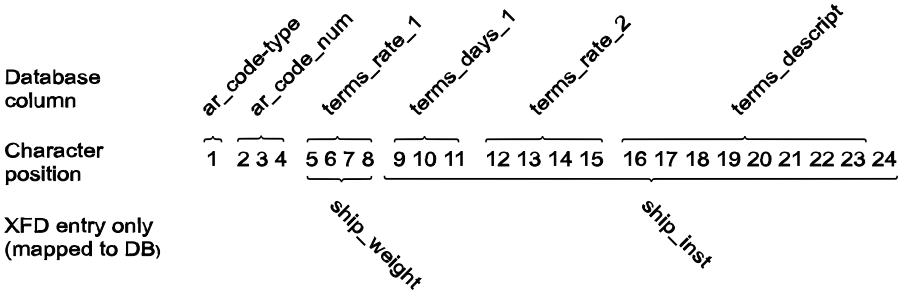
Some fields will not appear in the table, but the data dictionary will map them to the master field names. Acu4GL thus will eliminate redundancies and give you optimum performance.

```
01  ar-codes-record.
    03  ar-codes-key.
        05  ar-code-type       pic x.
        05  ar-code-num        pic 999.

01  ship-code-record.
    03  filler                 pic x(4).
    03  ship-weight            pic s999v9.
    03  ship-instruct          pic x(15).

01  terms-code-record.
    03  filler                 pic x(4).
    03  terms-rate-1           pic s9v999.
    03  terms-days-1           pic 9(3).
    03  terms-rate-2           pic s9v999.
    03  terms-descript         pic x(15).
```

The following diagram shows how Acu4GL creates database columns for some of the fields in the COBOL record, while the data dictionary maps other fields to those columns; this means that *all* the fields are accessible to the COBOL program.



See **section 3.2.2, "Defaults Used in XFD Files,"** for a description of the rules that Acu4GL follows as it builds the database table and an explanation of how you can override those rules when necessary.

3.2.2 Defaults Used in XFD Files

Several elements of COBOL require special handling when data dictionaries are built. These include multiple record definitions, REDEFINES, FILLER, and OCCUR. This section describes how the compiler handles each of these situations.

Note that in many instances you can override the default behavior described below by placing special comment lines in the FDs of your COBOL code. These comments are called *directives*, and are described in **Chapter 4, "Chapter 4: Using Directives."** For example, the WHEN directive allows you to use multiple definitions for a single set of data by specifying when each definition should be used.

Databases generally do not support the notion of multiple definitions for the same column. As the following paragraphs explain, whenever a COBOL program gives more than one definition for the same data, the compiler makes a choice about which definition to use in the dictionary. Then it disregards the rest.

KEY IS phrase

Fields named in KEY IS phrases of SELECT statements are included as column names. Other fields that occupy the same areas as the key fields (by either explicit or implicit redefinition) are not included by name, but are mapped to the key field column names by the data dictionary.

Remember, if the field named in the KEY IS phrase is a group item, it will not become a column in the database table unless a USE GROUP directive is used (see **section 3.2.1, "Understanding How the Database Table Is Formed"**). Instead, the subordinate fields of the group item will be used.

REDEFINES clause

Fields contained in a redefining item occupy the same positions as the fields being redefined. The compiler needs to select only one of the field definitions to use. The default rule that it follows is to use the fields in the item being redefined as column names; the data dictionary maps fields that appear subordinate to a REDEFINES clause to column names.

Multiple record definitions

This same rule extends to multiple record definitions. In COBOL, multiple record definitions are essentially redefinitions of the entire record area. This leads to the same complication that is encountered with REDEFINES: multiple definitions for the same data. So the compiler needs to select one definition to use.

Because the multiple record types can be different sizes, the compiler needs to use the largest one, so that it can cover all of the fields adequately. Thus, the compiler's rule is to use the fields in the largest record defined for the file. If more than one record is of the largest size, the compiler uses the first one.

Group items

Note that group items are, by default, never included in a data dictionary for the same reason that REDEFINES are excluded: they result in multiple names for the same data items. You can, however, choose to combine grouped fields into one data item by specifying the **USE GROUP** directive, described in Chapter 4.

FILLER data items

In a COBOL FD, FILLER data items are essentially placeholders. FILLER items are not uniquely named and thus cannot be uniquely referenced. For this reason, they are not placed into the ACUCOBOL-GT data dictionary. The dictionary maintains the correct mapping of the other fields, and no COBOL record positional information is lost.

Sometimes you need to include a FILLER data item, such as when it occurs as part of a key. In such a case, you could include it under a USE GROUP directive or give it a name of its own with the **NAME** directive, described in Chapter 4.

OCCURS clauses

An OCCURS clause always requires special handling, because the Acu4GL runtime system must assign a unique name to each database column. The runtime accomplishes this by appending sequential index numbers to the item named in the OCCURS clause.

For example, if the following were part of a file's description:

```
03  employee-table occurs 20 times.
    05  employee-number            pic 9(3)
```

these column names would be created in the database table:

```
employee_number_1
employee_number_2
.
.
.
employee_number_10
employee_number_11
.
.
.
employee_number_20
```

Note that the hyphens in the COBOL code are translated to underscores in database field names, and the index number is preceded by an underscore.

XFD location

If you are using object libraries, XFD files can be embedded into COBOL libraries through **cblutil**, and the runtime will find them. This is particularly effective when you are using the ACUCOBOL-GT Web Runtime and Acu4GL. It allows you to send all of the COBOL objects, as well as XFD files, in one package.

Please note that if an XFD file is located in an object library and is also located on the disk (in a directory pointed to by the **XFD_PREFIX** configuration variable), the XFD file in the object library is loaded first, unless the XFD_PREFIX configuration variable uses remote name notation.

3.2.3 Summary of Dictionary Fields

Fields defined with an OCCURS clause are assigned unique sequential names. Fields without names are disregarded.

When multiple fields occupy the same area, the compiler chooses only one of them unless you have a **WHEN** directive to distinguish them.

To choose:

- The compiler preserves fields mentioned in KEY IS phrases.
- It discards group items unless USE GROUP is specified.
- It discards REDEFINES.
- It uses the largest record if there are multiple record definitions.

3.2.4 Identical Field Names

In COBOL, you distinguish fields with identical names by qualification. For example, there are two fields named RATE in the following code, but they can be qualified by their group items. Thus, you would reference RATE OF TERMS-CODE and RATE OF AR-CODE in your program:

```
01  record-area.
    03  terms-code.
        05  rate                  pic s9v999.
        05  days                  pic 9(3).
        05  descript              pic x(15).
    03  ar-code.
        05  rate                  pic s9v999.
        05  days                  pic 9(3).
        05  descript              pic x(15).
```

However, database systems consider duplicate names an error. Thus, if more than one field in a particular file has the same name, the data dictionary will not be generated for that file.

The solution to this situation is to add a **NAME** directive (see Chapter 4) that associates an alternate name with one or both of the conflicting fields.

3.2.5 Long Field Names

To meet the SQL requirements of some RDBMSs, the Acu4GL runtime for those systems truncates long field names to the maximum allowed by the RDBMS. (In the case of the OCCURS clause described above, the truncation is to the original name, not the appended index numbers. However, the final

name, including the index number, is limited to the RDBMS maximum.) It's a good idea to make sure that your field names are unique (and meaningful) within the first 18 characters.

The database you are connecting to may allow longer names, and if so, more characters will be used. Names longer than 30 characters will generate a warning. Instead of allowing default truncation, you can use the **WHEN** directive to give a shorter alias to a field with a long name. Note that within the COBOL application you can continue to use the original name. The **NAME** directive affects only the data dictionary connection to the RDBMS table.

3.2.6 Naming the XFD File

The compiler needs to give a name to each XFD file (data dictionary) that is built. It attempts to build the name from your COBOL code, although there are some instances where the name in the code is nonspecific, and you must provide a name.

Each XFD name is built from a *starting name* that is derived (if possible) from the SELECT statement in your COBOL code. The following paragraphs explain how that occurs.

ASSIGN name is a variable

If the SELECT for the file has a variable ASSIGN name (ASSIGN TO *filename*), you must specify a starting name for the XFD file via a **FILE** directive in your code. This process is described in **Chapter 4, "Chapter 4: Using Directives."**

ASSIGN name is a constant

If the SELECT for the file has a constant ASSIGN name (such as ASSIGN TO "COMPFILE"), that name is used as the starting name for the XFD name.

ASSIGN name is generic

If the ASSIGN phrase refers to a generic device (such as ASSIGN TO "DISK"), the compiler uses the SELECT name as the starting name.

Forming the final XFD name

From the starting name, the final name is formed as follows:

- The compiler removes any extensions from the starting name.
- It constructs a "universal" base name by stripping out directory information that fits any of the formats used by the operating systems that run the ACUCOBOL-GT compiler.
- It reduces the base name to eight characters and converts it to lower case.
- It appends the letters ".xfd" to the base name.

Examples of XFD names

COBOL code	File name
`ASSIGN TO "usr/ar/customer.dat"`	customer.xfd
`SELECT TESTFILE, ASSIGN TO DISK`	testfile.xfd
`ASSIGN TO "-D SYS$LIB:HELP"`	help.xfd
`ASSIGN TO FILENAME`	(you specify)

Mapping other files to an XFD

At run time, it is possible to use a single XFD for files that have different names. For example, suppose a site has customer files that have identical structures but different names (such as CUST0001, CUST0002, and CUST0003). It's not necessary to have a separate XFD for each file, as long as their record definitions are the same.

The individual files can all be mapped to the same XFD via a runtime configuration variable called **XFD_MAP**. The following paragraphs describe how it works.

Suppose your COBOL application has a SELECT with a variable ASSIGN name, such as customer-file. This variable assumes different values (such as CUST0001 and CUST0002) during program execution.

Before compiling the application, you would use the **FILE** directive (see Chapter 4) to provide a base name for the XFD. Suppose you provide "CUST" as the base. The compiler would then generate an XFD named "cust.xfd". (The compiler always converts XFD names to lower case.)

To ensure that all customer files, each having a unique name, will use this same XFD, you make this entry in your runtime configuration file:

```
XFD_MAP    CUST* = CUST
```

The asterisk ("*") in the example is a wildcard that matches any number of characters. *Note that the extension ".xfd" should not be included in the map.* This statement would cause the XFD "cust.xfd" to be used for all files whose names begin with "CUST".

The XFD_MAP variable has this syntax:

```
XFD_MAP    [pattern = base-xfd-name]  ...
```

where *pattern* consists of any valid filename characters and may include "*" or "?". These two characters have special meanings in the pattern:

*	matches any number of characters
?	matches a single occurrence of any character

For example:

CUST????	matches CUST0001 and CUSTOMER; does not match CUST001 or CUST00001
CUST*	matches all of the above

CUST*1	matches CUST001 and CUST0001 and CUST00001; does not match CUSTOMER
*OMER	matches CUSTOMER; does not match CUST001 or CUST0001

The XFD_MAP variable is read during the open file stage of any Acu4GL products linked into the runtime.

XFD values can be replaced or be added to the end of existing XFD map values by setting the **XFD_MAP_RESETS** configuration variable. This variable determines whether setting the XFD_MAP adds to or replaces the existing value. When this variable is set to "0" (off, false, no), setting the XFD_MAP adds new value patterns to the end of the existing value. When it is set to "1" (on, true, yes), setting the XFD_MAP replaces the existing value with a new value. The default value is "1" (on, true, yes).

This variable may be useful if you need to include multiple XFD_MAP lines in a configuration file and want to avoid setting and resetting the variable. When multiple lines exist, all patterns are used in the order they are listed.

4 Using Directives

Key Topics

4.1 What Are Directives?

ACUCOBOL-GT® data dictionaries are based on your COBOL FDs. For this reason we call them extended FDs or XFDs. Each dictionary describes all of the fields for one file.

Directives are optional comments that you can use in your FDs to control how things look on the database side. Many applications won't use directives at all. They're most commonly used when a site intends to do a lot of work with the database management system outside of the COBOL application, and wants to control how the database table is built.

Directives are special comments placed into an FD in your COBOL source code. They guide the building of the data dictionaries, which in turn guide the building of the database table.

Each directive includes the letters "XFD". These three letters indicate to the compiler that the comment is to be used in data dictionary generation.

Directives offer you a great deal of control over how the database table is built. Among other things, they enable you to

- specify a column name to be used in the database table, in place of a COBOL field name
- map elementary items of a group item together into a single column
- cause numeric COBOL data to be treated as a text string in the database
- cause the fields from a specific record in a file to appear in the database table (rather than just the fields from the largest record)
- assign a column the DATE type, so that it will have the built-in functionality that dates have in the RDBMS
- give a name to the data dictionary file itself

Directives are always placed within a COBOL FD. They do *not* affect Procedure Division I/O statements, and they do *not* change your COBOL fields in any way. Rather, they guide the building of the data dictionaries, giving you a measure of control over the way COBOL data is *mapped* to database fields.

Data dictionaries may be built directly from your source code with no directives if the default mapping rules described in **Chapter 3, "Chapter 3: Data Dictionaries,"** and **Chapter 7, "Chapter 7: New and Existing Databases,"** are sufficient for your situation. If you would like to override the default mapping behavior or map a field to a different name, you must add directives to your COBOL code.

4.2 Directive Syntax

Place each directive syntax on a line by itself, immediately before the COBOL line to which it pertains.

Introduce each directive with a "$" in the Indicator Area (column 7 in standard ANSI source format), followed immediately by the letters "XFD", and then the directive itself. There should be no space between the $ and the XFD. Spaces are permitted elsewhere on the line as separators. For example, the NAME directive looks like this:

```
$XFD  NAME = EMP-NUMBER
```

An alternate ANSI-compliant way to introduce a directive is with an asterisk ("*****") in the Indicator Area. In this case, you begin the directive with the letters "XFD" and enclose the entire comment in double parentheses. For example:

```
*(( XFD  NAME = EMP-NUMBER ))
```

There should be no space between the asterisk and the double left parentheses. Spaces are permitted elsewhere on the line as separators.

You may use either form of the directive syntax (or both) in your applications. In addition, note that directives are case-insensitive and that hyphens and underscores are considered equivalent. Thus, for example,

```
$XFD VAR_LENGTH
$XFD VAR-LENGTH
$xfd var_length
$xfd var-length
```

are all recognized as equivalent by the Acu4GL® products.

Two or more directives that pertain to the same line of COBOL code may be combined on one comment line. The directives should be separated by a space or a comma. For example, to specify both USE GROUP and NUMERIC at the same time, you would add this line:

```
$XFD  USE GROUP, NUMERIC
```

The following pages describe each of the directives, in alphabetical order.

4.3 Directives Supported by the Acu4GL Interfaces

The Acu4GL interfaces support the following directives:

ALPHA
BINARY
COMMENT
COBOL-TRIGGER
DATE
FILE
NAME
NUMERIC
SECONDARY_TABLE
USE GROUP
VAR_LENGTH
WHEN
XSL

4.3.1 ALPHA

The ALPHA directive allows you to treat a data item as alphanumeric text in the database, when it is declared as numeric in the COBOL program.

Syntax

```
$XFD ALPHA
```

This is especially useful when you have numeric keys in which you occasionally store non-numeric data, such as LOW-VALUES or special codes. In this situation, treating the field as alphanumeric allows you to move any kind of data to it.

Example

Suppose you have specified KEY IS code-key. Then assume the following record definition:

```
01  code-record.
    03  code-key.
        05  code-num      pic 9(5).
```

In the database, group items are disregarded, so CODE-NUM is the actual key field. Suppose you needed to move a non-numeric value to the key:

```
MOVE "C0531" TO CODE-KEY.
WRITE CODE-RECORD.
```

In this case the results are not well-defined because a non-numeric value has been moved into a numeric field. The database might very well reject the record.

One way to solve this problem is to use the ALPHA directive. This causes the corresponding database field to accept alphanumeric data:

```
 01  code-record.
     03  code-key.
$XFD ALPHA
         05  code-num      pic 9(5).
```

As an alternative, you could specify the **USE GROUP** directive on the line before *code-key*. The USE GROUP directive implies that the field is alphanumeric.

4.3.2 BINARY

The BINARY directive is used to specify that the data in the field could be alphanumeric data of any classification. Absolutely any data is allowed.

The BINARY directive may not be used in combination with the **VAR_LENGTH** directive.

The method of storing fields declared as binary is database-specific. For example, with Informix databases, binary data is stored in char fields with an extra leading character. This character always contains a space. Oracle uses the data type raw for the field.

Syntax

```
$XFD BINARY
```

Example

You might use this directive when you need to store a key that contains LOW-VALUES; COBOL allows a numeric field to contain LOW or HIGH values, but these are invalid for a numeric field in the RDBMS:

```
 01  code-record.
     03  code-key.
         05  code-indic          pic x.
$XFD BINARY
         05  code-num            pic 9(5).
         05  code-suffix         pic x(3).
             .
             .
             .
 move low-values to code-num.
```

4.3.3 COBOL-TRIGGER

There is an XFD directive that tells Acu4GL that a COBOL trigger is to be executed.

Note: This directive must come immediately before the FD of the file for which the trigger is defined. It uses the standard XFD directive syntax [either $ or *((...))].

Syntax

```
XFD COBOL-TRIGGER=program-name
```

The following three parameters are passed to the COBOL program:

```
01  OPCODE.
    88  READ-BEFORE     VALUE "r".
    88  READ-AFTER      VALUE "R".
    88  WRITE-BEFORE    VALUE "w".
    88  WRITE-AFTER     VALUE "W".
    88  REWRITE-BEFORE  VALUE "u".
    88  REWRITE-AFTER   VALUE "U".
    88  DELETE-BEFORE   VALUE "d".
    88  DELETE-AFTER    VALUE "D".

01  FILE-RECORD        PIC X(MAX-RECORD-SIZE).

01  ERROR-CODE         PIC 99.
```

You can use C$PARAMSIZE to determine the size of the record passed, in case you have variable-length records. (See Appendix I, "Library Routines," in Book 4 of the ACUCOBOL-GT documentation set for more information on C$PARAMSIZE.)

- In cases where the record size *is* known, the actual record is passed [READ-AFTER (all), WRITE-BEFORE, WRITE-AFTER, REWRITE-BEFORE, REWRITE-AFTER].
- In cases where the record size is *not* known, the maximum record size is passed [READ-BEFORE (all), DELETE-BEFORE, DELETE-AFTER].

The contents of this field will be identical to the record area given to the file operation. The BEFORE images will have the value before the file operation, and the AFTER images will have the value after the file operation.

Use the ERROR-CODE parameter to signal to Acu4GL that an error occurred:

- If you set this parameter in a BEFORE trigger, the file operation will not execute.

- If you set this parameter in an AFTER trigger, the runtime will treat the file operation as an error, though in fact the file operation did execute. This means that, for sequential access (next or previous), the file position has changed and a subsequent next or previous will act as if the prior file operation succeeded.

Also, if you set an error in the REWRITE-AFTER or DELETE-AFTER, the record will not return to its prior state—the record will be modified or deleted in the database. We suggest that you use transactions if this behavior is not acceptable.

Please note that if the COBOL program cannot be called (for any reason), it is treated as having succeeded.

4.3.4 COMMENT

The COMMENT directive allows you to include comments in an XFD file. Because the information is embedded in a comment, it does not interfere with processing by Acu4GL products. Each comment entered with this directive will appear in the XFD file with the symbol "#" in column one.

Syntax

```
$XFD COMMENT text
```

4.3.5 DATE

The DATE directive's purpose is to store a field in the database as a date. Because there's no COBOL syntax that identifies a field as a date, you may want to add this directive to differentiate dates from other numbers, so that they enjoy the properties associated with dates in the RDBMS. See the description of the configuration variable **4GL_2000_CUTOFF** in section 8.2, "Runtime Configuration Variables," to view special options for two-digit dates.

Syntax

```
$XFD DATE=date-format-string
```

This directive implies the NUMERIC directive.

If no *date-format-string* is specified, six-digit (or six-character) fields are retrieved as YYMMDD from the database. Eight-digit fields are retrieved as YYYYMMDD.

The *date-format-string* is a description of the desired date format, composed of characters from the following list:

M Month (01–12)

Y Year (two or four digits)

D Day of month (01–31)

J Julian day (00000000–99999999)

E Day of year (001–366)

H Hour (00–23)

N Minute

S Second

T Hundredths of a second

Any other characters cause the date format string to be invalid, and result in a corrupt XFD error or a compile-time warning.

Each character in a date format string can be considered a place holder that represents the type of information stored at that location. The characters also determine how many digits will be used for each type of data.

For example, although you would typically represent the month with two digits, if you specify MMM as part of your date format, the resulting date will use three digits for the month, left-zero-filling the value. If the month is given as M, the resulting date will use a single digit and will truncate on the left.

Julian dates

Because the definition of Julian day varies, the DATE directive offers a great deal of flexibility for representing Julian dates. Many source books define the Julian day as the day of the year, with January 1st being 001, January 2nd being 002, and so forth. If you want to use this definition for Julian day, simply use "EEE" (day of year) in your date formats.

Other reference books define the Julian day as the number of days since some specific base date in the past. This definition is represented in the DATE directive with the letter "J" (for example, a six-digit date field would be preceded with the directive $XFD DATE=JJJJJJ). The default base date for this form of Julian date is 12/31/4714 BC.

You may define your own base date for Julian date calculations by setting the runtime configuration variable **4GL_JULIAN_BASE_DATE**.

Handling invalid dates

Acu4GL considers dates in the following range to be valid:

01/01/0001 to 12/31/9999

If Acu4GL considers a date valid and passes it to an RDBMS that considers it invalid, the results depend on the particular RDBMS. Some systems return an error code. Some store the date as NULL and do not return an error. Some store the date in an unpredictable fashion and do not return an error. We recommend that you determine the error handling procedures of the RDBMS in use.

If a COBOL program attempts to write a record containing a date that Acu4GL knows is invalid, Acu4GL inserts NULLs into the date field and writes the record.

If a COBOL program attempts to read into a record from a table with a NULL date field, zeroes are inserted into that field in the COBOL record.

For date fields having two-digit years, by default years 0 through 19 are inserted as 2000 through 2019, and years 20 through 99 are inserted as 1920 through 1999. You can change this behavior by changing the value of the variable **4GL_2000_CUTOFF**.

Note: If a field is used as part of a key, the field cannot be a NULL value. For information about converting date values using the **4GL_CONVERT_DATE_ZERO** variable, see section 8.2, "Runtime Configuration Variables."

Using group items

You may place the DATE directive in front of a group item, as long as you also use the **USE GROUP** directive.

Example 1

```
$xfd date
     03  date-hired    pic 9(8).
     03  pay-scale     pic x(3).
```

The column date-hired will have eight digits and will be type DATE in the database, with a format of YYYYMMDD.

Example 2

```
$XFD DATE, USE GROUP
     03  date-hired.
         05  yyyy      pic 9(4).
         05  mm        pic 9(2).
         05  dd        pic 9(2).
```

This also will produce a column named date-hired with eight digits and type DATE in the database, format YYYYMMDD.

Example 3

```
$XFD DATE=EEEYYYY
     03  date-sold      pic 9(7).
     03  sales-rep      pic x(30).
```

This will produce a column named date-sold with seven digits and type DATE in the database. The date will contain the day of the year (for example, February 1st would be 032), followed by the four-digit year, such as 1999).

4.3.6 FILE

The FILE directive supplies a *starting name* from which the data dictionary file name is formed. This directive is required only when the file name in the COBOL code is nonspecific. For example, use this directive when the SELECT for the file has a variable ASSIGN name (ASSIGN TO *variable_name)*. In this case, Acu4GL cannot form a file name automatically, and you must provide a name. The *starting name* serves as the basis for the dictionary name.

Syntax

```
$XFD FILE=name
```

This directive must appear on the line immediately preceding the file's FD.

Example

Suppose your SELECT statement has a variable ASSIGN name such as the one shown here:

```
select my-file
assign to my-variable.
```

Then you would need to add the FILE directive as shown here:

```
 select my-file
 assign to my-variable.
    .
    .
    .
$xfd  file=payroll
 fd my-file
```

Note that this directive must appear immediately before the file's FD.

See also

XFD_MAP configuration variable

4.3.7 NAME

The NAME directive assigns a database field name to the field defined on the next line.

Syntax

```
$XFD NAME=fieldname
```

This directive has several uses, as shown in the following examples.

Example 1 – non-unique field names

Within a database file, all field names must be unique. (Multiple database tables may include the same field name, but duplicates may not exist within a single table.) Unique field names are not required in COBOL, because names can be qualified by group items.

For example, this is acceptable in COBOL:

```
01  employee-record.
    03  date-hired.
        05  yy      pic 99.
        05  mm      pic 99.
        05  dd      pic 99.
    03  date-last-paid.
        05  yy      pic 99.
        05  mm      pic 99.
        05  dd      pic 99.
```

You need not change the field names in your COBOL program to access a database. Instead, you use the NAME directive to provide unique database names for the fields. For example:

```
 01  employee-record.
     03  date-hired.
         05  yy      pic 99.
         05  mm      pic 99.
         05  dd      pic 99.
     03      date-last-paid.
$xfd  name=year-paid
```

```
          05  yy      pic 99.
 $xfd  name=month-paid
          05  mm      pic 99.
 $xfd  name=day-paid
          05  dd      pic 99.
```

The dates portion of the resulting database table will look like this:

yy	mm	dd	year_paid	month_paid	day_paid
88	02	15	94	04	30

Example 2 – names not unique within first 18 characters or longer than 18 characters

Some SQL-based databases require that names be unique within 18 characters, and some require that names be no longer than 18 characters. For those systems, the Acu4GL runtime will automatically truncate longer COBOL names after the first 18 characters.

For names that are identical within the first 18 characters, or are not meaningful if shortened to the first 18 characters, use the NAME directive to assign them different database field names.

Suppose you had:

```
01  acme-employee-record.
    03  acme-employee-record-date-hired          pic x(6).
    03  acme-employee-record-date-last-paid      pic x(6).
```

You could add two NAME directives to differentiate the two item names by making them meaningful within 18 characters:

```
  01  acme-employee-record.
 $xfd name=date-hired
      03  acme-employee-record-date-hired        pic x(6).
 $xfd name=date-last-paid
      03  acme-employee-record-date-last-paid    pic x(6).
```

Note that your COBOL names have not changed. The new names are used only for the database fields.

Each time you compile your program and specify "–Fx" to create data dictionaries, any field names longer than 18 characters will be checked for uniqueness within the first 18. If any field names are identical for the first 18 characters, a compiler warning message will be issued. A warning of this type does not prevent the program from compiling, and the XFD is generated.

Example 3 – assigning shorter names

You may want to use the NAME directive to assign shorter names than those used in your COBOL programs. This makes the formation of interactive SQL queries easier and quicker. For example:

```
$XFD  NAME=EMPNO
     03  employee-number        pic x(8).
```

This directive causes the data dictionary to map EMPLOYEE-NUMBER to EMPNO in the database.

Example 4 – matching field names and COBOL FDs

If your database already exists, and a field name in the database does not match the name used in your COBOL FD, you can use a NAME directive to associate the two names. For example:

```
$xfd name=employee-no
     03  employee-number        pic x(8).
```

This directive causes the data dictionary to map EMPLOYEE-NUMBER in the COBOL program to EMPLOYEE-NO in the database.

Example 5 – renaming fields that begin with a numeric character

If your COBOL program uses field names that begin with a numeric character, use the NAME directive to assign a different name for use with your database. SQL will typically generate a syntax error when it encounters a column name that begins with a numeric character. For example:

```
    03  12-months-sales      pic 9(5)V99.
```

could be renamed this way:

```
$xfd  name=twelve-months-sales
```

```
03  12-months-sales    pic 9(5)V99.
```

4.3.8 NUMERIC

The NUMERIC directive allows you to treat a data item as an unsigned integer when it is declared as alphanumeric. You might use this when the data stored in the item is always numeric.

Syntax

```
$XFD NUMERIC
```

Example

```
$xfd numeric
 01  student-code      pic x(7).
```

4.3.9 SECONDARY_TABLE

Some RDBMSs, such as Sybase, permit subordinate tables. When this is the case, you can use the SECONDARY_TABLE directive to indicate that the next data item may be placed into a subordinate table if more than one table is necessary to accommodate the data.

Up to 26 subordinate tables can be created from a single record description. Each table name is based on the original table name, with a letter from A to Z appended. For example, if the original table were named my-table, subsequent subordinate tables would be given these names:

my-tableA
my-tableB
my-tableC
my-tableD

When the runtime accesses the data dictionary, it makes an initial pass through the data, taking all eligible data items for which the SECONDARY_TABLE directive is *not* specified.

SECONDARY_TABLE is ignored for certain data items. These include:

- any field that is part of a key
- any field that is an indicator for a WHEN directive

If the table size is not exceeded in the first pass, then, in a second pass, the runtime appends to the original table all items marked SECONDARY_TABLE that can be accommodated.

When the first table reaches a limit (either in total number of columns or total number of characters), that table is created, and a new table is begun. Items that did not fit into the previous table are placed into a subordinate table, in the order in which they are encountered.

The process repeats until all items have been accommodated.

It is permissible to place the SECONDARY_TABLE directive just before a level 01 record definition. In this case, it applies to all fields underneath the level 01.

Syntax

```
$XFD SECONDARY_TABLE
```

Example

```
 $xfd secondary_table
  01  description      pic x(80).
```

4.3.10 USE GROUP

The USE GROUP directive allows you to enter a group item into the database as a single field, instead of using the elements contained in the group. This is necessary if the item is stored in an existing database as a group, rather than as individual fields.

Combining fields of data into meaningful groups for database storage also improves I/O efficiency.

Syntax

```
$XFD USE GROUP
```

By default, the USE GROUP directive implies that the consolidated field is alphanumeric. If you want a numeric field, simply add the word **NUMERIC** at the end of the directive.

Example 1 – combining DATE and USE GROUP directives

For example, the directive in the following code combines the functions of the USE GROUP and **DATE** directives, and indicates that the date should be entered into the database as a single date-formatted data item instead of three distinct fields:

```
$XFD  USE GROUP, DATE
      03  date-entered.
          05  yy      pic 99.
          05  mm      pic 99.
          05  dd      pic 99.
```

Either a comma or a space (or both) may separate the word "DATE" from the words "USE GROUP".

Example 2 – forming larger units

Other fields with which you might use this directive include multi-part account numbers or department numbers, or keys that are referenced as a unit but not by their individual pieces. Here's an example of an item that might be grouped:

```
$xfd  use group
 01  gl-acct-no.
     03  main-acct     pic 9(4).
     03  sub-acct      pic 9(3).
     03  dept-no       pic 9(3).
```

If you are using an existing database in which certain fields are grouped, they must also be grouped in your COBOL FD.

If the database does not yet exist, keep in mind that combining fields into groups can improve execution speed. Whether to group fields or not also depends on whether they are always stored and used together. Someone who really knows how the data is being used might help to identify groups of fields that can be combined to speed processing.

4.3.11 VAR_LENGTH

By default, the compiler generates fixed-length fields in the XFD. The VAR_LENGTH directive requests that the data item that immediately follows the directive be assigned a type that implies variable length, if possible. This can save considerable space in your database.

The VAR_LENGTH directive cannot be used in combination with the **BINARY** directive.

The precise variable type that is assigned to the data item depends on which RDBMS is in use. Possible variable types that might be assigned are VARCHAR and VARBINARY.

Syntax

```
$XFD VAR_LENGTH
```

Example

For example, the directive in the following code indicates that the employee-name field should be entered into a SYBASE database as a VARCHAR data item.

```
$XFD  VAR_LENGTH
     03  employee-notes      pic x(300).
```

4.3.12 WHEN

The WHEN directive is used when you want to include multiple record definitions or REDEFINES in a database table. It's typically used when you want to force certain columns to be built that wouldn't be built by default (because you want to use them from the RDBMS side).

Recall that the key fields and the fields from the *largest* record are automatically included as explicit columns in the database table. *All fields* are stored and retrieved from the database, whether they appear as explicit columns or not. So you don't need to use WHEN unless you want to ensure that some additional fields are *explicitly listed by name* in the table.

WHEN declares that the field (or subordinate fields, if it is a group item) that immediately follow the directive *must* appear as a column (or columns) in the database table. It also states one condition under which the columns are to be used. The WHEN directive guarantees that the fields will be explicitly included in the table, as long as they aren't FILLER and don't occupy the same area as key fields.

The WHEN directive results in a table that unites all fields subordinate to the WHEN directive. This table might include more columns and could affect performance and storage requirements.

Syntax

```
$XFD WHEN field = value      (equals)
$XFD WHEN field <= value     (is less than or equals)
$XFD WHEN field < value      (is less than)
$XFD WHEN field >= value     (is greater than or equals)
$XFD WHEN field > value      (is greater than)
$XFD WHEN field != value     (is not equal to)
$XFD WHEN field = OTHER
```

or

```
$XFD WHEN field = value  (also <, <=, >, >=, !=)
```

The *value* may be an explicit data value (in quotes). The word OTHER can be used only with "=". It means use the following field(s) only if the WHEN condition(s) listed at the level of this field are not met.

Note: WHEN values must be on individual lines as single values in the code. WHEN values cannot be combined.

For example:

```
      03  ar-code-type                  pic x.
 $xfd  when ar-code-type = "s"
      03  ship-code-record              pic x(4).
 $xfd  when ar-code-type = "b"
      03  backorder-code-record redefines ship-code-record.
 $xfd  when ar-code-type = other
      03  obsolete-code-record redefines ship-code-record.
```

OTHER may be used before one record definition, and may be used once at *each level* within each record definition.

Note: A WHEN directive with condition OTHER *must* be used if there is a possibility that the data in the field will not meet any of the explicit conditions specified in the other WHEN directives. If this is not done, results are undefined.

Example 1

If the following code were compiled without directives, the underlined fields would appear explicitly in the database table. Note that the key fields would be included automatically, as would the fields from the largest record. FILLER would be ignored:

```
01  ar-codes-record.
    03  ar-codes-key.
        05  ar-code-type                pic x.
        05  ar-code-num                 pic 999.
01  ship-code-record.
    03  filler                          pic x(4).
    03  ship-instruct                   pic x(15).
01  terms-code-record.
    03  filler                          pic x(4).
    03  terms-rate-1                    pic s9v999.
    03  terms-days-1                    pic 9(3).
    03  terms-rate-2                    pic s9v999.
    03  terms-descript                  pic x(15).
```

If you add the WHEN directive as shown below, it causes the fields from the SHIP-CODE-RECORD to be included in the database table, and determines when specific database columns are used. The underlined fields then appear as columns in the database table:

```
 01  ar-codes-record.
     03  ar-codes-key.
         05  ar-code-type                pic x.
         05  ar-code-num                 pic 999.
$xfd  when ar-code-type = "s"
 01  ship-code-record.
     03  filler                          pic x(4).
     03  ship-instruct                   pic x(15).
$xfd  when ar-code-type = "t"
```

```
01  terms-code-record.
    03  filler                          pic x(4).
    03  terms-rate-1                    pic s9v999.
    03  terms-days-1                    pic 9(3).
    03  terms-rate-2                    pic s9v999.
    03  terms-descript                  pic x(15).
```

FILLER data items don't have unique names and are thus not used to form columns in the database table. You could use the NAME directive to give them a name if you really need to see them in the database table.

Note: In this example, the FILLER data items implicitly redefine key fields. Thus, they would be disregarded *even if you provided a name for them*.

Example 2

In the following code, in which no WHEN directives are used, the underlined fields will be explicitly named in the database table. (Key fields have the suffix "key" in their names in this example.)

Note that REDEFINES records simply re-map the same data area and are not explicitly included in the database table by default:

```
01  archive-record.
    03  filler                        pic x(33).
    03  archive-code                  pic 9(6).
    03  archive-location              pic 9(2).
    03  filler                        pic x(10).
01  master-record.
    03  animal-id-key.
        05  patient-id                pic 9(6).
        05  species-code-type         pic 9(5).
        05  species-name              pic x(6).
    03  service-code-key.
        05  service-code-type         pic 9(6).
        05  service-name              pic x(10).
    03  billing-code.
        05  billing-code-type         pic 9(4).
        05  plan-name                 pic x(8).
    03  office-info.
        05  date-in-office            pic 9(8).
```

```
        05  served-by-name             pic x(10).
    03  remote-info redefines office-info.
        05  van-id                     pic 9(4).
        05  proc-code                  pic 9(8).
        05  vet-name                   pic x(6).
```

If you add the WHEN directives shown below, you add several columns to the database table. The fields that then appear in the table are underlined:

```
$xfd when animal-id-key = "00000000000000000"
 01  archive-record.
     03  filler                         pic x(33).
     03  archive-code                   pic 9(6).
     03  archive-location               pic 9(2).
     03  filler                         pic x(10).
$xfd  when animal-id-key =  other
 01  master-record.
$xfd  use group
     03  animal-id-key.
         05  patient-id                 pic 9(6).
         05  species-code-type          pic 9(5).
         05  species-name               pic x(6).
     03  service-code-key.
         05  service-code-type          pic 9(6).
         05  service-name               pic x(10).
     03  billing-code.
         05  billing-code-type          pic 9(4).
         05  plan-name                  pic x(8).
$xfd when billing-code-type = "1440"
     03  office-info.
         05  date-in-office             pic 9(8).
         05  served-by-name             pic x(10).
$xfd when billing-code-type = other
     03  remote-info redefines office-info.
         05  van-id                     pic 9(4).
         05  proc-code                  pic 9(8).
         05  vet-name                   pic x(6).
```

Example 3

If your application has a REDEFINES whose field names are more meaningful than the fields they redefine, you might consider switching the order of your code, rather than using a WHEN directive. Use the less significant field names in the REDEFINES.

For example, you might change this:

```
03  code-info.
    05  filler                  pic 9(8).
    05  code-1                  pic x(10).
03  patient-info redefines code-info.
    05  patient-id              pic 9(4).
    05  service-code            pic 9(8).
    05  server-name             pic x(6).
```

to this:

```
03  patient-info.
    05  patient-id              pic 9(4).
    05  service-code            pic 9(8).
    05  server-name             pic x(6).
03  code-info redefines patient-info.
    05  filler                  pic 9(8).
    05  code-1                  pic x(10).
```

The fields that would appear in the database table by default are underlined above. This shows how the column names might become more meaningful when the order is reversed. Your application operates the same either way.

Note: When a WHEN directive condition is met, COBOL record definitions or REDEFINES records that are subordinate to other WHEN directives are not used. Database columns in rows that correspond to those records are set to the special database value NULL. This means that there is no value provided for those columns. NULL is not equivalent to zero, and it has special properties in the RDBMS. For example, you can select all rows for which a given column is NULL.

Example 4

This COBOL code:

```
 01  col-type                        pic x.
$xfd  when col-type = "a"
     03  def1                       pic x(2).
$xfd  when col-type = "b"
     03  def2 redefines def1        pic 9(2).
```

results in this database table:

col_type	def1	def2
a	Xx	<Null>
b	<Null>	10

4.3.13 XSL

If you are using the "-Fe" compiler option to generate XML-style XFD files, the XSL directive allows you to include a stylesheet reference in the XML header.

Syntax

```
$XFD XSL=stylesheet
```

where *stylesheet* is an alphanumeric literal indicating the appropriate stylesheet. The compiler includes the following line in all generated XML-style XFD files:

```
<?xml-stylesheet type="text/xsl" href="stylesheet"?>
```

For example:

```
$XFD XSL="myxsl.xsl"
```

generates this line:

```
<?xml-stylesheet type="text/xsl" href="myxsl.xsl"?>
```

5 Invalid Data

Key Topics

5.1 Illegal COBOL Data

This chapter lists which COBOL data items are considered invalid and explains what happens to them when they are stored in the database.

We also explain which database items are considered invalid and explain how these are translated to COBOL.

COBOL data (except key data) that is illegal by database standards is stored in the database as null unless you specifically override this default. (Use the configuration variable **4GL_ILLEGAL_DATA** to override.)

The following COBOL data usages are considered illegal by the database:

- In USAGE DISPLAY numbers, LOW-VALUES , HIGH-VALUES, and SPACES are all illegal.
- In COMP-2 numbers, HIGH-VALUES and SPACES are illegal.
- In COMP-3 numbers, HIGH-VALUES is illegal.
- All other numeric types fit one of the preceding cases (depending on their internal storage format).
- In DATE fields, the value “zero” is illegal, as are any other illegal conditions as defined by the date’s USAGE type (see above).
- In text fields, LOW-VALUES is illegal.
- BINARY numbers are always legal, and all values are legal in BINARY text fields.

5.1.1 Invalid Key Data

Any field that is part of a key is never stored as null. (A null key does not have a well-defined position in relation to other keys.) If a key field has an illegal value, the field is stored in the database as described here:

Illegal LOW-VALUES	minimum possible value (0 or -99999...) minimum possible date value (01/01/01)
Illegal HIGH-VALUES	maximum possible value (99999...) maximum possible date value (database dependent)
Illegal SPACES	zero (or 01/01/01 for a date field)

5.1.2 Invalid Data Other Than Keys

You have the option to convert illegal data, other than keys, into legal values by setting the configuration variable **4GL_ILLEGAL_DATA** to "Converted". When this is set, the Acu4GL® interface converts data the same way it does for keys.

4GL_ILLEGAL_DATA can also be set to "Null" to indicate that illegal COBOL data other than keys should be converted to null. This is the default behavior.

5.2 Invalid Database Data

When Acu4GL receives a null field from the database, it translates it to COBOL as follows:

Numbers (including dates): zero
Text (including binary): spaces

See the **DATE** directive in Chapter 4 for additional information about the handling of invalid dates.

6 Working with COBOL

Key Topics

6.1 Preparing and Compiling Your COBOL Program

After the environment has been set up for the Acu4GL® interface, you are ready to use the system. The following example illustrates how to set up a COBOL program to use your Acu4GL RDBMS product.

Note: If you are not familiar with the XFD directives described in **Chapter 4, "Chapter 4: Using Directives,"** you may want to read that chapter before continuing with this section.

The purchase-orders file from a COBOL program at an imaginary company will be stored in the database. This file contains the records that handle all of the information from the company's purchase orders.

Within the purchase-orders file, are two record types:

- the purchase-order header record. (There is one of these for each purchase-order form.)
- the purchase-order detail record. (There is one detail record for each line item in a purchase-order.)

The file is keyed off the *purchase-order-number*. We will build and examine the database table three times, to illustrate three different approaches to using the COBOL file description:

- First, no directives will be added to the COBOL code. The code will be compiled as is. Only the largest record will be included in the database table.
- Second, the WHEN directive will be added, and the program recompiled. This will cause all record formats to be included in the database.
- Finally, several fields will be grouped together and renamed as a matter of convenience.

Here's the sample code:

```
IDENTIFICATION DIVISION.
program-id.  purchase.
```

```
ENVIRONMENT DIVISION.
input-output section.
file-control.

select p-o-file
assign to disk "purch1"
organization is indexed
access mode is dynamic
record key is p-o-number
file status is p-o-status.

DATA DIVISION.
file section.
fd p-o-file.
01 p-o-record.
   03 p-o-division-number                pic 9(3).
   03 p-o-record-type                    pic x.
      88 header-record                   value "h".
      88 detail-record                   value "d".
   03 p-o-number                         pic 9(10).
   03 p-o-number-detail redefines
         p-o-number.
      05 picking-ticket-number           pic 9(6).
      05 shipping-region                 pic 9(2).
      05 p-o-customer-type               pic 9(2).
      05 p-o-customer-breakdown redefines
            p-o-customer-type.
         07 customer-category            pic x.
         88 p-o-customer-retail          value "r".
         88 p-o-customer-whlsale         value "w".
         07 customer-pay-format          pic x.
            88 is-net-30                 value "3".
            88 is-net-10                 value "1".
   03 p-o-date.
      05 p-o-yy                          pic 9(2).
      05 p-o-mm                          pic 9(2).
      05 p-o-dd                          pic 9(2).

01 p-o-detail-record.
   03 p-o-dept-number                    pic 9(3).
   03 p-o-record-type                    pic x.
   03 detail-p-o-number                  pic 9(10).
   03 p-o-shipping-info.
      05 p-o-quantity-to-ship            pic s9(4) comp.
      05 p-o-total-quantity              pic s9(4) comp.
```

```
   03 p-o-notes.
      05 notes-line occurs 3 times       pic x(40).
working-storage section.
01 p-o-status                            pic x(2).

PROCEDURE DIVISION.
level-1 section.
main-logic.

open output p-o-file.
close p-o-file.
stop run.
```

6.1.1 Compiling With No Directives

You can compile the preceding program *as is* with the "-Fx" option to generate the data dictionary. The compiled program will run and will build the database table shown at the end of this section.

Here's how the database table is built. First, any fields listed in the KEY IS clause of the SELECT are included (**p-o-number** in this example). Then the compiler takes the largest record (**p-o-detail-record**) and lists the fields that make up that record. The next two pages show the specific fields that are placed into the table.

All of the data from the COBOL program is stored in and retrieved from the database, even though not all fields are explicitly named in the database table. See **Chapter 3, "Chapter 3: Data Dictionaries,"** for a description of how this works.

The underlined fields are the only ones that will be entered into the table:

```
fd p-o-file.
01  p-o-record.
    03  p-o-division-number                  pic 9(3).
    03  p-o-record-type                      pic x.
        88  header-record                    value "h".
        88  detail-record                    value "d".
    03  p-o-number                           pic 9(10).
    03  p-o-number-detail redefines p-o-number.
        05  picking-ticket-number            pic 9(6).
        05  shipping-region                  pic 9(2).
```

```
           05  p-o-customer-type                pic 9(2).
           05  p-o-customer-breakdown redefines
                 p-o-customer-type.
               07  customer-category            pic x.
                   88  p-o-customer-retail      value "r".
                   88  p-o-customer-whlsale     value "w".
               07  customer-pay-format          pic x.
                   88 is-net-30                 value "3".
                   88 is-net-10                 value "1".
       03  p-o-date.
           05  p-o-yy                           pic 9(2).
           05  p-o-mm                           pic 9(2).
           05  p-o-dd                           pic 9(2).

   01  p-o-detail-record.
       03  p-o-dept-number                      pic 9(3).
       03  p-o-record-type                      pic x.
       03  detail-p-o-number                    pic 9(10).
       03  p-o-shipping-info.
           05  p-o-quantity-to-ship             pic s9(4) comp.
           05  p-o-total-quantity               pic s9(4) comp.
       03  p-o-notes.
           05  notes-line occurs 3 times        pic x(40).
```

As the table is built:

- Any hyphens in the COBOL field names are converted to underscores.
- Field names longer than 18 characters will cause a warning to be issued if they are not unique within the first 18 characters. Some databases, Informix for example, limit you to 18 total characters. We recommend that for portability and for end-user purposes you use the least number of characters. You can use the **NAME** directive for this to avoid changing your procedure division code.
- Fields in OCCURS clauses get special handling, because the runtime system must assign a unique name to each data item. So sequential index numbers are appended to the item named in the OCCURS. Limits vary between databases, so refer to the documentation that accompanied your RDBMS. In some instances, the name may be truncated by Acu4GL if necessary before the index is added.

Note: detail-p-o-number is not part of the table, because the key overlays this area.

This table is built in the database:

Column Name	Type
p_o_number	number(10) *
p_o_dept_number	number(3) *
p_o_record_type	char(1)
p_o_quantity_to_ship	number(4) *
p_o_total_quantity	number(4) *
notes_line_1	char(40)
notes_line_2	char(40)
notes_line_3	char(40)

* The actual database datatype may vary.

See the "Limits and Ranges" section found in the appendix of this manual that is specific to your RDBMS for a list of supported data types and their COBOL equivalents.

6.1.2 Compiling With the WHEN Directive

Suppose that you wanted *both record formats* to be placed into the table. This might be the case if you intended to do any work within your RDBMS. Add the **WHEN** directive in front of each record, as shown below (see **Chapter 4, "Using Directives"**). The underlined fields are the ones that will be entered into the table:

```
 fd p-o-file.
$xfd when p-o-record-type = "h"
 01  p-o-record.
     03  p-o-division-number                    pic 9(3).
     03  p-o-record-type                        pic x.
         88  header-record                      value "h".
```

```
        88  detail-record                          value "d".
    03  p-o-number                                 pic 9(10).
    03  p-o-number-detail redefines p-o-number.
        05  picking-ticket-number                  pic 9(6).
        05  shipping-region                        pic 9(2).
        05  p-o-customer-type                      pic 9(2).
        05  p-o-customer-breakdown redefines
              p-o-customer-type.
            07  customer-category                  pic x.
                88  p-o-customer-retail            value "r".
                88  p-o-customer-whlsale           value "w".
            07  customer-pay-format                pic x.
                88  is-net-30                      value "3".
                88  is-net-10                      value "1".
    03  p-o-date.
        05  p-o-yy                                 pic 9(2).
        05  p-o-mm                                 pic 9(2).
        05  p-o-dd                                 pic 9(2).
$xfd when p-o-record-type = "d"
01  p-o-detail-record.
    03  p-o-dept-number                            pic 9(3).
    03  p-o-record-type                            pic x.
    03  detail-p-o-number                          pic 9(10).
    03  p-o-shipping-info.
       05  p-o-quantity-to-ship                 pic s9(4) comp.
       05  p-o-total-quantity                   pic s9(4) comp.
    03  p-o-notes.
        05  notes-line occurs 3 times              pic x(40).
```

Note: **p-o-record-type** is entered into the table only *once*. The **detail-p-o-number** field is not part of the table, because the key overlays this area.

This table is built in the database:

Column Name	Type
p_o_division_number	number(3) *
p_o_record_type	char(1)
p_o_number	number(10) *
p_o_yy	number(2) *

Column Name	Type
p_o_mm	number(2) *
p_o_dd	number(2) *
p_o_dept_number	number(3) *
p_o_quantity_to_ship	number(4) *
p_o_total_quantity	number(4)
notes_line_1	char(40)
notes_line_2	char(40)
notes_line_3	char(40)

* The actual database datatype may vary.

6.1.3 Using Additional Directives

In this final approach, you decide to streamline the code a bit, so you introduce the following three changes:

- You apply the **USE GROUP** directive to the notes, because you don't need to access each note line individually from the database side. Grouping them improves execution speed and clarity.
- You rename the notes field for convenience.
- You apply the USE GROUP and **DATE** directives to the date, to give it all the properties of a date on the database side.

```
 fd p-o-file.
$xfd when p-o-record-type = "h"
 01  p-o-record.
     03  p-o-division-number                    pic 9(3).
     03  p-o-record-type                        pic x.
         88  header-record                      value "h".
         88  detail-record                      value "d".
     03  p-o-number                             pic 9(10).
     03  p-o-number-detail redefines p-o-number.
         05  picking-ticket-number              pic 9(6).
         05  shipping-region                    pic 9(2).
```

```
         05  p-o-customer-type                    pic 9(2).
         05  p-o-customer-breakdown redefines
               p-o-customer-type.
             07  customer-category                pic x.
                 88  p-o-customer-retail          value "r".
                 88  p-o-customer-whlsale         value "w".
             07  customer-pay-format              pic x.
                 88  is-net-30                    value "3".
                 88  is-net-10                    value "1".
$xfd use group, date
     03  p-o-date.
         05  p-o-yy                               pic 9(2).
         05  p-o-mm                               pic 9(2).
         05  p-o-dd                               pic 9(2).
$xfd when p-o-record-type = "d"
 01  p-o-detail-record.
     03  p-o-dept-number                          pic 9(3).
     03  p-o-record-type                          pic x.
     03  detail-p-o-number                        pic 9(10).
     03  p-o-shipping-info.
        05  p-o-quantity-to-ship                pic s9(4) comp.
        05  p-o-total-quantity                  pic s9(4) comp.
$xfd use group, name = notes
     03  p-o-notes.
         05  notes-line occurs 3 times           pic x(40).
```

Note: **p-o-record-type** is entered only *once* into the table. The **detail-p-o-number** field is not part of the table, because the key overlays this area.

This table is built in the database:

Column Name	Type
p_o_division_number	number(3)
p_o_record_type	char(1)
p_o_number	number(10)
p_o_date	date
p_o_dept_number	number(3)
p_o_quantity_to_sh	number(4)

Column Name	Type
p_o_total_quantity	number(4)
notes	char(120)

6.2 Creating File Descriptors and SELECT Statements

acu4glfd is a Windows-based utility that can help you analyze tables and create File Descriptors and SELECT statements. This utility uses ODBC technology to retrieve table information to list the fields in the table. When possible, the utility also determines a unique index and uses that index as the primary key.

Upon request, **acu4glfd** generates two kinds of output:

- a file descriptor (FD), listing the names of the columns in a table and their associated data types
- a SELECT statement for the file

You can COPY the FD and SELECT statement into your COBOL program, although you may need to make some changes, as indicated in this section.

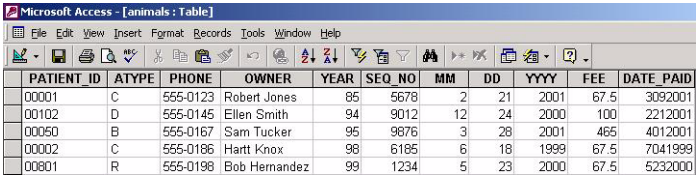
The **acu4glfd** utility is intended to help you jump start preparing your table for use with a COBOL program. The program determines the names of the columns, attempts to determine the appropriate ACUCOBOL-GT® data type for the column, and attempts to determine the primary key. **acu4glfd** does not attempt to identify alternate keys.

Remember, though, that database data is flat, and COBOL data can contain group items and depth. Because table data does not contain information about GROUP structure the way a COBOL program does, it may be necessary for you to tweak the code produced by **acu4glfd** and specify a picture before the program compiles properly. Any errors produced in this early compilation will assist in fine-tuning the FD for your COBOL program. This saves the time and effort of completely analyzing the table and coding the FD and SELECT statements yourself.

The **acu4glfd** utility operates only on data types that are understood by ODBC. Database drivers can define their own data types as well; these driver-defined data types are not recognized by **acu4glfd**. Therefore, the utility may not be able to identify all data types represented in the table. Refer to your database, driver, and ODBC documentation for additional information.

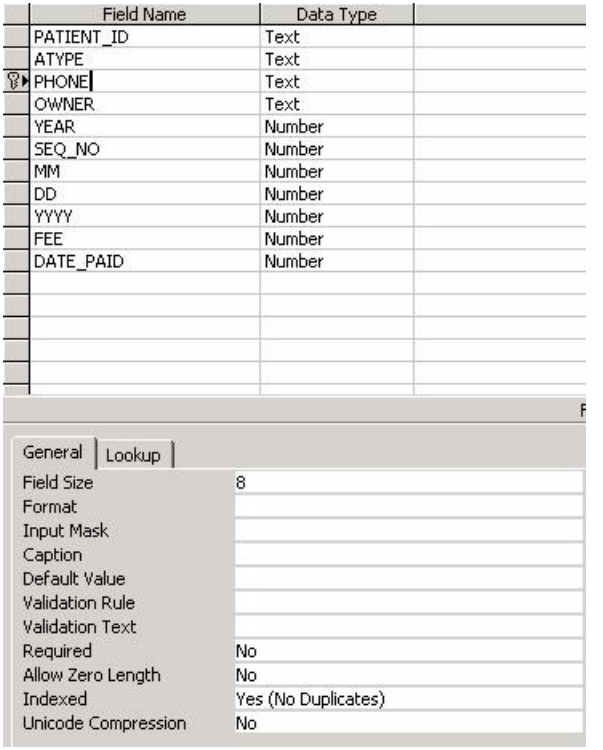
Before starting the **acu4glfd** utility, you must define your table/data as an ODBC data source. See your ODBC documentation for information about drivers and data sources.

Note that the examples are based on the following table in Microsoft Access.



PATIENT_ID	ATYPE	PHONE	OWNER	YEAR	SEQ_NO	MM	DD	YYYY	FEE	DATE_PAID
00001	C	555-0123	Robert Jones	85	5678	2	21	2001	67.5	3092001
00102	D	555-0145	Ellen Smith	94	9012	12	24	2000	100	2212001
00050	B	555-0167	Sam Tucker	95	9876	3	28	2001	465	4012001
00002	C	555-0186	Hartt Knox	98	6185	6	18	1999	67.5	7041999
00801	R	555-0198	Bob Hernandez	99	1234	5	23	2000	67.5	5232000

Note that the phone number (PHONE field) is the table's key.



Field Name	Data Type
PATIENT_ID	Text
ATYPE	Text
PHONE	Text
OWNER	Text
YEAR	Number
SEQ_NO	Number
MM	Number
DD	Number
YYYY	Number
FEE	Number
DATE_PAID	Number

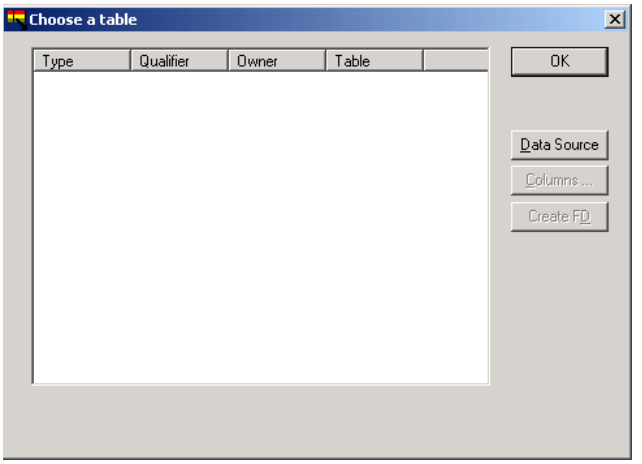
General | Lookup

Field Size	8
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	No
Allow Zero Length	No
Indexed	Yes (No Duplicates)
Unicode Compression	No

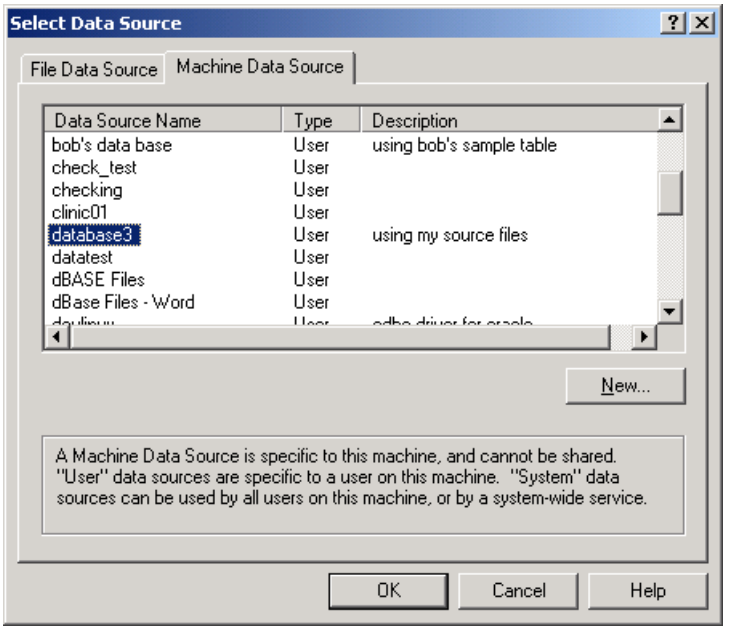
To generate a list of columns and data types and a SELECT statement

1. Start the utility by clicking on the **acu4glfd** icon or entering **acu4glfd** at the Windows command prompt. By default, "acu4glfd.exe" resides in your "acugt/bin" directory.

2. Click **Data Source** in the "Choose a table" dialog. (Click **OK** to terminate the application.)

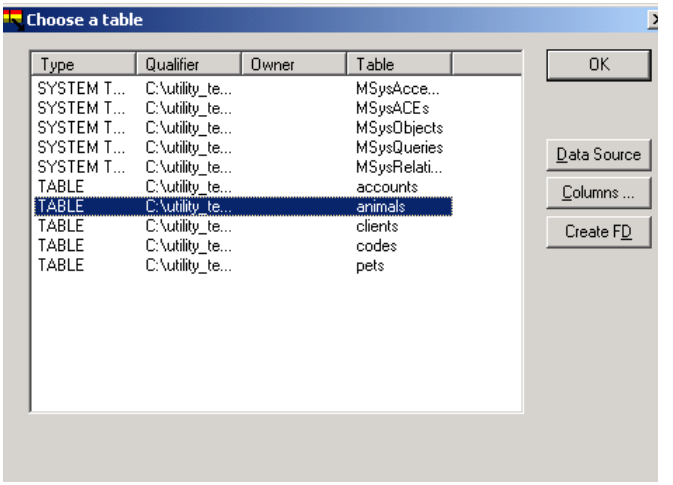


3. Select a data source from the standard Microsoft dialog that appears.



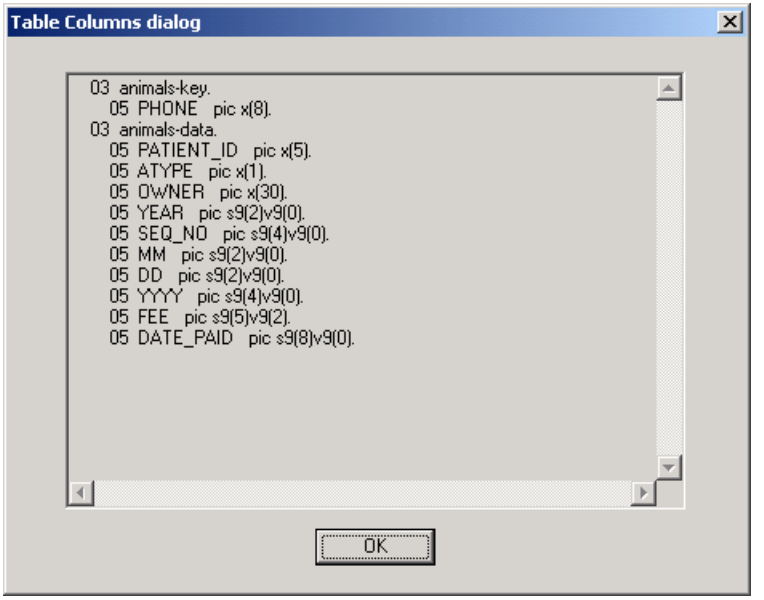
4. If necessary, enter your user name and password in the Authorization area of the Login dialog.

5. Select a table from the Choose a table dialog, which now contains a listing of the tables in the data source you selected in step 3. Note that you can choose only one table at a time.



The **Column** and **Create FD** buttons become active. To see a listing of columns and data types, proceed to **step 6**. To create an FD, go to **step 8**.

6. Click **Columns** to display a list of the columns in the table and their associated COBOL data types.



Note that PHONE appears first, even though it is not the first column in the table. That is because PHONE was defined as the table's key, so **acu4glfd** treats it as a unique index, moving it to the top of the list of columns.

7. Click **OK** to close this window.

8. Click **Create FD** to display the Save As dialog. Use this standard Windows dialog to indicate where the FD file should reside. Be sure to indicate to save as file type "File Descriptors." For this example, the FD is as follows:

```
*  animals.fd - Generated by Acu4GLfd from animals on
   2002/11/11
fd  animals.
01  animals-record.
    03  animals-key.
        05  PHONE    pic x(8).
    03  animals-data.
```

```
05  PATIENT_ID   pic x(5).
05  ATYPE   pic x(1).
05  OWNER   pic x(30).
05  YEAR   pic s9(2)v9(0).
05  SEQ_NO   pic s9(4)v9(0).
05  MM   pic s9(2)v9(0).
05  DD   pic s9(2)v9(0).
05  YYYY   pic s9(4)v9(0).
05  FEE   pic s9(5)v9(2).
05  DATE_PAID   pic s9(8)v9(0).
```

Note that in the FD, as in the column listing, PHONE, appears as the first item.

When **acu4glfd** determines a unique index, it creates a SELECT statement as well. By default, this SELECT is placed in the same directory as the FD. The base name is the same; however, the SELECT has an extension of ".sl". For this example, the SELECT is as follows:

```
*  animals.sl - Generated by Acu4GLfd from animals on
   2002/11/11
    select animals
    assign to disk "animals"
    organization is indexed
    access is dynamic
    record key is animals-key
    status is animals-status.
```

Note that the FD generated by **acu4glfd** uses the name of the table in the "fd" line. (See the sample code in **step 6**.) This will be the same name that appears in the "assign" line in the SELECT statement. (See **step 8** for the sample SELECT statement.)

You can now look at the FD and SELECT to determine if there are any changes you want to make immediately. Make those changes, COPY the FD and SELECT into your program, and try to compile the program. If compilation is successful, you can move on to the next step in your development effort. If compilation is not successful, use the information contained in the error messages to aid in troubleshooting the program.

7 New and Existing Databases

Key Topics

7.1 Databases

You'll probably find yourself in one of two situations as you begin to use this product. In situation one, the database files don't exist yet and will be brand new.

In situation two, the database files already exist, and you want to access that existing data from a COBOL application. There may or may not already be existing COBOL programs that access the database.

Each situation brings up its own questions. How do I declare my COBOL data so that it matches the database data? Are there any special fields in the database that must be accessed in a special way? How do I choose COBOL record names, field names, index names, and data types that best conform to the compiler, the Acu4GL® product, and the database engine? We'll ask and answer questions of that type in the next few sections.

7.2 Default Acu4GL Behavior

The following are automatically handled by the Acu4GL product:

- Acu4GL automatically converts upper case field names to lower case (or vice versa) when necessary, so don't concern yourself with case differences.
- Acu4GL automatically performs the data conversions needed to match the internal storage formats used by the database.
- If the database files don't already exist, they will be created for you automatically when you execute an OPEN OUTPUT verb. If this is your situation, you'll have no concerns about matching COBOL fields to database fields. The fields will match perfectly, because the database fields will be based on your COBOL code.
- You may encounter unexpected sequencing of data returned from a database. If your COBOL definition does not include alternate keys with duplicates, the data is returned in key order. If your COBOL definition does include alternate keys with duplicates, be aware that Acu4GL cannot control the sequence in which the data is returned. The database

query optimizer decides how to order the returned records. For a set of records with the same key value, the records may not be in historical order.

See also

A_MSSQL_ADD_IDENTITY
A_SYB_ADD_IDENTITY
A_INF_DUPLICATE_KEY

7.3 Accessing Existing Database Files

If you are accessing existing data in a relational database, you need to know how to declare your data so that it will match the database fields.

If your COBOL code isn't written yet, you should follow the guidelines given in the appendix of this manual for your specific RDBMS.

If your COBOL application already exists, you can make necessary adjustments by adding *directives* to your code. Directives are comments that guide the creation of the data dictionaries. The dictionaries in turn help map the COBOL fields to their equivalent database fields. Directives are described in **Chapter 4, "Chapter 4: Using Directives."**

7.3.1 How Do I Match Existing Text Fields?

To access character data, you simply declare the field as PICTURE X, with as many X's as appropriate.

7.3.2 How Do I Match Existing Numeric Fields?

Numeric fields are more database-specific. Most relational database systems accommodate INTEGER, DECIMAL, and DATE types. For information on the handling of other numeric types, see the appendix containing information specific to your RDBMS.

7.3.3 Field Names

If your COBOL application already exists, and if it must access a relational database that already exists, you may have to work around differences in the names of the fields, as well as naming conventions imposed by the RDBMS. For example, your program might use the name EMPLOYEE-NO, while the database uses the name EMP-NUMBER for the same item of information.

Resolving name conflicts

If naming differences exist, *you need not rename your COBOL fields, and you need not change the database.* This is because the ACUCOBOL-GT® compiler's "-Fx" option builds data dictionaries that map your COBOL fields to the correct database fields. The mapping is automatic if the names are the same. If the COBOL name differs from the database name, you enable the compiler to make the correct mapping by adding a **NAME** directive to your COBOL code.

Directives can also be used to produce other effects when data is mapped from COBOL to the database. **Chapter 4, "Chapter 4: Using Directives,"** describes directives in detail.

7.3.4 Index Names

Acu4GL uses a specific naming convention for indexes it creates. Indexes created with Acu4GL use the i<TABLENAME>_<key value> convention. If you are using indexes created outside of the Acu4GL application, you need to make sure the index names match the Acu4GL naming convention.

Performance is affected by the number of indexes you may have. If you use tables with multiple indexes, keep in mind that when a record is written or updated, locks are put onto all of the indexes, and they are all basically rewritten during the course of the write/update. This is a costly process. There may be multiple columns per index, and multiple indexes per table. Each rewrite implies a certain amount of wait time. Tables with a large number of indexes can be slow on writes/updates and possibly other operations in the database query optimizer become confused.

For more information about indexes and performance issues, see **Chapter 9, "Chapter 9: Performance and Troubleshooting."**

8 Compiler and Runtime Options

Key Topics

8.1 Compiler Options

Some file options (-Fa, -Fe, -Fx, and -F4) are used to generate XFD files (data dictionaries) that are used with Acu4GL. Other file options (-Fl, -Fs, and -Ft) can simplify the addition of transaction management facilities to existing programs. This section describes compiler options that are useful when working with Acu4GL.

-Fa This option tells the compiler to build data dictionaries (XFD files) for every indexed, relative, and sequential data file in the FDs of the program. It is the only option that builds XFDs for relative and sequential files. This option is also used for international character mapping. See also the "**-Fo**" compile-time option, which specifies the directory in which the data dictionaries are placed.

The "-Fa" option generates the most current format for the XFD files (Version 6), compatible with Acu4GL Version 6.0 and later.

-Fc This option causes the field names in generated XFD files to match exactly the source of the COBOL program that generated them.

-Fe This option causes XFD files to be generated in XML format rather than the standard flat text format. Acu4GL, AcuXDBC, and **alfred** can all read the XFD files in XML format. This option must be used in conjunction with the "**-Fx**" or "**-Fa**" options. The C$XML library routine can be used to parse the XML files if desired. In addition, by specifying the **XSL** directive, you can specify a stylesheet to be used when compiling

This option will not work in combination with "**-F4**". Version 4 XFD files cannot be generated in XML format.

-Fl This option enables single-locking rules rather than multiple-locking rules as the lock mode default. Normally, "WITH ROLLBACK" causes multiple locking rules to be in effect for a file. When "-Fl" is used, the "WITH ROLLBACK" clause does not affect whether single or multiple record locking rules are followed. Single locking becomes the default. You may enable multiple locking either by specifying "WITH LOCK ON MULTIPLE RECORDS" in a file's SELECT statement or by using "APPLY LOCK-HOLDING ON *file*" in the I-O CONTROL paragraph.

-Fo This option must be followed (as the next separate argument) by the directory that will hold the data dictionary files generated by the compiler when you use the "**-Fx**" option.

Type a space after the option and then give the name of the chosen directory. If this option is not used, the data dictionaries are placed into the current directory.

For example, to cause the dictionaries to be stored in the directory "/usr/inventory/dictionaries", you would enter:

```
-Fo  /usr/inventory/dictionaries
```

-Fs This option causes an implied START TRANSACTION verb before the first OPEN, CLOSE, WRITE, REWRITE, or DELETE and after each COMMIT or ROLLBACK. In effect, every file operation is part of a transaction. If this option is enabled and the compiler encounters a START TRANSACTION verb, it reports a warning and does not generate any code for the START TRANSACTION. The "-Fs" option provides an alternate way to program transactions and is often useful when you are converting from other COBOL or SQL implementations.

-Ft This option causes implied transactions for every OPEN, CLOSE, WRITE, REWRITE, or DELETE that is not part of an explicit transaction. Single file operations that are not part of a transaction are preceded by an implied START TRANSACTION and followed by an implied COMMIT. This option makes converting existing applications to a transaction system easier. Note that unlike most COMMITs, which unlock all of the file's currently locked records, the implied COMMIT does not unlock any records.

-Fx This option directs the compiler to build Version 5 data dictionaries (XFD files) for every indexed data file in the FDs of the program. If you require the older version of XFD files, specify the "**-F4**" option instead of "-Fx". If you use relative, sequential, or XML data files, use the "**-Fa**" option instead. Use the "**-Fo**" option to specify the directory in which the data dictionaries should be placed.

-F4 This option tells the compiler to build Version 4 data dictionaries (XFD files) for every indexed data file in the FDs of the program. This older version of the XFD files is compatible with Acu4GL Version 5.x and earlier. To build Version 4 XFDs for every indexed, relative, and sequential data file in your FDs, combine "-F4" with "**-Fa**", as in "ccbl -F4 -Fa".

See **section 8.2** for a description of runtime options.

8.2 Runtime Configuration Variables

There are several variables that can be set in your runtime configuration file that affect Acu4GL processing. Those listed below are applicable to any RDBMS with which Acu4GL communicates. Some configuration variables are database-specific; those variables are discussed in the individual database product sections found in this *User's Guide*.

Note: Most configuration variables can be read with the ACCEPT FROM ENVIRONMENT verb. If the variable to be read is numeric, the receiving field must be defined either as a numeric field or as an alphanumeric field of five or more characters. If it is defined as alphanumeric and is longer than five characters, the value that is read from the environment will occupy the leftmost five characters of the field and the remainder will be space-filled.

4GL_2000_CUTOFF

This variable is used to implement "date windowing" for Acu4GL. This variable determines which two-digit years will be considered to be in the twentieth century and which two-digit years will be considered to be in the twenty-first century.

This variable accepts a two-digit value.

- Two-digit years that are smaller than this value are considered part of the twenty-first century ("2000" is added to these dates).
- Two-digit years that are equal to or larger than this value are considered to be in the twentieth century ("1900" is added to them).

The default value for this variable is "20".

4GL_8_DIGIT_CUTOFF

This variable determines whether to use the 4GL_2000_CUTOFF for dates that don't use a format string and that are 8 digits. Setting this variable to "1" causes Acu4GL to apply the 4GL_2000_CUTOFF logic for 8-character-long dates. The default value is "0" (off, false, no). This configuration variable can also take values of "1" (on, true, yes).

4GL_COLUMN_CASE

This variable causes the Acu4GL product to leave the *case* of the field names found in the XFD *unchanged*. Normally all field names are converted to lower case by Acu4GL (except for Oracle, which converts all values to upper case), and all hyphens are converted to underscores.

- Valid values for this variable are UNCHANGED and LOWER.
- The default value is LOWER, which means that field names are converted to lower case and hyphens are converted to underscores.

If this variable is set to "Unchanged", all field names are used *unchanged* when the database table is created. Be aware that hyphens are not converted to underscores in this situation. Most databases do not accept hyphens in column names, so the XFD must be modified by hand to replace hyphens with underscores.

4GL_COMMIT_COUNT

The value of 4GL_COMMIT_COUNT indicates the conditions under which you want to issue an automatic COMMIT–WORK operation. Valid values are:

4GL_COMMIT_COUNT = 0 (Default)

When you set this variable to "zero", the runtime tracks the number of logical locks that are currently in effect. When the number of logical locks reaches zero, the runtime assumes that a transaction is complete and issues a COMMIT statement.

4GL_COMMIT_COUNT = *n*

When you set this variable to a nonzero value, the runtime tracks the number of WRITE, REWRITE, and DELETE operations, until the value of 4GL_COMMIT_COUNT is reached, at which time the runtime issues a COMMIT statement. The READ, START, and READ NEXT operations do not count toward this total, because the runtime is tracking data-altering operations rather than logical record locks. The disadvantage of this method is that when a COMMIT is issued, any record locks held by the runtime are released.

4GL_COMMIT_COUNT = -1

No commit is issued by the Acu4GL product. When 4GL_COMMIT_COUNT is set to "-1", two alternate ways to perform a commit or rollback are available:

1. Call **sql.acu** with COMMIT WORK or ROLLBACK WORK.
2. Use the COBOL verbs COMMIT and ROLLBACK, available in ACUCOBOL-GT.

4GL_COMMIT_COUNT is set to "-1" automatically when you use the transaction management facilities available in the ACUCOBOL-GT compiler. A COMMIT WORK is, however, issued on exit from the runtime (for example, on execution of a STOP RUN).

Examples

```
4GL_COMMIT_COUNT  0
```

A commit will be issued when no locks are held, either because all files that had locked records have been closed, or because a COBOL COMMIT verb has been issued. This is the default value.

Note that some data sources lose the current row when a COMMIT or ROLLBACK is executed. For these data sources, a setting of "0" for 4GL_COMMIT_COUNT can adversely affect performance.

```
4GL_COMMIT_COUNT  n
```

A commit will be issued after *n* operations. WRITE, REWRITE, and DELETE count towards *n*; READ, START, and READ NEXT do not.

```
4GL_COMMIT_COUNT  -1
```

No commit will be issued by the Acu4GL product. When 4GL_COMMIT_COUNT is set to "-1", the only way to perform a commit or rollback is to use the COBOL verbs COMMIT and ROLLBACK.

4GL_COMMIT_COUNT is set to "-1" internally when you use the transaction management facilities available in ACUCOBOL-GT.

A COMMIT will, however, be issued on exit from the runtime (for example, on execution of a STOP RUN).

Note: 4GL_COMMIT_COUNT can be set only in the configuration file. It cannot be set programmatically.

4GL_COMMIT_COUNT is available only in the Acu4GL for ODBC and Acu4GL for DB2 interfaces.

4GL_CONVERT_DATE_ZERO

If your data uses dates as part of Oracle or Informix database keys, the dates cannot be written as NULL or illegal values. To write data that includes NULL or illegal values, Acu4GL converts the NULL or illegal date values used in keys to a value of "01/01/01" when writing data to Oracle. Unfortunately, the code would not make a similar conversion when reading the same record, and those records written with "01/01/01" were not found. This could easily result in writing a record that cannot be read or deleted. When the 4GL_CONVERT_DATE_ZERO variable is set to the default setting "CONVERTED", Acu4GL is able to locate and return those records written using the date "01/01/01". This functionality can be disabled by setting the variable to "UNCHANGED."

4GL_DB_MAP

This variable enables you to set one map at a time, until all of the desired maps are set, by using FILE-PREFIX as a list of directories, each of which maps to a different database. Note that in this case "database" refers to "collection of tables" rather than distinguishing between (say) Oracle and SQL Server.

The syntax of the value of 4GL_DB_MAP is:

directory = database

where *directory* is matched against the file specification, and *database* is used when a directory matches. For example, if you have two databases, data01 and data02, and have COBOL programs written that reference data

files in /home/dataA and /home/dataB, you can set 4GL-DB-MAP as follows:

```
4GL-DB-MAP /home/dataA = data01
4GL-DB-MAP /home/dataB = data02
```

When the runtime attempts to open a file with a directory specification of /home/dataA, Acu4GL uses the table in database data01.

By setting this variable to blank, all maps will be reset to nothing, so no mapping will be performed. In this case, the Acu4GL interface behaves as it has always done in the past.

Currently, this feature is available with the following Acu4GL interfaces: MSSQL and Sybase. Any special instructions will be included in the section describing support.

See Also

The **4GL_USEDIR_LEVEL** variable, which determines how many levels of directory information to use as part of a table name. 4GL_DB_MAP interacts with 4GL_USEDIR_LEVEL, in the following way:

1. 4GL_DB_MAP is tested first, to determine which database to search for the table.
2. Then 4GL_USEDIR_LEVEL is used to change the final name of the table.

4GL_EXTRA_DB_COLS_OK

4GL_EXTRA_DB_COLS_OK allows the database table to have more columns than the COBOL program's corresponding file descriptor.

If the 4GL_EXTRA_DB_COLS_OK variable is set to "True" (on, yes), its default value, the number of columns in the database table do not have to match up with the number of fields in the COBOL program that is accessing the table. The database table can have more columns than the COBOL program references; however, the COBOL program may *not* have more fields than the database table.

Ensure that these "extra" columns are set correctly when new rows are added to the table. This is database-dependent. For example, under Sybase, the columns should either have a default value or should allow NULL. Ensuring that the column value is set is important because the SQL generated by the Acu4GL interface will not reference the extra columns when inserting rows. This configuration variable can also take values of "False" (off, no).

Note: Only the Acu4GL products for Sybase and Microsoft SQL Server support this variable. This behavior is the default for Informix and Oracle.

4GL_FULL_DATA_TEST

The configuration variable 4GL_FULL_DATA_TEST allows you to specify whether an entire field is tested for illegal data or if only the first byte is tested. The default value is "False" (0, off, no), meaning that only the first byte of data is tested. However, if you set 4GL_FULL_DATA_TEST to "True" (1, on, yes), the entire field is tested for illegal data.

4GL_IGNORED_SUFFIX_LIST

This variable is a space-delimited list of case-independent suffixes to ignore when opening Microsoft SQL Server tables, DB2 tables, or tables in an ODBC-compliant application.

The standard syntax for Microsoft SQL Server tables is server.database.owner.table, with a dot (".") as the separator. This syntax causes the filename "idx1.dat" to be interpreted as "the table dat owned by the user idx1." By setting this variable (which must be set in the configuration file, and cannot be set via SET ENVIRONMENT), the Acu4GL for Microsoft SQL Server product will ignore the listed suffixes.

There can be up to 10 ignored suffixes (more than that are not kept internally, so will be ignored.) The default value for this variable is "dat".

Example

```
4GL_IGNORED_SUFFIX_LIST dat idx txt
```

4GL_ILLEGAL_DATA

4GL_ILLEGAL_DATA determines how COBOL data that is considered illegal by the database will be converted before it is stored in the database.

The default value is NULL. This causes all illegal data (except key fields) to be converted to null before it is stored.

The value CONVERTED causes the following conversions to occur:

Illegal LOW-VALUES:	minimum possible value (0 or -99999...) minimum possible date value (01/01/01)
Illegal HIGH-VALUES:	maximum possible value (99999...) maximum possible date value (database dependent)
Illegal SPACES:	zero (or 01/01/01 for a date field)

Illegal data in key fields is always converted, regardless of the value of this configuration variable.

Oracle special options

The following values for 4GL_ILLEGAL_DATA apply only to Oracle character fields:

KEY_MOD

If a key field contains LOW-VALUES in the first byte, the field is considered invalid. The default behavior is to set the field to the minimum possible value. For character fields, this would result in a single character with the hex value of 0x01 being inserted. (For fixed-length fields, Oracle would space pad the field.)

Setting 4GL_ILLEGAL_DATA to the value KEY_MOD will result in only the first byte of an invalid key field being modified to the hex value of 0x01. The remainder of the field will remain unchanged.

This value can be useful for users converting from versions prior to 6.2 in which the interface's functionality matched this behavior rather than the documented behavior.

PASS_THROUGH

If the value of 4GL_ILLEGAL_DATA is set to PASS_THROUGH, all key field and non-key field values will be directly passed to Oracle after having trailing spaces removed for variable length fields. Any error detection on the fields will be handled by the Oracle database itself.

4GL_JULIAN_BASE_DATE

This variable sets the base date to be used for Julian date calculations. It is used in conjunction with the **DATE** directive.

To define your own base date for Julian dates, set this variable to a date having the format YYYYMMDD.

The default value is set internally at 01/01/4713 BC. (Dates other than the default must be greater than 01/02/4713 BC.)

For example, suppose your COBOL program uses dates in the form of the number of days since 12/31/1899. If you want to store these dates in a database accessed by Acu4GL, you can set your DATE directives to be $XFD DATE=JJJJJJ (make sure that the number of characters matches your date field). Then you should set the runtime configuration variable 4GL_JULIAN_BASE_DATE to “18991231”.

If you are using the runtime intrinsic functions that work with Julian dates, you should set the 4GL_JULIAN_BASE_DATE variable to “December 31, 1600” using the Gregorian calendar.

4GL_USEDIR_LEVEL

This variable provides a method of mapping filenames (with directory information) to table names. 4GL_USEDIR_LEVEL allows you to use files of the same name, which would normally go into separate directories, as separate names in a single database. This variable determines how many levels of directory information to use as part of the table name.

- If set to the default value of "0", all directory information is removed from the filename before it is used as a table name.
- When set to negative values, nothing is done differently.
- When set to a positive value, the number determines how much directory information to keep.

For example, given the file /home/data1/gl/master.dat, the following values result in the given table name:

4GL-USEDIR-LEVEL	Final resolved name
0	master.dat
1	glmaster.dat
2	data1glmaster.dat
3	or largerhomedata1glmaster.dat

This variable has an upper limit of 127 characters and is currently available with the following Acu4GL interfaces: MSSQL and Sybase. Any special instructions will be included in the section describing support.

See Also

The **4GL_DB_MAP** variable, which enables you to set one map at a time. 4GL_USEDIR_LEVEL interacts with 4GL_DB_MAP in the following way:

1. 4GL_DB_MAP is tested first, to determine which database to search for the table.
2. Then 4GL_USEDIR_LEVEL is used to change the final name of the table.

4GL_WHERE_CONSTRAINT

4GL_WHERE_CONSTRAINT allows you to override any value given in the external variable A4GL-WHERE-CONSTRAINT. This configuration variable can have a value with any length (limited only by machine memory).

If a value is set in 4GL_WHERE_CONSTRAINT, any value in the external variable is ignored. See **section 9.1.2, "The WHERE Constraint,"** for information on the external variable.

DEFAULT_HOST

If you are opening an *existing* file, most file systems linked into the runtime will be searched for the named file. If, however, you are creating a *new* file, you will need to tell the runtime which file system to use. You accomplish this with one of two configuration variables; the first is:

```
DEFAULT_HOST filesystem
```

This will designate the file system to be used for newly created files that are not individually assigned. If this variable is not given a value, the Vision file system is used.

Example

```
DEFAULT_HOST x
```

means that all new files will be *x* files unless individually assigned to another file system. For example,

```
DEFAULT_HOST INFORMIX
```

means that all new files will be Informix files, unless they are individually assigned to another file system.

The possible values for *x* are the following:

- INFORMIX
- ORACLE
- MSSQL
- ODBC
- DB2
- SYBASE

- EXTFH

See also

The **filename_HOST** variable, which specifies the file system to be used for *one specific file*

*filename*_HOST

The second variable that can be used to assign a newly-created file to a file system is *filename*_HOST. This configuration variable is used to assign an individual data file to a file system. Any file so assigned will use the designated file system, and not the one specified by DEFAULT_HOST.

Substitute the base name of the file for *filename*. The name you substitute should *not* include any directory names and should *not* include a file extension.

Example

For example, if the file CUSTFILE were the only file you wanted to assign to the Vision file system, and all other files were to be Informix files, you could specify:

```
DEFAULT_HOST   INFORMIX
CUSTFILE_HOST   VISION
```

in the configuration file. Other possible values for *filename*_HOST include:

- ORACLE
- MSSQL
- ODBC
- DB2
- SYBASE
- EXTFH

See also

The **DEFAULT_HOST** variable, which specifies the file system to be used for all files not specified by *filename*_HOST.

You can use DEFAULT_HOST and *filename*_HOST in combination to assign your new files in a default with exceptions manner. For example, the following directives:

```
DEFAULT_HOST VISION
afile_HOST ORACLE
bfile_HOST ORACLE
```

mean that all new files except *afile* and *bfile* will be assigned to Vision, and those two files will be assigned to Oracle.

Now suppose *myfile1a* and *myfile1b* both reside in mydb1, and *myfile2a* and *myfile2b* both reside in mydb2. With the following directives:

```
DEFAULT_HOST VISION
a-mssql-database mydb1
myfile1a-HOST mssql
myfile1b-HOST mssql
myfile2a mydb2.dbo.myfile2a
myfile2b mydb2.dbo.myfile2b
mydb2-HOST mssql
```

the runtime will use the SQL Server interface for *myfile1a*, *myfile1b*, *myfile2a*, and *myfile2b* and will find all the tables in the correct database. When the COBOL program opens *myfile2a*, the runtime translates that name to *mydb2.dbo.myfile2a* and looks at the basename to determine the filesystem to use (mydb2). It then looks for mydb2-HOST to determine which filesystem to send the file requests to.

You can change the values of the filename-HOST and DEFAULT_HOST variables during program execution by including the following in your code:

```
SET ENVIRONMENT "filename_HOST" TO filesystem
```

or

```
SET ENVIRONMENT "DEFAULT_HOST" TO filesystem
```

This enables you to change file systems during the execution of your program. (This is not the typical way to specify a file system; it is usually designated in the runtime configuration file and is not changed in the COBOL program.)

FILE_TRACE

This variable allows you to start file tracing without opening the debugger. Set this variable to a non-zero value to save information about all file OPENs, READs, and WRITES in the error file. This is equivalent to specifying "tf *n*" from the debugger (where *n* is an integer). The default is "0." See **section 3.1.4, "File Tracing,"**of the *ACUCOBOL-GT User's Guide* for more information about the file trace feature.

XFD_DIRECTORY

XFD_DIRECTORY tells the runtime system the name of the directory that contains the data dictionaries built by the ACUCOBOL-GT compiler. The default value is the current directory.

Example

To tell the runtime that the dictionaries are stored in the directory /usr/inventory/dictionaries, enter the following:

```
xfd-directory   /usr/inventory/dictionaries
```

Note: This configuration variable is ignored if you have specified a search path for XFD files using the XFD_PREFIX configuration variable.

See also

XFD_PREFIX runtime configuration variable, which enables you to specify a path to search for XFD files. XFD_PREFIX extends the functionality of XFD_DIRECTORY.

See **section 8.1, "Compiler Options,"** for information about the "-Fo" option, which tells the compiler where to put the dictionaries. Unless you have moved the dictionaries, you should use the same value in XFD_PREFIX (or for XFD_DIRECTORY) that you used with the "–Fo" option.

XFD_MAP

XFD_MAP tells the runtime system to associate certain filenames with a particular XFD. This enables you to use one XFD for many different files. You can use XFD_MAP to add or replace the existing value depending, on the setting of the **XFD_MAP_RESETS** variable.

The XFD_MAP variable has this syntax:

```
XFD_MAP    [pattern = base-xfd-name]   ...
```

where *pattern* consists of any valid filename characters and may include "*" or "?". These two characters have special meanings in the pattern:

*	matches any number of characters
?	matches a single occurrence of any character

For example:

CUST????	matches CUST0001 and CUSTOMER; does not match CUST001 or CUST00001
CUST*	matches all of the above
CUST*1	matches CUST001 and CUST0001 and CUST00001; does not match CUSTOMER
*OMER	matches CUSTOMER; does not match CUST001 or CUST0001

Note that XFD_MAP cannot be read from the ACCEPT FROM ENVIRONMENT statement.

XFD_MAP_RESETS

This configuration variable determines whether setting the **XFD_MAP** adds to or replaces the existing value. When this variable is set to "0" (off, false, no), setting the XFD_MAP adds new value patterns to the end of the existing value. When it is set to "1" (on, true, yes), setting the XFD_MAP replaces the existing value with a new value. The default value is "1" (on, true, yes). This variable may be useful if you need to include multiple XFD_MAP lines in a configuration file, and want to avoid setting and resetting the variable. When multiple lines exist, all patterns are used in the order they are listed.

XFD_PREFIX

When this variable is assigned a value, it is used to locate the XFD file, and the variable **XFD_DIRECTORY** is ignored. This variable enables you to define a specific series of directories to search for XFD files, rather than indicating only one. The runtime searches each directory in the order specified until an XFD file matching the name of the desired file is found and then assumes that is the correct file.

Therefore, when using the XFD_PREFIX variable, be aware that the runtime:

- Searches each named directory and not its subdirectories.
- Stops searching once a matching file name is found, even if other files of the same name are located in a different directory later in the series.
- Does not validate that the file specifications (such as key or record size parameters) of the XFD file match those in the COBOL program. Be sure the correct file you want to use is in the path specified.

You can specify a directory path that contains embedded spaces if you surround the path with quotation marks. You separate entries using a semi-colon (;). For example:

```
XFD_PREFIX C:\"Sales Data";C:\Customers
```

You can specify up to 4096 characters for this variable.

Note: The default for XFD_PREFIX is empty. If this variable is set to any other value, the configuration variable XFD_DIRECTORY (in which you specify only one directory) is ignored.

See also

XFD_DIRECTORY runtime configuration variable

Sections 5.2.1 and 5.2.2 in the *ACUCOBOL-GT User's Guide* for more information about client-enabled runtimes and remote notation.

9 Performance and Troubleshooting

Key Topics

9.1 Performance Issues

This chapter provides guidelines for improving system performance. Also included is an alphabetical listing of the error messages that can occur during the compilation of your program. Each message is followed by an explanation and a recommended recovery procedure.

Adding a relational database management system brings significant complexity to any computer system. A few key guidelines may help to improve the performance of the system and prevent problems as the database grows.

9.1.1 Guidelines

The following sections describe some areas for you to consider as you prepare your database and your COBOL program.

Guideline 1 - Database administrator

A site with a database management system needs a Database Administrator (DBA).

The administrator is critical in any large database installation. The DBA checks performance statistics and memory usage, performs general maintenance and backup for the database, initiates traces, executes database utilities, sets buffer sizes, determines how often buffers are flushing, and, in general, understands how database settings and events affect overall performance.

The DBA also performs periodic tuning of the database, including:

- using monitoring tools
- allocating table spaces
- examining output of the query optimizer to ensure that the most efficient paths are being used
- updating table statistics
- creating indexes

If a site experiences a performance problem, this usually means there is a bottleneck somewhere in the system. The DBA can help to isolate the bottleneck. Once the bottleneck is identified, the site can determine whether to apply more resources or correct the situation that is stressing the existing resources.

Guideline 2 - Understand COBOL operations and database operation

Some COBOL operations are particularly stressful to a database. The more the application uses these operations, the more likely it will slow the performance of the RDBMS.

The more you understand about your RDBMS and how it operates, the more you can help your COBOL applications to work efficiently with it.

File Input and Output

Consider these standard COBOL I/O operations:

- READ
- REWRITE
- WRITE
- DELETE
- OPEN

Each has an associated *cost* in terms of database performance. Each asks the database to do something that takes time. So if there are I/O operations that are *optional* in your COBOL application, you may want to remove them.

For example, let's examine file OPENs.

Developers sometimes OPEN and CLOSE files unnecessarily, using the OPEN–CLOSE pairing as a way to demarcate each new procedure:

```
OPEN file-A
procedural code
CLOSE file-A
```

But it's important to know that file OPENs are expensive in terms of performance. If you can eliminate non-essential OPENs from portions of your application, you can probably make an improvement in processing speed.

READ operations can also affect performance. All COBOL I/O is based on key indexes. Examining the output of your query optimizer allows you to determine if the most efficient execution path is being used for READs. The execution path for a given query can change as your data changes and as the size of the tables changes. It is also affected by database initialization parameters and any statistical information that you may have gathered on your tables. It might be helpful to know that, typically, the index path is the most efficient for Acu4GL® applications.

Transactions

Large *transactions* are also very expensive. The main problem here is that the database will hold locks on indexes and rows throughout an entire transaction. Thus, the database is creating additional overhead for an extended period of time if the transaction is lengthy.

In addition, complex information tracking must take place to ensure that transactions can be successfully rolled back.

Often application designers decide to err on the side of safety when applying transaction management to a mature application. Which operations should be included in a single transaction? The safe approach is to group everything that is related into one transaction. But this scheme is expensive—even more so when a database is involved. The lengthier the transaction, the longer the locks are held and system resources are tied up. The extensive data verification in COBOL programs only prolongs this.

If performance is an issue, give some thought to dividing each transaction into smaller and more efficient subgroups.

Tables with multiple indexes

If you use tables with multiple indexes, keep in mind that when a record is written or updated, locks are put onto all of the indexes, and they are all basically rewritten during the course of the write/update. This is a costly

process. There may be multiple columns per index, and multiple indexes per table. Each rewrite implies a certain amount of wait time. Tables with a large number of indexes can be slow on writes/updates and possibly other operations in the database query optimizer become confused.

There are two things you can do in this circumstance:

- Restructure your data
- Use the Vision file system for some of your data

Restructuring the data

The benefits of data restructuring may be significant.

For example, if you have any situations in which two indexes start out with the same column or set of columns, you may be able to improve performance appreciably by changing your data definition.

Suppose two indexes both start with MONTH, DAY, YEAR. These identical first three columns can cause the RDBMS's query optimizer to choose the wrong index, in which case you will generate a significant amount of unnecessary and unproductive overhead. Restructuring one of the indexes can make a difference.

Using Vision files

If you cannot restructure your data but are finding certain operations to be too expensive, you might want to consider moving some data into the Vision indexed file system.

Guiding the data searches

You can guide the data searches that result from database queries, and thus improve performance, by making use of an external variable called A4GL_WHERE_CONSTRAINT. This process is explained in **section 9.1.2, "The WHERE Constraint."**

Guideline 3 - Program and database interaction

A database can interact with your COBOL program in surprising ways. When you introduce a database to a mature COBOL application, your application may experience errors you have not seen before. Being aware of this possibility helps you to prepare your response.

For example, your existing application without the database may rarely exceed the limit on the number of data files that can be open simultaneously. But when you add a database, you increase the likelihood of this significantly. This is because the query processing may often require temporary tables. Also, the ORDER BY clause (used in SQL statements to sequence retrieved data) opens temporary work files. So you may find that these files cause you to reach the limit sooner. (Note that proper tuning of the query optimizer can reduce the number of temporary files required.)

When you upgrade to a new version of the RDBMS, be careful. The new software components may interact with your applications differently than their predecessors did. This is to be expected. It's important to rerun your pre-installation simulations (see **Guideline 4 - Plan for growth**) to determine whether your system resources are still adequate. Investigate any differences in the two simulations. You may have to make adjustments to compensate for the differences in the RDBMS versions.

Upgrading or switching to a new database may also mean that you need to modify your makefile to coordinate with the settings of the new database. You can edit your makefile manually, or use one of the editing tools distributed with Acu4GL. For more information, see the appendix specific to the Acu4GL interface you are using.

If you notice a change in performance with a new release of your RDBMS, keep in mind that certain database settings can have a significant effect on speed. Fine-tuning your settings can make a difference.

Several other options to help you improve performance are listed below:

- Try one of the many third-party monitoring tools available.
- Break up data onto several connected hard drives to free up space and resources.
- Limit the number of indexes you have assigned with each database table.

Guideline 4 - Plan for growth

We cannot emphasize enough the importance of planning ahead for growth. You need to be able to predict the system resources that your application will require when it reaches a full load, both in terms of users and database size.

Before you choose and install hardware, it is best to run a simulation. Your hardware vendor or RDBMS vendor can help you to simulate a large number of users on a given platform with a given RDBMS.

Setting up and running a simulation that includes your own application does cost money. But if you are moving into an installation of any size, the consequences of not knowing what to expect can be far greater than the cost of the simulation.

A potentially costly mistake is to test in your office with a small database and a small number of users. Databases respond differently to small amounts of data and a small number of users than they do to large amounts of data and a large number of users. Functionally, the database works differently as the load increases. You might think of this adjustment as switching into a different gear. Significant changes in the load on your database can lead to large increases in overhead. The behaviors and loads you will encounter as the database expands cannot be determined from a linear projection based on the smaller scenario.

9.1.2 The WHERE Constraint

The Acu4GL WHERE constraint is an external variable that gives the developer some control over the data searches that result from database queries. It can help to improve performance in some situations. This section describes its purpose and shows how it is implemented.

COBOL data processing is based on keyed READ operations following a positioning operation. Records are read until the key value being processed changes. Because traditional COBOL data processing is based on a B+ tree file system, the overhead for such operations is relatively minor.

RDBMS data processing introduces a new level of complexity to data processing. The database's *query optimizer* receives the SQL query for the COBOL operation being performed and then builds a *working set* of data that satisfies that query. Because the database optimizer has many different possible execution methods, this can result in poor performance if the optimizer chooses a query execution path that is less than optimal.

Performance degradation may also result from the fact that queries generated by COBOL operations result in *unbounded index queries*. Unbounded queries are generated because COBOL positioning operations (Start Not Less Than and Start Not Greater Than) provide only one of the bounding conditions for the working set, instead of both an upper and lower boundary.

As an example, consider the case where an application needs to process all items in a warehouse on aisle 17, shelf 8, and bin 2. If each of these items is a field in a key, the COBOL program might generate the following query for a READ operation:

Note: The following example applies to the Acu4GL for Oracle product. The SQL generated will be different for the different products.

```
SELECT * FROM warehouse_items WHERE
   aisle = 17 and
   shelf = 8 and
   bin = 2
ORDER BY aisle ASC, shelf ASC, bin ASC;
```

This query achieves the desired result but has one problem. For the COBOL program to end its processing, it must read a record that has a new value for *bin*. The COBOL application has no way of specifying an upper boundary for the read operation, so when all rows of data from bin 2 have been read, Acu4GL will attempt to read the next record by generating the following query:

```
SELECT * FROM warehouse_items WHERE
   aisle = 17 and
   shelf = 8 and
   bin > 2
ORDER BY aisle ASC, shelf ASC, bin ASC;
```

This query will cause the database query optimizer to gather all records pertaining to items on the remainder of shelf 8 to build its working set. This is excessive from the COBOL application's point of view, because the COBOL program needs only the *first record* of the working set to determine that it has finished processing.

This problem can be even more serious if the application is processing the last bin on a shelf. Because there are no more bins on that shelf, the query would drop down a level and generate the following:

```
SELECT * FROM warehouse_items WHERE
   aisle = 17 and
   shelf > 8
ORDER BY aisle ASC, shelf ASC, bin ASC;
```

This would select all items on the remainder of that aisle of the warehouse, which could be a very large working set if each aisle had 130 shelves!

In reality, most of the time the database query optimizer will not build the entire working set if it has been properly tuned, but will instead perform INDEXED READS to process the query. This means that the query optimizer will traverse an index tree to retrieve the records, much as COBOL index files do, as opposed to using combinations of indexes and sort and merge operations.

It can be helpful for the COBOL developer to influence precisely which information is to be returned. If the application developer knows at compile time (or before the query is executed) the precise scope of the record processing that needs to be performed by the read operations, the developer can more finely tune the information being retrieved.

Acu4GL provides a method by which the COBOL programmer can provide additional information to the database query optimizer by providing more specific selection information to the Acu4GL product. This selection information is added to the WHERE clause of the SQL queries generated by the Acu4GL product. This can be particularly useful in providing upper boundaries to queries being generated, with the result that the working set is smaller.

The developer may provide upper boundaries on the key segments for a select, or any other selection criteria needed to constrain the working set to just the desired subset of rows. This additional information is added to generated queries with the AND condition. It is not possible for the

application developer to specify a larger working set than would otherwise have resulted. The developer may only constrain the working set to a smaller subset.

Using WHERE constraints from COBOL

The following steps are required for using the WHERE constraint.

Step 1: Declare an external variable.

To make use of WHERE constraints from COBOL, the application must declare an external variable for communication with the Acu4GL product. This variable is declared as follows:

```
77  a4gl-where-constraint  pic x(300) external
```

Step 2: Modify your COBOL procedures.

Your COBOL application should move the information that you want added to the WHERE clause to the new external variable before a COBOL positioning operation such as START or READ is performed. The additional constraint will then be applied to any SQL read query performed on that file until a new positioning operation is performed.

The additional query information is also stored in Acu4GL's *cursor cache*, so that if the same read conditions occur in later processing, the existing closed cursor can be reused with new bind variables instead of being regenerated.

Be sure to fill the external variable before a positioning operation (START or READ). The WHERE constraint affects only READ NEXT operations preceded by a positioning operation. The WHERE constraint does not affect a READ NEXT that was not preceded by a positioning operation (such as a READ NEXT without a START immediately after opening a file).

Example

In your COBOL program you include this statement:

```
Move "ftest_key > 3 and ftest_key < 6" to
   A4GL_WHERE_CONSTRAINT.
Inspect A4GL_WHERE_CONSTRAINT replacing trailing spaces by
   low-values.
```

```
START FTEST-FILE KEY NOT LESS FTEST-KEY.
```

These results occur:

```
CURSOR 0:
SELECT *,ROWID FROM ftest WHERE (ftest_key1_seg1 = ? AND
ftest_key1_seg2 >= ?) AND (ftest_key > 3 and ftest_key < 6)
ORDER BY ftest_key1_seg1 ASC, ftest_key1_seg2 ASC;

CURSOR 1:
SELECT *,ROWID FROM ftest WHERE (ftest_key1_seg1 > ?) AND
(ftest_key > 3 and ftest_key < 6) ORDER BY ftest_key1_seg1 ASC,
ftest_key1_seg2 ASC;
```

Limitations

- WHERE constraints added by the COBOL program may not be portable between databases.
- The application may specify conditions of such complexity that they confuse the database query optimizer, resulting in poor performance. Be sure to examine the results of the optimizer trace facilities to ensure optimal performance.
- Take care to prevent the COBOL application from sending information that will result in a syntax error that will not be detected until runtime.
- The WHERE constraint should reference only columns in the primary table.

9.2 Troubleshooting

The remainder of this chapter lists the possible error messages that can occur during compilation. Recommended recovery procedures are given for each situation.

Note: It is possible that you will experience slower performance simply because the Acu4GL application is limited because of the rules of COBOL. If you want to keep track of performance levels, there are many third-party tools available to help monitor performance.

9.2.1 Compiler Errors

The errors listed below could occur when you compile with the "-Fx" option. *In some cases, the data dictionary cannot be built until you remove the error condition.* (Data dictionary errors do not, however, prevent the object code from being generated.)

Bad picture for DATE: *keyname*

The PICTURE must be six or eight bytes in length, either alphanumeric or numeric with no sign.

Data missing from key segment *keyname*

This occurs when some part of the named key cannot be placed in the dictionary; the dictionary cannot be generated in this situation.

This usually occurs because of filler. For example:

```
01  my-record.
    03  my-key.
        05  filler           pic xx.
        05  field-1          pic xx.
```

If my-key is declared as a record key, you will receive this error because the area of the key described by filler is not included in the dictionary.

To correct this error, ensure that every character that is part of the key is included in some field that is part of the dictionary. Use an XFD to give a field name to each filler, to ensure that fillers are included:

```
 01  my-record.
     03  my-key.
$xfd name=myfiller
         05   filler           pic xx.
         05   field-1          pic xx.
```

Directive word too long: *keyname*

With one exception the words contained in a directive, including field names, cannot exceed 30 characters. The value of a **WHEN** directive may consist of up to 50 characters.

`GROUP expected after USE`

The **USE GROUP** directive must include both words.

`Missing '=' in XFD directive`

The **NAME** directive requires an equal ("=") sign. The **WHEN** directive requires a comparison operator.

`Missing field name after WHEN`

A valid field name, or the word "OTHER", must be specified with the **WHEN** directive.

`xxx: unknown XFD directive`

The compiler did not recognize the directive you used. The *xxx* is the directive found. Check for a typographical error.

`Value should be a name: xxx`

This error occurs when the item to the right of an "=" should be a name, and it isn't. For example, it would be an error to use a quoted string with the **NAME** directive: $XFD NAME=some text.

The *xxx* in the message is the value found.

`Value should be numeric: xxx`

This error occurs when the item to the right of an "=" should be numeric and it isn't. The *xxx* in the message is the value found.

`Value should be a literal: xxx`

This error occurs when the item to the right of an "=" should be a literal, and it isn't. The *xxx* in the message is the value found. A literal is either a quoted string or a numeric integer.

`Variable file name requires File directive`

This message occurs when the compiler cannot assign a name to the .xfd file, because the ASSIGN phrase for the file names a variable file name. In this case, you must use a **FILE** directive to name the .xfd file.

WHEN variable xxx not found in record

This happens if you have a **WHEN** directive that mentions a variable that doesn't exist in the record.

9.2.2 Compiler Warnings

***xxx* not unique in first 30 characters**

This message occurs if a field name is not unique within the first 30 characters. The *xxx* is the name found. You can either change the field name or apply the **NAME** directive.

The warning listed below can occur when you compile with the "-Zx" or "-Fx" option. The data dictionary will be built, and Acu4GL will operate correctly. The warning informs you of a special database situation that is not advisable.

Field xxx causes duplicate database data

This warning means that your record definition should be restructured. Your current definition is set up in such a way that:

- you have overlapping key fields, and
- both keys must be represented in the database as *separate items*.

Acu4GL will handle this situation correctly. It will keep the overlapping keys updated simultaneously, so that they always have the same value. However, the warning alerts you that *you have the same data represented twice in the database*. This is dangerous, because someone at the site might access the database via SQL and accidentally change only one of the keys.

Here's an example of the problem and a description of how to correct it (the example assumes that both *key-1* and *key-2* have been declared as keys):

```
01  order-record.
    03  key-1.
        05  field-a                pic x(5).
        05  field-b                pic 9(5).
        05  key-2
            redefines field-b       pic x(3).
```

This example will generate the warning message.

Because *key-2* is a key, it must also be represented in the database. It doesn't correspond exactly to any other data field, so it must be entered as a separate column in the database.

In the COBOL view of the file, *key-1* and *key-2* overlap. But the requirements of database storage force the same data (known to COBOL as *field-b*) to be physically represented twice in the database. Any updates to the data from any ACUCOBOL-GT® program will correctly update both columns. Updates from *outside* of ACUCOBOL-GT carry no such guarantee.

In this example, you can correct the situation by breaking *field-b* into two columns, so that *key-2* corresponds exactly to another data field:

```
01  order-record.
    03  key-1.
        05  field-a                      pic x(5).
        05  field-b.
            07  field-b1                 pic x(3).
            07  field-b2                 pic 9(2).
        05  key-2
            redefines field-b             pic x(3).
```

9.2.3 Retrieving Runtime Errors

You can determine the meanings of your database error codes by referring to the database documentation. Here are methods for storing the complete error code and some helpful text that describes it.

You can retrieve a secondary error code by using selected runtime options, or by calling the library routine C$RERR (described in Appendix I in Book 4, *Appendices*, of the ACUCOBOL-GT documentation set). Note that you can pass two parameters to C$RERR for interface errors (rather than just one). The first parameter retrieves the code; the second parameter retrieves a message associated with the error condition.

Note: For Oracle users: Because 9D takes two bytes and Oracle secondary errors can be four bytes, the first parameter should be PIC x(6).

You can retrieve runtime errors in three ways, as described in the following sections:

- **Retrieving messages using the "-x" runtime option**
- **Retrieving messages using the debugger**
- **Retrieving messages using C$RERR**

Unless noted otherwise, these methods apply to all RDBMs supported by the Acu4GL family of interfaces. For a listing of runtime errors, refer to the "Troubleshooting" section of the appendix that pertains to your product.

9.2.3.1 Retrieving messages using the "-x" runtime option

At run time, if you specify an error file and use the "-x" option, the runtime puts the extended error code and some text associated with the error into the error file. (When run against an Informix database, you'll see three levels of error codes in the file: ACUCOBOL-GT error, Informix database error, and ISAM error.) A value of "zero" for any level means no error at that level.

For example:

```
runcbl -le errfile -x  myprog
```

where:

-l	causes the contents of the runtime configuration file to be included in the error output
e (or -e)	causes the error output to be placed in the file named immediately after the option
errfile	is the user-specified name of the error file
-x	causes the secondary error numbers to be included
myprog	is the name of your object file

The text of the error would then have this format in the file:

```
*** File system error value = 3 ***
*** Dictionary (.xfd) file not found***
File error 9D,03 on filename
Dictionary (.xfd) file not found
```

9.2.3.2 Retrieving messages using the debugger

Occasionally you may receive an error message that means syntax error. (In Informix, for example, this is error code 201.) You can examine the error file and determine the cause of the problem if you receive this error code. You'll need to rerun the program, specifying the options shown below, and turning on Trace Files (TF) when execution begins:

```
runcbl -dle errfile -x yourprog
```

Notice that the only change from Method One is the "-d" option, which turns on the debugger. The source code does *not* need to be compiled in debug mode.

After you press **Enter**, you will be at the debugger screen. Enter the following command:

```
tf  n
```

where *n* is a number from 1 to 9. The higher the number you enter, the more debugging information you receive. See Chapter 3 of the *ACUCOBOL-GT User's Guide* for more information on using the debugger.

"FILE TRACE ON" is echoed on the screen. Now enter the following command:

```
g
```

You are now running your program normally. Proceed until you encounter the error condition, and then exit. Your error file contains the error information described in Method One, above, and also contains the SQL queries that the Acu4GL product constructed. Examining these queries can help to determine the cause of the syntax error. Please contact Technical Services if you need help.

9.2.3.3 Retrieving messages using C$RERR

You might want to separate the error codes and their associated text, and store them in variables. The variables can then be displayed to the screen or handled in whatever way you deem appropriate.

You saw an example of the usage of C$RERR in the sample program in **Chapter 2, "Chapter 2: Getting Started."** The simplified example shown below uses the library routine C$RERR with two parameters to retrieve the complete error code (first parameter) and its associated text (second parameter).

Note: See "**For Informix**," at the end of this section, for code that is specific to Informix.

```
DATA DIVISION.
   .
   .
working-storage section.
01  file-status                  pic xx.
01  error-status.
    03 primary-error             pic x(2).
    03 secondary-error           pic x(40).
01  error-text                   pic x(40).

PROCEDURE DIVISION.
   .
   .
get-file-err.
call "C$RERR" using error-status, error-text, status-type.
display "FILE ERROR:", primary-error.
display "DATABASE ERROR:",  secondary-error.
display error-text.
accept omitted.
stop run.
```

Here's an example of the output you might get from this:

```
FILE ERROR: 9D
DATABASE ERROR: 1608
```

A network error was encountered when results were sent to the front end. Check the error log for more information.

For Informix

Remember that the extended code can consist of two parts (database error and ISAM error), separated by a comma. In the example shown below, we use the library routine C$RERR with two parameters to retrieve the complete error code (first parameter) and its associated text (second parameter). Then we use the UNSTRING verb to separate the code into its parts:

```
DATA DIVISION
   .
   .
   .
working-storage section.
01  file-status                          pic xx.
01  error-status.
    03 primary-error                     pic x(2).
    03 extended-error                    pic x(40).
01  secondary-error                      pic x(10).
01  isam-error                           pic x(40).
01  error-text                           pic x(40).

PROCEDURE DIVISION
   .
   .
   .
get-file-err.
   call "C$RERR" using error-status, error-text.
   unstring extended-error delimited by "," into
      secondary-error, isam-error.

   display "FILE ERROR:", primary-error.
   display "DATABASE ERROR:", secondary-error.
   display "ISAM ERROR:", isam-error.

   display error-text.
   accept omitted.
   stop run.
```

Here's an example of the output you might get from this:

```
FILE ERROR: 9D
DATABASE ERROR: 350
ISAM ERROR: 108
Index already exists on column.
```

Using the Informix error number and "finderr"

An additional method, also for Informix only, allows you to take the Informix error number and use the "finderr" syntax to discover error information. More information on this syntax can be found in your Informix documentation.

The syntax is:

```
finderr xxxxx
```

(where *xxxxx* is the Informix error number).

10 General Questions and Answers

Key Topics

10.1 Introduction

This chapter provides answers to some common questions that can apply to any RDBMS or ODBC-compliant data source supported by the Acu4GL® family of interfaces. Be sure to check the appendix pertaining to Acu4GL for ODBC or your RDBMS for additional information:

Appendix A: **Appendix A: Acu4GL for Informix Information**

Appendix B: **Appendix B: Acu4GL for Microsoft SQL Server Information**

Appendix C: **Appendix C: Acu4GL for Oracle Information**

Appendix D: **Appendix D: Acu4GL for ODBC Information**

Appendix E: **Appendix E: Acu4GL for Sybase Information**

Appendix F: **Appendix F: Acu4GL for DB2 Information**

10.2 Questions and Answers

Question: I can't seem to get Acu4GL to create the files in my database or ODBC data source. They keep coming up as Vision files.

Answer: Check to see that the **DEFAULT_HOST** variable is set in the runtime configuration file or the environment. Setting DEFAULT_HOST in the environment overrides the setting in the runtime configuration file.

Enter one of the following commands, depending on your operating system.

For UNIX systems:

```
runcbl  -vv
```

For Windows systems:

```
wrun32 -vv
```

to make sure that the version number of the Acu4GL interface for your RDBMS is reported. This tells you that the Acu4GL product has been installed successfully.

Note: In Windows environments, if "-vv" does not return the Acu4GL product information, make sure the runtime has the name you used and is the first so-named executable on the PATH, and that the ".dll" is in the same directory.

If you are running Acu4GL for Sybase, and "-vv" does not return the Sybase Acu4GL product information, make sure the linked runtime has the name you used and is the first so-named executable on the PATH. If you cannot locate a runtime that displays a Sybase Acu4GL product version number using "-vv", you must create one by re-linking the runtime. If you must do that, make sure that USE_SYBASE is set to "1" in the file "filetbl.c" before you relink.

Question: Can I use both an RDBMS or ODBC data source and Vision at the same time?

Answer: Yes, you can. In the configuration file, set **DEFAULT_HOST** to the file system you want the runtime to use automatically. Then, for select files, assign them to an alternate file system with the variable **filename_HOST**. For example, to put CUSTFILE into a specified RDBMS and everything else into Vision, you would add:

```
DEFAULT_HOST   Vision
CUSTFILE_HOST  name_of_file_system
```

where *name_of_file_system* is one of the following: Informix, Microsoft SQL Server, Oracle, ODBC, Sybase, or DB2.

Question: How can I find out what an error message is?

Answer: If you run your application with the "-x" option, you will receive extended error numbers that include those returned by your RDBMS or ODBC application.

Note: For Informix, two types of errors are returned: the Informix error and the ISAM error.

For Oracle, the error code must include five digits. If your error code does not already include five digits, you must add leading zeros to complete this requirement. For example, if your error code is code number 150, the syntax for this option would be OERR ORA 00150.

If you have sent the errors to an output file with the "-e" option, the runtime will also attempt to include the text that explains the errors. You can also use the C$RERR library routine. You can also retrieve error codes from within your COBOL program. See the example in **section 9.2.3, "Retrieving Runtime Errors."**

Question: Do my XFD files have to be in the same directory as my object files?

Answer: No. You can instruct the compiler to put the XFD files in an alternate directory with the "-Fo" compiler option. Then at run time, make sure you have the configuration variable XFD_PREFIX set to include that same directory. See **Chapter 8, "Chapter 8: Compiler and Runtime Options,"** for information on the "-Fo" option and the XFD_PREFIX configuration variable.

Question: Why aren't my KEYs being retrieved in the correct order?

Answer: If your KEY field is numeric or alphanumeric, you may have illegal data in the field. For example, if you've used LOW-VALUES or HIGH-VALUES to mark control records, those values are considered invalid and can cause the records containing them to be retrieved in an unexpected sequence.

To enable special values such as these to be processed, use the **BINARY** directive in front of the key field. This allows data of any classification to be processed. Either designate an individual field as binary, or specify USE GROUP, BINARY in front of a group of fields.

The method of storing variables declared as binary is database-specific. For example, for Informix databases, these are stored as char fields with an extra leading character that always contains a space. For Oracle databases, to use another example, variables declared as binary are stored as raw fields. Refer to your database documentation for information specific to your data source or RDBMS.

Question: Is it possible to use the same XFD file for data files with different names, if they all have the same structure? This would be useful when I create several customer files that use the same record definitions.

Answer: At run time, it is possible to use a single XFD for files that have different names. For example, suppose a site has customer files that have identical structures but different names (CUST0001, CUST0002, CUST0003, and so on). It's not necessary to have a separate XFD for each file, so long as their record definitions are the same.

The individual files can all be mapped to the same XFD via a runtime configuration variable called **XFD_MAP**. Here's how it works.

Suppose your COBOL application has a SELECT with a variable ASSIGN name, such as customer-file. This variable assumes different values (such as CUST0001 and CUST0002) during program execution.

Before compiling the application, you would use the FILE directive to provide a base name for the XFD. Suppose you provide CUST as the base. The compiler would then generate an XFD named cust.xfd. (The compiler always converts XFD names to lower case.)

To ensure that all customer files, each having a unique name, will use this same XFD, you make this entry in your runtime configuration file:

```
XFD_MAP  CUST* = CUST
```

The asterisk (*) in the example is a wildcard that matches any number of characters. Note that the extension .xfd should not be included in the map. This statement would cause the XFD cust.xfd to be used for all files whose names begin with CUST.

The XFD_MAP variable has this syntax:

```
XFD_MAP   [pattern = base-xfd-name]  ...
```

where pattern consists of any valid filename characters and may include "*" or "?". These two characters have special meanings in the pattern:

*	matches any number of characters
?	matches a single occurrence of any character

For example:

CUST????	matches CUST0001 and CUSTOMER; does not match CUST001 or CUST00001
CUST*	matches all of the above
CUST*1	matches CUST001 and CUST0001 and CUST00001; does not match CUSTOMER
*OMER	matches CUSTOMER; does not match CUST001 or CUST0001

The XFD_MAP variable is read during the open file stage of any Acu4GL products linked into the runtime.

A Acu4GL for Informix Information

Key Topics

A.1 Getting Started with Acu4GL for Informix

Important: Micro Focus does not provide the Informix libraries along with the Acu4GL® product. The Informix Libraries are necessary for providing a proper connection between Informix and ***extend*** products. If you do not already possess them, please consult your database documentation or database vendor for information on obtaining these libraries.

You need to order Embedded ESQL for C. If you are using the version 5.1 series, you need the version 5.1 0.UC1.client libraries. If you are using the version 7.2 series or higher, you need the version 7.2x or higher.

To find out whether you can use a client/server environment to communicate with other database versions, contact IBM.

Follow the directions provided in your Informix documentation for installation of the Informix libraries, or, depending on what your arrangement is with Informix, contact Informix Technical Support for any questions you may have.

Before you begin using Acu4GL for Informix on a new system, you must install and configure the Informix RDBMS. Your Informix vendor provides installation instructions.

Several steps that must be performed before you begin using Acu4GL for Informix on a new system are:

- Install and configure the Informix RDBMS.
- Install Acu4GL for Informix.
- Designate a database.

  Note: Acu4GL for Informix does *not* support ANSI-mode databases.

- Prepare and compile your COBOL program.

Your Informix vendor provides installation instructions for the Informix RDBMS.

Please refer to the file SERVERS_7.2 in the release directory of your Informix distribution for more information about version 7.2, 7.3, and above.

A.1.1 Installation Preparation

The Informix Acu4GL product is an add-on module that must be linked with the ACUCOBOL-GT® runtime system. (Note that your versions of Acu4GL and the runtime must be the same.) For this reason, you'll need a C compiler to install Acu4GL. Acu4GL is compatible with Informix versions 5.x1, 7.2, 7.3, 9 and later.

Note: Please make sure that you have obtained the proper Informix libraries for your version of Informix. See **Section A.1, "Getting Started with Acu4GL for Informix,"** for information on these libraries.

Acu4GL is shipped using either TAR or CPIO format, depending on the type of machine you have. The label on the product medium tells you which format has been used.

Create a directory on your machine to hold the product and then type one of the following commands:

```
tar xfv device
```

or

```
cpio -icvBd < device
```

In either case, *device* is the appropriate hardware device name (for example, /dev/rdiskette or /dev/rmt0).

Note: Sites using Texas Instruments System 1500 should add an uppercase "T" to the cpio options ("-icvBdT").

A.1.2 Installation Steps

To install the Acu4GL for Informix product, perform the following steps:

Step 1: Install Informix.

Install the Informix RDBMS and make sure it is operational. Micro Focus does not provide this product. Make sure you install the libraries in the appropriate directories, based on the Informix documentation.

We recommend that you obtain Interactive SQL (**ISQL**) or **dbaccess** if possible. This is not mandatory, but it will give you quite a bit of flexibility. **ISQL** allows you to do database work outside of COBOL, including interactive queries and reports. ACUCOBOL-GT's standard file utility, **vutil**, cannot be used on Informix tables.

One way to test the installation of Informix is to access Interactive SQL and examine the stores database.

Step 2: Create a new runtime.

Create a new runtime system that includes the Informix Acu4GL product.

Note: In the following directions, the term "runtime system" refers to the runtime shared object on systems, where the ACUCOBOL-GT runtime is a shared object, and to **runcbl** on other systems, where the runtime is static. The runtime is a shared object on the following systems: AIX 5.1 and later, HP-UX 11 and later, and Solaris 7 and later. To check, look at the contents of the "lib" subdirectory of your ACUCOBOL-GT installation. If the files "libruncbl.so" or "libruncbl.sl" reside in that directory, the runtime is a shared object on your system.

2a. Backup

Make a backup copy of your newly installed **runcbl** or **libruncbl.so**.

2b. Create a "Makefile.inf" file

The "inf_inst" program can be run to create a "Makefile.inf" file.

Note: Please read the reference section at the start of the "inf_inst" script for the latest information regarding the "Makefile.ini" and database or platform differences.

---

1. First, create the Makefile.inf file by entering the following command:

   ```
   sh inf_inst
   ```

   in the …/install directory.

2. Next, you are asked to enter the directory where ACUCOBOL-GT is installed. Type the information and press **Enter**.

3. After that, you are asked which version of Informix you have installed. You have four choices:

   - For version 5.x1, enter "5".
   - For version 7.2 32-bit, enter "7".
   - For version 7.3 or for version 9, enter "3".
   - For version 7.3 64-bit, enter "4".

4. Once you have entered the version, add the names of your C routines (".o"), if any, to the line starting with "SUBS=".

Note: The "if_inst" script generates a makefile that may require slight modifications. Make sure that the FSI_LIBS line in your makefile is correct.

Now you are ready to relink your ACUCOBOL-GT runtime.

2c. Link the runtime system

1. Make sure you are in the directory containing the ACUCOBOL-GT runtime system. Then, at the UNIX prompt, enter the following command:

   ```
   make clean
   ```

   to ensure that you have a clean directory in which to build your runtime.

2. Now enter the following command:

   ```
   make -f Makefile.inf
   ```

   or

   ```
   makerun
   ```

   This compiles "sub.c" and "filetbl.c" and then links the runtime system.

2d. Verify the link

1. Enter the following command:

   ```
   runcbl -vv
   ```

2. Check to see that the version of the Informix Acu4GL is reported to your screen. This will verify that the link was successful.

3. If no Acu4GL version number is displayed, this means that something isn't set up properly.

   a. Check filetbl.c to make sure that USE_INFORMIX is set to "1" (Step 2b).

   b. Then check Makefile for accuracy (see Step 2c), and relink (Step 2d).

Shared libraries

If you have relinked the ACUCOBOL-GT runtime and receive an error message of this type when you try to execute it:

```
Could not load library; no such file or directory

Can't open shared library . . .
```

this may mean that your operating system is using shared libraries and cannot find them. This can occur even if the shared libraries reside in the same directory in which you are currently located.

Different versions of the UNIX operating system resolve this issue in different ways, so it is important that you consult your UNIX documentation to resolve this error.

Some versions of UNIX require that you set an environment variable that points to shared libraries on your system. For example, on an IBM® RS/6000® running AIX® 4.1, the environment variable LIBPATH must point to the directory in which the shared libraries are located. On HP/UX, the environment variable that must be set to point to shared libraries is SHLIB_PATH. On UNIX SVR4, the environment variable is LD_LIBRARY_PATH.

Be sure to read the system documentation for your operating system to determine the appropriate way to locate shared libraries.

A second way to resolve this type of error is to link the libraries into the runtime with a static link. Different versions of the C development system use different flags to accomplish this link. Please consult the documentation for your C compiler to determine the correct flag for your environment.

Step 3: Copy runcbl to the correct directory.

If the runtime is a statically linked **runcbl**, copy the new executable to a directory mentioned in your execution path. This file needs to have execute permission for everyone who will be using the compiler or runtime. This step is not necessary when the runtime system is a shared library.

The ACUCOBOL-GT license file for the runtime ("runcbl.alc") and the license file for the Acu4GL product to Informix ("runcbl.ilc") must be copied into the same directory as the runtime executable.

If you rename your runtime executable, be sure to rename your license files to use the same base name, with the extensions unchanged. For example, if you rename your runtime to be "myprog", then the license file for the Acu4GL product for Informix should be renamed "myprog.ilc", and the license file for the runtime should be renamed "myprog.alc".

The remaining files can be left in the directory in which they were unloaded.

Step 4: Compile the sample program.

1. If you are using the C-shell, enter the command **rehash**.
2. Now try compiling and running the sample program with the following commands:

```
ccbl   -Zd -Fx  demo.cbl
runcbl demo.acu
```

Note: You can also run the sample program using runtime flags, if necessary.

A.1.3 Designating a Database

Use the configuration variable **DEFAULT_HOST** to establish Informix as your file system:

```
DEFAULT_HOST  informix
```

Next, select the particular Informix database to be accessed by your COBOL application. Identify this database to the runtime system via the **DATABASE** configuration variable:

```
DATABASE  database-name
```

The database you choose might be one that has been in use at the site for some time and already contains data, or it might be a new, blank database that has just been created and named.

Note: The Acu4GL product for Informix allows you to create a file with an OPEN OUTPUT statement, just as you can create Vision indexed files. The Informix equivalent of a Vision file is a table, not a database. You need to have an existing database for your Informix tables, just as you must have an existing directory for Vision files.

To create a new, blank database, use the tools available from Informix.

A.2 Filename Translation

As you prepare to work with Acu4GL for Informix, you may find it helpful to understand the rules around filename interpretation and to understand how the names of tables and XFD files are formed and work together.

When the ACUCOBOL-GT compiler generates XFD files, it uses lowercase letters to name the XFD file. In addition, the compiler changes hyphens to underscores when naming the XFD file.

Through configuration variables, the runtime translates the file name in the COBOL program into the filename that is passed to the **open()** function in the runtime. The **open()** function determines which file system to pass the request to, but does not change the name of the file. For additional information on configuration variables, see Appendix H in Book 4, *Appendices*, of the ACUCOBOL-GT documentation set.

At this point, Acu4GL for Informix translates the file name to lowercase letters and changes hyphens to underscores. This "new" name is the one that Acu4GL for Informix will use in the future for references to the database table.

A.3 Configuration File Variables

This section lists the runtime configuration file variables that are specific to Acu4GL for Informix. Configuration file variables that are generally applicable to any RDBMS with which Acu4GL communicates are discussed in **section 8.2, "Runtime Configuration Variables."** See **DEFAULT_HOST** in section 8.2 "Runtime Configuration Variables," and **DATABASE**, later in this chapter, for important information on setting these variables.

A_INF_DUPLICATE_KEY

This variable determines how Acu4GL for Informix handles the processing of alternate keys with duplicates. When reading on an alternate key with duplicates, it is possible that not all records will be returned. The probability of this scenario increases if the direction of the read changes (for example, switching from read next to read previous). This generally occurs when:

- The the data set was imported from an external file
- The the database has undergone large amounts of modifications, resulting in the re-use of row IDs
- The the initial query returned the record set in a way such that the records are not in row ID order

Three potential values can be set for this variable:

Note: These settings can affect performance. If you need assistance, see your Database Administrator.

Zero ("0")

If this variable is set to zero ("0"), no ROWID is added to the SELECT statements for reading in a forward direction, but a ROWID is added to SELECT statements for reading in a backward direction if the file is not a VIEW.

Note: We don't recommend setting this variable to zero ("0"), because ROWIDs may not be in ascending order. This method can be used if the direction in which the application reads records does not change, but it is unknown if this will work for all situations. We provide this method for backwards compatibility only.

One ("1")

If this variable is set to one ("1"), a ROWID is added to the ORDER BY clause of all SELECT statements reading tables on an alternate key that allow duplicates. A ROWID is not added if the file is a VIEW. This assures a predictable ordering of keys and is the default setting.

Two ("2")

If this variable is set to two ("2"), the primary key segments are added to the ORDER BY clause of all SELECT statements reading tables or VIEWs on an alternate key that allow duplicates.

A_INF_NO_TRANSACTION_ERROR

This configuration file variable allows you to enable or disable the "9D, 255 - Not in transaction" Informix error. The default setting for this variable is "1" and does not allow this message to be displayed. If this variable is set to zero ("0"), the 9D, 255 error is returned if a stop run caused a commit or rollback to be sent to Acu4GL for Informix.

A_INFORMIX_ERROR_FILE

This configuration file variable allows you to map errors using a text file to supplement the default method of providing errors. By adding the name of the file that contains the actual error mapping to A_INFORMIX_ERROR_FILE, database-specific errors are mapped to COBOL errors.

Note: Several messages related to COBOL I/O cannot be supplemented.

Example

A sample syntax for this configuration file variable would be:

```
A_INFORMIX_ERROR_FILE=INFerrs
```

where:

INFerrs is a file of a specified format containing a mapping of database-specific errors to COBOL errors. One such entry might be:

```
1 DUPLICATE_RECORD
```

DATABASE

DATABASE specifies the name of the specific database to be accessed. You cannot open any database files until you have set this variable.

Example

```
DATABASE stores
```

indicates the stores database is to be accessed.

Note: If you choose, you can set the DATABASE variable in the startup script for the session.

INF_LOGIN

INF_LOGIN indicates the user name under which you want to connect to the database system. This is an optional variable. If INF_LOGIN and INF_PASSWD are not set, a default login is performed. If these variables are set, there must be a matching UNIX login name and password.

Example

To connect to the database with the user name MYNAME, you would specify:

```
INF_LOGIN  MYNAME
```

in the configuration file.

See also

INF_PASSWD configuration variable

INF_PASSWD

The variable INF_PASSWD should be set to the password assigned to the database account associated with the user name specified by INF_LOGIN. If INF_LOGIN and INF_PASSWD are not set, a default login is performed. If these variables are set, there must be a matching UNIX login name and password.

Example

For example, if the account with the user name has the associated password “CW021535”, specify

```
INF_PASSWD  CW021535
```

in the configuration file or the environment.

See also

INF_LOGIN configuration variable

MAX_CURSORS

MAX_CURSORS is the number of cursors (parameterized queries) in Acu4GL's cursor cache. The default value is 100. The range is 1–100. If you set this variable to a number less than the maximum of 100, you may reduce the number you specify in the Informix system parameter OPEN_CURSORS by a like amount. The value of Informix's OPEN_CURSORS should exceed MAX_CURSORS by at least 7.

Example

To allow your application to use up to 23 cursors, specify

```
MAX_CURSORS   23
```

in the configuration file.

A.4 Informix Performance

Informix databases include many advanced concepts, such as parallel queries, multi-processor support, and virtual processors.

To increase performance for large sites on powerful multi-processor machines, Informix has changed some of the database default configurations, moving to a "cost based" optimization for its default query mode. This means that the query optimizer for Informix OnLine makes its decisions about optimization based purely on costs, without considering translation isolation mode. With this default, OnLine does not give preference to index scans (nested-loop joins and key base reads) over table scans (other join methods).

The behavior can be especially problematic in the case of benchmark testing. When the table is created, a mass insert of the records is performed on the database table. The indexes are created for the table, but the database has not yet gathered information about the distribution of data in the underlying table, or the usefulness of a particular index. The lack of statistical information within the database regarding the data in the tables results in possible poor decisions on the part of the query optimizer. The optimizer is

unable to determine that an index exists that best matches the requirements of the query and will choose an alternate execution approach that will cause drastically poorer performance of the benchmark application.

Example

As an example of this problem, we will examine performance times from ACUCOBOL-GT's "iobench" program.

Test	**Informix 5.1 SunOS 4.1.3**	**Informix 7.2 Intel 486 System 5.4**	**Informix 9**
Write Sequential	1.2	1.8	3.33
Sort Sequential	3.3	3.4	4.30
Load Index 1	13.8	16.1	24.74
Read Index 1	19.5	732.9	1085.90
Update Index 1	4.5	2.8	56.33
Load Index 2	24.2	29.6	28.33
Update Index 2	13.6	10.0	56.83
Total	80.1	796.6	1259.66

The times between the two machines and databases that the benchmark were run on are comparable for all times except for the read operation. The read operation shows a significant performance degradation.

Isolating the problem

To isolate the source of the problem, you can insert into your COBOL program a call to Acu4GL's utility program **sql.acu**. Before the Read benchmark test, we inserted the call:

```
CALL "sql.acu" USING SQL-COMMAND.
```

In the working-storage section, SQL-COMMAND is defined as follows:

```
77 SQL-COMMAND      pic x(75) value "SET EXPLAIN ON".
```

This call instructs the Informix database engine to print out the query optimizer's execution plan to a file in the current directory called "sqexpain.out". Here is a sample of the optimizer's output:

```
QUERY:
------
SELECT *, rowid FROM idx1 WHERE idx_1_key >= ? ORDER
BY idx_1_key ASC

Estimated Cost: 2
Estimated # of Rows Returned: 3
Temporary Files Required For: Order By

1) informix.idx1: SEQUENTIAL SCAN

    Filters: informix.idx1.idx_1_key >= '0000000900'
```

This output shows that the Informix query optimizer is performing a sequential scan on the table for each set of start/read operations. The optimizer is also making use of a temporary file during processing to sort the information in key order, as described by the "ORDER BY" clause. This clause is necessary to ensure that the COBOL application receives the records in the required sequence.

Problem resolutions

To correct the problems with the query optimizer, you can take one of several approaches:

1. Override the Informix database parameter in the **database configuration files**.
2. Override the new database parameter for a given **user session**.
3. Provide the **query optimizer** with more information so that it can choose a more efficient method of returning data.

Method 1: Configuration files

The Informix database engine reads a system configuration file each time it is started. Two configuration files apply: the file "onconfig.std" is used as a template for creating database configuration files when new databases are created; the "onconfig.<database>" file is the configuration file for a given database.

In the database configuration file, locate the line

```
OPTCOMPIND    2    # To hint the optimizer
```

Modify this line to read

```
OPTCOMPIND    0    # To hint the optimizer
```

The OPTCOMPIND configuration parameter helps the optimizer choose an appropriate join method for your application. A setting of "0" indicates that when appropriate indexes exist, the optimizer chooses index scans (nested-loop joins), without considering the cost, over table scans (sort-merge joins or hash joins).

This new setting takes effect the next time you shut down and restart the ONLINE database engine. The altered setting is then applied to all operations with the query optimizer. To ensure that this change applies to newly created databases, you should modify the "onconfig.std" file.

Method 2: Altering a user session

You can modify how the query optimizer executes queries on an individual basis by setting a UNIX environment variable in the user's environment. This environment variable should be set *before* executing your COBOL application. It cannot be set with the COBOL "SET CONFIGURATION" or "SET ENVIRONMENT" verbs. Use the syntax "setenv OPTCOMPIND 0" or "OPTCOMPIND=0; export OPTCOMPIND", depending on which shell is being used.

Method 1 and method 2 resulted in the following "iobench" times:

Test	**Informix 5.1 SunOS4.1.3**	**Informix 7.2 Intel 486 System 5.4**	**Informix 9**
Write Sequential	1.2	1.8	3.25
Sort Sequential	3.3	3.3	4.33
Load Index 1	13.8	16.6	24.92
Read Index 1	19.5	22.4	47.23
Update Index 1	4.5	4.1	5.57
Load Index 2	24.2	22.9	28.19
Update Index 2	13.6	10.9	6.22
Total	80.1	81.9	119.71

We can verify the changes that were made by examining the output of the query optimizer. The above test resulted in these results:

```
QUERY:
------
SELECT *, rowid FROM idx1 WHERE idx_1_key >= ? ORDER BY idx_1_key ASC
Estimated Cost: 4
Estimated # of Rows Returned: 3
1) informix.idx1: INDEX PATH
    (1) Index Keys: idx_1_key
        Lower Index Filter: informix.idx1.idx_1_key >= '0000000900'
```

Both method 1 and method 2 above are useful if you are processing new data. They have the drawback, however, that they permanently constrain the execution paths that the query optimizer has to choose from. To allow the query optimizer the greatest flexibility in working with data that exists day-to-day on your system, you will want to choose the next method.

Method 3: Provide the query optimizer with information

Database tables that are in use on a frequent basis can benefit from providing the query optimizer with more information. This allows the optimizer to work at its best in returning information from the database. You provide the optimizer with information of the tables by issuing the following SQL command:

```
UPDATE STATISTICS HIGH FOR TABLE <tablename>;
```

Executing this command updates the Informix system database catalog tables SYSTABLES, SYSCOLUMNS, SYSINDEXES, and SYSDISTRIB. The optimizer uses this data to determine the best execution path for queries. The database server does not update this statistical data automatically, however. Statistics are updated only when you issue an UPDATE STATISTICS statement.

Informix recommends that you run UPDATE STATISTICS in high mode for all columns that head an index. For the fastest execution time of the UPDATE STATISTICS statement, you must execute one UPDATE STATISTICS statement in the high mode for each such column. For each multi-column index, run UPDATE STATISTICS in low mode for all of its columns.

You may want to perform the UPDATE STATISTICS on the entire table periodically as shown above, instead of just on the indexes.

For more information on optimizing Informix performance and general maintenance issues, please refer to your Informix-OnLine documentation.

A.5 Technical Tips

Switching file systems

Each time a file is opened, the file system identified by the *filename*_HOST configuration variable (if present) or the DEFAULT_HOST variable is used. You can change the value of these variables in your code by including:

```
SET ENVIRONMENT "filename_HOST" TO filesystem
```

or

```
SET ENVIRONMENT "DEFAULT_HOST" TO filesystem
```

just before you open the file. SET ENVIRONMENT thus enables you to change file systems during the execution of your program. The *filesystem* value for Informix is "Informix". The value for Vision is "Vision".

If you change to the Informix file system in the midst of a program, be sure to specify the database to be used:

```
SET ENVIRONMENT "DEFAULT_HOST" TO "INFORMIX"

SET ENVIRONMENT "DATABASE" TO "STORES"
```

Note that the database cannot be changed if there are tables open in the active database. *Be sure to close all tables before making a change.*

Remember that SET ENVIRONMENT is not the typical way to specify a file system. Normally the file system is designated in the runtime configuration file and is not changed in the COBOL program.

A.6 Supported Features

OPEN ALLOWING READERS is not supported by Informix. You determine in your runtime configuration file how this phrase will be interpreted. Set the variable STRENGTHEN_LOCKS to "1" (one) to cause this phrase to be treated as OPEN ALLOWING NO OTHERS. Set the variable to "0" (zero) to cause the phrase to be treated as OPEN ALLOWING ALL. The default value is "0".

Only single-record locking is supported, unless the program is within a transaction.

If you attempt to REWRITE a record that contains a SERIAL data type, you will receive an Informix error. You might want to change SERIAL fields to SMALLINT, as we do for the orders table in the demonstration. However, you may encounter the situation in which the user has a table with a serial column and cannot change it. There may be a solution to this, and the solution is determined by whether you need to reference this serial column when you do your READs.

If the user does not need to access these columns directly, you can build your FDs without this field. Because the current version of Acu4GL for Informix lists the columns from the XFD, and Acu4GL for Informix allows the table to have more columns than the FD, it should be possible for you not to include the serial columns in the FD. This has the effect that, on an INSERT, the unmentioned serial columns would be populated by the next sequential value.

If it is necessary for the user to access these columns, consider having two FDs. One FD would list the serial column, included for reading. Then use another FD, this time without the serial column, for your writes or rewrites.

Informix does not support record encryption, record compression, or alternate collating sequences. You may include these options in your program; they will be disregarded if they are specified.

The ACUCOBOL-GT utility program **vutil** cannot be used with Informix files. Instead, use utilities supplied by Informix.

Whenever you are using the library routine RENAME, you must specify that you are using *indexed files*. This information is passed by the value "I" in the fourth parameter. If you want to delete an Informix table with the DELETE FILE verb, make sure the verb references an *indexed file*.

Acu4GL passes all transaction operations on to the database. Passing all transaction flags to the database may have the unexpected effect of releasing a lock because of the COMMIT of a transaction. Acu4GL for Informix performs REWRITE and DELETE operations on the current record through the record lock using the syntax "WHERE CURRENT OF". If this is a problem, Acu4GL can be told to use an alternate method of explicitly specifying the primary key in the REWRITE and DELETE operations by setting the configuration variable: "4GL_POSITIONED_MODIFICATIONS = 0". The default is "1". You should note that database systems will not perform well with applications compiled with the "-ft" option. You should consider explicitly coding the transactions.

A.7 Limits and Ranges

The following limits exist for the Informix-SE file system:

Maximum indexed key size: 120 bytes
Maximum number of fields per key: 8

The following limits exist for the Informix-OnLine file system:

Maximum indexed key size: 256 bytes
Maximum number of fields per key: 16

With the Acu4GL product for Informix, only one database can be open at a time. Within this one database, a maximum of 100 tables can be open at the same time. You can set this table maximum to a lower number via the runtime configuration variable **MAX_CURSORS**.

The following Informix data types are not currently supported:

INTERVAL
BLOB
FLOAT
SMALLFLOAT

Acu4GL for Informix supports the following data types; conversions between COBOL and database formats are as shown:

COBOL to	**INFORMIX**
PIC X PIC XX	CHAR VARCHAR
PIC 9 PIC 99X PIC 999 PIC 9999	SMALLINT
PIC 9(5) PIC 9(6) PIC 9(7)X PIC 9(8) PIC 9(9)	INTEGER
All other PIC 9's	DECIMAL

INFORMIX to	**COBOL**
INTERGERX	PIC S9(10) or PIC S9(9) COMP-4
SMALLINT	PIC S9(5) or PIC S9(4) COMP-4
DECIMAL (6,2)	PIC (4)V99
MONEY(4)	PIC (2)V99
SERIAL	PIC 9(9)
DATE	PIC 9(6) or PIC 9(8)
DATETIME	PIC 9(12) or PIC 9(14)
VARCHAR (max, min)	PIC X(max)

Opening a file I-O exclusive doesn't hold lock on Informix. This is a limitation on the way the Informix Acu4GL product works. Since we do not do any actual operations on the table except to get information about it, Informix does not tell us the file is locked. Users should get a lock condition when they do a read with lock.

Performing error handling on the read/write will yield a 99 record locked on any read operation performed—even if the file is open for input.

Other limits are described in Appendix B in Book 4, *Appendices*, of the ACUCOBOL-GT documentation.

A.8 Runtime Errors

This section lists the Acu4GL error messages that could occur during execution of your program. **Chapter 9** provides information on compile-time errors and also provides several methods for retrieving runtime errors.

An explanation and a recommended recovery procedure follow each message.

Runtime errors will have this format:

9D,xx

The 9D indicates a file system error and is reported in your FILE STATUS variable. The *xx* is a secondary, or extended, error code. You can retrieve an extended error code by using selected runtime options or by calling the library routine C$RERR. Note that you can pass two parameters to C$RERR for interface errors (rather than just one). The first parameter retrieves the code; the second parameter retrieves a message associated with the error condition. This process is explained in detail on the following pages.

When the extended error code is greater than 99, the error is explained in the Informix documentation. Error codes less than 99 are explained here:

01 DATABASE is not defined in the environment

You must specify which database you are using. Use the **DATABASE** configuration variable.

02 Attempt to open more than one database at once

This can happen if you specify a database, open a file, then specify a different database with SET ENVIRONMENT, and try to open another file. Informix does not allow files to be open from two databases simultaneously.

03 Dictionary (.xfd) file not found

The dictionary file for one of your COBOL files cannot be located. Be sure you have specified the correct directory via your **XFD_PREFIX** configuration variable. You may need to recompile with the "-Fx" option to re-create the dictionary. See **section 8.1** for information on compiler options.

04 Corrupt dictionary file

The dictionary file for one of your COBOL files is corrupt and cannot be read. Recompile with "-Fx" to re-create the dictionary.

05 Too many fields in the key
(more than 8 for Informix-SE, more than 16 for Informix-OnLine)

Check your key definitions and redefine the key that is illegal, and then recompile with the "-Fx" option.

There are additional 9D extended error numbers; these have values of 100 or more. Often they have two parts that are separated by a comma (such as 9D 350,108). These are Informix error codes. The first part is the database error; the second part (if any) is the ISAM error.

Informix database error numbers are typically greater than 200. ISAM errors, when present, fall between 100 and 199.

Note: See your Informix documentation regarding the "finderr" syntax for more information.

---

06 Mismatched dictionary file

The dictionary file (.xfd) for one of your files conflicts with the COBOL description of the file FD. The *xx* indicates a tertiary error code that is defined by the host file system. You can determine the exact nature of the mismatch by referring to the host system's error values.

The tertiary error code may have any of these values:

01 – mismatch found but exact cause unknown (this status is returned by the host file system)

02 – mismatch found in file's maximum record size

03 – mismatch found in file's minimum record size

04 – mismatch found in the number of keys in the file

05 – mismatch found in primary key description

06 – mismatch found in first alternate key description

07 – mismatch found in second alternate key description

The list continues in this manner for each alternate key.

255 Not in transaction

A stop run has caused a commit or rollback to be sent to Acu4GL for Informix. By default, this message is disabled. If you would like to enable this message, set the **A_INF_NO_TRANSACTION_ERROR** variable to zero ("0").

A.9 Common Questions and Answers

This section contains some questions and answers specific to Acu4GL for Informix. Refer to **Chapter 10** in this book for additional questions and answers that pertain to the Acu4GL family of products.

Question: How do I create new databases?

Answer: You have two choices: (1) you can use an Informix product, such as **ISQL**, to create a new database, or (2) you can use the program **sql.acu** provided with your Acu4GL product.

To use the **sql.acu** utility to create an empty database and grant access privileges to other users, enter the following command:

```
runcbl sql.acu
```

The program pauses to accept an SQL command. Enter the following:

```
CREATE DATABASE database-name
```

Note: If you want to use the transaction logging facility available in ACUCOBOL-GT, you must enable it for the database you are creating. The precise SQL command depends on the Informix engine you are using, as shown below.

For Informix-OnLine, enter the following command to use transaction logging:

```
CREATE DATABASE database-name WITH LOG;
```

For Informix-SE, enter the following command to use transaction logging:

```
CREATE DATABASE database-name WITH LOG IN
   '/acct/f1992/acct_log';
```

If you are using an Informix database with transactions enabled, you *must* use ACUCOBOL-GT's transaction management capabilities. (Either use START TRANSACTION and COMMIT, or compile with "-fs", or use the **sql.acu** program to issue BEGIN WORK and COMMIT.) Using a transaction-enabled database without ACUCOBOL-GT's transaction management capabilities results in the records not being locked and can generate error messages.

The program pauses to accept an SQL command. Now enter the following:

```
GRANT DBA TO PUBLIC
```

Then press **Enter** again to exit the program.

The database name may be up to ten characters and must contain only letters, digits, and underscores. The first character must be a letter.

The statement GRANT DBA TO PUBLIC gives the Database Administrator access privileges to all other users.

Question: Is it possible to use both Informix-OnLine and Informix-SE on the same machine?

Answer: Yes. You'll need to tell the runtime which database engine to use. You do this by setting the environment variable SQLEXEC as shown here:

Informix-OnLine

```
SQLEXEC =  ${INFORMIXDIR}/lib/sqlturbo
```

Informix-SE

```
SQLEXEC =  ${INFORMIXDIR}/lib/sqlexec
```

Setting SQLEXEC tells the Informix utility programs and the ACUCOBOL-GT runtime which engine to access. This variable must be set before you execute the runtime. Only one engine (either OnLine or SE) can be used for any given execution of the runtime.

INFORMIXDIR is an environment variable that you must set to the location of your Informix product.

Question: What files do I need to link my C routines into Acu4GL?

Answer: From your ACUCOBOL-GT runtime medium you need:

sub.c
sub85.c
filetbl.c
config85.c
sub.h
libruncbl.a
libvision.a
libacuterm.a
clntstub.o

If you are using AcuServer™, instead of linking clntstub.o, see the relinking instructions in your *AcuServer User's Guide*.

From your Acu4GL medium, you need all of the following that are present:

infemb7.o - for version 7.2
infemb73.o - for version 7.3
infemb.oo - for version 5.1
inf.o
cur.o
* libsql.a
* libgen.a
* libos.a
* libasf.a
* netlib.a

* These files are provided by Informix and should be located in the appropriate Informix directory.

Instructions for linking are given in **section A.1.2, "Installation Steps."**

Question: I could not find the secondary error number for a 9D in the documentation. What does the number mean?

Answer: The errors fall into three categories:

01 – 99 are Acu4GL codes and are described in **section A.8** of this manual.

100 – 199 are ISAM errors and are explained in Informix manuals.

200 and up are Informix database errors and are explained in Informix manuals.

Question: Can the Acu4GL for Informix product support a full date/time format?

Answer: The finest time granularity that Informix will support is one hundredths of a second. To achieve this, you must be sure the code is correct in the **DATE** directive and specify a date-format-string in your COBOL application, as opposed to just a date. See **section 4.3.5, "DATE,"** for additional information.

Question: Are there any ACUCOBOL-GT library routines that do not work with, or would not make sense to use with Acu4GL for Informix?

Answer: Yes. There are two ACUCOBOL-GT library routines that either don't work with or do not make sense to use with Acu4GL for Informix: C$COPY and C$RECOVER.

B Acu4GL for Microsoft SQL Server Information

Key Topics

B.1 Microsoft SQL Server Concepts Overview

A quick overview of some basic design concepts underlying the Microsoft SQL Server Database Management System will help you interface your COBOL program to it.

Servers

A Microsoft SQL Server "server" is one copy of the database engine executing on a computer. A server has a name, and when a program wants to access the database controlled by a server, the program asks for a connection to that server by name. Multiple servers can be executing on a single machine, controlling different databases. The default name that Microsoft SQL Server gives to a server is DSQUERY. The naming of servers is discussed in section B.4 "Acu4GL for Microsoft SQL Server Configuration File Variables" under the configuration variable **A_MSSQL_DEFAULT_CONNECTION**.

Table ownership

Table names in Microsoft SQL Server have the form *database.owner.table_name*. Within Microsoft SQL Server, if you are the owner of a given table, you can refer to it as just *table_name*. If you are *not* the owner, you must refer to it with the *owner* of the table as a prefix. Different owners can thus have tables of the same name. However, this is *not* true when you use the Acu4GL® for Microsoft SQL Server interface.

Acu4GL for SQL Server works a little differently. It automatically determines the owner name it will use to reference a table. It is therefore essential that there *not* be multiple tables with the same name in a single database, even though the tables have different owners. If there are multiple tables, the Acu4GL for Microsoft SQL Server product will not necessarily find the correct one, and no diagnostic will be issued.

Note that table names include dots (.) as separators. Because of this, you must make sure there are no extensions on COBOL file names that will be converted to table names. For example, if you were to have a COBOL file named IDX1.DTA, Acu4GL for Microsoft SQL Server would attempt to open a table DTA with owner IDX1. You can avoid this problem either by

renaming your COBOL file in your source program, or by using an ACUCOBOL-GT® runtime configuration file variable to map the file name to an allowable file name, such as:

```
IDX1.DTA   IDX1
```

In the above example, IDX1.DTA is the name in the ASSIGN clause of the file's SELECT statement.

If you map your file name to a new name, we recommend that you simply drop the extension to form the new name. Here's why. The compiler uses the base file name—without the extension—to create the XFD file name (IDX1.XFD). The runtime needs to be able to locate this file. But if you've mapped the file name to something completely different (such as MYFILE), the runtime will look for an XFD file named MYFILE.XFD. So you'd have to remember to change the name of IDX1.XFD to MYFILE.XFD in the XFD directory. Save yourself this extra step by simply dropping the extension when you map the name. Also, see the configuration variable **4GL_IGNORED_SUFFIX_LIST** for an alternate method of removing file extensions.

Security

Security is implemented in the Microsoft SQL Server RDBMS. A user is required to log in to the RDBMS before any file processing can occur. Acu4GL for Microsoft SQL Server provides both a default and a user-configurable method for implementing this.

Generally, it is best for someone with Database Administrator (DBA) privileges to create and drop the tables, allowing others only the permissions to add, change, or delete information contained in them.

See the Microsoft SQL Server documentation for more details on DBA privileges.

B.2 Installation and Setup

You must perform several steps before you begin using Acu4GL for Microsoft SQL Server on a new system. The following topics provide this information:

- **Installing on a Client Machine**
- **Setting Up a User Account**
- **Setting Up the User Environment**
- **Designating the Host File System**

The Microsoft SQL Server RDBMS, version 6.5 or later, must be installed and configured *prior* to the installation of Acu4GL for Microsoft SQL Server.

Microsoft SQL Server's "Query Analyzer" product, an interactive query tool, is also necessary for installing the ACUCOBOL-GT stored procedures.

Micro Focus does not provide these products.

Note: The default sort order for Microsoft SQL Server is case-independent. Make sure that this is what you really want when you install the server. Acu4GL for Microsoft SQL Server will use the sort order as installed and cannot change its behavior to do case-dependent ordering if that is not what the server uses.

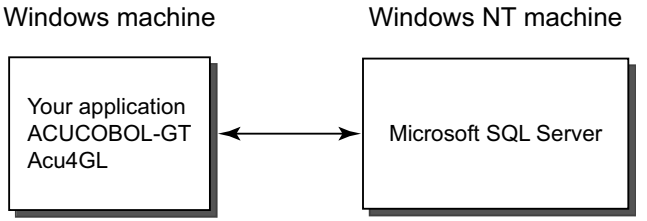
First you must install the files from the ***extend*** installation media onto the client machine. Then follow the instructions below for the server machine. Final setup steps on the client machine complete the installation.

Be sure to use the 64-bit libraries with a 64-bit runtime and 64-bit Acu4GL for Microsoft SQL Server, and the 32-bit libraries with a 32-bit runtime and 32-bit Acu4GL for SQL Server.

B.2.1 Installing on a Client Machine

The Acu4GL interface for Microsoft SQL Server is an add-on module. The product installation includes a .DLL file that is detected at run time. It is not necessary for 32-bit users to relink the runtime.

Acu4GL for SQL Server can be executed on a machine that is running any 32-bit Windows platform from Microsoft. The following diagrams show the supported machine configurations:



Note that the client machine may be the same machine as the server.

Installation instructions for each of these configurations are given in the next section. The instructions describe the steps you must follow on both the client machine and the server machine.

CD-ROM installation

Instructions for installing your Acu4GL product from the ACUCOBOL-GT CD-ROM are contained on the *Quick Start* card that accompanied the product. Please refer to it for installing your ***extend*** products.

Once the installation is complete, please return to this appendix for setting up your Acu4GL product.

To create the lock tables and stored procedures on a Windows server

Complete the following steps to create the lock tables and stored procedures on a Windows NT server machine.

Step 1: Install SQL Server

Microsoft SQL Server, version 6.5 or later, must be installed and configured on the Windows NT server machine *before* you install Acu4GL for Microsoft SQL Server on the client machine. Follow the instructions from your RDBMS vendor.

Step 2: Copy the batch file

MS_INST.CMD is a batch file from ACUCOBOL-GT that creates the MS_INST.SQL file, which is the collection of stored procedures necessary for executing the Acu4GL product. Copy MS_INST.CMD to your server machine into a directory of your choice.

Step 3: Execute the batch file

To execute the batch file, enter

```
MS_INST LockDatabase
```

where *LockDatabase* is the database you want to use for the internal ACUCOBOL-GT lock tables. If this database does not already exist, it will be created.

Everyone who will use the Acu4GL for Microsoft SQL Server product must have write access to this database.

This step creates MS_INST.SQL, the collection of stored procedures necessary for executing Acu4GL for SQL Server.

Step 4: Install the ACUCOBOL-GT stored procedures

To install the ACUCOBOL-GT stored procedures in your Microsoft SQL database, execute the generated MS_INST.SQL file with the SQL Query Analyzer or the SQL Server Management Studio if you are using Microsoft SQL Server 2005.

By default, the stored procedures are installed into the master database. However, you may choose another database in which to store them.

This completes the setup on the Windows NT server machine.

Note: If you are upgrading from an earlier version of Acu4GL, be sure to install the new stored procedures. Micro Focus always upgrades stored procedures in such a way that they are compatible with older versions of the product, so installing new stored procedures over old ones does not affect your ability to run with an older version of the interface software. Your new version of Acu4GL for Microsoft SQL Server may not run properly without the corresponding stored procedures.

It can be difficult to maintain multiple copies of stored procedures; therefore, we recommend that you continue to create the stored procedures in the master database. If your installation does not permit this, you do have the flexibility to create the stored procedures elsewhere. However, to facilitate maintenance of the stored procedures, we recommend that you create as few databases as possible.

Installation steps for a Windows client

Installation instructions for Microsoft SQL Server are provided by Microsoft and need to be read and understood to install the Windows client. Be sure to choose your communication method at the client machine. To do this, follow the instructions provided with your SQL Server software.

B.2.2 Setting Up a User Account

Acu4GL for Microsoft SQL Server must be able to connect to a user account. You may either set up one general account for all users, or an account for each individual user. To set up an account, you must have DBA privileges.

See **sp_addlogin** and **sp_adduser** in the *Microsoft SQL Server Commands Reference Manual* for additional information on setting up user accounts.

B.2.3 Setting Up the User Environment

The user's account should have been set up correctly to access the Microsoft SQL Server RDBMS system. This includes environment variables such as DSQUERY. See your Microsoft SQL Server documentation for more details.

In addition to setting the variables required for Microsoft SQL Server, you must do the following:

- Ensure that your execution path contains the name of the directory in which you placed your Acu4GL-enabled runtime.
- Set the **A_MSSQL_LOGIN** and **A_MSSQL_PASSWD** variables, either in your environment or in the ACUCOBOL-GT runtime configuration file. (If you don't do this, Acu4GL will use the value of the USER environment variable as your Microsoft SQL Server login name, with no password). For security reasons, it is best to set the password variable from your COBOL program by asking the user to enter a password and then executing

  ```
  SET ENVIRONMENT "A_MSSQL_PASSWD" TO user-entry
  ```

- You may want to make and use a personalized copy of the configuration file to avoid impacting other users. The *ACUCOBOL-GT User's Guide* describes how to use the A_CONFIG environment variable, or the "-c" runtime option, to identify a personal configuration file.

For detailed information on **A_MSSQL_LOGIN** and **A_MSSQL_PASSWD**, see section B.4, "Acu4GL for Microsoft SQL Server Configuration File Variables", in this appendix.

B.2.4 Designating the Host File System

If you are opening an *existing* file, all file systems linked into the runtime will be searched for the named file. If, however, you are creating a *new* file, you will need to tell the runtime which file system to use. You accomplish this with one of two runtime configuration file variables:

```
DEFAULT_HOST filesystem
```

or

```
filename_HOST filesystem
```

DEFAULT_HOST

Use the DEFAULT_HOST variable to designate the file system to be used for newly created files that are not individually assigned. For example,

```
DEFAULT_HOST MSSQL
```

means that all new files will be Microsoft SQL Server tables unless otherwise specified by the second configuration variable, *filename*_HOST.

*filename*_HOST

Use the *filename*_HOST variable to assign an individual data file to a file system. Any file so assigned will use the designated file system, and not the one specified by DEFAULT_HOST. The syntax is:

```
filename_HOST filesystem
```

where *filename* is the file name, without any extension, named in the ASSIGN TO clause of your SELECT statement. For example,

```
myfile_HOST VISION
```

means that *myfile* will be under the Vision file system.

You can use these runtime configuration file variables in combination to assign your new files in a default with exceptions manner; for example, this set of entries:

```
DEFAULT_HOST VISION
afile_HOST MSSQL
bfile_HOST MSSQL
```

means that all new files except *afile* and *bfile* will be assigned to Vision, and those two files will be assigned to Microsoft SQL Server.

You can also change the values of these variables during program execution by including in your code:

```
SET ENVIRONMENT "filename_HOST" TO filesystem
```

or

```
SET ENVIRONMENT "DEFAULT_HOST" TO filesystem
```

This enables you to change file systems during the execution of your program. This is not the typical way to specify a file system; normally it is designated in the runtime configuration file and is not changed in the COBOL program.

Note: The ACUCOBOL-GT interface to Microsoft SQL Server allows you to create a Microsoft SQL Server table with an OPEN OUTPUT statement, just as you can create Vision indexed files. The Microsoft SQL Server equivalent of a Vision file is a table, not a database. You must create a database for your Microsoft SQL Server tables before you run the COBOL program that creates the tables, just as you must create a directory for your files before you run a COBOL program that creates Vision files.

You are now ready to use the **sql.acu** program, as defined in **Chapter 2**. After you learn about and use this utility, you will next find out about preparing and compiling your COBOL program, followed by learning to use the demonstration program, which can also be found in Chapter 2.

B.3 Filename Translation

As you prepare to work with Acu4GL for Microsoft SQL Server, you may find it helpful to understand the rules around filename interpretation and to understand how the names of tables and XFD files are formed and work together.

When the ACUCOBOL-GT compiler generates XFD files, it uses lowercase letters to name the XFD file. In addition, the compiler changes hyphens to underscores when naming the XFD file.

Through configuration variables, the runtime translates the file name in the COBOL program into the filename that is passed to the **open()** function in the runtime. The **open()** function determines which file system to pass the request to, but does not change the name of the file.

However, Acu4GL for Microsoft SQL Server needs the name of the file to find the appropriate XFD file. To do this, Acu4GL for Microsoft SQL Server changes the name of the file to lowercase letters and changes hyphens to underscores. Note, however, that this is performed only on a local copy of the file. Once the XFD file is found, the filename reverts to the name that was originally passed to the **open()** function. Characters that are illegal in identifiers, such as a hyphen ("-"), are trapped by the database, and Acu4GL for Microsoft SQL Server will neither find the file nor create a new one.

B.4 Configuration File Variables

This section lists the runtime configuration file variables that are specific to Microsoft SQL Server. Configuration file variables that are generally applicable to any RDBMS with which Acu4GL communicates are discussed in **section 8.2, "Runtime Configuration Variables."**

A_MSSQL_ADD_IDENTITY

When is set to the default of "On" (true, yes), A_MSSQL_ADD_IDENTITY adds an extra column to any table created by the Acu4GL for Microsoft SQL Server product. The extra column will have the identity property and will be included on all indexes that are not unique. Otherwise, when A_MSSQL_ADD_IDENTITY is set to "False" (false, no), no extra column is added.

Note: The COBOL FD should not include the identity column.

Example

```
A_MSSQL_ADD_IDENTITY TRUE
```

On keys that allow duplicates, this variable has been found to vastly improve performance. Note that the default value is "True" for a file that allows duplicates, and is "False" for a file with no duplicate keys.

A_MSSQL_ADD_TIMESTAMP

Using a timestamp column is the only way to absolutely ensure that modifications made to a row are not overwriting someone else's changes. When it's reading a table that is open for I/O, the Acu4GL for Microsoft SQL Server product uses BROWSE MODE if a timestamp column exists. When the Acu4GL product is creating a table, if the value of A_MSSQL_ADD_TIMESTAMP is TRUE, a timestamp column is included in the table. (Note that your COBOL FD should *not* include the timestamp column.) While the default value is "Off" (false, no), this configuration variable can also take values of "On" (true, yes).

Note: Microsoft discourages the use of BROWSE MODE on Select statements because of performance reasons. Therefore, unless it is absolutely necessary, we discourage setting this variable to "On".

Example

```
A_MSSQL_ADD_TIMESTAMP  1
```

Note: If you do not use this option, be prepared for REWRITE statements to fail with an error stating that someone else has modified the row. Because of the AcuLocks tables, this can happen only from non-ACUCOBOL-GT applications.

A_MSSQL_APPROLE_NAME

This variable, in conjunction with A_MSSQL_APPROLE_PASSWD, allows Acu4GL to use approles. Before connecting to a database, set A_MSSQL_APPROLE_NAME to the name of a role. Note that you must also set A_MSSQL_APPROLE_PASSWD to a password for that role. By using these two variables, you set the runtime to use this approle.

For more information on approles, please see the SQL Server documentation.

See also

A_MSSQL_APPROLE_PASSWD

A_MSSQL_APPROLE_PASSWD

This variable, in conjunction with A_MSSQL_APPROLE_NAME, allows Acu4GL to use approles. Before connecting to a database, set A_MSSQL_APPROLE_NAME to the name of a role. Then set A_MSSQL_APPROLE_PASSWD to a password for that role. By using these two variables, you set the runtime to use this approle.

For more information on approles, please see the SQL Server documentation.

See also

A_MSSQL_APPROLE_NAME

A_MSSQL_CURSOR_OPTION_1, A_MSSQL_CURSOR_OPTION_2, A_MSSQL_CURSOR_OPTION_3

These configuration variables allow you to fine-tune the declaration of cursors in the Acu4GL for Microsoft SQL Server product. In general, cursors are declared with the following syntax:

```
DECLARE cursor_name option_1 CURSOR option_2 FOR
  <select....> option_3
```

In other words, different phrases can go in each of the option_*X* places. Also, different versions of SQL Server allow different options in each of those places. Because of this, the Acu4GL product allows customization of the cursor declaration via these three variables. The values of these variables are placed verbatim into the declare phrase when building a cursor. Note that any errors in the values of these variables may cause your Acu4GL product to be inoperable. Be sure to read the SQL Server documentation to determine what phrases are allowed in each case.

The default values are as follows (the limit is 65 characters for each option):

OPTION_1: “SCROLL”
OPTION_2: blank
OPTION_3: “For read only”

This was added because of an issue in SQL Server 6.5 and SQL Server 7.0 that would return incorrect results if this phrase was not used.

A_MSSQL_DATABASE

A_MSSQL_DATABASE specifies the name of the specific database to be accessed. You cannot open any database files until you have set this variable.

Example

```
A_MSSQL_DATABASE stores
```

indicates the stores database is to be accessed.

A_MSSQL_DEADLOCK_LOOPS

Use A_MSSQL_DEADLOCK_LOOPS if you expect that more than one user will be opening a lot of tables at the same time. This configuration variable can be used to instruct Acu4GL to re-execute an INSERT statement that could not execute because a row in the AcuOpenTables table was locked, or to return an error if the user chooses not to run the query again.

The default for A_MSSQL_DEADLOCK_LOOPS is “0”, which causes the interface to return an error 9D,1205 indicating that a table is locked.

Set A_MSSQL_DEADLOCK_LOOPS to a positive numeric value to cause Acu4GL for SQL Server to re-execute by the number specified by the query that tried to open the table, and thus caused the deadlock. Note that it can be as long as 10 seconds until SQL Server detects the deadlock, and the application appears to “hang” while the repeated attempts to re-execute the query are in progress.

Set A_MSSQL_DEADLOCK_LOOPS to "-1" or "MESSAGE" to cause Acu4GL for SQL Server to display a message box containing the text of the SQL Server error message and the option to rerun the query.

```
SQL Server has returned an error
(text of message from SQL Server)
Do you want to retry the operation?
```

If the user answers **Yes**, the interface reruns the query. If the user answers **No**, the interface returns an error 9D. Setting A_MSSQL_DEADLOCK_LOOPS to "-1" or "MESSAGE" is the preferred action; the time it takes to inform users of the problem allows other connections to finish opening the tables, giving the AcuOpenTables table time to remove the deadlock.

A_MSSQL_DEFAULT_CONNECTION

A_MSSQL_DEFAULT_CONNECTION specifies the name of the server to which the runtime will connect. This variable is checked only if the DSQUERY environment variable has not been set. If neither DSQUERY nor A_MSSQL_DEFAULT_CONNECTION is set, the server is named *localhost*. To reference tables in another server, open the file *servername.database.owner.table*.

Example

Suppose you have two servers, one named TOM and one named HARRY. If most of the tables you want to access are on the server HARRY, then you should set:

```
A_MSSQL_DEFAULT_CONNECTION  HARRY
```

For those occasions when you want to access the TOM server, you could open the file this way:

```
TOM.stores.johndoe.purch1
```

A_MSSQL_DEFAULT_OWNER

In SQL Server, tables are named *database.user.tablename*. The value of A_MSSQL_DEFAULT_OWNER specifies the name of the *user*. If this variable is set, it must match the owner of a table. Tables are referenced as *owner.table*, where *owner* is the value of A_MSSQL_DEFAULT_OWNER. If this variable is set to a value for which *owner.table* does not exist, the interface fails to access the table. Note that newly created tables are owned by the users who create them, not the owner named in A_MSSQL_DEFAULT_OWNER. The default value of this configuration variable is blank, causing the interface to call an internal stored procedure to determine the actual owner of the table. Note that we discourage multiple tables with the same *tablename* in a single database.

A_MSSQL_DO_NOT_TRANSLATE_CHAR

If a SQL Server database is configured with a collation that conflicts with the Locale of the client program, characters that can not be represented in the database's collation are turned into question marks. This was not the behavior prior to version 8.1.2.

Setting this variable to “ON”, causes all character fields, except those for date fields, to be treated as binary, storing them in the database and retrieving from the database without translation. This provides the behavior seen prior to 8.1.2 and is useful if the client is using a different locale than the database's collation.

The default setting is “OFF” and means that character fields, except for date fields, will be translated.

A_MSSQL_FAST_ACCESS

A_MSSQL_FAST_ACCESS is a configuration variable that is set from your COBOL program. Files opened while this variable is set to a nonzero value will be optimized for forward sequential access.

We implemented this option to substantially improve the READ NEXT performance in some cases. For example, testing the ***extend*** benchmark program "iobench.cbl" in three ways, yielded the following results for the READ AND SKIP operation:

No FAST_ACCESS:	72.76 seconds
FAST_ACCESS, ROWCOUNT 0:	148.88 seconds
FAST_ACCESS, ROWCOUNT 10:	8.09 seconds

("10" is the perfect value for ROWCOUNT in this benchmark, because the program does a START, 10 READ NEXT operations, and then does it again.)

For certain reporting programs, this option can dramatically improve performance. However, please note the following restrictions.

Files must be open INPUT or open IO with MASS-UPDATE. FAST_ACCESS gives a performance boost only when no locking is required. In files that allow locking, a record must be reread after being locked; this prevents an uninterrupted forward sequential traversal.

Files opened with FAST_ACCESS use a dedicated connection for reading from the file. Since connections are memory-intensive (both on the client, and on the server), the number of files opened with FAST_ACCESS should be kept to a minimum. In the event that opening a connection fails, the file open will still continue, but FAST_ACCESS mode will be disabled, with the following message appearing in the trace file:

```
FAST_ACCESS mode not available.
```

Also, a new connection technically uses a new concurrent Microsoft SQL Server license from Microsoft.

Files opened with FAST_ACCESS will not participate in transactions and may even cause the runtime to hang if transactions are used, especially if the FAST_ACCESS file is updated within the transaction. We suggest that if you use transactions, you don't use FAST_ACCESS. At the minimum, if you use transactions, we suggest that you use FAST_ACCESS only for files open INPUT.

Files opened with FAST_ACCESS cannot be read backwards. In other words, READ PREVIOUS will not work with FAST_ACCESS files. In fact, if you try to READ PREVIOUS on a file opened with FAST_ACCESS, you will get an error 9D,20.

The ANSI standard states that READ NEXT after a READ will return the next record. Some applications depend on this, and some applications just want to read dynamically from a file, and don't use the positioning facility. Because of this ambiguity, files that are opened with FAST_ACCESS cannot be READ dynamically. If you try to READ on a file opened with FAST_ACCESS, you will get an error 9D,20.

There are no restrictions on WRITE, REWRITE, and DELETE. However, these operations use the cursor-based connection, not the dedicated connection. This is the reason transactions may hang.

While the default value is "0" (off, false, no), this configuration variable can also take values of "On" (true, yes).

A_MSQL_LIMIT_DROPDOWN

This configuration option is closely related to **A_MSSQL_USE_DROPDOWN_QUERIES**.

When a sequence of START and READ NEXT/PREVIOUS operations are performed by an application, Acu4GL/MSSQL will generate a sequence of queries to return the set of records matching the application's request. To improve performance, the interface will generate a sequence of "drop down" queries based upon the key of reference's key segments going from the most specific subset using the most number of segments to the most general using the least number of segments. This functionality is turned on by setting the configuration variable A_MSSQL_USE_DROPDOWN_QUERIES to "TRUE".

For example if a key is described by:

```
03 MY-ALTKEY.
   05 MY-ALTKEY-SEG1        PIC X(2).
   05 MY-ALTKEY-SEG2        PIC X(2).
   05 MY-ALTKEY-SEG3        PIC X(2).
```

Then a START followed by a sequence of READ NEXT operations might generate the selection criteria of:

```
WHERE MY-ALTKEY-SEG1 = :w0 AND MY-ALTKEY-SEG2 = :w1 AND
  MY-ALTKEY-SEG3 >= :w3
WHERE MY-ALTKEY-SEG1 = :w0 AND MY-ALTKEY-SEG2 > :w1
WHERE MY-ALTKEY-SEG1 > :w0
```

This can improve performance because the target for each query is kept to a minimal size. If a set of records is not required, the database does not need to spend the time building the working set. When a "DROP DOWN" does occur however, the subsequent working set can require a large amount of time to process because of the potential magnitude of records. Normally there is not a way for a COBOL application to instruct the interface to stop processing when it has finished with the records based on a given key segment.

To address this, a new variable has been introduced:

A_MSQL_LIMIT_DROPDOWN

This variable allows an application to direct the interface not to perform "drop down" query generation and instead return "end of file" when the records matching the current query have been exhausted.

INSTRUCTIONS for use

Set the configuration variable to one of the following settings:

OFF

This is the current default. The interface will perform "drop down" queries.

PARTIAL

If the record positioning was performed by a START with a SIZE clause such that the initial positioning was performed using fewer than the total number of columns in the key, the process will cease after all records matching the START columns have been exhausted.

FULL

If the record positioning was performed by a START without a SIZE clause, the process will cease after all records matching the START columns have been exhausted.

ALL

Regardless of what form of START was used for the initial positioning, the process will cease after all records matching the START columns have been exhausted.

This variable is performed at the time of each start, so the application may change its value.

A_MSSQL_LOCK_DB

A_MSSQL_LOCK_DB specifies the name of the database that holds the lock table.

See also

Section B.6, "Table Locking"

A_MSSQL_LOGIN

A_MSSQL_LOGIN indicates the user name under which you want to connect to the database system.

Example

To connect to the database with the user name MYNAME, you would specify:

```
A_MSSQL_LOGIN  MYNAME
```

in the runtime configuration file.

If A_MSSQL_LOGIN is not set, the runtime will use the value of your USER environment variable as your Microsoft SQL Server login name. For this automatic login to succeed, you must have set up a user with the same name as your computer login name.

See also

A_MSSQL_PASSWD runtime configuration file variable

Section B.2.2, "Setting Up a User Account".

A_MSSQL_MAX_CHARACTERS

A_MSSQL_MAX_CHARACTERS indicates the maximum number of bytes the Acu4GL product will allow in a table row.

Microsoft SQL Server places a limit on the number of bytes per table row. The Acu4GL product adheres to this limit, but sometimes it cannot accurately count how many bytes a particular row contains (because of overhead bytes that Microsoft SQL Server adds). This variable enables the developer to set the Acu4GL product's upper bound.

You might want to try reducing it if you discover that a row cannot be added to a table. By reducing the upper bound, you may be able to prevent the problem.

If Microsoft SQL Server increases the maximum number of bytes allowed in a row (in a future release of the product), you can increase the value of this variable to take advantage of the new limit.

The A_MSSQL_MAX_CHARACTERS variable has a default value of "1962".

A_MSSQL_MAX_COLUMNS

A_MSSQL_MAX_COLUMNS indicates the maximum number of columns the Acu4GL product will allow in a table. The default value is "250".

Microsoft SQL Server places a limit on the number of columns per table. The Acu4GL product will adhere to this limit, but sometimes it cannot accurately count how many columns a table contains (because a column has been added to a table without the Acu4GL product's knowledge). This variable enables the developer to set the Acu4GL product's upper bound. You might want to try reducing the upper bound if you discover that a table cannot be created for some reason. By reducing the upper bound, you allow for uncountable columns and thus may be able to prevent the problem.

If SQL Server increases the maximum number of columns allowed per table (in a future release of the product), you could increase the value of this variable to take advantage of the new limit. Consult your SQL Server documentation to decide the new maximum number of columns allowed per table.

A_MSSQL_NATIVE_LOCK_TIMEOUT

This is one of two locking methods available with Acu4GL for Microsoft SQL Server. The methods are accessed via two configuration variables: A_MSSQL_VISION_LOCKS_FILE and A_MSSQL_NATIVE_LOCK_TIMEOUT. The lock method used is determined as follows: If A_MSSQL_VISION_LOCKS_FILE is set to the name of a Vision file that can be open I/O and has the correct structure, the Vision file is used to hold lock information. If A_MSSQL_NATIVE_LOCK_TIMEOUT is set to a positive value, native locking is used. Otherwise, the AcuLocks table is used to hold locks.

This locking method enables you to use Microsoft SQL Server native locks if you explicitly code transactions in your COBOL program. You can access this method by setting the configuration variable A_MSSQL_NATIVE_LOCK_TIMEOUT to a positive value. This value will be the number of seconds that a connection will wait for a timeout to occur. When such a timeout occurs (for any reason), the Acu4GL product assumes that the timeout was caused by a locked record, and will return error 99 (record locked). If you set this variable but do not explicitly code transactions in your COBOL program, record locking will not occur. Note that the Acu4GL product will wait the number of seconds specified, and your application may seem to "hang" if the timeout is too long. On the other hand, if the timeout is too short, you may get record locked errors when the network is slow.

Microsoft SQL Server uses a page-locking mechanism, and so this method of locking records may cause your application to return spurious record locked errors caused by a record being locked on the same page as the record you are trying to access. SQL Server 7.0 uses row-level locking, but because of the timing issue, we discourage the use of this option.

Note: Even with this variable set, the interface still needs the AcuOpenTables table.

See also

A_MSSQL_VISION_LOCKS_FILE configuration variable

Section B.6, "Table Locking"

A_MSSQL_NO_CACHED_READ

By default, the Acu4GL for MSSQL runtime applies the same READ logic on a previously-read record as is used for the Vision indexed file system. Instead of sending another request to MSSQL to read the same record, the runtime will get the record from cache. This behavior is referred to as "Cached Read" and can improve performance.

A_MSSQL_NO_CACHED_READ controls this behavior. By default cached read is enabled. It can be disabled by setting A_MSSQL_NO_CACHED_READ to "true" or "1".

If the last operation was a successful READ with LOCK and the next operation is a READ with LOCK on the same key with the same key value then the cached record will be returned without accessing the database. A trace file entry is made that says "cached read".

A_MSSQL_NO_COUNT_CHECK

When performing a REWRITE, the interface checks to see that a record was actually updated. If not, it will return an error 23. Setting this variable ON will cause that check not to happen. This will improve performance on

REWRITE, at the risk of missing an error. While the default value is "Off" (false, no), this configuration variable can also take the value of "On" (true, yes).

A_MSSQL_NO_DBID

The interface stores the Database ID in the AcuLocks and AcuOpenTables tables, to distinguish different tables in different databases. Sometimes this causes problems. So setting this variable ON will cause the interface to use a Database ID of "0", instead of the actual ID of the database. While the default value is "Off" (false, no), this configuration variable can also take the value of "On" (true, yes).

A_MSSQL_NO_RECORD_LOCKS

Setting this variable ON will cause all READS to be treated as READ NO LOCK, which can improve performance (but has the obvious consequences). While the default value is "Off" (false, no), this configuration variable can also take the value of "On" (true, yes).

A_MSSQL_NO_TABLE_LOCKS

Setting this variable ON will cause the interface to not use the AcuOpenTables table, which causes all table locking to be disabled. This can improve performance on OPEN and CLOSE statements. While the default value is "Off" (false, no), this configuration variable can also take the value of "On" (true, yes).

A_MSSQL_NO_23_ON_START

When A_MSSQL_NO_23_ON_START is set to a nonzero value, START does not detect whether records actually exist. Because it does not detect the existence of records, it is possible, when using this variable, to do a START

without error, and for the next READ NEXT to return END_OF_FILE, contrary to the ANSI standard. While the default value is "Off" (false, no), this configuration variable can also take the value of "On" (true, yes).

Example

```
A_MSSQL_NO_23_ON_START number
```

where *number* can be a zero or nonzero value.

A_MSSQL_NT_AUTHENTICATION

The A_MSSQL_NT_AUTHENTICATION configuration variable indicates whether Microsoft SQL Server will authenticate users based on their Windows login.

If this variable is set to "True" (on, yes), the Acu4GL for Microsoft SQL Server interface attempts to log users on using the SQL Server Windows NT authentication mode. (See your SQL Server documentation for information about this authentication mode.) When A_MSSQL_NT_AUTHENTICATION is enabled, the A_MSSQL_LOGIN and A_MSSQL_PASSWD configuration variables are no longer needed or used. (They are still available if you are not using Windows NT authentication mode.) If you have not set up SQL Server itself to allow this type of authentication, setting this variable to TRUE causes all login attempts to fail. See your SQL Server documentation for information on how to set up this type of authentication.

The default is "False" (off, no), indicating that the Acu4GL for Microsoft SQL Server interface will not use Windows NT authentication mode when logging users on and will continue to use login/password authentication.

A_MSSQL_PACKETSIZE

The A_MSSQL_PACKETSIZE variable sets the size of network packets. Setting this variable can affect performance, since fewer and larger network calls can improve performance.

This variable must be set in the configuration file, and has no effect if set in a COBOL program via SET CONFIGURATION or SET ENVIRONMENT. The value of this variable is the largest size that the transport layer uses for network packets (although the underlying library may reduce the size specified; this is out of the control of the interface.) The largest value that can be specified is "32767". The default depends on which version of the client libraries are linked into the runtime, although "512" is the most common default.

Use this configuration variable to tune your database performance. To set the packet size to 8192 use:

```
A_MSSQL_PACKETSIZE   8192
```

Setting this variable to "0" or to a negative value will cause the Acu4GL product to use the default value.

A_MSSQL_PASSWD

The variable A_MSSQL_PASSWD should be set to the password assigned to the database account associated with the user name specified by A_MSSQL_LOGIN.

Examples

If the account with the user name in A_MSSQL_LOGIN has the associated password CW021535, you would specify:

```
A_MSSQL_PASSWD  CW021535
```

in the runtime configuration file.

For better security, you can accept a password from the user during program execution; set the A_MSSQL_PASSWD variable based on the response:

```
ACCEPT RESPONSE NO-ECHO.
SET ENVIRONMENT "A_MSSQL_PASSWD" TO RESPONSE.
```

Note: If the user has been set up without a password, this variable need not be set.

See also

A_MSSQL_LOGIN runtime configuration file variable

A_MSSQL_ROWCOUNT

This variable has an effect only if you are reading on a key that does not allow duplicates, or if you have added an Identity column to the table. A_MSSQL_ROWCOUNT determines how many rows are returned by a SELECT statement sent to the server.

This variable can be used to speed up the Acu4GL product. For example, if you know you will be reading only one record at a time, and reading from a unique key, you can set A_MSSQL_ROWCOUNT to "1", thus speeding up the processing.

If you know you are going to be reading records ten rows at a time, set A_MSSQL_ROWCOUNT to "10". If you don't have any information about how many rows are going to be requested, set this variable to "0", which is the default.

Note: Setting this variable to a non-optional value can actually degrade performance, since the interface may be forced to issue more SELECT statements once the rowcount has been determined. Use caution when setting this variable.

See also

A_MSSQL_ADD_IDENTITY runtime configuration file variable

A_MSSQL_SELECT_KEY_ONLY

This variable directs the interface to select key columns only when searching for records. Its use can improve READ performance on large tables with many rows.

When set to “True” (on, yes), the default value, A_MSSQL_SELECT_KEY_ONLY causes the interface to select only key columns when searching for records and then select the entire row of the single record that must be returned to the COBOL program. This improves performance on large tables with many rows.

Setting A_MSSQL_SELECT_KEY_ONLY to “False” (off, no) does not affect how the select is created for files open I/O (since the record must be locked and then the rest of the data fetched), but causes the interface to select all the columns of the table for files open INPUT.

A_MSSQL_SKIP_ALTERNATE_KEYS

A_MSSQL_SKIP_ALTERNATE_KEYS determines whether alternate keys are used to form indexes during table creation. The default value of this variable is “0”, which means it’s okay to use alternate keys. This configuration variable can also take values of “1” (on, true, yes).

If you set the variable to a nonzero value (such as “1”), alternate keys are *not* used to form indexes, which speeds up processing if many writes or rewrites are being performed. (Note that a value of “1” may slow processing if the application is reading sequentially using an alternate key.)

A_MSSQL_TRANSLATE_TO_ANSI

Setting this variable to “True” (on, yes) causes the Acu4GL interface to call the same translation function used by the Windows runtime to translate characters going to the server into the OEM character set, and to translate characters coming from the server to ANSI.

The default is “False” (off, no), which indicates that the Acu4GL interface does not call the translation function, but passes the data as is to the library.

A_MSSQL_UNLOCK_ON_EXECUTE

Setting this variable to "True" (on, yes) causes all invocations of I$IO using the EXECUTE opcode to unlock all records. Normally, records are unlocked when a transaction finishes. But if users do their own transaction management using **sql.acu** (which calls I$IO using the EXECUTE opcode), the interface never knows to unlock records, because it doesn't check the text sent to the database to see if it is associated with a transaction. The default value is "Off" (false, no).

A_MSSQL_USE_DROPDOWN_QUERIES

Setting A_MSSQL_USE_DROPDOWN_QUERIES to a nonzero value causes selects sent to the database to be of the drop-down variety, instead of a single large query.

For example, if you have a file with three fields in the primary key, (keyseg1, keyseg2, keyseg3), and your COBOL program does a START, the following query is sent to the database:

```
select (columns) from (table) where
((keyseg1 = value1 and keyseg2 = value2 and
keyseg3 > value3) or (keyseg1 = value1 and
keyseg2 > value2) or (keyseg1 > value1))
order by keyseg1, keyseg2, keyseg3
```

If you use drop-down queries, the following collection of queries is sent instead:

```
select (columns) from (table) where (keyseg1
= value1 and keyseg2 = value2 and keyseg3 >
value3) order by keyseg1, keyseg2, keyseg3
```

When that set is finished, we then send:

```
select (columns) from (table) where (keyseg1
= value1 and keyseg2 > value2) order by
keyseg1, keyseg2, keyseg3
```

And when that set is finished, we then send:

```
select (columns) from (table) where (keyseg1
```

```
> value1) order by keyseg1, keyseg2, keyseg3
```

There are advantages and disadvantages to each method. If you use the A4GL_WHERE_CONSTRAINT variable, you should probably set A_MSSQL_USE_DROPDOWN_QUERIES to "0", because the WHERE constraint will limit the result set sufficiently that the larger query will be more efficient. See **Section 9.1.2, "The WHERE Constraint,"** for additional information.

If you usually START files and read to the end, you should set A_MSSQL_USE_DROPDOWN_QUERIES to "0", because a fewer number of queries need to be sent to the database. On the other hand, if you START files and stop reading after some condition, but haven't used the WHERE constraint, you may get more efficient access by setting this variable to "1" and using the drop-down style of query. In either case, we recommend that you run some tests to see which value of this variable makes the most sense for your application.

This variable is accessed only during a positioning operation, so you can set it at different times for different tables.

While the default value is "0" (off, false, no), this configuration variable can also take values of "On" (true, yes).

Example

```
A_MSSQL_USE_DROPDOWN_QUERIES number
```

where *number* can be a zero or nonzero value.

See also **A_MSQL_LIMIT_DROPDOWN**.

A_MSSQL_VISION_LOCKS_FILE

This locking method causes the lock table (AcuLocks) to be a Vision file instead of an SQL table. This can be accessed via the configuration variable A_MSSQL_VISION_LOCKS_FILE, which must be set to the name of the Vision file that will hold the lock information. Note that you must also set a

configuration variable that specifies this file as a Vision file (using a _host variable). This file must be accessible to all users accessing the Microsoft SQL Server, either through a common directory, or through AcuServer.

Also included with the Acu4GL for Microsoft SQL Server product is a small COBOL program that will manage this Vision file ("lockmgr"). This program should be run with the same runtime that you normally use to access Microsoft SQL Server tables, and with the same configuration variables set. The program detects whether Acu4GL for Microsoft SQL Server is available and detects the A_MSSQL_VISION_LOCKS_FILE variable to determine which file to manage. This program displays all the records in the lockfile and gives options for removing single records (by highlighting the desired record to remove), or removing all shown records, and also for restricting the shown records by PID, Table, and Database. This program also creates the Vision lock file and is the only method of creating the file. You can refresh the display by selecting the Restrict button and then pressing OK without restricting the display further.

If everything is working correctly, there should be no records in this table when there are no users accessing Microsoft SQL Server through Acu4GL. The source for this program is in the "sample/acu4gl" directory.

See also

A_MSSQL_NATIVE_LOCK_TIMEOUT configuration file variable

Section B.6, "Table Locking""

B.5 Using the Database Table

The database table is built and accessed automatically when the COBOL application is executed. To the end user, the interaction between the COBOL and the Microsoft SQL Server database is invisible; queries are generated and data is exchanged in fractions of a second, and the application proceeds without interruption.

You can also access the database information directly from Microsoft SQL Server, at your option.

B.6 Table Locking

By default, Microsoft SQL Server doesn't support the type of record and table locking that COBOL expects. For this reason, the Acu4GL for Microsoft SQL Server product implements its own locking method. This is accomplished with the addition of two tables to a database. You choose which database will hold these tables during installation of the Acu4GL for Microsoft SQL Server product.

Before using the locking tables, you must execute the included **ms_inst.sql** script. (See the installation instructions you used from this manual for the exact procedure. They can be located in the table of contents.) If you don't perform this step, the first time you try to execute a COBOL program that opens a Microsoft SQL Server table, you will receive error 9D,11, "ACUCOBOL Lock Table Incorrect".

AcuLocks table

The first locking table is called AcuLocks; it holds the record locks. The columns in this table are

- the DBID
- the Table ID
- the Process ID of the process holding the lock
- the primary key of the record that is locked

There is a unique index on the DBID, the Table ID, and the Key Value, so that inserts into this table are automatically rejected if another user holds a lock on the row in question. This also gives the Database Administrator the information needed to determine who has locks set, and whether the user in question still has a connection to the server.

There is a non-unique index on DBID, Table ID, Table Number, Process ID and Key Value that speeds searching of the AcuLock table when a lock is released and prevents dead locks.

AcuOpenTables table

The second locking table is called AcuOpenTables; it holds information about open tables.

The columns in this table are

- the DBID
- the Table ID
- the process ID (PID) of the process that has the table open
- the Open Mode (Input, Output, I/O or Extend)
- whether Multiple records can be locked
- whether the file can be open for I/O by any other users
- whether the file can be open at all by any other users
- whether Mass Update was specified in the open

There is a trigger on this table, which will automatically reject opens that are not allowed based on other users' open modes.

There are 5 non-unique indices on this table, AcuOpenTablesMass, AcuOpenTablesMulti, AcuOpenTablesPrimary, AcuOpenTablesRead and AcuOpenTablesWrite. All of the indices contain the DBID and Table ID, but AcuOpenTablesMulti, AcuOpenTablesMass, AcuOpenTablesRead, and AcuOpenTablesWrite contain an additional column for whether the file can have multiple records locked, whether the file was opened for mass updates, whether the file can be opened for I/O by other users, and whether the file can be opened at all by any other users. These indices speeds searching of the AcuOpenTables table and prevents dead locks.

By using these lock tables, the Acu4GL for Microsoft SQL Server product is able to support all the types of locking ordinarily supported by ACUCOBOL-GT. No special runtime configuration variables are required.

This method of locking is all that is needed if no applications other than COBOL programs are going to be using the Acu4GL for Microsoft SQL Server product. But if your site has other applications that access the Microsoft SQL Server databases, you must use a method of locking that is inherent to Microsoft SQL Server.

Another method of locking that Microsoft SQL Server supports internally is the result of time stamping and the use of BROWSE MODE (see the discussion of BROWSE MODE in the *Microsoft SQL Server Commands Reference Manual*). If a table has a time stamp column, the Acu4GL for Microsoft SQL Server product will use browse mode. This will allow the server to detect whether another application has modified a record while an ACUCOBOL-GT application has had it locked.

For information about alternative locking methods, see the configuration variables **A_MSSQL_NATIVE_LOCK_TIMEOUT** and **A_MSSQL_USE_DROPDOWN_QUERIES**.

B.7 Stored Procedures

A stored procedure is a collection of SQL statements residing on the server, stored as text in a table in the database. Stored procedures provide an efficient environment for Acu4GL because, once they are executed on the server, they do not need to be parsed and optimized each time they are executed. (However, if the server goes down, the stored procedure will be parsed and optimized again the first time the procedure is called after the database restarts.)

If you run a set of stored procedures for the database where data is manipulated (your production database), this database must be the setting for **A_MSSQL_LOCK_DB**. If you run the stored procedures against both the lock database and the production database, the lock database can be the setting for A_MSSQL_LOCK_DB. Note that if you run one version of stored procedures against the lock database and another version against the production database, the stored procedures in the lock database override those in the production database. Therefore, we recommend that you always update your stored procedures with each new installation of Acu4GL, so that these procedures are consistent when you run them against the tables in your database.

This section discusses two types of stored procedures:

- Procedures that you may want to add to Acu4GL for SQL Server
- Procedures provided in Acu4GL for SQL Server that you, as the developer or administrator, may find useful

---

Note: If you are upgrading from an earlier version of Acu4GL, be sure to install the new stored procedures. We always upgrade stored procedures in such a way that they will be compatible with older versions of the product, so installing new stored procedures over old ones does not affect your ability to run with an older version of the interface software.

B.7.1 Developer- or Site-supplied Stored Procedures

This section provides information on stored procedures you may want to create. It also supplies some example code.

The Acu4GL for Microsoft SQL Server interface checks for these stored procedures when opening a file and will use them in certain circumstances, such as when A_MSSQL_NO_23_ON_START is set to "No".

These stored procedures are:

***tablename*_insert (where *tablename* is the name of the table being accessed)**

***tablename*_update**

The INSERT and UPDATE stored procedures takes as arguments all columns in order listed in the XFD. They expects no columns to be returned.

***tablename*_delete**

The DELETE stored procedure takes as arguments the primary key fields in the order they are listed in the XFD. It expects no columns to be returned.

***tablename*_read**

The READ stored procedure takes as arguments the primary key fields in the order they are listed in the XFD (in the key section). It expects the columns to be returned in order of condition. So all condition 0 columns are retrieved first, then condition 1 columns, etc. For each condition, the columns are expected in order given in the XFD

***tablename*_start*nnn* (where *nnn* is the key number to start on)**

The startnnn stored procedure takes as input a varchar which is the mode (">", ">=", "=", "<", or "<="), and then all the key columns for that key. If no records match the criteria, the stored procedure must raise an error (using the TRANSACT-SQL keyword "raiserrror") with a value of 22006, "Record not found". If the stored procedure fails to do this, then the interface will not return error 23 to the COBOL program.

Note: The Acu4GL for Microsoft SQL Server interface does not create these stored procedures or check their accuracy. It is possible to create stored procedures in such a way as to make the Acu4GL for Microsoft SQL Server product completely inoperable. The Acu4GL product uses these stored procedures for performance reasons only.

Sample XFD

Sample code for developer-supplied stored procedures is based on the following example of an XFD:

```
XFD,03,FTEST2-FILE,FTESTDAT
# ftestdat.xfd - generated by ACUCOBOL-GT v4.2 Alpha 1
   (8/22/99)
# Generated Sun Aug  22 07:54:28 1999
00031,00031,003
01,0,004,00000
01
FTEST2-KEY
01,1,004,00004
02
FTEST2-KEY1-SEG1
FTEST2-KEY1-SEG2
01,0,004,00008
01
```

```
FTEST2-ALTKEY2
000
0006,00006
00000,00004,16,00004,+00,000,000,FTEST2-KEY
00004,00002,16,00002,+00,000,000,FTEST2-KEY1-SEG1
00006,00002,16,00002,+00,000,000,FTEST2-KEY1-SEG2
00008,00004,16,00004,+00,000,000,FTEST2-ALTKEY2
00012,00009,00,00009,+00,000,000,FTEST2-NUMBER
00021,00010,16,00010,+00,000,000,FTEST2-INFO
```

*tablename*_insert

*tablename*_insert is used to WRITE a record to the file. The parameters passed to the stored procedure are the values of all the columns in the row, in the order of the columns in the database. The timestamp column and identity column (if present in the table) are not passed to the stored procedure.

Given the **Sample XFD**, you might want to create the following stored procedure for writing records to a file:

```
create procedure ftestdat_insert
@ft2_key char(4),
@ft2_key1_seg1 char(2),
@ft2_key1_seg2 char(2),
@ft2_altkey2 char(4),
@ft2_number char(9),
@ft2_info char(10)
as
insert into ftestdat (ftest2_key, ftest2_key1_seg1,
ftest2_key1_seg2, ftest2_altkey2, ftest2_number,
ftest2_info) values (@ft2_key, @ft2_key1_seg1,
@ft2_key1_seg2, @ft2_altkey2, @ft2_number, @ft2_info)

grant execute on ftestdat_insert to public
```

*tablename*_delete

*tablename*_delete is used to DELETE a record from the file. The parameters passed to the stored procedure are the values of the primary key, in the order they are listed in the XFD.

Based on the **Sample XFD**, you might want to create the following stored procedure for deleting records from a file:

```
create procedure ftestdat_delete
@ft2_key char(4)
as
delete from ftestdat where ftest2_key = @ft2_key

grant execute on ftestdat_delete to public
```

*tablename*_read

*tablename*_read is used to read a random record (READ, not READ NEXT or READ PREVIOUS). The parameters passed to the stored procedure are the values of the primary key, in the order they are listed in the XFD. The expected rowset is the columns in the first table (if secondary tables are used), or the columns of the table (if secondary tables are not necessary).

This stored procedure is very similar to *tablename*_start.

*tablename*_update

*tablename*_update is used to REWRITE a record from the file. The parameters passed to the stored procedure are the values of all the columns in the row, in the order of the columns in the database. The timestamp column and identity column (if present in the table) are not passed to the stored procedure.

For example, based on the **Sample XFD**, you might want to create the following stored procedure for rewriting a record:

```
create procedure ftestdat_update
@ft2_key char(4),
@ft2_key1_seg1 char(2),
@ft2_key1_seg2 char(2),
@ft2_altkey2 char(4),
@ft2_number char(9),
@ft2_info char(10)
as
update ftestdat set
ftest2_key = @ft2_key,
ftest2_key1_seg1 = @ft2_key1_seg1,
ftest2_key1_seg2 = @ft2_key1_seg2,
ftest2_altkey2 = @ft2_altkey2,
ftest2_number = @ft2_number,
ftest2_info = @ft2_info
```

```
where ftest2_key = @ft2_key

grant execute on ftestdat_update to public
```

*tablename*_start*nnn*

*tablename*_start*nnn* is used to START a file. The *nnn* value is the key number to start on, and will be 0 filled. For example, the start procedure for the primary key for table mytab will be “mytab_start000”.

Note: If **A_MSSQL_NO_23_ON_START** is set to “Yes”, the start stored procedure is disabled.

Because there can be up to 119 alternate keys, the Acu4GL product does not search for a start procedure unless, or until, it is used. The parameters passed to the stored procedure are a 2-char mode [it is a varchar(2) field], with one of the following values: “>”, “>=”, “=”, “<=”, or “<”. The rest of the parameters are the columns of the key used to start. Because the ANSI specification for START includes information about the size of the key being used (and in particular allows partial keys), the start procedure is used only if an entire key is given to the start verb. This procedure is also special in that it does not return data, but needs to raise an error condition if the start fails. The way to raise the error condition from within the stored procedure is to include code similar to the following:

```
raiserror 22006 "Record not found"
```

The code “22006” is very important. It is the code searched for in setting the error condition from within the Acu4GL product. If you use a different number, your starts may succeed when they should actually fail.

For example, based on the **Sample XFD**, you might want to create the following stored procedure to start a file:

```
create procedure ftestdat_start001
@mode varchar(2),
@ft2_key1_seg1 char(2),
@ft2_key1_seg2 char(2)
as

if exists (select 1 from ftestdat where
```

```
(ftest2_key1_seg1 = @ft2_key1_seg1 and
((@mode = ">=" and ftest2_key1_seg2 >=@ft2_key1_seg2) or
(@mode = ">" and ftest2_key1_seg2 > @ft2_key1_seg2) or
(@mode = "=" and ftest2_key1_seg2 = @ft2_key1_seg2) or
(@mode = "<" and ftest2_key1_seg2 < @ft2_key1_seg2) or
(@mode = "<=" and ftest2_key1_seg2 <= @ft2_key1_seg2))))
return
if exists (select 1 from ftestdat where
(((@mode = ">=" or @mode = ">") and
ftest2_key1_seg1 > @ft2_key1_seg1) or
((@mode = "<=" or @mode = "<") and
ftest2_key1_seg1 < @ft2_key1_seg1)))
return
raiserror 22006 "Record not found"

grant execute on ftestdat_start001 to public
```

B.7.2 Built-in Stored Procedures

Several stored procedures come with Acu4GL for SQL Server. One, **sp_AcuInit**, provides a means for customized initialization. The others return information based on the AcuOpenTables and AcuLocks tables.

Note: You will see that the names of several stored procedures end in "_1", indicating the first version of the stored procedure. Whenever a stored procedure is updated, the extension is updated by one. This is why you can install new stored procedures without overwriting older ones. Be sure to install all stored procedures when you install Acu4GL for SQL Server.

---

sp_AcuInit

If the stored procedure **sp_AcuInit** exists in the master database, it is executed when the connection is made to the server. This is a procedure you can set up to do customized initialization. This stored procedure does not take any parameters and does not return any results. This optional stored procedure is executed for all connections by the Acu4GL interface to the database, not just the primary connection.

As an example of customized initialization, you can use this stored procedure to remove stale locks by calling **sp_AcuRemoveUnusedLocks_1** (which is installed when all the other Acu4GL stored procedures are installed) or to limit access to the database by certain users during certain hours. If **sp_AcuInit** returns an error, the connection is denied and the error is reported to the COBOL program. The method for returning an error is to execute the Transact-SQL statement "raiserror". See your SQL Server documentation for information about Transact-SQL and stored procedures.

The **sp_AcuInit** procedure, if it exists in the master database, is executed whenever Acu4GL makes a new connection, including FAST_ACCESS connections, to the database. Therefore, any customization that you've indicated via **sp_AcuInit** applies to all connections a transaction makes to the database, not just the primary connection.

sp_AcuRemoveUnusedLocks_1

Use this stored procedure to determine who is logged in and to remove Process IDs that are no longer active on the system. You can call this from **sp_AcuInit** each time a user connects to the database to ensure that the lock table contains only active locks. This stored procedure must reside in the same database as the AcuOpenTables and AcuLocks tables; it is placed there automatically when these tables are created during installation.

sp_AcuTableReport_1

Use this stored procedure to learn who is using the tables in the database at the time this procedure is run. Run this procedure before running the **sp_AcuZeroUserCount** stored procedure, so that you can contact users to inform them that the database will be closing. This stored procedure must reside in the same database as the AcuOpenTables and AcuLocks tables; it is placed there automatically when these tables are created during installation.

sp_AcuUserCount_1

You can run **sp_AcuUserCount** from the query analyzer to learn how many users have a particular table open. You can use this to track table and database activity to ensure that your database is running as efficiently as

possible. This stored procedure must reside in the same database as the AcuOpenTables and AcuLocks tables; it is placed there automatically when these tables are created during installation.

sp_AcuZeroUserCount_1

Use **sp_AcuZeroUserCount** to remove all locks on a table and close it. Be sure to run this stored procedure on all tables in the database if you will be shutting down the database for any reason. This stored procedure must reside in the same database as the AcuOpenTables and AcuLocks tables; it is placed there automatically when these tables are created during installation.

B.8 Limits and Ranges

The following limits exist for the Microsoft SQL Server file system:

Maximum number of columns per key: 16
Maximum number of columns: 250
Maximum number of bytes in a single row when using Acu4GL for Microsoft SQL Server: 1962

To achieve the same sort or retrieval sequence under Microsoft SQL Server as under the Vision file system, key fields that contain signed numeric data must be preceded by a **BINARY** directive.

Acu4GL for Microsoft SQL Server supports the data types shown below; when it's creating tables, the following conversion rules are used, in the sequence shown:

COBOL	SQL Server
DATE directive	`DATETIME`
BINARY directive	`VARBINARY(n) (if SIZE < 255)` `IMAGE (if SIZE ≥ 255)`
VAR_LENGTH directive	`VARCHAR(n) (if SIZE < 255)`
Usage FLOAT	`REAL (if SIZE = 4)`

COBOL	SQL Server
Usage DOUBLE	FLOAT (if SIZE = 8)

Any other numeric usage:

PIC 9(*n*)V9(*m*)	SMALLINT (if *m* = 0 and *n* < 5) INT (if *m* = 0 and *n* < 10) DECIMAL(*n* + *m*, *m*)(otherwise)

Any other usage:

PIC X(*n*)	CHAR(*n*) (if *n* < 255) TEXT (if *n* > 255)

Other limits are described in Appendix B in Book 4, *Appendices*, of the ACUCOBOL-GT documentation set.

B.9 Runtime Errors

This section lists the Acu4GL error messages that could occur during execution of your program. **Chapter 9** provides information on compile-time errors and also provides several methods for retrieving runtime errors.

Each message is followed by an explanation and a recommended recovery procedure.

Runtime errors will have this format:

9D,xx

The 9D indicates a file system error and is reported in your FILE STATUS variable. The *xx* is a secondary error code. These are the secondary errors reported directly from Acu4GL:

9D,01 Read error on dictionary file

An error occurred while reading the XFD file; this probably means the XFD is corrupt. Recreate the XFD file.

9D,02 Corrupt dictionary file

The dictionary file for one of your COBOL files is corrupt and cannot be read. Recompile with the "-Fx" option to re-create the dictionary. See **section 8.1** for information on compiler options.

9D,03 Dictionary (.xfd) file not found

The dictionary file for one of your COBOL files cannot be located. Be sure you have specified the correct directory via your **XFD_PREFIX** runtime configuration file variable. You may need to recompile with the "-Fx" option to create the dictionary.

9D,04 Too many fields in the key

There are more than 16 fields in a key. Check your key definitions and restructure the key that is illegal, then recompile with "-Fx".

9D,05 (no message associated with this error)

A date given to the Acu4GL interface is invalid and cannot be converted.

9D,06 Mismatched dictionary file

The dictionary file (.xfd) for one of your files conflicts with the COBOL description of the file FD. The *xx* indicates a tertiary error code that is defined by the host file system. You can determine the exact nature of the mismatch by referring to the host system's error values.

The tertiary error code may have any of these values:

01 – mismatch found but exact cause unknown (this status is returned by the host file system)
02 – mismatch found in file's maximum record size
03 – mismatch found in file's minimum record size
04 – mismatch found in the number of keys in the file
05 – mismatch found in primary key description
06 – mismatch found in first alternate key description

07 – mismatch found in second alternate key description

The list continues in this manner for each alternate key.

9D,11 ACUCOBOL-GT stored procedures not found
or
ACUCOBOL-GT lock table missing

The installation of Acu4GL for Microsoft SQL Server creates a number of stored procedures and tables. At least one of these was not found.

9D,12 A column of a key is of data type TEXT or IMAGE, which is illegal

Columns that are part of an index may not be of type TEXT or type IMAGE. Check your key definition.

9D,13 Internal error

Multiple records were found with the same index (this is a Microsoft SQL Server error).

9D,14 DB library function returned an unexpected error

dbinit (Microsoft SQL Server) failed. An error message from Microsoft SQL Server is displayed on the terminal.

9D,16 Trying to rename a table across databases

RENAME works only within the same database.

9D,17 Cache error

(internal error) The internal process cache has been corrupted. Please contact Technical Services.

9D,18 Primary Key error

An error occurred when creating the primary key for a secondary table.

9D,19 Table Size Error

The table is larger than Microsoft SQL Server will accept, either in the number of columns or in the number of bytes in the row. Use the **SECONDARY_TABLE** directive to get around this.

9D,20 (no message associated with this error)

This error tells you that you are trying to do something with a FAST_ACCESS table that is not allowed. The description of the **A_MSSQL_FAST_ACCESS** configuration variable in section B.4 details the restrictions for tables opened this way.

9D,21 (no message associated with this error)

There was a problem accessing one or more Vision locks files and processing of this request cannot continue. The tertiary error is an error code returned from Vision.

9D,22007

The error sub-code 22007 indicates tha an attempt was made to write non-numeric data into a numeric field (database column). It is reported with error code 9D. For example:

```
report_status called from 3688
[1] 22018 0 - Invalid character value for cast specification
 No more messages
 Leaving, Execute
Leaving, execute
 >>>file status = 9D,22007
```

24

Note that not all database errors are reported as 9D errors. The Acu4GL interface translates the following database errors to file status 24:

1101, 1105, 1118, 1510, 1703, 1803, 4504, 4706.

Each of these errors is about the requested operation failing due to space considerations in the database. The runtime trace file must be examined to determine which error occurred and how to resolve it.

B.10 Common Questions and Answers

This section contains some questions and answers specific to Acu4GL for Microsoft SQL Server. Refer to **Chapter 10** for additional questions and answers that pertain to the Acu4GL family of products.

Question: When I try to open a file for output, I get the error 9D,2714. There is already an object named "*" in the database. Why?

Answer: One of your record's data items probably has the same name as a Microsoft SQL Server reserved word. Locate the column by comparing a file trace of the CREATE TABLE to Microsoft SQL Server's list of reserved words. Apply the **NAME** directive to the field in the FD that is associated with the invalid column, then recompile the program to create a new XFD file.

Question: Can I open tables in different databases?

Answer: Yes. Use a file name like:

```
database.owner.tablename
```

Note that, because Acu4GL for Microsoft SQL Server automatically determines an owner, you can also specify a file name like "database..tablename". The two dots are mandatory in this case.

Question: Can I use multiple servers (on the same machine or on different machines) on my network?

Answer: Yes. Each server has a unique name.

Question: I'm getting an error 9D,11 ACUCOBOL-GT lock table missing. I know that I added the lock table during installation.

Answer: This is probably a permissions problem. All users must have READ, WRITE, UPDATE, and DELETE access to AcuLocks (and therefore to the database that contains it). Be sure to check your permissions.

Question: I keep receiving an error message saying that my login is invalid. But I'm sure I'm using the correct username and password.

Answer: All usernames, passwords, and database names are case-sensitive. Be sure that you are typing the names exactly as they are set up.

Question: Are there any ACUCOBOL-GT library routines that do not work with or would not make sense to use with Acu4GL for Microsoft SQL Server?

Answer: Yes. There are two ACUCOBOL-GT library routines that either don't work with or do not make sense to use with Acu4GL for Microsoft SQL Server: C$COPY and C$RECOVER.

C Acu4GL for Oracle Information

Key Topics

C.1 Oracle Concepts Overview

Acu4GL for Oracle is based on the Oracle Call Interface (OCI), an API that makes the ACUCOBOL-GT code more portable to the maximum number of platforms.

A quick overview of some basic design concepts underlying the Oracle Database Management System will help you interface your COBOL program to it.

Transactions

The Oracle RDBMS is a transaction-based system. All of the work that you perform while using Oracle must occur within a transaction, whether that work is being done through Acu4GL for Oracle or another 4GL application**.** When you use Acu4GL for Oracle, a transaction is implicitly started for you by the database engine itself with the first file I/O operation performed on a file associated with Oracle. Because all operations with Acu4GL for Oracle occur within a transaction, any record locked during processing remains locked until either a COMMIT WORK or ROLLBACK WORK is issued. This action results in behavior similar to the LOCK ON MULTIPLE RECORDS clause in COBOL.

The benefits of a transaction management system are best illustrated by an example. A COBOL application that handles order entry might perform these steps to accept an order:

1. Write an invoice record.
2. Update a customer record.
3. Write a payroll record for sales commissions.
4. Update an inventory record.

This series of four file operations is a logical unit. If the program were interrupted, and completed only some of the four file operations, then the files would be in an inconsistent state. For example, if the program terminated unexpectedly after it updated the customer record, but before it updated the inventory record, then a subsequent run might access non-existent inventory.

The solution to this problem is to provide a method for the programmer to define a set of operations that should either all occur or all not occur. Then, if the program encounters an error or terminates, the files are left in a consistent state.

All file operations that are part of a transaction are logged. Once logged, they can either be committed or rolled back (undone) by the program.

If a program dies or the system fails, the log file can be used to reconstruct complete transactions, returning all files to a consistent state. Transaction logging thus offers these two facilities:

- It provides the programmer with the ability to define transactions and the ability to commit them or "undo" them (usually in response to an error condition). This "undo" facility is called a "rollback."
- It provides the ability to reconstruct files into a consistent state after a program dies or system failure occurs. This operation is called "recovery."

Record-locking issues in transactions

Applications that are written for transaction management systems, or that perform work in small "operation-based" logical units, benefit greatly from Oracle's transaction management systems. Applications that are not written for transaction management encounter difficulty with record locking when operating against a system that enforces transaction management.

The difficulty can occur with an application that is performing more than one logical task at a time. Any operation that modifies or reads data in an I/O mode without the WITH NO LOCK phrase causes a lock to be placed in the database system. As a result, the application may have many more record locks present than would be expected by the normal rule of COBOL file locking. The application would act similarly as to when the LOCKS ON MULTIPLE RECORDS clause in COBOL is used. This can best be illustrated by an example:

1. The user is entering a customer's order.
2. As each line item is entered into the order, the inventory file is modified to reflect that items have been removed from the stock on hand.

3. The user must switch to a different part of the application to perform a different task, perhaps as a result of a phone call from a new customer.

4. All of the records that were locked, or modified, by the application before the switch remain locked because the first order is not complete. No COMMIT or ROLLBACK has been issued to complete the transaction. All of the records locked by the transaction remain locked until the application ends the transaction.

5. Because one order is open and not yet committed, other applications may be locked out of certain order items if they are still locked by the processing of the first order. The second order entry may be held up until the first order is completed.

6. Note that the first application is not locked out. A process can read its own locked records.

Acu4GL and record locking

Acu4GL provides semi-automated ways to handle transaction logging based on the setting of the **COMMIT_COUNT** environment variable. You can also directly alter your source code to deal with this issue. Individual users determine how much work they wish to do to conform to the Oracle transaction management system by choosing the method that best fits their needs and resources. The following methods are listed in order of increasing amount of work:

COMMIT_COUNT = 0 (Default)

When you set this variable to zero (“0”), the runtime tracks the number of logical locks that are currently in effect. When the number of logical locks reaches zero, the runtime assumes that a transaction is complete and issues a COMMIT statement.

COMMIT_COUNT = *n*

When you set this variable to a nonzero value, the runtime tracks the number of WRITE, REWRITE, and DELETE operations, until the value of COMMIT_COUNT is reached, at which time the runtime issues a COMMIT statement. The READ, START, and READ NEXT operations do not count toward this total, because the runtime is tracking data-altering operations

rather than logical record locks. The disadvantage of this method is that when a COMMIT is issued, any record locks held by the runtime are released.

COMMIT_COUNT = –1

No commit is issued by the Acu4GL product. When COMMIT_COUNT is set to "-1", two alternate ways to perform a commit or rollback are available:

1. Call **sql.acu** with COMMIT WORK or ROLLBACK WORK.
2. Use the COBOL verbs COMMIT and ROLLBACK, available in ACUCOBOL-GT.

COMMIT_COUNT is set to "-1" automatically when you use the transaction management facilities available in the ACUCOBOL-GT compiler. A COMMIT WORK is, however, issued on exit from the runtime (for example, on execution of a STOP RUN).

COMMIT Verb in COBOL

This method forces a COMMIT to be sent to Oracle. It can be used in conjunction with other modes of COMMIT handling. For non-ORACLE files, this is equivalent to the UNLOCK ALL verb.

Explicitly Coded Transactions

This method provides the greatest flexibility in that transactions are specifically tailored for the user's application. This method also requires the most work for traditional COBOL programs in which transaction modules may not be clearly defined.

Acu4GL for Oracle

The Oracle system parameter open_cursors should be set to enhance communication with Acu4GL. See **section C.2.4, "Checking System Parameters,"** for more information.

Table ownership

Table names in Oracle have the form *owner.table_name*. If you are the owner of a given table, you can refer to it as just *table_name*. If you are not the owner, you must refer to it with the owner of the table as a prefix. Acu4GL for Oracle provides a user-configurable method (the **USER_PATH** configuration variable) for implementing this.

Security

Security is implemented in the Oracle RDBMS. A user is required to log in to the RDBMS before any file processing can occur. Acu4GL for Oracle provides both a default and a user-configurable method for implementing this. Oracle's security considerations pose several challenges for COBOL programmers. Because of the various levels of permissions, certain operations are not allowed unless you have Database Administrator (DBA) privileges. These restricted operations include:

- creating a table under another user's name
- dropping a table owned by another user

Oracle versions 9*i* and later also provide additional security levels; however, you must consult the Oracle documentation to determine if any of these permission levels are appropriate for your site.

Generally, it is best for someone with DBA privileges to create and drop the tables, allowing others only the permissions to process information contained in them. A table can be referenced either by *owner.table_name* or by a public synonym that you have created for the table. See the Oracle documentation for more details on DBA privileges and public synonyms.

Note: By default, Acu4GL for Oracle always checks to see if a public synonym is available for a file at open time regardless of what the **USER_PATH** is set to. If the name of a table owned by a current user is the same as a public synonym, the user-owned table is chosen.

C.2 Installation and Setup

The following topics list the steps you must perform before you begin using Acu4GL for Oracle.

The following sections describe the steps you must take before you begin using Acu4GL for Oracle on a new system:

- **Windows Installation Steps**
- **UNIX Installation Steps**
- **Completing the Installation**
- **Checking System Parameters**
- **Setting Up a User Account**
- **Setting Up the User Environment**
- **Designating the Host File System**
- **Setting Up the Search Path**
- **Handling Transactions**

C.2.1 Windows Installation Steps

Installation of the product

The Oracle RDBMS, version 9*i* or later, must be installed and configured prior to the installation of Acu4GL for Oracle. Consult the Oracle documentation if you have any questions regarding this step.

This section details installation instructions for Acu4GL for Oracle on your Windows machine. UNIX installation instructions can be found in **section C.2.2, "UNIX Installation Steps,"** of this appendix.

CD-ROM installation

Instructions for installing your Acu4GL product from CD-ROM are contained on the *Quick Start* card that accompanied the product. Please refer to it for installing the ACUCOBOL-GT development system and Acu4GL.

Once the installation is complete, please refer to this appendix for setting up your Acu4GL product.

Regarding relinking for Windows users

Relinking is *not* required for Windows users (but it *is* required for UNIX users and is explained in **section C.2.2, "UNIX Installation Steps"**). Windows users access Oracle through a DLL file detected at runtime, "a4ora32.dll". For compatibility purposes, this file is a copy of the file "a4oraoci.dll" and is created during the installation process.

Note: If users are using Oracle client software to connect to a database not on the current machine, they must set up an alias for the database they are connecting to using Oracle's "Net Configuration Assistant" or a similar tool. For example, you would use such a tool to set up an alias called "sun10" that has these settings:

Protocol: TCP/IP
Host Name: SUN10
Database Instance: ORCL

You should then test this alias using SQL*Plus with a login of "username@sun10". Once this configuration is verified, you can set the **ORA_LOGIN** variable in the COBOL configuration file to that value, for example, "ORA_LOGIN username@sun10". If you prefer, this setting can be broken up using the new configuration variable A_ORA_DATABASE as follows:

```
ORA_LOGIN username
ORA_PASSWD userpass
A_ORA_DATABASE remotesrv
```

Information on Oracle's "Net Configuration Assistant" can be found in your Oracle documentation.

C.2.2 UNIX Installation Steps

The Acu4GL for Oracle product on UNIX is an add-on module that must be linked with the ACUCOBOL-GT runtime system. For this reason, you'll need a C compiler to install the Acu4GL product. To interface, you must use the ACUCOBOL-GT compiler and runtime; the version of the runtime must match the version of Acu4GL.

The Acu4GL product is shipped using either TAR or CPIO format, depending on the type of machine you have. The label on the medium shipped to you tells you which format has been used.

From your Acucorp directory, choose where you want to install the Acu4GL product (or create a new directory for it) and then enter one of the following commands:

```
tar xfv device
```

or

```
cpio -icvBd < device
```

This will copy the files from the distribution medium to your Acucorp directory structure. *device* is the appropriate hardware device name (for example, /dev/rdiskette or /dev/rmt0). Sites using Texas Instruments System 1500 should add an uppercase "T" to the **cpio** options (-icvBdT).

Contents of the medium

Note that each Acu4GL product has its own *license file* that must be located in the same directory as the ACUCOBOL-GT runtime. For Oracle, the license file is distributed with the name "runcbl.olc".

C.2.3 Completing the Installation

With appropriate installation settings and procedures, Acu4GL communicates with versions 9*i* or later of Oracle. To install the Acu4GL product, perform the following steps.

Step 1: Install Oracle

The Oracle RDBMS must be installed and configured *prior* to the installation of Acu4GL for Oracle.

We also recommend:

- Oracle's SQL*Plus® product, which is an interactive query tool. This is not mandatory, but it will give you quite a bit of flexibility. SQL*Plus allows you to do database work outside of COBOL, including interactive queries, table creation, table modification, and creation of views, constraints, and relationships between tables.

- Oracle's "scott/tiger" test schema, which can be used for testing.

Micro Focus does not provide these products.

Make sure all environment variables Oracle requires, such as ORACLE_HOME, ORACLE_SID, and LD_LIBRARY_PATH (or equivalent, if necessary), are set up in the environment of each user who will be accessing Acu4GL for Oracle (see your Oracle documentation for the list of variables required). Verify Oracle's setup and configuration using an Oracle tool, such as SQL*Plus. Also verify that the PATH references the necessary Oracle subdirectories.

Step 2: Create a new runtime system

Note: In the following directions, the term "runtime system" refers to the runtime shared object on systems where the ACUCOBOL-GT runtime is a shared object and to **runcbl** on other systems, where the runtime is static. The runtime is a shared object on the following systems: AIX 5.1 and later, HP-UX 11 and later, and Solaris 7 and later. To check, look at the contents of the "lib" subdirectory of your ACUCOBOL-GT installation. If the files "libruncbl.so" or "libruncbl.sl" reside in that directory, the runtime is a shared object on your system.

Complete steps 2a through 2c below to create a new runtime that includes the Oracle Acu4GL product.

You may also link your own C routines with the runtime system.

2a. Execute the script

The **ora_inst** script is an interactive shell script that determines which libraries your version of Oracle has, and then creates a makefile suitable for linking Acu4GL for Oracle. In the instructions below, ***bold italicized*** text within a message indicates that the script inserts what you had previously typed.

Execute the shell script **ora_inst** by typing the following command:

```
sh  ora_inst
```

Note: You may exit the script at any time by pressing the system interrupt key (usually CTRL+C).

This is the first message that you see:

```
If this script dies with an error like "VAL=0: command not
found", try executing it with a bourne shell, as in sh ora_inst
```

If the script does terminate, and if re-entering the command does not work, call Technical Services.

When the script begins executing, the following message is displayed (no response is needed):

```
We first need to determine where you have installed ORACLE.
```

If the environment variable ORACLE_HOME has been set, the script continues, and the next message you see is the version inquiry below. If ORACLE_HOME has *not* been set to the Oracle directory name, the installation script displays this message:

```
Enter the directory where ORACLE is installed:
```

Type the name of the directory in which you installed Oracle, and then press **Enter**. The script then determines whether a directory by that name exists; if not, the script displays the following message:

```
entered directory isn't a directory.
```

and returns to the "Enter the directory" prompt to give you another chance. After you have entered a legitimate directory name, the script responds with:

```
Creating Makefile.ora ...
You should now be able to execute a make
with the command make -f Makefile.ora in
the … <ACUCOBOL-GT directory>/lib
```

These are just informative messages.

Note: Be sure to review the "makefile.ora" file and make adjustments as necessary.

2b. Link the runtime system

If you need to link in your own C routines, add them to the **SUBS=** line of the file "Makefile.ora". See Chapter 6 in *A Guide to Interoperating with ACUCOBOL-GT* for details on linking C subroutines.

Make sure you are in the directory containing the ACUCOBOL-GT runtime system. Then, at the UNIX prompt, enter the following command:

```
make clean
```

to ensure that you have a clean directory in which to build your runtime.

Now enter the following command:

```
make -f Makefile.ora
```

or

```
makerun
```

This compiles “sub.c” and “filetbl.c” and then links the runtime system.

Note: Make sure you have correctly set all of the flags specific to your platform correctly before you relink your library and object files.

If the make fails because of an out-of-date symbol table, execute the following command:

```
ranlib *.a
```

and then execute the make again. If the make fails for any other reason, call Technical Services.

If you are relinking your runtime under SCO UNIX, you must add “-lrpc -lsocket” to the LIBS section of “Makefile.ora”. If you place them on a line by themselves, remember to place a backslash at the end of the previous line. You must also remove these two libraries from the “ACUSERVER_LIBS” line.

2c. Verify the link

Enter the following command:

```
./runcbl -vv
```

to verify the link. This returns version information on all of the products linked into your runtime system. Make sure it reports the version of Acu4GL for Oracle.

Shared libraries

If you have relinked the ACUCOBOL-GT runtime and receive an error message of this type when you try to execute it:

```
"Could not load library; no such file or directory"
"Can’t open shared library . . . "
```

this may mean that your operating system is using shared libraries and cannot find them. This can occur even if the shared libraries reside in the same directory in which you are currently located.

Different versions of the UNIX operating system resolve this in different ways, so it is important that you consult your UNIX documentation to resolve this error.

Some versions of UNIX require that you set an environment variable that points to shared libraries on your system. For example, on an IBM RS/6000® running AIX® 4.1, the environment variable LIBPATH must point to the directory in which the shared libraries are located. On HP/UX, the environment variable that must be set to point to shared libraries is SHLIB_PATH. On UNIX SVR4, the environment variable is LD_LIBRARY_PATH. Note that the correct version of the ORACLE_HOME/lib should be in your LD_LIBRARY_PATH. This might be ORACLE_HOME/lib64 or ORACLE_HOME/lib, depending on what the default is for your system. If a lib32 directory exists, the 64-bit libraries may be the default. Please refer to your Oracle release notes for more information.

Be sure to read the system documentation for your operating system to determine the appropriate way to locate shared libraries.

A second way to resolve this type of error is to link the libraries into the runtime with a static link. Different versions of the C development system use different flags to accomplish this link. Please consult the documentation for your C compiler to determine the correct flag for your environment.

Step 3: Copy runcbl to the correct directory

If the runtime system is a statically linked **runcbl**, copy the new executable to a directory mentioned in your execution path. This file needs to have *execute* permission for everyone who will be using the runtime system. The copy step is not necessary when the runtime system is a shared library.

The ACUCOBOL-GT license file for the runtime (“runcbl.alc”) and the license file for the Acu4GL product to Oracle (“runcbl.olc”) must be copied into the same directory as the runtime executable. If you rename your runtime executable, be sure to rename your license files to base name, with the extensions unchanged. For example, if you rename your runtime to be

"myprog.bin", the license file for the Acu4GL product for Oracle should be renamed "myprog.olc", and the license file for the runtime should be renamed "myprog.alc."

The remaining files can be left in the directory into which they were installed from the distribution medium.

C.2.4 Checking System Parameters

Acu4GL makes use of cursor caching to process Oracle RDBMS data efficiently. A cursor is a prepared query, saved in a parameterized format. When a like function is to be performed on a different record (such as a GET NEXT), the task can be performed by executing the saved query with new parameters. This improves performance by eliminating the need to regenerate the query. The number of cursors available to your COBOL application may need to be adjusted, as described below.

Setting the parameters

In summary, what you need to do is:

- Set the Oracle system parameter *open_cursors* to indicate the total number of cursors you want to have. To do this, look in Oracle's "init.ora" and "initSID.ora", or any other database initialization files used at your site, and add or modify the line:

  ```
  open_cursors = nnn
  ```

- The number of cursors can be equal to the number of keys times the number of segments per key for each file.

  For example, if you have five files, each of which has three keys, and each key has three segments, you should set open_cursors to a minimum of "45".

Note: Setting OPEN_CURSORS should be performed only by your System Administrator or Database Administrator, because it affects all applications using the database.

C.2.5 Setting Up a User Account

Acu4GL for Oracle must be able to connect to a user account. You may either set up one general account for all users or an account for each individual user. To set up an account, you must have Database Administrator (DBA) privileges.

Note: You may opt to use Oracle's Enterprise Manager on Windows platforms to simplify the login process. If you decide to use the Enterprise Manager, please skip this section, and refer to the Enterprise Manager documentation.

Configured login

If you want to specify a database login name other than the system login name, you can use configured login. The SQL command syntax to enroll a new user by configured login is:

```
GRANT CONNECT, RESOURCE TO db_login_name IDENTIFIED BY
    password
```

db_login_name is the name you want to use to log in to the database.

For more details on login, see your site's system administrator and the Oracle *Database Administrator's Guide*.

C.2.6 Setting Up the User Environment

The user's account should have been set up correctly to access the Oracle RDBMS system. This includes environment variables such as ORACLE_HOME and ORACLE_SID. See your Oracle documentation for more details.

In addition to the variables required for Oracle, you must do the following:

Ensure that your execution path contains the name of the directory where you placed your newly linked Acu4GL runtime executable.

If you are not using the automatic login procedure, you will need to set the **ORA_LOGIN**, **ORA_PASSWD**, and possibly **A_ORA_DATABASE** variables either in your environment or in the ACUCOBOL-GT runtime configuration file. For security reasons, it is best to set the password variable from your COBOL program by asking the user to enter a password and then executing:

```
SET ENVIRONMENT ORA_PASSWD TO user-entry
```

You may want to make and use a personalized copy of the configuration file to avoid impacting other users. The *ACUCOBOL-GT User's Guide* describes how to use the A_CONFIG environment variable, or the "-c" runtime option, to identify a personal configuration file.

If you will be accessing files that you do not personally own and that do not have public synonyms, you will need to set the configuration variable **USER_PATH**.

For detailed information on ORA_LOGIN, ORA_PASSWD, and USER_PATH, see section C.4, "Acu4GL for Oracle Configuration File Variables," in this appendix.

C.2.7 Designating the Host File System

If you are opening an *existing* file, all file systems linked into the runtime are searched for the named file. If, however, you are creating a *new* file, you must tell the runtime which file system to use. You accomplish this with one of two configuration file variables; the first is:

```
DEFAULT_HOST filesystem
```

This designates the file system to be used for newly created files that are not individually assigned. For example,

```
DEFAULT_HOST ORACLE
```

means that all new files will be Oracle files unless otherwise specified by the second configuration variable, which is:

```
filename_HOST filesystem
```

where *filename* is the file name, without any extension, named in the ASSIGN TO clause of your SELECT statement. This configuration variable is used to assign an individual data file to a file system. Any file so assigned will use the designated file system, and not the one specified by DEFAULT_HOST. For example,

```
myfile_HOST VISION
```

means that *myfile* will be under the Vision file system.

You can use these configuration file variables in combination to assign your new files in a default with exceptions manner; for example, this set of entries:

```
DEFAULT_HOST ORACLE
afile_HOST VISION
bfile_HOST VISION
```

means that all new files except *afile* and *bfile* will be assigned to Oracle, and those two files will be assigned to Vision.

You can change the values of these variables during program execution by including in your code:

```
SET ENVIRONMENT "filename_HOST" TO filesystem
```

or

```
SET ENVIRONMENT "DEFAULT_HOST" TO filesystem
```

This enables you to change file systems during the execution of your program. (This is not the typical way to specify a file system; normally it is designated in the runtime configuration file and is not changed in the COBOL program.)

Note: Acu4GL for Oracle allows you to create a table with an OPEN OUTPUT statement, just as you can create Vision indexed files. The Oracle equivalent of a Vision file is a table, not a database. You must create a database for your Oracle tables before you run the COBOL program that creates the tables, just as you must create a directory for your files before you run a COBOL program that creates Vision files.

C.2.8 Setting Up the Search Path

If you own a file because you created it with OPEN OUTPUT from COBOL, or because you created it using SQL, you do not need to do anything more to access the file. However, if you want to access files that are owned by other users on the system and do not have public synonyms, you must provide the runtime with information on how to locate these files.

The USER_PATH configuration variable is used by the runtime to locate files. It functions in much the same way as ACUCOBOL-GT's FILE_PREFIX variable (see Chapter 2 in Book 1, *User's Guide,* of the ACUCOBOL-GT compiler documentation). The syntax for this variable is:

```
USER_PATH  user1 [user2]...
```

Where the *user* argument may either be the name of a user on the system or a period (**.**), which indicates the files owned by yourself. For example, if you have the following settings:

```
ORA_LOGIN    =   OPS$ORACLE
ORA_PASSWD   =   DATA_FERIT
USER_PATH    =   barbara scott.
```

and then attempt to open for I/O the file "myfile", the runtime uses the following search path:

```
OPEN I-O       barbara.myfile
OPEN I-O       scott.myfile
OPEN I-O       ops$oracle.myfile
```

Remember that another option for allowing access to a file to all users on the system is to create a public synonym for the file.

C.2.9 Handling Transactions

Issuing a COMMIT WORK database command terminates a transaction and makes all work permanent and accessible to other users. The Acu4GL for Oracle runtime issues this command automatically when no locks are held, but you can cause it to happen under other conditions by setting the value of the **COMMIT_COUNT** configuration file variable, which is discussed in detail in section C.4, "Acu4GL for Oracle Configuration File Variables".

For direct control over the transaction logging facility in Oracle, use the transaction management features available with ACUCOBOL-GT compiler. See Chapter 5 in Book 1, *User's Guide,* of the ACUCOBOL-GT compiler documentation.

Note: You are now ready to use the **sql.acu** program. Go to **section 2.4, "Using the "sql.acu" Program,"** to learn about using this Micro Focus-provided utility.

After you learn about and use this utility, you are ready to find out about preparing and compiling your COBOL program, followed by learning to use the demonstration program. All of this information can also be found in **Chapter 2, "Chapter 2: Getting Started."**

C.3 Oracle's Instant Client

Instant Client, a free download from Oracle, allows you to run your applications without installing the standard Oracle client or having an ORACLE_HOME.

Note: Setting up Oracle Instant Client can be challenging and should be performed only by advanced users.

To use Oracle's Instant Client with Acu4GL for Oracle, download the "Basic Lite" and "SQL*Plus" packages from Oracle. Follow Oracle's instructions to install and configure the products and test their connection using SQL*Plus *before* proceeding to work with Acu4GL.

Once you have completed this process, test the installation. We recommend that you go to a different directory and then type the following command:

```
sqlplus scott@bigdb/tiger
select user from dual;
```

On UNIX, if this test is successful, relink the runtime to use the Instant Client. For instructions on relinking, see **section C.2.3, "Completing the Installation."** The **ora_inst** script will recognize the Instant Client structure; just tell it the directory into which you installed the Instant Client when asked for the ORACLE_HOME setting.

On Windows, if this test is successful, you are ready to use the runtime. The Windows runtime loads the Oracle Instant Client dynamically from the specified path. To verify that you have loaded the application successfully, enter

```
wrun32 -v
```

on the command line.

You should see "Acu4GL ORACLE/OCI file system" included on the ACUCOBOL-GT window that is displayed.

C.4 Filename Translation

As you prepare to work with Acu4GL for Oracle, you may find it helpful to understand the rules around filename interpretation and to understand how the names of tables and XFD files are formed and work together.

When the ACUCOBOL-GT compiler generates XFD files, it uses lowercase letters to name the XFD file. In addition, the compiler changes hyphens to underscores when naming the XFD file.

Through configuration variables, the runtime translates the file name in the COBOL program into the filename that is passed to the **open()** function in the runtime. The **open()** function determines which file system to pass the request to, but does not change the name of the file. For additional information on configuration variables, see Appendix H in Book 4, *Appendices*, of the ACUCOBOL-GT documentation set.

At this point, Acu4GL for Oracle translates the file name to uppercase letters and changes hyphens to underscores. This "new" name is the one that Acu4GL for Oracle will use in the future for references to the database table.

C.5 Configuration File Variables

This section lists the runtime configuration file variables that are specific to Acu4GL for Oracle. Configuration file variables that are generally applicable to any RDBMS with which Acu4GL communicates are discussed in **section 8.2, "Runtime Configuration Variables."**

A_ORA_DATABASE

A_ORA_DATABASE specifies the name of a network service that you have set up to refer to a database using an Oracle tool such as "Net Configuration Assistant," or by manually editing "tnsnames.ora".

See also

ORA_LOGIN configuration variable

ORA_PASSWD configuration variable

A_ORACLE_ERROR_FILE

This configuration file variable allows you to map errors using a text file to supplement the default method of providing errors. By adding the name of the file that contains the actual error mapping to A_ORACLE_ERROR_FILE, database-specific errors are mapped to COBOL errors.

Note: Some messages related to COBOL I/O cannot be supplemented.

Example

A sample syntax for this configuration file variable would be:

```
A_ORACLE_ERROR_FILE=ORCLerrs
```

where:

ORCLerrs is a file of a specified format containing a mapping of database-specific errors to COBOL errors. One such entry might be:

```
1 DUPLICATE_RECORD
```

A_ORA_HINTS

This variable enables the Oracle Hints feature. The default value is "1" (on, true, yes). To turn off this feature, set it to "0" (off, false, no).

Oracle Hints can improve database performance considerably in some situations, by "strongly suggesting" to the oracle query optimizer that it use a given index to process a query.

Note that keys must use the ACUCOBOL-GT naming convention for naming indexes in the database to use this function. For example, the primary key on the file "ftest" would be named "iftest_0".

A_ORA_KEEP_START_CURSOR

When a COBOL application performs a start operation, Acu4GL/Oracle builds a sequence of one or more SQL queries to satisfy the requirements of the start operation. When a result set is found that matches the criteria of the start operation, that key value is saved for use on a subsequent READ NEXT/PREVIOUS operation. The cursor used to locate the target key value is not left open in order to maximize concurrency. When a READ NEXT/PREVIOUS operation is subsequently performed, a new query is generated. This could result in a sequence of queries such as the following.

For the START operation:

```
  WHERE ( partial_key1_seg1 = :w0 AND partial_key1_seg2 =
:w1  ) ORDER BY
   partial_key1_seg1 ASC, partial_key1_seg2 ASC,
partial_key1_seg3 ASC
```

For the READ NEXT operation:

```
WHERE ( partial_key1_seg1 = :w0 AND partial_key1_seg2 >= :w1
 ) ORDER BY
```

```
    partial_key1_seg1 ASC, partial_key1_seg2 ASC,
partial_key1_seg3 ASC
```

Hence, THE configuration variable:

A_ORA_KEEP_START_CURSOR

The potential settings for this variable are:

FALSE	the current default. The interface will function as in previous versions.
TRUE	the cursor used for the start operation will be maintained in an open state for use by subsequent operations. Setting this variable to TRUE will eliminate the generation of the second SQL query for the READ operation.

A_ORA_LIMIT_DROPDOWN

When a sequence of START and READ NEXT/PREVIOUS operations are performed by an application, Acu4GJ/Oracle generates a sequence of queries to return the set of records matching the application's request. To improve performance, the interface generates a sequence of "drop-down" queries based upon the key of reference's key segments going from the most specific subset using the most number of segments to the most general using the least number of segments.

For example if a key is described by:

```
03 MY-ALTKEY.
   05 MY-ALTKEY-SEG1        PIC X(2).
   05 MY-ALTKEY-SEG2        PIC X(2).
   05 MY-ALTKEY-SEG3        PIC X(2).
```

then a START followed by a sequence of READ NEXT operations might generate the selection criteria of:

```
WHERE MY-ALTKEY-SEG1 = :w0 AND MY-ALTKEY-SEG2 = :w1 AND
  MY-ALTKEY-SEG3 >= :w3
WHERE MY-ALTKEY-SEG1 = :w0 AND MY-ALTKEY-SEG2 > :w1
WHERE MY-ALTKEY-SEG1 > :w0
```

This can improve performance in that the target for each query is kept to a minimal size. If a set of records is not required, the database does not need to spend the time building the working set. When a "DROP DOWN" does occur however, the subsequent working set can require a large amount of time to process, because it may be an order of magnitude greater number of records. Normally there is not a way for a COBOL application to instruct the interface to stop processing when it has finished with the records based on a given key segment.

We have introduced a new variable to provide this instruction:

A_ORA_LIMIT_DROPDOWN

This variable allows an application to direct the interface not to perform "drop-down" query generation and instead return "end of file" when the records matching the current query have been exhausted.

The potential settings for this variable are:

Setting	**Description**
OFF	The current default. The interface will perform "drop-down" queries.
PARTIAL	If the record positioning was performed by a START with a SIZE clause, such that the initial positioning was performed using fewer than the total number of columns in the key, the process will cease after all records matching the START columns have been exhausted.

Setting	Description
FULL	If the record positioning was performed by a START without a SIZE clause, the process will cease after all records matching the START columns have been exhausted.
ALL	Regardless of what form of START was used for the initial positioning, the process will cease after all records matching the START columns have been exhausted.

This variable is performed at the time of each start, so the application may change its value.

A_ORA_MAX_FILE_CURSORS

This configuration variable allows you to set the maximum number of cursors that the runtime is allowed to have per file. This is similar to the configuration variable MAX_CURSORS found in Acu4GL for Oracle prior to Version 6.2.0, except that the number is on a per-file basis as opposed to the maximum number of cursors available to the runtime in total.

The default value for A_ORA_MAX_FILE_CURSORS is "0", meaning that the runtime uses all of the cursors that it can. Because each file must have access to at least two cursors, set this variable to a value greater than "2" if you want to specify a particular value. If all the allotted cursors for a file have been used, the least recently used cursor for that file is freed.

Note that this variable is read only at startup time, and should, therefore, be set in the configuration file.

A_ORA_NLS_SORT

This configuration variable forces the sort order of returned records to be in a binary order. Set A_ORA_NLS_SORT to "1" to turn off the default binary order and use the native language sort instead. The default is "0", which means that the sort order will not be the NLS sort order and will, instead, be binary.

This variable must be set before the first database file is accessed.

A_ORA_USE_PRIMARY

Acu4GL/Oracle adds the segments of the primary key to the "ORDER BY" clause when reading on an alternate key that allows duplicates. This is done to insure that the keys are returned in a predictable order should the application change the direction it is reading the information in. Your application can now control whether or not this occurs.

Setting the variable A_ORA_USE_PRIMARY to FALSE will prevent the addition of the primary key segments to the "ORDER BY" clause for alternate keys allowing duplicates. The default of TRUE will maintain the current functionality of adding the primary key segments. This variable is checked at the time of file open and may be set differently for different files.

A_ORA_WAIT_LOCK

This configuration variable determines the type of Oracle locks used: wait or nowait locks. The default is “FALSE”, nowait locks. This variable is read at file open time for each file.

A_ORACLE_ERROR_FILE

This configuration file variable allows you to map errors using a text file to supplement the default method of providing errors. By adding the name of the file that contains the actual error mapping to A_ORACLE_ERROR_FILE, database-specific errors are mapped to COBOL errors.

Note: Some messages related to COBOL I/O cannot be supplemented.

Example

A sample syntax for this configuration file variable would be:

```
A_ORACLE_ERROR_FILE=ORCLerrs
```

where:

ORCLerrs is a file of a specified format containing a mapping of database-specific errors to COBOL errors. One such entry might be:

```
1 DUPLICATE_RECORD
```

COMMIT_COUNT

The value of COMMIT_COUNT indicates the conditions under which you want to issue an automatic COMMIT_WORK operation.

Examples

COMMIT_COUNT **0**

A commit is issued when no locks are held, either because all files that had locked records have been closed, or because a COBOL COMMIT verb has been issued. This is the default value.

COMMIT_COUNT ***n***

A commit is issued after *n* operations. WRITE, REWRITE, and DELETE count towards *n*; READ, START, and READ NEXT do not.

COMMIT_COUNT **-1**

No commit is issued by the Acu4GL product. When COMMIT_COUNT is set to "-1", there are two alternate ways to perform a commit or rollback:

- One way is to call **sql.acu** with COMMIT WORK or ROLLBACK WORK.
- The second way is to use the COBOL verbs COMMIT and ROLLBACK, available in ACUCOBOL-GT.

COMMIT_COUNT is set to "-1" internally when you use the transaction management facilities available in the ACUCOBOL-GT compiler.

A COMMIT WORK, however, is issued on exit from the runtime (for example, on execution of a STOP RUN).

See also

Section 2.4, "Using the "sql.acu" Program"

Section C.7, "Supported Features"

ECN-GL443

In previous versions of Acu4GL, if an Oracle database was created with NLS_CHARACTERSET set to AL32UTF8 and NLS_LENGTH_SEMANTICS to CHAR, a START within a COBOL program using Acu4GL to access the database encountered a 23 error, on a key that should have be found.

Version 8.1.2, ECN-GL443 modified this behavior. the new default behavior of the product is to cause the START to locate the desired record. If you encounter a problem with the new behavior, using the configuration variable ECN_GL443 and setting it to "N" will revert the product back to the original behavior.

ORA_LOGIN

ORA_LOGIN indicates the user name under which you want to connect to the database system.

Example

To connect to the database with the user name MYNAME, you would specify:

```
ORA_LOGIN  MYNAME
```

in the configuration file.

See also

ORA_PASSWD configuration variable

A_ORA_DATABASE configuration variable

Section C.2.5, "Setting Up a User Account"

ORA_PASSWD

The variable ORA_PASSWD should be set to the password assigned to the database account associated with the user name specified by ORA_LOGIN.

Example

For example, if the account with the user name has the associated password "CW021535", you would specify:

```
ORA_PASSWD  CW021535
```

in the configuration file or the environment.

For a full login from a client to a remote server, you could have either of the following two settings:

```
ORA_LOGIN username
ORA_PASSWD userpass
A_ORA_DATABASE remotesrv
```

or

```
ORA_LOGIN username@remotesrv
ORA_PASSWD userpass
```

In this example, *remotesrv* is the net service name you have set up, as defined above in the section dealing with A_ORA_DATABASE.

See also

ORA_LOGIN configuration variable

A_ORA_DATABASE configuration variable

USER_PATH

USER_PATH indicates the user name or names to be used when Acu4GL searches for files. The order of the names is significant. The syntax for this variable is:

```
USER_PATH  user1 [user2]...
```

where the *user* argument may be either the name of a user on the system, or a period (“.”), which indicates the files owned by yourself.

The type of OPEN being issued for the file determines the effects of this setting.

Examples

If an OPEN INPUT or OPEN I/O is issued, and a USER_PATH variable is defined in the runtime configuration file, Acu4GL searches for a user of the named file in the following sequence of places:

1. The list of users in USER_PATH.
2. The public synonyms

The first valid file is opened.

If USER_PATH is defined and the current user is the owner of the file, the current user must be included as one of the users (as indicated by a “.” in the USER_PATH). If this is not the case, even though the current user has created the table, it will not be found and a file error “35” will result. This circumstance can occur if the file was created with the OPEN OUTPUT phrase and “.” is not an element in USER_PATH. When the table is created, the current user will be the owner of that table. When the runtime attempts to open the table, the runtime will not look for tables owned by the current user unless USER_PATH is not set, or “.” is part of the USER_PATH setting.

If an OPEN INPUT or OPEN I/O is issued, and *no* USER_PATH variable is in the runtime configuration file, Acu4GL searches for a user of the named file in the following sequence of places:

1. The user named for login (ORA_LOGIN or OPS$*username*)
2. Public synonyms

Acu4GL opens the first file that has a valid combination of user and file name.

If an OPEN OUTPUT is issued (whether USER_PATH is present or not), a new table is created with the owner being:

1. The name specified in ORA_LOGIN
2. The name constructed by automatic login (see **section C.2.5, "Setting Up a User Account."**)

See also

Chapter 2 in the *ACUCOBOL-GT User's Guide* contains a discussion of the FILE_PREFIX configuration variable and general guidelines on the use of file search paths.

Section C.2.8, "Setting Up the Search Path"

C.6 Using the Database Table

The database table is built and accessed automatically when the COBOL application is executed. To the end user, the interaction between the COBOL and the Oracle database is transparent, as though they were part of one process.

You can also access the database information directly from Oracle, at your option.

C.7 Supported Features

OPEN ALLOWING READERS is not supported by Oracle. You determine in your runtime configuration file how this phrase will be interpreted. Set the variable STRENGTHEN_LOCKS to "1" to cause this phrase to be treated as OPEN ALLOWING NO OTHERS. Set the variable to "0" to cause the phrase to be treated as OPEN ALLOWING ALL. The default value is "0".

Transactions are enforced in Oracle. Within a transaction, records locked by I/O operations that modify, delete, or otherwise result in a COBOL record-lock, remain locked until a COMMIT WORK or a ROLLBACK WORK is issued, or until your COBOL program encounters a COMMIT or ROLLBACK verb. See the description of the **COMMIT_COUNT** configuration variable.

Oracle does not support record encryption, record compression, or alternate collating sequences. You may include these options in your program; they will be disregarded if they are specified.

The ACUCOBOL-GT utility program **vutil** cannot be used with Oracle files. Instead, use utilities supplied by Oracle.

Whenever you are using the library routine RENAME, you must specify that you are using *indexed files*. This information is passed by the value "I" in the fourth parameter.

Unless you have DBA privileges, or special privileges provided by Oracle Version 9*i*, you can't:

- Delete a table you don't own
- Create a table under another user's name

C.8 Limits and Ranges

The following limits exist for the Oracle 9*i*® file system:

Maximum indexed key size: 250 bytes
Maximum number of fields per key: 16
Maximum number of columns: 254
Maximum length of a char field: 255

Note: The limits and ranges differ between releases of Oracle.

Acu4GL for Oracle supports the following data types.

COBOL to	**Oracle**
PIC X	CHAR
PIC X(*n*)	CHAR(*n*)
PIC X(*n*)	VARCHAR2 *
PIC X(*n*)	RAW(*n*) **
PIC 9(*n*)	NUMBER(*n*)
PIC 9(*n*)	RAW(*n*) **
PIC 9(*n*)V9(*m*)	NUMBER(*n*+*m*, *m*)
example: PIC999V99	NUMBER(5,2)

Oracle to	**COBOL**
CHAR	PIC X
CHAR(*n*)	PIC X(*n*)
VARCHAR2	PIC X(*n*) *
DATE	PIC 9(6) or PIC 9(8)
NUMBER(*n*)	PIC 9(*n*)
NUMBER(*n, m*)	PIC 9(*n*-*m*)V9(*m*)
RAW(*n*)	PIC X(*n*) ** or PIC 9(*n*) **
example: NUMBER(4,3)	PIC9V999

* indicates that the **VAR_LENGTH** directive is required.

** indicates that the **BINARY** directive is required.

Internally, Acu4GL for Oracle uses VARCHAR to prevent key fields that are all spaces from being converted to null.

These Oracle data types are not currently supported:

LONG
LONG RAW

Other limits described in Appendix B of Book 4, *Appendices*, of the ACUCOBOL-GT documentation set apply.

C.9 Runtime Errors

This section lists the Acu4GL error messages that could occur during execution of your program. **Chapter 9, "Chapter 9: Performance and Troubleshooting,"** provides information on compile-time errors and also provides several methods for retrieving runtime errors.

Each message is followed by an explanation and a recommended recovery procedure.

Runtime errors will have this format:

9D,xxxx

The *9D* indicates a file system error and is reported in your FILE STATUS variable. The *xx* is a secondary, database-specific error code. These are the secondary errors reported directly from Acu4GL for Oracle:

9D,03 Dictionary (.xfd) file not found

The dictionary file for one of your COBOL files cannot be located. Be sure you have specified the correct directory via your **XFD_PREFIX** configuration variable. You may need to recompile with the "-Fx" option to create the dictionary. See **section 8.1** for additional information on compiler options.

9D,04 Corrupt dictionary file

The dictionary file for one of your COBOL files is corrupt and cannot be read. Recompile with "-Fx" to re-create the dictionary. Note that this error may be caused by inappropriate application of a directive. Check all directives, and call Technical Services if you have questions.

9D,05 Too many fields in the key (more than 16 for Oracle)

Check your key definitions and redefine the key that is illegal, and then recompile with "-Fx".

9D,06 Mismatched dictionary file

The dictionary file (.xfd) for one of your files conflicts with the COBOL description of the file FD. The *xx* indicates a tertiary error code that is defined by the host file system. You can determine the exact nature of the mismatch by referring to the host system's error values.

The tertiary error code may have any of these values:

01 – mismatch found but exact cause unknown (this status is returned by the host file system)
02 – mismatch found in file's maximum record size
03 – mismatch found in file's minimum record size
04 – mismatch found in the number of keys in the file
05 – mismatch found in primary key description
06 – mismatch found in first alternate key description
07 – mismatch found in second alternate key description

The list continues in this manner for each alternate key.

9D,1001 Invalid Cursor

The prepare portion is aged out of cache. Re-enter the code while invalidating the prepare portion.

C.10 Common Questions and Answers

This section contains some questions and answers specific to Acu4GL for Oracle. Refer to **Chapter 10** for additional questions and answers that pertain to the Acu4GL family of products.

Question: What files do I need to link my C routines into Acu4GL?

Answer: You need to modify the file "makefile.ora" to add your C routines to the file and then relink your runtime with the command "make -f makefile.ora".

Question: When I try to open a file for output, I get the error 9D,904 Invalid Column Name. Why?

Answer: One of your record's data items probably has the same name as an Oracle reserved word. Locate the column by comparing a file trace of the CREATE TABLE to Oracle's list of reserved words and then apply the **NAME** directive to the column in question.

Question: I created a table the last time I ran the application, and now I can't find it. Why?

Answer: This is probably a file (table) ownership problem. If you were running the application under a different user name when you created the file, that table may not be visible unless the creating user's name is listed in your **USER_PATH** configuration variable.

Question: Can the Acu4GL for Oracle product support a full date/time format?

Answer: The finest time granularity that Oracle supports is one second. Fractions of a second are not supported. In order to achieve this, you must be sure the "xfd date" code is correct and specify a date-time string in your COBOL application, as opposed to just a date. See **section 4.3.5, "DATE,"** for additional information.

Question: Are there any ACUCOBOL-GT library routines that do not work with or would not make sense to use with Acu4GL for Oracle?

Answer: Yes. There are two ACUCOBOL-GT library routines that either don't work with or do not make sense to use with Acu4GL for Oracle: C$COPY and C$RECOVER.

D Acu4GL for ODBC Information

Key Topics

D.1 ODBC Concepts

With Acu4GL for ODBC, your Windows-based application can access the most common database formats found on desktop computer systems, as well as relational database management systems on UNIX or Windows NT servers.

This section gives a brief overview of concepts you'll need to work with ODBC.

D.1.1 What Is ODBC?

Open Database Connectivity (ODBC) is a library of standardized data access functions. It's intended to give programmers a way to access and manipulate data in a variety of data sources.

ODBC operates in a way that is similar to Informix ESQL/C and Oracle Pro*COBOL®. The main difference is that ODBC offers the advantage of being able to access almost any data source.

With traditional call-level interfaces (CLI), you are required to learn the Application Programming Interface (API) for each data source. If you want to port an application from one data source to another, you must write a complete new access module.

ODBC was designed expressly to access almost any data source. It offers a standard API that can be used to manipulate data in a Microsoft Access database on your PC, or connect from your PC to an Oracle database on a UNIX host. It can even access files that are not databases, such as Btrieve data files and Excel spreadsheets.

D.1.2 Origins of ODBC

ODBC is able to access a wide variety of file systems because it takes advantage of their common properties and common standards.

For example, most of the popular database systems on the market today bear a strong resemblance to each other, both in functionality and in the methods they use to access data. This is because the major vendors' products are based in whole or in part on the *relational* database concepts of E. F. Codd. ODBC makes use of these commonalties.

Shared standards also lend power to ODBC. Several groups have been working to set standards in the UNIX environment, where much of the database technology originated. These groups include the ANSI standards committee, the X/Open consortium, and the SQL Access Group. Together, these groups have devised much of the core technology that Microsoft has organized into ODBC.

D.1.3 Restrictions

ODBC does *not* make your application instantly portable to any database or data source that has ODBC support. Many layers of software are involved in the data retrieval process, and each layer adds its own restrictions.

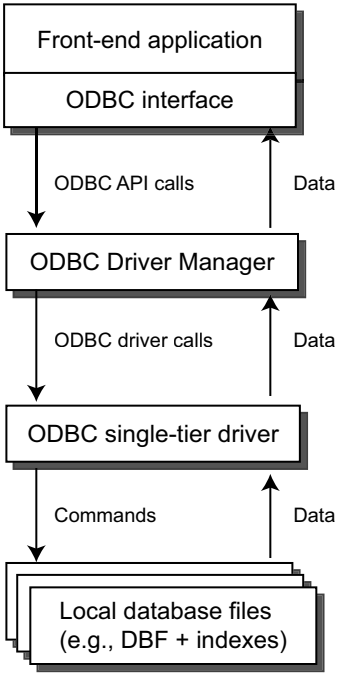
The major component that makes an ODBC connection work is the ODBC *driver*. (See **section D.1.4, "ODBC Structure,"** for more information.) The driver processes API function calls, submits SQL requests to a specific data source or DBMS, and returns results.

ODBC drivers are available from a variety of vendors and offer very different levels of support and performance. The *sources of the data* that the drivers connect to also offer very different capabilities in areas such as record locking, file operations, and transaction management. These differences in performance and capability can have a significant impact on your application.

D.1.4 ODBC Structure

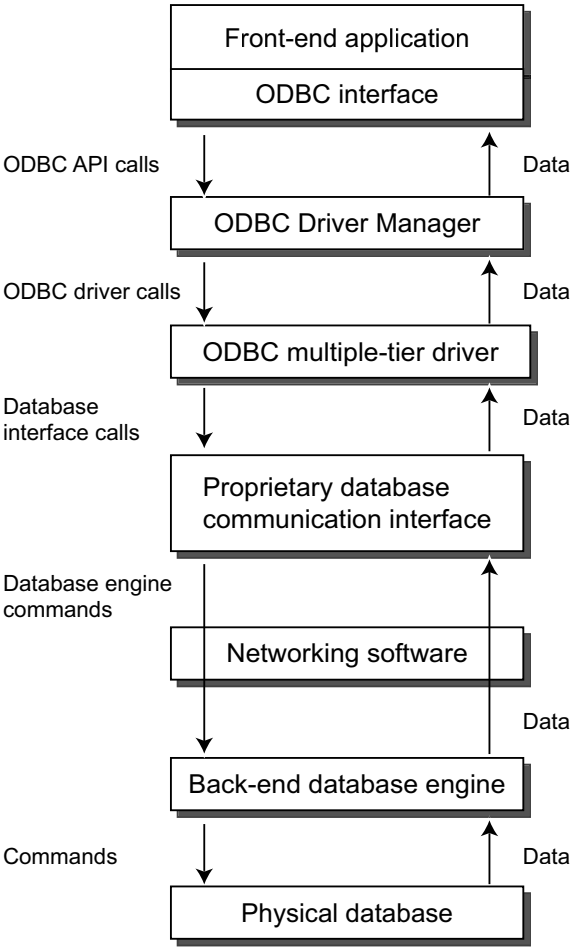
A quick overview of the components of ODBC will help you interface your COBOL program to ODBC-compliant data sources.

ODBC requires four components: an ODBC-enabled application, the ODBC driver manager, ODBC database drivers, and data sources. The following diagram indicates how these components work together in a single-tier environment.



Single-tier ODBC architecture

The following diagram indicates how these components work together in a multiple-tier environment.



Multiple-tier ODBC architecture

Application

The application is your COBOL program, or more precisely, the ACUCOBOL-GT® runtime system together with Acu4GL for ODBC. The runtime generates the appropriate SQL queries to accomplish the file I/O task that your COBOL program is executing. This includes all file positioning and record locking. When the query results are returned to the ACUCOBOL-GT runtime, they are mapped either to a COBOL record or to the appropriate COBOL status code, if one exists.

Driver Manager

The driver manager loads the ODBC driver. The driver manager then passes requests to the driver and returns results to the ACUCOBOL-GT runtime. For the Microsoft Windows environment, the driver manager is called ODBC32.DLL for 32-bit architectures.

The driver manager is typically included with the ODBC driver and is often installed along with the driver. For some products, such as Microsoft Access and Excel, the driver manager is installed automatically if you choose the ODBC support option while installing the product. (The ODBC support option tends to be part of the custom install process.)

Driver

The driver processes ODBC requests and returns data to the driver manager and your application. If necessary, the driver translates the runtime's SQL request into a form that is understood by the target data source.

Your application is not limited to communicating through just one driver, however. By setting the ACUCOBOL-GT configuration variable **A_ODBC_DATASOURCE**, you can direct the ACUCOBOL-GT runtime to connect to a different driver for any given file. Simply set this variable before opening the indexed data file. The Acu4GL interface for ODBC will then handle connecting to the new data source and performing any operations necessary to open the file.

The driver is the most important feature of ODBC. ODBC drivers for a given data source may be available from many different vendors. They vary greatly in performance and in the options they provide for a given data source. Be sure to carefully review a data source driver to determine its level of performance.

Your Acu4GL for ODBC disks include a program that tests your ODBC driver for the various functions the Acu4GL product requires. This program is called DRVTST32.EXE. You initiate it by selecting the appropriate icon in the Acu4GL for ODBC program group. Please refer to **section D.9, "Driver Requirements,"** for instructions on using the driver test program. This section also describes the minimal requirements that your driver must meet to work with Acu4GL for ODBC.

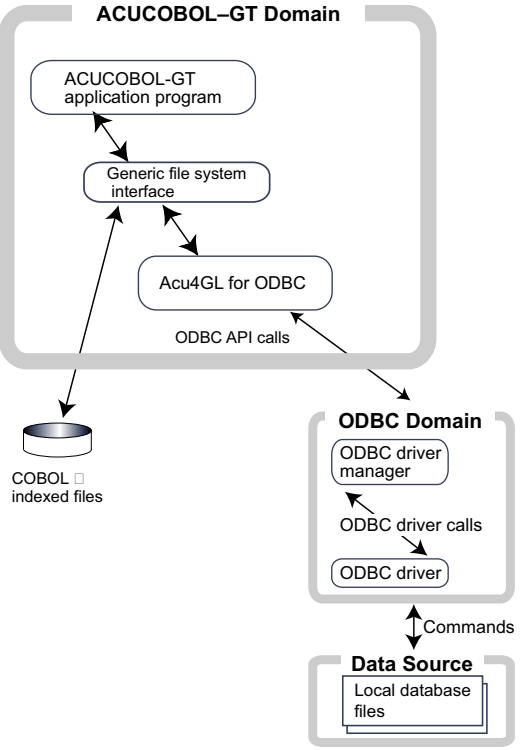
Data Sources

Data sources are the files or databases accessed by a driver. Each data source must be identified during setup.

Some data sources are simple to identify. For example, to access a text file, you would provide the name and location of the file. Some are more complex, though. For example, to access a UNIX-hosted database such as Oracle, you would identify the Windows drivers, Windows applications, network connections, and database information.

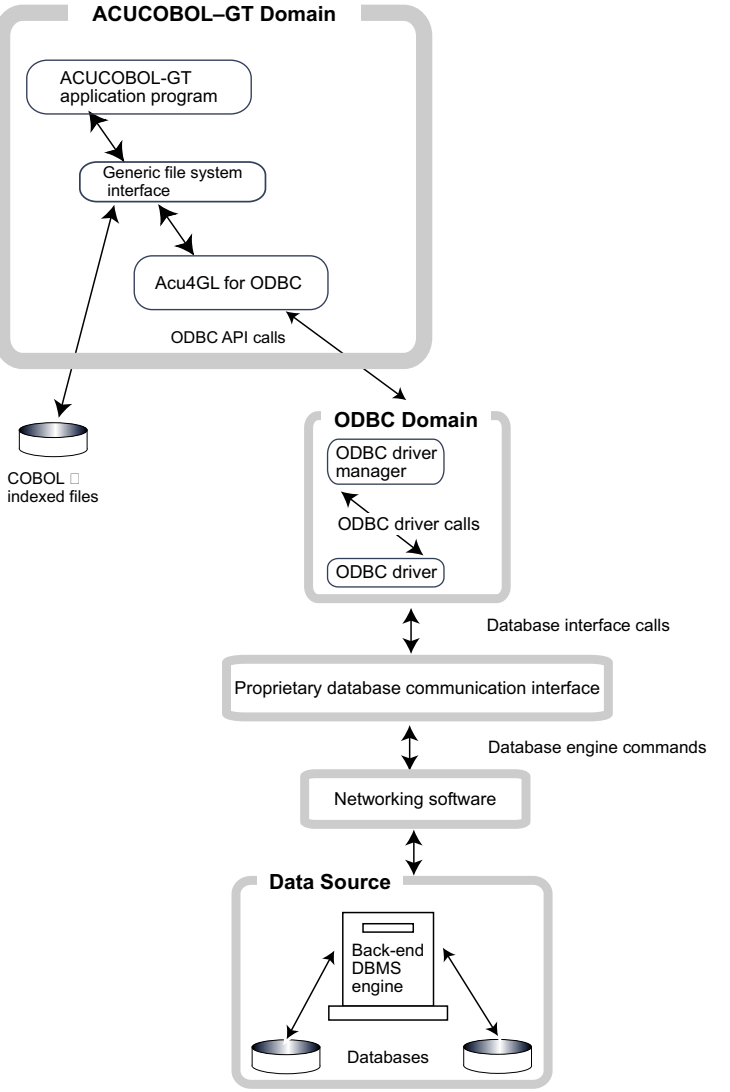
In Windows XP, you set up an ODBC data source in the Windows control panel, Administrative Tools dialog. After you double-click the Data Sources (ODBC) icon, an ODBC Data Source Administrator dialog is displayed. You must designate the driver, the data source, and any other information necessary to access the data. Setup procedures are described in detail in **section D.2, "Acu4GL for ODBC Installation and Setup."**

The following diagram shows how the Acu4GL product fits into a *single-tier* environment.



Acu4GL for ODBC in a single-tier environment

The following diagram shows how Acu4GL fits into a *multiple-tier* environment.



Acu4GL for ODBC in a multi-tier environment

D.2 Acu4GL for ODBC Installation and Setup

The following sections describe the steps you must perform *before* you begin using Acu4GL for ODBC on a new system:

- **ODBC Installation**
- **Installation of Acu4GL for ODBC**
- **Setting Up Data Sources**
- **Setting Up the User Environment**
- **Designating the Host File System**
- **Designating the Host Data Source**

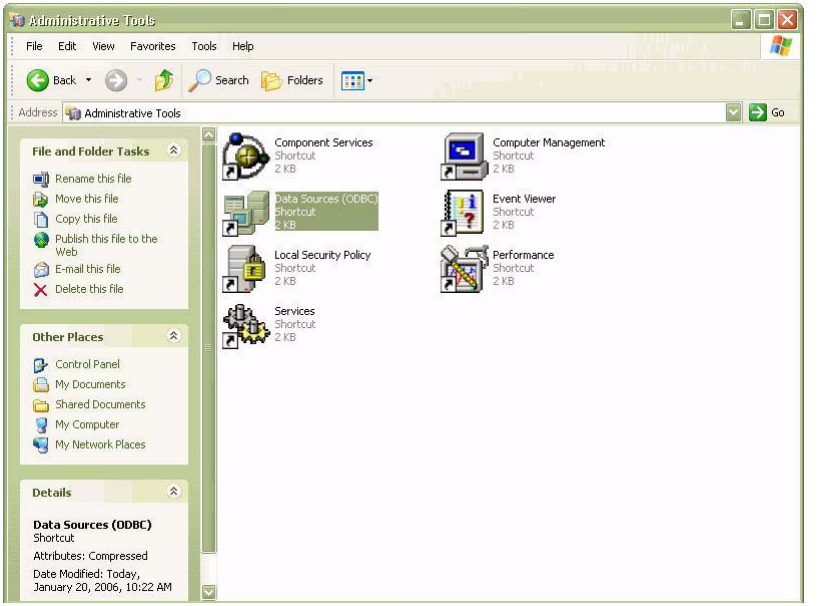
D.2.1 ODBC Installation

ODBC must be installed and configured *before* you install Acu4GL for ODBC. This product is not provided by Micro Focus, but probably was installed along with any ODBC drivers or data sources that you have on your system. For example, if you have installed Microsoft Access on your computer, the Access drivers and ODBC may have been automatically installed.

Follow these steps to determine whether you already have ODBC:

1. Open the Windows control panel.

2. On Windows XP and newer versions of Windows, double-click the Administrative Tools icon, then select Data Sources (ODBC). On older versions of Windows, select the 32-bit ODBC icon directly from the control panel.



3. Double-clicking the ODBC icon brings up the ODBC Data Source Administrator dialog showing all the data sources currently configured for your system.

   If the ODBC icon is not in your control panel, ODBC is probably not installed on your system. To install ODBC on your system, you have two options:

   - Buy an ODBC driver from a driver or database vendor and install it. ODBC is installed automatically.

   - Buy and install a Windows-based ODBC data source. The ODBC driver and ODBC will be one of the install options.

D.2.2 Installation of Acu4GL for ODBC

The Acu4GL product for ODBC is an add-on module. The ODBC Acu4GL product installation includes a .DLL file that is detected at run time.

The Acu4GL product for ODBC can be executed only on a machine that is running a 32-bit Windows platform or a 64-bit Windows platform with a 32-bit runtime and 32-bit drivers. Note that 32-bit drivers may be able to access a 64-bit database. Consult your driver vendor to be sure.

CD-ROM installation

Instructions for installing your Acu4GL product from the ACUCOBOL-GT Development Suite CD-ROM are contained on the CD booklet that accompanied the product. Please refer to it for installing the ACUCOBOL-GT Development Suite, which includes Acu4GL.

Once the installation is complete, please refer to this appendix for setting up your Acu4GL product.

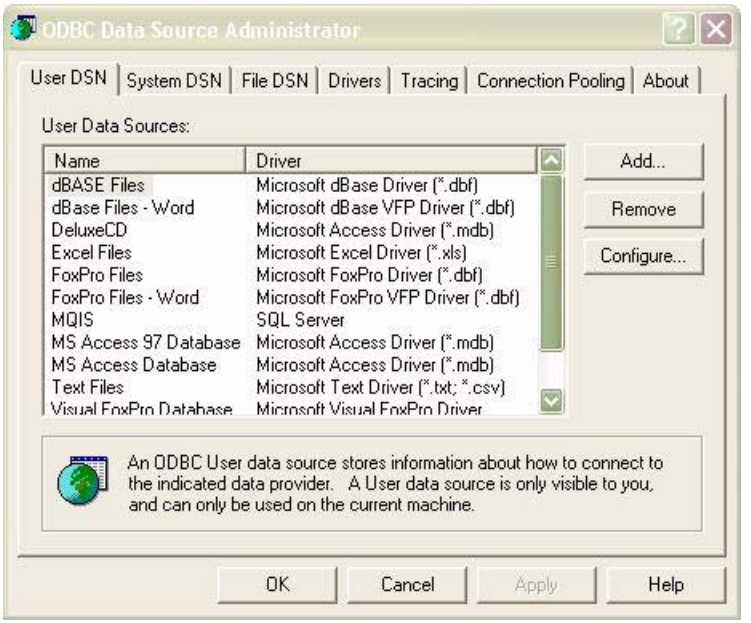
Be sure to set up your data source in the Windows control panel as instructed in **section D.2.3, "Setting Up Data Sources,"** and then enter the data source name in the configuration variable **A_ODBC_DATASOURCE**, as described in section D.4. If your driver requires them, be sure to set **A_ODBC_LOGIN** and **A_ODBC_PASSWD**. Also set **DEFAULT_HOST**. Other configuration variables that may make finding and accessing tables faster and more precise are **USER_PATH** and **A_ODBC_CATALOG**

D.2.3 Setting Up Data Sources

You will need to set up a data source name for any item you want to be able to access from Acu4GL for ODBC. The exact steps required for this will vary for each individual driver. The general steps for setting up an ODBC data source are:

1. Open the Windows control panel and select Administrative Tools.

2. Double-click the Data Sources (ODBC) icon. The ODBC Data Source Administrator dialog is displayed. This dialog lists all the data sources currently configured for your system.



3. Double-click on the data source you plan to use to access a setup dialog. A subsequent window lets you add a description of the data source and allows you to supply other information the driver requires.

If the data source you want is not listed, choose **Add** to indicate which driver to use for the data source you will create. Remember, the exact steps required for setting up a driver vary greatly between drivers and driver vendors. Please refer to the driver's documentation for precise details on this step. We recommend that you test the setup of your new data source. Many data sources come with a testing utility for this purpose.

D.2.4 Setting Up the User Environment

In addition to setting the variables required for ODBC, you will need to do the following:

1. Ensure that your execution path contains the name of the directory in which you placed the runtime executable that includes Acu4GL for ODBC.

2. Set the **A_ODBC_LOGIN** and **A_ODBC_PASSWD** variables, either in your environment or in the ACUCOBOL-GT runtime configuration file, if they are required for your data source. Also, consider whether or not setting the **USER_PATH** variable would be useful in your set up situation. For security reasons, it is best to set the password variable from your COBOL program by asking the user to enter a password and then executing:

   ```
   SET ENVIRONMENT "A_ODBC_PASSWD" TO user-entry
   ```

3. You may want to make and use a personalized copy of the configuration file to avoid impacting other users. The ACUCOBOL-GT documentation describes how to use the A_CONFIG environment variable or the "-c" runtime option to identify a personal configuration file.

4. Define any other configuration file variables you require for your unique environment. Possible variables define error map file location, locking method, commit count, commit timing, and BINARY/CHAR type conversion.

For detailed information on ACUCOBOL-GT configuration file variables, see **section D.5, "Configuration File Variables."**

D.2.5 Designating the Host File System

Each time the COBOL application creates a *new* file, it needs to know which file system to use. The runtime checks the file system to use when the file is opened. You provide the name of the file system with one of two runtime configuration file variables. The first is:

```
DEFAULT_HOST filesystem
```

This will designate the file system to be used for newly created files that are not individually assigned. For example,

```
DEFAULT_HOST ODBC
```

means that all new files will be ODBC files unless otherwise specified by the second configuration variable, which is:

```
filename_HOST filesystem
```

where *filename* is the file name, without any extension, named in the ASSIGN TO clause of your SELECT statement. This configuration variable is used to assign an individual data file to a different file system. Any file so assigned will use the designated file system, and not the one specified by DEFAULT_HOST. For example,

```
myfile_HOST VISION
```

means that *myfile* will be under the Vision file system. The ability to designate a different file system for certain files enables you to tailor your application to a specific customer's needs or to implement an incremental conversion for a customer. With relational databases, this is particularly useful in that it allows you to tune an application for processing speed and resource requirements.

You can use these runtime configuration file variables in combination to assign your new files in a default with exceptions manner; for example, this set of entries:

```
DEFAULT_HOST VISION
afile_HOST ODBC
bfile_HOST ODBC
```

means that all new files except *afile* and *bfile* will be assigned to Vision, and those two files will be assigned to ODBC.

You can also change the values of these variables during program execution by including in your code:

```
SET ENVIRONMENT "filename_HOST" TO filesystem
```

or

```
SET ENVIRONMENT "DEFAULT_HOST" TO filesystem
```

This enables you to change file systems during the execution of your program. This is not the typical way to specify a file system; normally it is designated in the runtime configuration file and is not changed in the COBOL program.

Note: The Acu4GL for ODBC product allows you to create an ODBC table with an OPEN OUTPUT statement, just as you can create Vision indexed files, if the data source driver allows for table creation. The ODBC equivalent of a Vision file is a table, not a database. You must create a database for your ODBC tables *before* you run the COBOL program that creates the tables, just as you must create a directory for your files *before* you run a COBOL program that creates Vision files.

D.2.6 Designating the Host Data Source

You must tell the runtime system which data source to use for your ODBC file. You accomplish this by setting the configuration variable **A_ODBC_DATASOURCE** to the exact name that you set up in the Windows ODBC driver manager. You can set this variable in your COBOL configuration file if you will be using only one data source. This variable is described in section D.4., "Acu4GL for ODBC Configuration File Variables."

For example, if the data source you want to select is listed in the control panel ODBC window as:

```
MS Access Databases (Access Data (*.mdb))
```

you should add the following line to your configuration file:

```
A_ODBC_DATASOURCE MS Access Databases
```

If you used the **Add** option to configure a new data source driver, set A_ODBC_DATASOURCE to the personalized name that you entered in the Add window.

If you do not know the data source name in advance, or if you intend to use more than one data source, you may set the data source name dynamically at run time. In your COBOL program, you would add code similar to this prior to the statement that opens the file:

```
SET ENVIRONMENT "A_ODBC_DATASOURCE" TO "MS Access Databases"
```

You are now ready to use the **sql.acu** program, as outlined in **Chapter 2, "Chapter 2: Getting Started."**

After you learn about and use this utility, you next find out about preparing and compiling your COBOL program, followed by learning to use the demonstration program. See **section 6.1, "Preparing and Compiling Your COBOL Program,"** and **section 2.5, "The Demonstration Program,"** for additional information.

D.3 Filename Translation

As you prepare to work with Acu4GL for ODBC, you may find it helpful to understand the rules around filename interpretation and to understand how the names of tables and XFD files are formed and work together.

When the ACUCOBOL-GT compiler generates XFD files, it uses lowercase letters to name the XFD file. In addition, the compiler changes hyphens to underscores when naming the XFD file.

Through configuration variables, the runtime translates the name in the COBOL program into the filename that is passed to the **open()** function in the runtime. The **open()** function determines which file system to pass the request to, but does not change the name of the file.

Acu4GL for ODBC translates the name to lowercase letters and changes hyphens to underscores unless the configuration variable, **4GL_COLUMN_CASE**, is set. (See **section D.6** for information on this variable.) This "new" name is the one that Acu4GL for ODBC will use in the future for references to database tables.

D.4 Decimal Points

Acu4GL for ODBC reads the decimal point character from the environment variable DECIMAL_POINT. If DECIMAL_POINT is set, Acu4GL uses that character. If the variable is not set, Acu4GL uses the decimal character that is encoded in the XFD file.

The two most common decimal indicators are the period "."and the comma "," characters. The comma is used often in European code and is often indicated in COBOL programs by the "DECIMAL POINT IS COMMA" clause.

D.5 Configuration File Variables

This section lists the runtime configuration file variables that are specific to Acu4GL for ODBC. Configuration file variables that are generally applicable to any RDBMS with which Acu4GL communicates are discussed in **section 8.2, "Runtime Configuration Variables."**

A_ODBC_ALTERNATE_COMMIT_LOGIC
A_ODBC_USE_SQLTABLES

4GL_MAX_DATE

It is possible that invalid dates can be incorrectly written to the database. Invalid dates are written as NULL, but if the date is in the key, problems can occur. To avoid invalid date problems in this instance, use the 4GL_MAX_DATE configuration variable to set the high-value date.

The format is:

```
4GL_MAX_DATE = YYYYMMDD
```

The maximum date for 4GL_MAX_DATE is "20991231" (December 31, 2099).

See also

4GL_MIN_DATE configuration file variable

4GL_MIN_DATE

As noted with the configuration variable 4GL_MAX_DATE, invalid dates can be incorrectly written to the database. Invalid dates are written as NULL, but if the date is in the key, problems can occur. To avoid invalid date problems in this instance, use the 4GL_MIN_DATE configuration variable to set the low-value date.

The format is:

```
4GL_MIN_DATE = YYYYMMDD
```

The default date for 4GL_MIN_DATE is "19000101" (January 1, 1900).

See also

4GL_MAX_DATE configuration file variable

A_ODBC_ALTERNATE_COMMIT_LOGIC

The primary purpose of the configuration variable 4GL_COMMIT_COUNT is to provide for applications that must communicate with a transaction-oriented database but have not explicitly coded transactions into their COBOL application.

The configuration variable A_ODBC_ALTERNATE_COMMIT_LOGIC determines how the interface will respond to the setting of 4GL_COMMIT_COUNT. When A_ODBC_ALTERNATE_COMMIT_LOGIC is set to "0" or "FALSE" (the default), the value of 4GL_COMMIT_COUNT is checked only at startup. The interface will issue a commit when the criteria set by 4GL_COMMIT_COUNT is met, regardless of the current transaction state.

When A_ODBC_ALTERNATE_COMMIT_LOGIC is set to "1", the value of 4GL_COMMIT_COUNT is checked after each WRITE, REWRITE, DELETE, or UNLOCK operation. A commit is issued, however, only if the runtime is not currently in a transaction.

Example

```
4GL_COMMIT_COUNT=1
A_ODBC_ALTERNATE_COMMIT_LOGIC=0
```

Each WRITE operation is committed to the database immediately. This prevents an application from being able to rollback a WRITE.

See Also

4GL_COMMIT_COUNT runtime configuration variable

A_ODBC_ALTERNATE_COMMIT_LOGIC configuration variable

A_ODBC_CATALOG

This variable indicates the catalog name to be used when Acu4GL searches for objects in the database. Note that not all data sources will support a catalog. Using A_ODBC_CATALOG with other variables, such as USER_PATH and A_ODBC_TABLE_TYPES, can speed up the finding of tables in large databases. It can also prevent an error 9D,14: “More than one table with the same name,” thereby enabling access to tables with identical names, but with different catalogs.

For example, If USER_PATH and A_ODBC_CATALOG are used, the form of the SQL statement will be modified from:

```
select COL1, ... from TABLENAME ...
```

to:

```
SELECT COL1, ... FROM [catalog.][username.]TABLENAME ...
```

where *catalog* and *username* will be filled in if provided.

See Also

USER_PATH configuration variable

A_ODBC_TABLE_TYPES configuration variable

A_ODBC_COMMIT_ON_BEGIN

ODBC has no START TRANSACTION method. Everything since the last COMMIT or ROLLBACK is considered part of the current transaction.

To ensure that the previous transaction has been ended before a new one begins, set A_ODBC_COMMIT_ON_BEGIN to a nonzero value. This causes each COBOL START TRANSACTION to first issue a COMMIT to all applicable ODBC drivers in use. This ensures that the previous transaction has been ended before the new one starts.

If this variable is not set, or is set to "0", a COBOL ROLLBACK may affect file I/O that occurred before the most recent COBOL START TRANSACTION. While the default value is "0" (off, false, no), this configuration variable can also take values of "On" (true, yes).

A_ODBC_DATASOURCE

Set A_ODBC_DATASOURCE to the exact name of the data source that you established in the Windows ODBC driver manager. You can set this variable in your COBOL configuration file if you will be using only one data source.

For example, if the data source you want to select is listed in the Windows ODBC driver manager window as:

```
MS Access Databases (Access Data (*.mdb))
```

you should add the following line to your configuration file:

```
A_ODBC_DATASOURCE MS Access Databases
```

If you do not know the data source name in advance, or if you intend to use more than one data source, you may set the data source name dynamically at run time. In your COBOL program, you would add code similar to this prior to the statement that opens the file:

```
SET ENVIRONMENT "A_ODBC_DATASOURCE" TO "data source name"
```

There is no default value for this variable. If you do not enter a name, ODBC will use the default ODBC driver.

A_ODBC_ERROR_MAP_FILE

Because there are so many drivers available for ODBC, you may find that data source error codes don't necessarily map well to COBOL error codes. To solve this problem, Acu4GL for ODBC allows you to create an error map file to map native database errors to COBOL errors. Create this file using the guidelines described on the following page, and then use the configuration file variable, A_ODBC_ERROR_MAP_FILE, to indicate the name and location of the file you created.

Example

If the file used for mapping is called MAP, and this file is located in the directory C:\ODBC, you would specify:

```
A_ODBC_ERROR_MAP_FILE  c:\ODBC\MAP
```

in the runtime configuration file. There is no default value for this variable.

Guidelines for creating a map file

Although you can check your data source documentation for error code information, the easiest way to determine what error codes need to be mapped to more appropriate COBOL codes is through trial and error. As users use Acu4GL for ODBC, they may report receiving error messages that don't make sense based on their situation. Research these errors and try to determine a more appropriate COBOL error response.

When you create your error map file, use the following guidelines:

- Begin comment lines with "#". Blank lines are also considered comments.
- Break the rest of the file into sections, with each section header comprising all the information enclosed in brackets from the data source error function.

  For example, if the data source returns this error:

```
OdbcOneInfo: State: S1000, Native Error: -346
[Visigenic][ODBC Informix 5 Driver][Informix]Could not update
a row in the table.
```

make your section header:

```
[Visigenic][ODBC Informix 5 Driver][Informix]
```

- Include two fields in each line in the section: the internal error number, and an ACUCOBOL-GT mapping string.

  Using the same example, if you wanted to map the Visigenic Informix driver error, "-346 Could not update a row in the table" to the COBOL error, "Not found", you would include this line in the section:

  ```
  -346 E_NOT_FOUND
  ```

  Other Visigenic Informix error maps would follow in the same section. If you use other drivers, you could use multiple sections.

  The valid values for the second field are as follows:

  ```
  E_SYS_ERR
  E_PARAM_ERR
  E_TOO_MANY_FILES
  E_MODE_CLASH
  E_REC_LOCKED
  E_BROKEN
  E_DUPLICATE
  E_NOT_FOUND
  E_UNDEF_RECORD
  E_DISK_FULL
  E_FILE_LOCKED
  E_REC_CHANGED
  E_MISMATCH
  E_NO_MEMORY
  E_MISSING_FILE
  E_PERMISSION
  E_NO_SUPPORT
  E_NO_LOCKS
  ```

Through experience, we have discovered specific ways to better map errors for some drivers. For a list of these driver error mapping suggestions, look at the file "odbcerrs" on your installation disks.

A_ODBC_ISOLATION_LEVEL

Use the A_ODBC_ISOLATION_LEVEL configuration variable to set the isolation level. The default ordering of isolation levels is:

SQL_TXN_READ_COMMITTED
SQL_TXN_READ_UNCOMMITTED
SQL_TXN_REPEATABLE_READ
SQL_TXN_SERIALIZABLE

You can change the isolation level by setting the configuration variable A_ODBC_ISOLATION_LEVEL to an integer in the configuration file. The settings are:

Isolation level	**Setting in configuration file**
SQL_TXN_READ_UNCOMMITTED	1
SQL_TXN_READ_COMMITTED	2
SQL_TXN_REPEATABLE_READ	3
SQL_TXN_SERIALIZABLE	4

For example, to set the isolation level to SQL_TXN_READ_COMMITTED, add the following entry to the configuration file:

```
A_ODBC_ISOLATION_LEVEL 2
```

If the user-set isolation level is not supported by the driver, the default method of selecting a level is used.

A_ODBC_LOCK_METHOD

Use this variable to specify the locking method that the Acu4GL product should use when accessing your data source. Possible values are:

none
SETPOS
SETSTMTOPTION
UPDATECOLUMN

Example:

```
A_ODBC_LOCK_METHOD UPDATECOLUMN
```

The default value is none, meaning that the Acu4GL product will do nothing special to lock your data. This does not necessarily mean that your data won't be locked. Locking depends mainly on your data source and driver. No value simply tells Acu4GL for ODBC not to perform any locking functions.

Note: Please note that we cannot guarantee that setting this variable will have any effect for a particular ODBC driver. Locking depends mainly on the data source and the driver. You can use this variable to attempt to induce row locking. If none of the settings cause any rows to be locked, or if the machine hangs, please report this behavior to Technical Services.

SETPOS

Specify SETPOS as your lock method to tell Acu4GL for ODBC to perform the following locking sequence:

When setting up a statement handle for accessing the ODBC data source, the Acu4GL product calls a function called SQLSetScrollOptions, with values (SQL-CONCUR-LOCK, SQL-SCROLL-KEYSET-DRIVER, 1).

When fetching rows from the data source, the Acu4GL product calls SQLExtendedFetch instead of SQLFetch.

Finally, it calls SQLSetPos, with the values (1, SQL-POSITION, SQL-LOCK-EXCLUSIVE).

If any of these functions do not exist in the ODBC driver, then A_ODBC_LOCK_METHOD reverts to none.

Note: Even if these functions do exist, there is no guarantee that any rows will be locked using this sequence of calls. Contact your driver vendor to determine whether or not this sequence will be effective for your data source.

SETSTMTOPTION

Specify SETSTMTOPTION as your lock method for ODBC version 2.0 (and later) drivers. This option is similar to SETPOS, except ODBC version 2 defines a new function called SQLSetStmtOption, which performs the task of SQLSetScrollOptions, but has more functionality.

In particular, when using this method of locking, the Acu4GL product will call SQLSetStmtOption a number of times, with values (SQL-CONCURRENCY, SQL-CONCUR-LOCK), (SQL-CURSOR-TYPES, SQL-CURSOR-KEYSET-DRIVEN), and (SQL_KEYSET-SIZE, 1). Again, if the function does not exist, A_ODBC_LOCK_METHOD reverts to none.

UPDATECOLUMN

Specifying UPDATECOLUMN as your lock method performs an entirely different type of locking. Instead of trying to lock a row while reading it, this method creates a new statement handle for the data source. Then, after fetching the data from the data source, it resubmits an SQL query to select the same row (based on the primary key) and adds an UPDATE clause. Last, it fetches the data from the data source. Because this method has the most overhead, we don't recommend it for slow drivers, such as Microsoft Access, although it is much more likely to succeed in locking records.

A_ODBC_LOGIN

A_ODBC_LOGIN indicates the user name under which you want to connect to the database system. Not all data sources require a user login name. Those that do may have case requirements. Check your data source documentation to determine if login is case-sensitive.

Example

To connect to the database with the user name MYNAME, you could specify:

```
A_ODBC_LOGIN  MYNAME
```

or

```
A_ODBC_LOGIN  myname
```

in the runtime configuration file. There is no default value for this variable. If no login is specified, none will be used.

See also

A_ODBC_PASSWD runtime configuration file variable

A_ODBC_NO_NULL_COLUMNS

The interface will never insert NULL into the database if this is set to TRUE or ON or YES. While the default value is "Off" (false, no), this configuration variable can also take values of "On" (true, yes).

A_ODBC_PASSWD

The variable A_ODBC_PASSWD should be set to the password assigned to the database account associated with the user name specified by **A_ODBC_LOGIN**. Not all data sources require a password. Those that do may have case requirements. Check your data source documentation to determine if password is case-sensitive.

Examples

If the account with the user name in A_ODBC_LOGIN has the associated password "CW021535", you would specify:

```
A_ODBC_PASSWD  CW021535
```

in the runtime configuration file.

For better security, you can accept a password from the user during program execution; set the A_ODBC_PASSWD variable based on the response:

```
ACCEPT RESPONSE NO-ECHO.
SET ENVIRONMENT "A_ODBC_PASSWD" TO RESPONSE.
```

Note: If the user has been set up without a password, this variable need not be set. There is no default value for this variable. If no password is specified, none will be used.

A_ODBC_PRINT_LOG

This variable is used for debugging only, and is best used under the direction of an ***extend*** Technical Services representative.

A_ODBC_QUOTE_IDENTIFIERS

A_ODBC_QUOTE_IDENTIFIERS tells the Acu4GL product whether to test for the IDENTIFIER-QUOTE-CHAR, and to use it if it is found. If this variable is nonzero, and if the driver returns information about the IDENTIFIER-QUOTE-CHAR, all identifiers sent in SQL to the driver will be quoted. This is useful for using Microsoft Excel, for example, if the spreadsheet has column names with quotes or other strange characters. Notice that some drivers mistakenly report that they can handle quoted identifiers, which is why this variable is needed.

If you set this variable to a nonzero value, and start getting errors about "table not found" or "column not found," this variable should not be used when communicating with that driver.

While the default value is "0" (off, false, no), this configuration variable can also take values of "On" (true, yes).

Note: We have found that with Excel, this variable is useful. With Oracle, this variable is not recommended.

Refer to **section D.6** for information on using mixed-cased identifiers.

A_ODBC_STRICT_EQUAL

When Acu4GL for ODBC executes a START … KEY EQUAL *keyval*, the interface generates a SELECT statement to select all rows with a key equal to the key columns in *keyval*. When fetching the data, Acu4GL eventually detects that there is no more data. If A_ODBC_STRICT_EQUAL is set to "True" at the time the START is executed, the interface returns the error END OF FILE (error 10) at this point. If A_ODBC_STRICT_EQUAL is set to "False" (the default), Acu4GL for ODBC then generates a new SELECT statement to continue reading records until the end of the file rather than the subset of records represented by *keyval*, and these records may be discarded by the program. (Note that this is program-dependent behavior.) Setting A_ODBC_STRICT_EQUAL to "True" causes the interface to stop reading records earlier, which can make SELECT statements generated by Acu4GL for ODBC more efficient.

Note the following conditions:

- A_ODBC_STRICT_EQUAL must be set to "True" when the START is executed.
- The START must be a KEY EQUAL start.
- There must be no COMMITs between the START and the end of the file. Because most ODBC drivers close all cursors on COMMIT and ROLLBACK (according to the ANSI 92 SQL standard), the interface must regenerate a SELECT the first time it attempts to execute a READ NEXT after a COMMIT or ROLLBACK. For most drivers, this means that the **4GL_COMMIT_COUNT** configuration variable should be set to "-1", so that no COMMITs are issued by Acu4GL for ODBC. See **4GL_COMMIT_COUNT** in Chapter 8 for additional information.
- A_ODBC_STRICT_EQUAL is most useful with alternate keys that allow duplicates.

Note: This same behavior can be accomplished with the 4GL_WHERE_CONSTRAINT, but using A_ODBC_STRICT_EQUAL requires less programming.

While the default value is “False” (off, no, 0), this configuration variable can also take the value “True” (on, yes, 1).

A_ODBC_TABLE_TYPES

This variable is used to specify a table type (TABLE, VIEW, and so forth) that should be looked for when selecting a database table. Using A_ODBC_TABLE_TYPES with other variables, such as USER_PATH and A_ODBC_CATALOG, can speed up the finding of tables in large databases.

This information will be passed to the API function call SQLTables() as the TableType parameter. Please refer to your driver’s documentation for supported types.

Example

```
A_ODBC_TABLE_TYPES  TABLE,VIEW
```

See also

USER_PATH configuration variable

A_ODBC_CATALOG configuration variable

A_ODBC_UNSIGNED_TINYINT

Some ODBC drivers may assign the TINYINT data type, a signed 1-byte variable that can store values from -128 to 127, to an internal unsigned type that can store values from 0 to 255. To determine if your ODBC driver does this, set a PIC s99 variable to a negative value and then write a record. If a “value out of range” message is returned, you must set A_ODBC_UNSIGNED_TINYINT to a nonzero value. As a result, PIC s99 variables are stored in a larger type (usually INT) that allows negative values. While the default value is “0” (off, false, no), this configuration variable can also take values of “On” (true, yes).

A_ODBC_USE_CATALOG

Not all data sources will support a catalog, or the data sources may return a catalog name that is not useful. For example, Microsoft Access returns the full path to the database, which is not needed since this information is detailed in the datasource. The default behavior of the runtime is to not use the catalog in the actual SQL queries. This configuration variable enables you to modify this behavior.

The default value is "FALSE", which will cause the catalog portion of the table name to be treated as blank in SQL queries. Setting this variable to "TRUE" will cause the value of the catalog returned by SQLTables ODBC function to be used in subsequent SQL queries.

See also

USER_PATH configuration variable

A_ODBC_CATALOG configuration variable

A_ODBC_USE_CHAR_FOR_BINARY

Some data sources have restrictions on the number of binary large objects (BLOBs) that can be placed into a single table. If your data source has such restrictions, specify A_ODBC_USE_CHAR_FOR_BINARY in the configuration file. The possible values for this variable are:

0 = use BINARY type
1 = use CHAR type

If A_ODBC_USE_CHAR_FOR_BINARY is set to a nonzero value, you can store data that uses the BINARY directive as hexadecimal encoded CHAR types. This allows you to work around your data source restriction. While the default value is "0" (off, false, no), this configuration variable can also take values of "On" (true, yes).

Example

```
A_ODBC_USE_CHAR_FOR_BINARY 1
```

A_ODBC_USE_SPACE_IN_DATES

Some ODBC drivers require spaces when sending dates or timestamps to the data source. For data sources that require spaces between the "d" and the quote (‘), set the configuration variable A_ODBC_USE_SPACE_IN_DATES.

For example:

```
Set MYDATE={d '1999-3-20'}
```

(note that there is a space between *d* and *'*)

Setting the A_ODBC_USE_SPACE_IN_DATES configuration variable to "1" causes the Acu4GL for ODBC product to use syntax as above, instead of:

```
Set MYDATE={d'1999-3-20'}
```

(note that there is *not* a space between *d* and *'*)

Keep in mind that it is possible that all data sources may allow a space between the *d* and the *'*, and that it is always safe to set this variable.

While the default value is "0" (off, false, no), this configuration variable can also take values of "1" (on, true, yes).

Example

```
A_ODBC_USE_SPACE_IN_DATES 1
```

A_ODBC_USE_SQLCOLUMNS

For some large databases, the API function SQLColumns(), which is called to get a description of the columns in a table, sometimes takes a long time to execute. If you are experiencing such problems, you can use the API function call SQLDescribeCol() instead, which can improve performance for large databases.

To turn on this new functionality, set the A_ODBC_USE_SQLCOLUMNS configuration variable to FALSE.

The default value of this variable is TRUE, which means the interface will use the SQLColumns() API function.

A_ODBC_USE_SQLTABLES

For some large databases, the API function SQLTables(), which is called to get a list of the tables and information about them, sometimes takes a long time to execute. If you are experiencing such problems, you can instruct the interface to build a test SQL query and use the API function call SQLNumResultCols() to determine if the table exists. This may improve performance for large databases.

To turn on this new functionality, set the A_ODBC_USE_SQLTABLES configuration variable to FALSE.

The default value of this variable is TRUE, which means the interface will use the SQLTables() API function.

USER_PATH

USER_PATH indicates the user name or names (schemas) to be used when Acu4GL searches for files. The order of the names is significant. The syntax for this variable is:

```
USER_PATH  user1 [user2]...
```

where the *user* argument may be either the name of a user (schema) on the system, or a period (“.”), which indicates the files owned by yourself.

The type of OPEN being issued for the file determines the effects of this setting.

Examples

If an OPEN INPUT or OPEN I/O is issued, and a USER_PATH variable is defined in the runtime configuration file, Acu4GL searches for a user of the named file in the list of users in USER_PATH. The first valid file is opened.

If USER_PATH is defined and the current user is the owner of the file, the current user must be included as one of the users, as indicated by a "." or the setting of your login schema (A_ODBC_LOGIN) in the USER_PATH. If this is not the case, even though the current user has created the table, it will not be found and a file error "35" will result. This circumstance can occur if the file was created with the OPEN OUTPUT phrase, and "." is not an element in USER_PATH. When the table is created, the current user will be the owner of that table. When the runtime attempts to open the table, the runtime will not look for tables owned by the current user unless USER_PATH is not set, or unless "." or the user's current schema is part of the USER_PATH setting.

If an OPEN INPUT or OPEN I/O is issued, and *no* USER_PATH variable is in the runtime configuration file, Acu4GL searches for a user of the named file in the user named for login (A_ODBC_LOGIN). Acu4GL opens the first file that has a valid combination of user and file name.

If an OPEN OUTPUT is issued (whether USER_PATH is present or not), a new table is created with the owner being the name specified in A_ODBC_LOGIN.

Using USER_PATH with other variables such as A_ODBC_CATALOG can speed up the finding of tables in large databases. It can also prevent an error 9D,14: "More than one table with the same name," thereby enabling access to tables with identical names, but with different schemas.

For example, If the USER_PATH and A_ODBC_CATALOG are used, the form of the SQL statement will be modified from:

```
select COL1, ... from TABLENAME ...
```

to:

```
SELECT COL1, ... FROM [catalog.][username.]TABLENAME ...
```

where *catalog* and *username* will be filled in if provided.

See Also

A_ODBC_LOGIN configuration variable

A_ODBC_CATALOG configuration variable

D.6 Mixed-case SQL Identifiers

Identifiers in SQL-92 are not case-sensitive. However, drivers that do not conform strictly to SQL-92 may require mixed-case identifiers.

If your driver enforces a mixed case, do the following:

1. Compile your source COBOL program with the "-Fc" option to generate mixed-case identifiers in your XFD files.
2. Modify the filename in your COBOL program to match the target identifier's case.
3. In your configuration file, set the configuration variable **4GL_COLUMN_CASE** to "UNCHANGED". Possible values are UPPER, LOWER, UNCHANGED.

D.7 Record and Table Locking

Acu4GL for ODBC supports record locking for multi-user systems, as long as your ODBC driver supports record locking. To complement your driver's locking method, Acu4GL for ODBC can initiate three locking sequences:

SETPOS
SETSTMTOPTION
UPDATECOLUMN

As described in **section D.4**, you specify which type of locking you want to use with the configuration file variable, **A_ODBC_LOCK_METHOD**. However, if your driver doesn't support locking, Acu4GL for ODBC by itself cannot provide locking.

Record locking with ODBC drivers is a semi-random activity. Different ODBC drivers have different capabilities and even implement the same capabilities in different ways. For this reason, you will have to do some experimentation to determine which locking method to specify for A_ODBC_LOCK_METHOD. Only through trial and error will you be able to determine which method will lock records in the most efficient way for your particular ODBC driver.

D.8 Limits and Ranges

The following limits exist for the ODBC protocol:

Maximum number of columns per key: depends on driver/database
Maximum number of columns: depends on driver/database

Acu4GL allows an unlimited number of columns, but the database may have a limit, and the driver may have a smaller limit than the database. If you have too many columns, you will receive a database error on the table create (for example, open output).

To achieve the same sort or retrieval sequence under ODBC as under the Vision file system, place a **BINARY** directive immediately before each key field that contains signed numeric data. High values and low values can cause problems in key fields. If you want data that uses the BINARY directive to be stored as hexadecimal encoded CHAR types, you can specify **A_ODBC_USE_CHAR_FOR_BINARY** in the configuration file. Please note that this could potentially increase the size of your key beyond the driver maximum.

Other limits are described in Appendix B in Book 4, *Appendices*, of the ACUCOBOL-GT documentation.

D.9 Driver Requirements

Your ODBC driver must include the following functions:

- all Core ODBC driver functions
- the Level 1 function SQLColumns
- the Level 1 function SQLTables

Depending on the method of record locking you choose, your driver may also need to support some of the following function calls. See the description for **A_ODBC_LOCK_METHOD** in section D.4, "Acu4GL for ODBC Configuration File Variables."

- SQLSetStmtOption

- SQLSetScrollOptions
- SQLExtendedFetch
- SQLSetPos

To test the capabilities of your ODBC driver, we have included a driver test program on your Acu4GL for ODBC installation disks. You can also consult your driver documentation to ensure that it meets these requirements.

ODBC driver test

Acu4GL for ODBC includes a driver test program designed to test your ODBC driver for compatibility with Acu4GL for ODBC. The driver test also lists all the tables/columns for the driver selected. This can help design COBOL FD statements for existing tables. To start the program, select the DRVTST32.EXE icon in the program group where Acu4GL for ODBC has been installed.

User interaction for this program is minimal. The only time you need to interact with the program is when it first starts and you are asked for the name of the ODBC driver to connect to. You will see a dialog very similar to the ODBC setup dialog, prompting you to select a data source.

Once you have selected a data source, you may be asked to log on to the data source (this depends on the data source you select). Enter your login and password, if necessary.

After that, the program queries the ODBC driver, and creates a file called "ACUCOBOL.RPT" in the current directory. This file is filled with information about the driver you chose, but is in a format that is most useful to ***extend*** Technical Services. When you are finished running the driver test program, send the "ACUCOBOL.RPT" file to ***extend*** Technical Services, and they can help you determine the compatibility level of your driver.

If you execute this program multiple times, information will be appended to the original "ACUCOBOL.RPT" file.

D.10 Data Type Mapping

Overview of ODBC data types

ODBC defines certain generic data types, which every data source driver maps to its own internal types. Each driver needs to be queried about which types it supports, because different drivers support different types.

The types that Acu4GL for ODBC uses, if they exist in the driver, are:

```
SQL_CHAR
SQL_VARCHAR
SQL_DECIMAL
SQL_NUMERIC
SQL_SMALLINT
SQL_INTEGER
SQL_REAL
SQL_FLOAT
SQL_DOUBLE
SQL_TINYINT
SQL_BIGINT
SQL_BINARY
SQL_VARBINARY
SQL_LONGVARBINARY
SQL_DATE
SQL_TIMESTAMP
```

For example, Informix maps these types like this:

```
SQL_CHAR                  CHAR
SQL_VARCHAR               VARCHAR
SQL_DECIMAL               DECIMAL or MONEY
SQL_NUMERIC               no such type
SQL_SMALLINT              SMALLINT
SQL_INTEGER               INTEGER or SERIAL
SQL_REAL                  REAL
SQL_FLOAT                 no such type
SQL_DOUBLE                FLOAT
SQL_TINYINT               no such type
SQL_BIGINT                no such type
SQL_BINARY                no such type
SQL_VARBINARY             no such type
SQL_LONGVARBINARY         BYTE
SQL_DATE                  DATE
SQL_TIMESTAMP             DATETIME YEAR TO FRACTION(5)
```

Informix also defines a type that ODBC calls SQL_LONGVARCHAR, and Informix calls TEXT, which the Acu4GL for ODBC product doesn't use.

Notice that Informix has two types that match the ODBC SQL_DECIMAL type, and two that match the SQL_INTEGER type. The Acu4GL product will usually use the first type that it finds that matches an ODBC type, unless there are restrictions on that type. For example, the SERIAL type is limited in that a table can have only one such column, while a typical table may have more than one column of integer data.

So the Acu4GL for ODBC product will use the Informix INTEGER type, instead of the SERIAL type for integer data.

For another example, MS Access maps these types like this:

SQL_CHAR	CHAR
SQL_VARCHAR	TEXT
SQL_DECIMAL	no such type
SQL_NUMERIC	CURRENCY
SQL_SMALLINT	SHORT
SQL_INTEGER	LONG or COUNTER
SQL_REAL	SINGLE
SQL_FLOAT	no such type
SQL_DOUBLE	DOUBLE
SQL_TINYINT	BYTE
SQL_BIGINT	no such type
SQL_BINARY	BINARY
SQL_VARBINARY	VARBINARY
SQL_LONGVARBINARY	LONGBINARY
SQL_DATE	no such type
SQL_TIMESTAMP	DATETIME

Access also defines types, SQL_BIT = BIT and SQL_LONGVARCHAR = LONGTEXT which the Acu4GL for ODBC product doesn't use.

Mapping COBOL data types to ODBC data types

When the Acu4GL for ODBC product creates a table, it uses what it determines to be the best match of a data type for any particular column.

This means that the database column will be able to hold any data that the COBOL data type can hold, and is as close as possible to the type of data that the COBOL program is using. This determination is based in part on what

data types the data source has available. Obviously, if a data source doesn't support some data type, the Acu4GL for ODBC product can't use it with that data source. The actual algorithm used is rather complicated, but the general rules are as follows:

User preferences take precedence. This means that the XFD directives specified are checked first. Therefore, when data should be of type DATE or BINARY, a DATE or BINARY type is located and used, if the data source supports it.

If the COBOL data type is usage float or usage double, a data source type of FLOAT, REAL, or DOUBLE is used, depending on what is available. If none of these is available, the Acu4GL product abides by the next rule.

If the COBOL data type is numeric, a numeric type is used in the data source. The numeric type chosen depends on how large the COBOL data type is, and how many digits to the right of the decimal point (if any) there are.

- For example, if the COBOL data type is PIC 99, the data source types checked for are TINYINT, SMALLINT, INTEGER, BIGINT, DECIMAL, NUMERIC, CHAR. The first of these that exists is the type that will be used for that column.
- For another example, if the COBOL data type is PIC 9(6)v99, the data source types checked for are DECIMAL, NUMERIC, DOUBLE, FLOAT, REAL, CHAR. Again, the first of these types that exists is the type that will be used for that column.
- Anything else will use CHAR.

Occasionally, you may encounter a data source type that supports only type CHAR (which is the only data type that is guaranteed to exist, according to the ODBC specification). Under these data sources, all the COBOL data types will be mapped to CHAR types.

Mapping ODBC data types to COBOL data types

Sometimes developers are in a situation where they need to create a COBOL File Description based on an existing data source table. The most important thing to understand in this situation is that there is almost nothing that you can do wrong! When the Acu4GL product opens a data source table, the only thing it checks is that the column names match the COBOL data names.

When the Acu4GL product reads data from the data source, it essentially does a COBOL-style MOVE from the native data type to the COBOL data type, whatever it is. And since most types have a CHAR representation (in other words, you can actually display most data types, using a standard ODBC-capable tool), using PIC X(nn) for each column will work perfectly well.

A better general rule is to use a COBOL type that closely matches the data source data type, but don't worry about getting an exact fit. So you can use PIC 9(9) whenever the data source has an INTEGER type.

If you have more information about the data source type, you might be able to use a different COBOL representation. For example, if you know that a particular column in an ODBC data source has values only in the range 0–999, you could use PIC 999 for your COBOL data. The COMP-type you use is really determined by your own preferences, and should have little bearing on the COBOL data type you choose.

If you want to somehow choose your COBOL data types so that there is a best fit, you can use the following mapping:

```
SQL_CHAR               PIC X(nn)     nn =size of item
SQL_VARCHAR            PIC X(nn)     nn = maximum size of item
SQL_DECIMAL            PIC 9(n)v9(m)
SQL_NUMERIC            PIC 9(n)v9(m)
SQL_SMALLINT           PIC 9(5) COMP-5
SQL_INTEGER            PIC 9(9) COMP-5
SQL_REAL               USAGE FLOAT
SQL_FLOAT              USAGE FLOAT
SQL_DOUBLE             USAGE DOUBLE
SQL_TINYINT            PIC 9(3) COMP-5
SQL_BIGINT             PIC 9(9) COMP-5
SQL_BINARY             PIC X(nn)
SQL_VARBINARY          PIC X(nn)
SQL_LONGVARBINARY      PIC X(nn)
SQL_DATE               PIC 9(6) or PIC 9(8)
SQL_TIMESTAMP          USAGE DISPLAY
```

Note: The BINARY data types are usually of a form that COBOL can't understand anyway. You will usually just read these columns, and rewrite them unchanged. If you have more information about the data in the columns, you might be able to do something else, but this requires more knowledge about the columns.

The DECIMAL, NUMERIC, DATE, and TIMESTAMP types usually have special representations in a data source, which really doesn't match any COBOL data type exactly. When the Acu4GL product binds the data (a technical term), it asks the data source to return it in character form, so the most efficient COBOL data type would probably be USAGE DISPLAY.

D.11 Troubleshooting

This section lists the Acu4GL error messages that could occur during execution of your program. **Chapter 9** provides information on compile-time errors and also provides several methods for retrieving runtime errors.

An explanation and a recommended recovery procedure follow each message.

Note: Often, errors are generated for the simple reason that the ODBC driver version you are using is incompatible with Acu4GL for ODBC. If you encounter errors when using Acu4GL for ODBC, run the driver test program found on your installation disks to determine if you are using a compatible driver version. Refer to **section D.9, "Driver Requirements,"** for more details on the driver test program.

D.11.1 Runtime Errors

Runtime errors will have this format:

9D,xx

The 9D indicates a file system error and is reported in your FILE STATUS variable. The *xx* is a secondary error code. These are the secondary errors reported directly from Acu4GL:

9D,01Read error on dictionary file

An error occurred while reading the XFD file; this probably means the XFD is corrupt. Recompile with "-Fx" to re-create the dictionary. See **section 8.1** for additional information on compiler options.

9D,02 Corrupt dictionary file

The dictionary file for one of your COBOL files is corrupt and cannot be read. Recompile with "-Fx" to re-create the dictionary.

9D,03 Dictionary (.xfd) file not found

The dictionary file for one of your COBOL files cannot be located. Be sure you have specified the correct directory via your **XFD_PREFIX** runtime configuration file variable. You may need to recompile with "-Fx" to create the dictionary.

9D,04 Too many fields in the key

There are more than 16 fields in a key. Check your key definitions and restructure the key that is illegal, and then recompile with the "-Fx" option.

9D,06Mismatched dictionary file

The dictionary file (.xfd) for one of your files conflicts with the COBOL description of the file FD. The *xx* indicates a tertiary error code that is defined by the host file system. You can determine the exact nature of the mismatch by referring to the host system's error values.

The tertiary error code may have any of these values:

01 – mismatch found but exact cause unknown (this status is returned by the host file system)
02 – mismatch found in file's maximum record size
03 – mismatch found in file's minimum record size
04 – mismatch found in the number of keys in the file
05 – mismatch found in primary key description
06 – mismatch found in first alternate key description
07 – mismatch found in second alternate key description

The list continues in this manner for each alternate key.

9D,11ODBC connection missing needed functionality

Because ODBC has so many implementations, we require certain minimal functionality. This includes the ability to list columns, list primary keys, list existing tables, and support Level 1 SQL. Your ODBC driver lacks one or more of these capabilities.

9D,12ODBC library function returned an unexpected error

One of the ODBC library functions returned an error that was not expected.

9D,13Illegal size or type of data for variable xxx

An elementary data item in your FD was larger than 255 bytes, or there is no ODBC type that matches the current data type.

9D,14More than one table with the same name

When tables were listed, more than one table was found with the same name. Consider whether or not setting the USER_PATH configuration variable will resolve the issue. This variable will enable tables with identical names, but with different ownership (located in a different schema) to be found and accessed.

9D,15ODBC driver missing needed types

The driver doesn't support the data types needed for Acu4GL for ODBC.

D.11.2 Native SQL Errors

You may encounter the following native SQL errors when using Acu4GL for ODBC. These errors are generated by your data source, so the exact wording and error number will vary.

9D,???? Too many BLOBs (binary large objects)

Some databases have restrictions on how many BLOBs can be placed into a single table. To work around this restriction, specify the configuration variable **A_ODBC_USE_CHAR_FOR_BINARY**. This allows you to encode binary data in hexadecimal and write it out as CHAR data instead of BINARY.

9D,???? Invalid Column Name OR Reserved Word

You probably specified a column name using a word that has been reserved for your data source. Locate the column by comparing a file trace of the CREATE TABLE to your data source's list of reserved words. Apply the **NAME** directive to the field in the FD that is associated with the invalid column, then recompile the program to create a new XFD file.

9D,16 Maximum Length for Literal Characters Exceeded

The interface queries the data source about the maximum size of a character literal in an SQL statement, and if there are any fields in the COBOL data record that exceed that limit, the interface fails to open the file, with an error 9D,16.

Error 9D,16 means that there is at least one COBOL field that exceeds the reported maximum length for character literals, and that you may not be able to insert or update at least one column in the database.

D.12 Common Questions and Answers

This section contains some questions and answers specific to Acu4GL for ODBC. Refer to **Chapter 10** for additional questions and answers that pertain to the Acu4GL family of products.

Question: Can I open tables in different databases and share the data?

Answer: Yes. But you must set up the data sources one at a time, as described in this section. For example, you could first set your data source to be Oracle and open Oracle tables, and then you could change the data source to Access and open Access tables.

Remember, you can set up data sources dynamically at run time by adding a line like this before the statement that opens the file:

```
SET ENVIRONMENT "A_ODBC_DATASOURCE" TO "data source name"
```

Question: I keep receiving an error message saying that my login is invalid. But I'm sure I'm using the correct username and password.

Answer: All usernames, passwords, and database names are case-sensitive. Be sure that you are typing the names exactly as they are set up.

Question: I'm using Microsoft Access version 2.0, and I'm having trouble accessing/writing tables from my ACUCOBOL-GT program.

Answer: Check the version of the Access driver you are using. Microsoft Access version 2.0 requires version 2.0 of the driver. If you are using a previous version of the driver, the runtime will generate errors. To determine the version number of your driver, you can run the Acu4GL for ODBC driver test program DRVTST32.EXE (32-bit version).

Question: I'm noticing some performance degradation when accessing my ODBC data source. What is the cause of this?

Answer: You may notice some performance impact if you were previously accessing Vision indexed files directly. This is because ODBC adds a software layer between your applications and your data sources. In return for minor performance impact, you can reap the benefits of database independence and enhanced portability. Overall performance depends on several factors, including your network configuration and your specific data source.

Question: I wrote data with my Acu4GL for Oracle (or Informix) product. When I try to read it with Acu4GL for ODBC, the data is not the same.

Answer: By necessity, the BINARY data type is implemented differently in Acu4GL for ODBC than it is in Acu4GL for Oracle or Informix. For this reason, writing data with one Acu4GL product and then reading it with another will probably produce different results. Future releases of Acu4GL for Oracle and Informix will address this problem.

Question: When I create a table, I get an error saying I have placed too many BLOBs (binary large objects) in my table.

Answer: Some databases have restrictions about the number of BLOBS that can be placed into a single table. To work around this restriction, you can specify the configuration variable **A_ODBC_USE_CHAR_FOR_BINARY**. This allows you to encode binary data in hexadecimal and write it out as CHAR data instead of BINARY. Refer to section D.4, "Acu4GL for ODBC Configuration File Variables," for more information on this variable.

Question: Are there any ACUCOBOL-GT library routines that do not work with or would not make sense to use with Acu4GL for ODBC?

Answer: Yes. The C$COPY and C$RENAME ACUCOBOL-GT library routines do not work with Acu4GL for ODBC.

E Acu4GL for Sybase Information

Key Topics

E.1 Sybase Concepts Overview

A quick overview of some basic design concepts underlying the Sybase Database Management System will help you interface your COBOL program to it.

Servers

A Sybase server is one copy of the database engine executing on a computer. A server has a name, and when a program wants to access the database controlled by a server, the program asks for a connection to that server by name. Multiple servers can be executing on a single machine, controlling different databases. The default name that Sybase gives to a server is SYBASE. Naming of servers is discussed in section E.6, "Acu4GL for Sybase Configuration File Variables," under the configuration variable **A_SYB_DEFAULT_CONNECTION**.

Table ownership

Table names in Sybase have the form *database.owner.table_name*. Within Sybase, if you are the owner of a given table, you can refer to it as just *table_name*. If you are *not* the owner, you must refer to it with the *owner* of the table as a prefix. Different owners can thus have tables of the same name. However, this is *not* true when you use Acu4GL for Sybase.

Acu4GL for Sybase works a little differently. It automatically determines the owner name it will use to reference a table. It is therefore essential that there *not* be multiple tables with the same name in a single database, even though the tables have different owners. If there are, the Acu4GL for Sybase product will not necessarily find the correct one, and no diagnostic will be issued.

Note that table names include dots (".") as separators. Because of this, you must make sure there are no extensions on COBOL file names that will be converted to table names. If you were to have a COBOL file named "IDX1.DAT", Acu4GL for Sybase would attempt to open a table DAT with owner IDX1. You can avoid this problem either by renaming your COBOL file in your source program, or by using an ACUCOBOL-GT® runtime configuration file entry to map the file name to an allowable file name, such as:

```
IDX1.DAT  IDX1
```

If you map your file name to a new name, we recommend that you simply drop the extension to form the new name. Here's why. The compiler uses the base file name—without the extension—to create the XFD file name ("IDX1.XFD"). The runtime needs to be able to locate this file. But if you've mapped the file name to something completely different (such as "MYFILE"), the runtime will look for an XFD file named "MYFILE.XFD". So you'd have to remember to change the name of "IDX1.XFD" to "MYFILE.XFD" in the XFD directory. Save yourself this extra step by simply dropping the extension when you map the name.

Security

Security is implemented in the Sybase RDBMS. A user is required to log in to the RDBMS before any file processing can occur. Acu4GL for Sybase provides both a default and a user-configurable method for implementing this.

Generally, it is best for someone with Database Administrator (DBA) privileges to create and drop the tables, allowing others only the permissions to add, change, or delete information contained in them.

See the Sybase documentation for more details on DBA privileges.

Alternate REWRITE method

If the above stored procedure is not available, then the update query passed to the database has been optimized to update only dirty columns. This should improve performance.

E.2 Installation and Setup

The following sections describe several steps that must be performed before you begin using Acu4GL for Sybase on a new system:

- **Acu4GL for Sybase Installation**
- **Setting Up a User Account**
- **Setting Up the User Environment**
- **Designating the Host File System**

E.2.1 Sybase RDBMS Installation

The Sybase RDBMS must be installed and configured prior to the installation of Acu4GL for Sybase. Acu4GL can interface to Sybase version 11.0 and later.

Sybase's **isql** product, which is an interactive query tool, is also necessary for installing the ACUCOBOL-GT stored procedures.

The Acu4GL product does not need **isql** after installation, but it is a tool that can give you quite a bit of flexibility. It allows you to do database work outside of COBOL, including interactive queries, table creation, table modification, and creation of views, constraints, and relationships between tables.

Micro Focus does not provide these products.

E.2.2 Acu4GL for Sybase Installation

The Acu4GL product for Sybase is an add-on module. The Sybase Acu4GL product installation includes a .DLL file that is detected at run time. If you are working in a 32-bit environment, it is not necessary to relink the runtime.

However, the relinking process is necessary for UNIX users, and is explained below in this section.

The Acu4GL product can be executed on a machine that is running UNIX or Windows.

Installation instructions for each of these configurations are given in the following sections. The instructions in the sections that follow describe the steps you must follow on both the client machine and the server machine.

Be sure to use the 64-bit libraries with a 64-bit runtime and 64-bit Acu4GL for Sybase, and the 32-bit libraries with a 64-bit runtime and 32-bit Acu4GL for Sybase.

E.2.3 UNIX Client Installations

The Acu4GL product is shipped using either TAR or CPIO format, depending on the type of machine you have. The media label tells you which format has been used. You can start by copying all of the files onto your client machine. If you are using a second machine as your server (this is optional), move the server files (identified below) to that second machine.

You will be installing Acu4GL into the same directory where your ACUCOBOL-GT runtime is located. Change to that directory and type one of the following commands:

```
tar  xfv device
```

or

```
cpio  -icvBd < device
```

This will copy the files from the distribution medium to your directory. *device* is the appropriate hardware device name (for example, /dev/rmt0). Sites using Texas Instruments System 1500 should add an uppercase "T" to the **cpio** options (-icvBdT).

E.2.3.1 UNIX Client Installation Steps

Acu4GL communicates with Sybase version 11.0 or later. The client (your application) and the server (the RDBMS) may be located on the same UNIX machine, or on different UNIX machines. The server may also be located on a Windows NT machine.

To install the Acu4GL product, first perform the following steps on the UNIX client machine:

Step 1: Install Sybase client library

The Open Client DB-Library/C from Sybase (version 11.0 or later) must be installed on the client machine before you install Acu4GL for Sybase. Follow the installation instructions in your Sybase documentation.

Step 2: Create a makefile

The script **syb_inst** is an interactive shell script that determines which libraries your version of Sybase has, and then creates a makefile suitable for linking Acu4GL with an ACUCOBOL-GT runtime for UNIX. It also generates the SQL script used to install ACUCOBOL-GT stored procedures.

Execute the shell script **syb_inst** by entering the following command on your client machine:

```
sh  syb_inst
```

Note: You may exit the script at any time by pressing the system interrupt key (usually CTRL+C).

If you entered only **syb_inst** instead of the full command, this message may appear:

```
VAL=0: command not found
VAL:  Undefined variable
```

This can be fixed by entering **sh syb_inst**.

When the script begins executing, you see the following message:

```
During the execution of this script, we will create an SQL
script which you will need to execute from isql, using a
command like:

"isql -Usa -Ppassword < syb_inst.sql"

This script will add some stored procedures, and create some
tables in a database that you specify during the execution of
this script.
```

In order to implement locking from ACUCOBOL-GT, we need to create a lock table. This should be a pretty small table, but we need to decide which database to create this table in. You should now enter the name of the database you want the lock tables created in. If the database does not exist, it is created.

The following question is asked until you enter a valid database name:

```
Which database would you like to create this table in?
```

This database will eventually be located on your server machine. Enter a valid database name. (*Invalid* names are master, model, temp, or sybsystemprocs.) Names must start with a letter or underscore, must contain only letters, digits, and underscores, and may be up to 30 characters long. Any other entry will be modified.

Next you will see the following message:

```
Saving any old version of syb_inst.sql...
```

Old versions of "syb_inst.sql" are saved to "syb_inst*nnn*.sql" where *nnn* starts at 001 and goes to 999. If a file "syb_inst.sql" exists, you see:

```
Saving syb_inst.sql as syb_inst nnn.sql
```

for some value of *nnn*. Then you see:

```
Creating syb_inst.sql...
```

This uses the file "syb_inst.in" as a template, and creates "syb_inst.sql", using the database name you entered above.

You also see:

```
Creating cblconfig.syb...
```

This is a sample file of configuration variables you may want to add to your "cblconfig" file.

You then see the following message:

```
In order to use the Acu4GL for Sybase interface, you need to
relink the runtime system.  We are assuming that the Makefile
that came with your runtime system still has the "FSI_SUBS="
and "FSI_LIBS=" lines in it, and will base our changes on that
Makefile.  However, we will create a new file called
Makefile.syb, that has these changes in it.

Do you want to set up the Makefile for this system?
```

Type **y** or **Y** to continue.

You then see the following message:

```
We next need to determine where you have installed the SYBASE
client libraries.
```

The following prompt is repeated until you enter the directory that contains the Sybase files:

```
Enter the directory where the SYBASE client libraries are
installed
```

Type the full pathname of the directory containing the Sybase client libraries, and press **Enter**. Note that this is the directory that contains the Sybase "lib" directory. For example, if the full path for the file "libsybdb.a" is "/usr/sybase/lib/libsybdb.a", you would type "/usr/sybase".

The script checks for a file "lib/libsybdb.a" in the directory you entered. You then see one of the following messages:

```
We seem to have the ACUCOBOL-GT library files in this
directory.

We need to find the ACUCOBOL-GT library files...Enter the
directory where ACUCOBOL-GT is installed:
```

If you are asked to enter the directory name, type the full pathname of the directory containing ACUCOBOL-GT, then press **Enter**.

Before the Makefile is created, we need to check for some alternate optional libraries. You see the message:

```
    Checking for -lnsl or -lnsl_s
```

If one of the libraries exists, you see:

```
using -lnsl
    -or-
using -lnsl_s
```

If you instead see a message like this one:

```
    syb_inst: cc: not found
```

it's likely that your PATH environment variable does not include the location of your compiler. In this case, exit from the script, fix the PATH variable, and start the script again.

Finally, you see the message:

```
Creating Makefile.syb ...
```

```
Makefile.syb created.  You should be able to execute
     "make -f Makefile.syb"
```

and the script exits.

Step 3: Move the makefile

Copy "Makefile.syb" to the ACUCOBOL-GT library subdirectory if it is not already there.

Now you are ready to relink your ACUCOBOL-GT runtime.

Step 4: Link the runtime system

Note: In the following directions, the term "runtime system" refers to the runtime shared object on systems where the ACUCOBOL-GT runtime is a shared object and to **runcbl** on other systems, where the runtime is static. The runtime is a shared object on the following systems: AIX 5.1 and later, HP-UX 11 and later, and Solaris 7 and later. To check, look at the contents of the "lib" subdirectory of your ACUCOBOL-GT installation. If the files "libruncbl.so" or "libruncbl.sl" reside in that directory, the runtime is a shared object on your system.

Make sure you are in the directory containing the ACUCOBOL-GT runtime system. Then, at the UNIX prompt, enter the following command:

```
make clean
```

to ensure that you have a clean directory in which to build your runtime.

Now enter the following command:

```
make -f Makefile.syb
```

This compiles "sub.c" and "filetbl.c" and then links the runtime system.

Step 5: Verify the link

Enter the following command:

```
./runcbl -vv
```

to verify the link. This returns version information on all of the products linked into your runtime system. Make sure it reports the version of Acu4GL for Sybase.

Shared libraries

If you have relinked the ACUCOBOL-GT runtime and receive an error message of this type when you try to execute it:

```
"Could not load library; no such file or directory"

"Can't open shared library . . . "
```

this may mean that your operating system is using shared libraries and cannot find them. This can occur even if the shared libraries reside in the same directory where you are currently located.

Different versions of the UNIX operating system resolve this in different ways, so it is important that you consult your UNIX documentation to resolve this error.

Some versions of UNIX require that you set an environment variable that points to shared libraries on your system.

For example, on an IBM RS/6000 running AIX 4.1, the environment variable LIBPATH must point to the directory where the shared libraries are located.

On HP/UX, the environment variable that must be set to point to shared libraries is SHLIB_PATH. On UNIX SVR4, the environment variable is LD_LIBRARY_PATH.

Be sure to read the system documentation for your operating system to determine the appropriate way to locate shared libraries.

A second way to resolve this type of error is to link the libraries into the runtime with a static link. Different versions of the C development system use different flags to accomplish this link. Please consult the documentation for your C compiler to determine the correct flag for your environment.

Step 6: Copy runcbl to the correct directory

If the runtime is a system is a statically linked executable, copy the new executable to a directory listed in your execution path. This file needs to have execute permission for everyone who will be using the compiler or runtime system. The copy step is not necessary when the runtime system is a shared library.

The ACUCOBOL-GT license file for the runtime ("runcbl.alc") and the license file for the Acu4GL product to Sybase ("runcbl.ylc") must be copied into the same directory as the runtime executable.

If you rename your runtime executable, be sure to rename your license files to use the same base name, with the extensions unchanged.

For example, if you rename your runtime to be "myprog", the license file for the Acu4GL product for Sybase should be renamed "myprog.ylc", and the license file for the runtime should be renamed "myprog.alc".

The remaining files can be left in the directory to which they were unloaded from the distribution medium.

Step 7: Use rehash

If you are using the C shell, enter the command **rehash**. This tells the C shell that there is a new executable in the path.

This completes the installation process on the client machine.

E.2.3.2 UNIX or Windows NT server Intallations

Perform the following steps on the server machine (the client and server may be located on the same machine):

Step 1: Install Sybase

The Sybase server, version 11.0 or later, must be installed and configured on your server machine. Follow the Sybase installation instructions in your Sybase documentation.

Step 2: Move the ACUCOBOL-GT stored procedures

Move "syb_inst.sql" to your server machine (the one hosting the Sybase RDBMS). This file was created in step 2 of the client installation.

Step 3: Install the ACUCOBOL-GT stored procedures

You must have Database Administrator privileges to install the ACUCOBOL-GT stored procedures on the server. (You need to enter the password for the Database Administrator.) Enter the following command:

```
isql -Usa -Ppassword  <  syb_inst.sql
```

("syb_inst.sql" is created during the installation of the client)

By default, the stored procedures are installed into the master or sybsystemprocs databases. However, you may choose another database in which to store them. Follow these steps if you want to install the stored procedures in a database other than the default:

1. Before you execute the "syb_inst.sql" query, modify the query file to use the desired database instead of the master or sybsystemprocs database. After you have modified the query file, execute it as described above.

2. Create in each database that contains tables accessed by Acu4GL the stored procedures that were previously in the master or sybsystemprocs database. Note that the stored procedures are executed without being prefixed by a database name, and Sybase requires that such stored procedures must be in the master or sybsystemprocs database.

The interface also searches for and executes the **sp_AcuInit** stored procedure from this location. This stored procedure is executed when a connection is made to the database; therefore, **sp_AcuInit** should reside in the default database for each user if it is not created in the master database. See **section E.6.2, "Built-in Stored Procedures,"** for information on the **sp_AcuInit** stored procedure.

This completes the setup on the UNIX or Windows NT server machine.

Note: If you are upgrading from an earlier version of Acu4GL, be sure to install the new stored procedures. We always upgrade stored procedures in such a way that they are compatible with older versions of the product, so installing new stored procedures over old ones does not affect your ability to run with an older version of the interface software. Your new version of Acu4GL for Sybase may not run properly without the corresponding stored procedures.

It can be difficult to maintain multiple copies of stored procedures; therefore, we recommend that you continue to create the stored procedures in the master or sybsystemprocs database. If your installation does not permit this, you do have the flexibility to create the stored procedures elsewhere. However, to facilitate maintenance of the stored procedures, we recommend that you create as few databases as possible.

The following sections describe the next steps you must take before using Acu4GL for Sybase for the first time:

- **Setting Up a User Account**
- **Setting Up the User Environment**
- **Designating the Host File System**
- **Using the "sql.acu" Program**
- **Preparing and Compiling Your COBOL Program**
- **The Demonstration Program**

E.2.4 Windows Client and UNIX or Windows NT Server Installations

First you must install the files from the installation media onto the client machine. Then follow the instructions below for the appropriate server machine. Finally, some setup steps on the client machine complete the installation.

Instructions for installing your Acu4GL product from the ACUCOBOL-GT CD-ROM are contained on the *Quick Start* CD booklet that accompanied the product. Please refer to it for installing your products.

Once the installation is complete, please refer to this appendix for setting up your Acu4GL product.

E.2.4.1 UNIX Server Machine Installations

To install the Acu4GL for Sybase on a UNIX server machine:

Step 1: Install Sybase

The Sybase RDBMS, version 11.0 or later, must be installed and configured before you install Acu4GL for Sybase. Follow the Sybase installation instructions in your Sybase documentation.

Sybase's **isql** product, which is an interactive query tool, is also necessary for installing the ACUCOBOL-GT stored procedures.

The Acu4GL product does not need **isql** after installation, but it's a tool that can give you quite a bit of flexibility. It allows you to do database work outside of COBOL, including interactive queries, table creation, table modification, and creation of views, constraints, and relationships between tables.

Micro Focus does not provide these products.

Step 2: Copy files

Copy "syb_inst", "syb_cfg.in", and "syb_inst.in" to your UNIX server machine.

Step 3: Create stored procedures

The file "syb_inst" is an interactive shell script that generates the SQL script used to install ACUCOBOL-GT stored procedures.

Execute the shell script "syb_inst" by entering the following command on your server machine:

```
sh  syb_inst
```

Note: You may exit the script at any time by pressing the system interrupt key (usually CTRL+C).

If you entered only **syb_inst** instead of the full command, this message may appear:

```
VAL=0: command not found
VAL:  Undefined variable
```

This can be fixed by entering **sh syb_inst**.

When the script begins executing, you will see the following message:

```
During the execution of this script, we will create an SQL
script which you will need to execute from isql, using a
command like:

"isql -Usa -Ppassword < syb_inst.sql"

This script will add some stored procedures, and create some
tables in a database that you specify during the execution of
this script.
```

In order to implement locking from ACUCOBOL-GT, we need to create a lock table. This should be a pretty small table, but we need to decide which database to create this table in. You should now enter the name of the database you want the lock tables created in. If the database does not exist, it is created.

The following question is asked until you enter a valid database name:

```
Which database would you like to create this table in?
```

This database will eventually be located on your server machine. Enter a valid database name. (*Invalid* names are master, model, temp, or sybsystemprocs.) Names must start with a letter or underscore, must contain only letters, digits, and underscores, and may be up to 30 characters long. Any other entry will be modified.

Next you see the following message:

```
Saving any old version of syb_inst.sql...
```

Old versions of "syb_inst.sql" are saved to "syb_inst*nnn*.sql" where *nnn* starts at 001 and goes to 999. If a file "syb_inst.sql" exists, you see:

```
Saving syb_inst.sql as syb_instnnn.sql
```

for some value of *nnn*. Then you see:

```
Creating syb_inst.sql...
```

This uses the file "syb_inst.in" as a template, and creates "syb_inst.sql", using the database name you entered above.

You also see:

```
Creating cblconfig.syb...
```

This is a sample file of configuration variables you may want to add to your cblconfig file.

You then see the following message:

```
Do you want to set up the Makefile for this system?
```

Type **n** or **N** to quit the script.

Step 4: Install the ACUCOBOL-GT stored procedures

You must have Database Administrator privileges to install the ACUCOBOL-GT shared procedures on the server. (You need to enter the password for the Database Administrator.) Enter the following command:

```
isql -Usa -Ppassword  <  syb_inst.sql
```

By default, the stored procedures are installed into the master or sybsystemprocs databases. However, you may choose another database in which to store them. Follow these steps if you want to install the stored procedures in a database other than the default:

1. Before you execute the "syb_inst.sql" query, modify the query file to use the desired database instead of the master or sybsystemprocs database. After you have modified the query file, execute it as described above.

2. Create in each database that contains tables accessed by Acu4GL the stored procedures that were previously in the master or sybsystemprocs database. Note that the stored procedures are executed without being prefixed by a database name, and Sybase requires that such stored procedures must be in the master or sybsystemprocs database.

The interface also searches for and executes the **sp_AcuInit** stored procedure from this location. This stored procedure is executed when a connection is made to the database; therefore, **sp_AcuInit** should reside in the default database for each user if it is not created in the master database. See **section E.6.2, "Built-in Stored Procedures,"** for information on the **sp_AcuInit** stored procedure.

This completes the setup on the UNIX or Windows NT server machine.

Note: If you are upgrading from an earlier version of Acu4GL, be sure to install the new stored procedures. We always upgrade stored procedures in such a way that they are compatible with older versions of the product, so installing new stored procedures over old ones does not affect your ability to run with an older version of the interface software. Your new version of Acu4GL for Sybase may not run properly without the corresponding stored procedures.

It can be difficult to maintain multiple copies of stored procedures; therefore, We recommend that you continue to create the stored procedures in the master or sybsystemprocs database. If your installation does not permit this, you do have the flexibility to create the stored procedures elsewhere. However, to facilitate maintenance of the stored procedures, We recommend that you create as few databases as possible.

E.2.4.2 Windows NT Server and Windows Client Installations

Complete the following steps to install the Acu4GL product on a Windows NT server machine.

Step 1: Install SQL Server

Sybase, version 11.0 or later, must be installed and configured on the Windows NT server machine before you install Acu4GL for Sybase on the client machine. Follow the instructions from your RDBMS vendor.

The **isql** product, which is an interactive query tool, is also necessary for installing the ACUCOBOL-GT stored procedures. The Acu4GL product does not need **isql** after installation, but it is a tool that can give you quite a bit of flexibility. It allows you to do database work outside of COBOL, including interactive queries, table creation, table modification, and creation of views, constraints, and relationships between tables.

Micro Focus does not provide these products.

Step 2: Copy the batch file

The file "syb_inst.cmd" is a batch file from Micro Focus that will create the "syb_inst.sql" file, which is the collection of stored procedures necessary for executing the Acu4GL product. Copy "syb_inst.cmd" to your server machine into a directory of your choice.

Step 3: Execute the batch file

To execute the batch file, enter

```
SYB_INST LockDatabase
```

where *LockDatabase* is the database you want to use for the internal ACUCOBOL GT lock tables. If this database does not already exist, it will be created.

Everyone who will use the Acu4GL for Sybase product must have write access to this database.

This step creates "syb_inst.sql", which is the collection of stored procedures necessary for executing the Acu4GL product.

Step 4: Install the ACUCOBOL-GT stored procedures

You must have Database Administrator privileges to do this step. You need to enter the password for the Database Administrator.

Type:

```
ISQL -USA -Ppassword  <  SYB_INST.SQL
```

Another way to accomplish the same result is to use **ISQL/W** to execute syb_inst.sql as a query. See your Sybase documentation for how to do this.

By default, the stored procedures are installed into the master or sybsystemprocs databases. However, you may choose another database in which to store them. Follow these steps if you want to install the stored procedures in a database other than the default:

1. Before you execute the "syb_inst.sql" query, modify the query file to use the desired database instead of the master or sybsystemprocs database. After you have modified the query file, execute it as described above.

2. Create in each database that contains tables accessed by Acu4GL the stored procedures that were previously in the master or sybsystemprocs database. Note that the stored procedures are executed without being prefixed by a database name, and Sybase requires that such stored procedures must be in the master or sybsystemprocs database.

The interface also searches for and executes the **sp_AcuInit** stored procedure from this location. This stored procedure is executed when a connection is made to the database; therefore, **sp_AcuInit** should reside in the default database for each user if it is not created in the master database. See **section E.6.2, "Built-in Stored Procedures,"** for information on the **sp_AcuInit** stored procedure.

This completes the setup on the UNIX or Windows NT server machine.

Note: If you are upgrading from an earlier version of Acu4GL, be sure to install the new stored procedures. We always upgrade stored procedures in such a way that they are compatible with older versions of the product, so installing new stored procedures over old ones does not affect your ability to run with an older version of the interface software. Your new version of Acu4GL for Sybase may not run properly without the corresponding stored procedures.

It can be difficult to maintain multiple copies of stored procedures; therefore, we recommend that you continue to create the stored procedures in the master or sybsystemprocs database. If your installation does not permit this, you do have the flexibility to create the stored procedures elsewhere. However, to facilitate maintenance of the stored procedures, we recommend that you create as few databases as possible.

E.2.4.3 Windows client Installations

Complete the following steps to install the Acu4GL product on a Windows client machine.

Step 1: Choose your communication method

Be sure to choose your communication method at the client machine. To do this, follow the Sybase instructions for selecting a Net-Library.

After you make your selection, you will be prompted for directory names and for the server internet address or host name. The installation utility will then make an entry in the SQL.INI file, which points to the server. The SQL.INI file is located in the INI subdirectory under the directory where your Net-Library is stored.

The name of the server that is placed into SQL.INI is the name you must use in your COBOL configuration file with the variable **A_SYB_DEFAULT_CONNECTION**. Any name listed in brackets in that file can be used as a server name.

Step 2: Verify that your client is communicating with your server

To verify that you are connected, use the Sybase client utility **SYBPING** to ping the server. Make sure you receive a response to the ping.

The following sections describe the next steps to take before using Acu4GL for Sybase for the first time:

Setting Up a User Account

Setting Up the User Environment

Designating the Host File System

Using the “sql.acu” Program

Preparing and Compiling Your COBOL Program

The Demonstration Program

E.2.5 Setting Up a User Account

Acu4GL for Sybase must be able to connect to a user account. You may either set up one general account for all users or an account for each individual user. To set up an account, you must have DBA privileges.

See **sp_addlogin** and Sybase **sp_adduser** in the *Sybase Commands Reference Manual.*

E.2.6 Setting Up the User Environment

The user's account should have been set up correctly to access the Sybase RDBMS system. This includes environment variables such as SYBASE and DSQUERY. See your Sybase documentation for more details.

In addition to setting the variables required for Sybase, you will need to do the following:

- Ensure that your execution path contains the name of the directory in which you placed your newly linked Acu4GL runtime executable.
- Set the **A_SYB_LOGIN** and **A_SYB_PASSWD** variables, either in your environment or in the ACUCOBOL-GT runtime configuration file (if you don't do this, Acu4GL will use your UNIX login name as your Sybase login name, and no password). For security reasons, it is best to set the password variable from your COBOL program by asking the user to enter a password and then executing:

  ```
  SET ENVIRONMENT "A_SYB_PASSWD" TO user-entry
  ```

You may want to make and use a personalized copy of the configuration file to avoid impacting other users. The ACUCOBOL-GT *User's Guide* describes how to use the A_CONFIG environment variable, or the "-c" runtime option, to identify a personal configuration file.

For detailed information on **A_SYB_LOGIN** and **A_SYB_PASSWD**, see section E.6, "Acu4GL for Sybase Configuration File Variables."

E.2.7 Designating the Host File System

If you are opening an *existing* file, most, but not all, file systems linked into the runtime will be searched for the named file. If, however, you are creating a *new* file, you must tell the runtime which file system to use. You accomplish this with one of two runtime configuration file variables; the first is:

```
DEFAULT_HOST filesystem
```

This designates the file system to be used for newly created files that are not individually assigned. For example,

```
DEFAULT_HOST SYBASE
```

means that all new files will be Sybase files unless otherwise specified by the second configuration variable, which is:

```
filename_HOST filesystem
```

where *filename* is the file name, without any extension, named in the ASSIGN TO clause of your SELECT statement. This configuration variable is used to assign an individual data file to a file system. Any file so assigned uses the designated file system, and not the one specified by **DEFAULT_HOST**. For example:

```
myfile_HOST VISION
```

means that *myfile* will be under the Vision file system.

You can use these runtime configuration file variables in combination to assign your new files in a default with exceptions manner; for example, this set of entries:

```
DEFAULT_HOST VISION
afile_HOST SYBASE
bfile_HOST SYBASE
```

means that all new files except *afile* and *bfile* will be assigned to Vision, and those two files will be assigned to Sybase.

You can also change the values of these variables during program execution by including in your code:

```
SET ENVIRONMENT "filename_HOST" TO filesystem
```

or

```
SET ENVIRONMENT "DEFAULT_HOST" TO filesystem
```

This enables you to change file systems during the execution of your program. This is not the typical way to specify a file system; normally it is designated in the runtime configuration file and is not changed in the COBOL program.

Note: The interface to Sybase allows you to create a Sybase table with an OPEN OUTPUT statement, just as you can create Vision indexed files. The Sybase equivalent of a Vision file is a table, not a database. You need to create a database for your Sybase tables before you run the COBOL program that creates the tables, just as you need to create a directory for your files before you run a COBOL program that creates Vision files.

You are now ready to use the **sql.acu** program. You can find information on this program in **section 2.4, "Using the "sql.acu" Program."**.

After you learn about and use this utility, you will next find out about preparing and compiling your COBOL program, followed by learning to use the demonstration program. See **section 6.1, "Preparing and Compiling Your COBOL Program,"** and **section 2.5, "The Demonstration Program,"** for additional information.

E.3 Filename Translation

As you prepare to work with Acu4GL for Sybase, you may find it helpful to understand the rules around filename interpretation and to understand how the names of tables and XFD files are formed and work together.

When the ACUCOBOL-GT compiler generates XFD files, it uses lowercase letters to name the XFD file. In addition, the compiler changes hyphens to underscores when naming the XFD file.

Through configuration variables, the runtime translates the file name in the COBOL program into the filename that is passed to the **open()** function in the runtime. The **open()** function determines which file system to pass the request to, but does not change the name of the file.

However, Acu4GL for Sybase needs the name of the file to find the appropriate XFD file. To do this, Acu4GL for Sybase changes the name of the file to lowercase letters and changes hyphens to underscores. Note, however, that this is performed only on a local copy of the file. Once the XFD file is found, the filename reverts to the name that was originally passed to the **open()** function. Characters that are illegal in identifiers, such as a hyphen ("-"), are trapped by the database, and Acu4GL for Sybase will neither find the file nor create a new one.

E.4 Configuration File Variables

This section lists the runtime configuration file variables that are specific to Acu4GL for Sybase. Configuration file variables that are generally applicable to any RDBMS with which Acu4GL communicates are discussed in **section 8.2, "Runtime Configuration Variables."**

A_SYB_ADD_IDENTITY

When it is set at the default of "True", A_SYB_ADD_IDENTITY adds an extra column to any table created by the Acu4GL for Sybase product. The extra column will have the identity property and will be included on all indexes that are not unique. Otherwise, when A_SYB_ADD_IDENTITY is set at "False", an extra column is *not* added. While the default value is "Off" (false, no), this configuration variable can also take values of "On" (true, yes).

Note: The COBOL FD should not include the identity column.

Example

```
A_SYB_ADD_IDENTITY TRUE
```

On keys that allow duplicates, this variable has been found to vastly improve performance. Note that the default value is "True" for a file that allows duplicates, and is "False" for a file with no duplicate keys.

A_SYB_ADD_TIMESTAMP

Using a timestamp column is the only way to absolutely ensure that modifications made to a row are not overwriting someone else's changes. (See the discussion of BROWSE MODE in the *Sybase Commands Reference Manual*.) When reading a table that is open for I/O, the Acu4GL for Sybase product uses BROWSE MODE if a timestamp column exists. When the Acu4GL product is creating a table, if the value of A_SYB_ADD_TIMESTAMP is "1", a timestamp column is included in the table. (Note that your COBOL FD should *not* include the timestamp column.) While the default value is "Off" (false, no), this configuration variable can also take values of "On" (true, yes).

Example

```
A_SYB_ADD_TIMESTAMP  1
```

See also

BROWSE MODE in the *Sybase Commands Reference Manual*

A_SYB_CHECK_DELETE_SP

This configuration variable controls whether the Acu4GL for Sybase interface checks for the *tablename*_delete stored procedure when opening a table. If A_SYB_CHECK_DELETE_SP is set to "False" (no, off, 0), the Acu4GL interface does not check for the stored procedure. Therefore, even if *tablename*_delete exists, it is not used if A_SYB_CHECK_DELETE_SP is set to "False". The default is "True" (yes, on, 1) to check for the stored procedure.

A_SYB_CHECK_INSERT_SP

This configuration variable controls whether the Acu4GL for Sybase interface checks for the *tablename*_insert stored procedure when opening a table. If A_SYB_CHECK_INSERT_SP is set to "False" (no, off, 0), the Acu4GL interface does not check for the stored procedure. Therefore, even

if *tablename*_insert exists, it is not used if A_SYB_CHECK_INSERT_SP is set to "False." The default is "True" (yes, on, 1) to check for the stored procedure.

A_SYB_CHECK_READ_SP

This configuration variable controls whether the Acu4GL for Sybase interface checks for the *tablename*_read stored procedure when opening a table. If A_SYB_CHECK_READ_SP is set to "False" (no, off, 0), the Acu4GL interface does not check for the stored procedure. Therefore, even if *tablename*_read exists, it is not used if A_SYB_CHECK_READ_SP is set to "False." The default is "True" (yes, on, 1) to check for the stored procedure.

A_SYB_CHECK_UPDATE_SP

This configuration variable controls whether the Acu4GL for Sybase interface checks for the *tablename*_update stored procedure when opening a table. If A_SYB_CHECK_UPDATE_SP is set to "False" (no, off, 0), the Acu4GL interface does not check for the stored procedure. Therefore, even if *tablename*_update exists, it is not used if A_SYB_CHECK_UPDATE_SP is set to "False." The default is "True" (yes, on, 1) to check for the stored procedure.

A_SYB_CURSOR_OPTION_1, A_SYB_CURSOR_OPTION_2, A_SYB_CURSOR_OPTION_3

These configuration variables allow you to fine-tune the declaration of cursors in the Acu4GL for Sybase product. In general, cursors are declared with the following syntax:

```
DECLARE cursor_name option_1 CURSOR option_2 FOR
   <select....> option_3
```

In other words, different phrases can go in each of the option_*X* places. Also, different versions of Sybase allow different options in each of those places. Because of this, the Acu4GL product allows customization of the cursor declaration via these three variables. The values of these variables are placed verbatim into the declare phrase when building a cursor. Note that any errors in the values of these variables may make your Acu4GL product inoperable. You should refer to the Sybase documentation to determine what phrases are allowed in each case.

The default values are as follows (limit is 65 characters for each option):

OPTION_1: “SCROLL”
OPTION_2: blank
OPTION_3: “For read only”

Users of Sybase version 11 should set the value of OPTION_3 to “at isolation read uncommitted” to prevent page locks.

A_SYB_DATABASE

A_SYB_DATABASE specifies the name of the specific database to be accessed. You cannot open any database files until you have set this variable.

Example

```
A_SYB_DATABASE stores
```

indicates the stores database is to be accessed.

A_SYB_DEADLOCK_LOOPS

Use A_SYB_DEADLOCK_LOOPS if you expect that more than one user will be opening a lot of tables at the same time. This configuration variable can be used to instruct Acu4GL to re-execute an INSERT statement that could not execute because a row in the AcuOpenTables table was locked, or to return an error if the user chooses not to run the query again.

The default for A_SYB_DEADLOCK_LOOPS is "0", which causes the interface to return an error 9D,1205 indicating that a table is locked.

Set A_SYB_DEADLOCK_LOOPS to a positive numeric value to cause Acu4GL for Sybase to re-execute by the number specified the query that tried to open the table, and, thus, caused the deadlock. Note that it can be as long as 10 seconds until Sybase detects the deadlock, and the application appears to "hang" while the repeated attempts to re-execute the query are in progress.

Set A_SYB_DEADLOCK_LOOPS to "-1" or "MESSAGE" to cause Acu4GL for Sybase to display a message box containing the text of the Sybase error message and the option to rerun the query.

```
Sybase has returned an error
(text of message from Sybase)
Do you want to retry the operation?
```

If the user answers "yes," the interface reruns the query. If the user answers "no," the interface returns an error 9D. Setting A_SYB_DEADLOCK_LOOPS to "-1" or "MESSAGE" is the preferred action; the time it takes to inform users of the problem allows other connections to finish opening the tables, giving the AcuOpenTables table time to remove the deadlock.

A_SYB_DEFAULT_CONNECTION

A_SYB_DEFAULT_CONNECTION specifies the name of the server to which the runtime will connect. Acu4GL for Sybase checks this variable only if the DSQUERY environment variable has not been set. (Note that Sybase recommends assigning a value to the DSQUERY environment variable.) If neither DSQUERY nor A_SYB_DEFAULT_CONNECTION is set, the default server is named SYBASE, just as for **isql**. To reference tables in another server, open the file *servername.database.owner.table*.

Example

Suppose you have two servers, one named TOM and one named HARRY. If most of the tables you want to access are on the server HARRY, you should set:

```
A_SYB_DEFAULT_CONNECTION  HARRY
```

For those occasions when you want to access the TOM server, you could open the file this way:

```
TOM.stores.johndoe.purch1
```

See also

The Sybase documentation on **isql**

A_SYB_EXTRA_PROC

A_SYB_EXTRA_PROC can be used to keep modifications to the AcuLocks table out of transactions. If these modifications are kept out of transactions, users can read a record, even if they cannot REWRITE it. Otherwise, they are locked out. A_SYB_EXTRA_PROC works by creating a separate connection for these modifications.

When this variable is set to a nonzero value, an extra connection is used for the following three procedures:

- Modifying the AcuLocks table
- Modifying the AcuOpenTables table
- Modifying the IMAGE data. Note that two connections are required when updating IMAGE or TEXT data. This is a major reason for setting A_SYB_EXTRA_PROC.

The default value is “0” (off, false, no). A nonzero value can also be represented by “On” (true, yes).

If this variable is set to “On”, the extra connection is used to send TEXT or IMAGE data to the server. The first connection is used to deliver non-TEXT and non-IMAGE data to the server. When a WRITE or REWRITE is executed, the interface program INSERTS or UPDATES the non-TEXT and non-IMAGE data by using placeholder data in the TEXT or IMAGE columns.

The TEXT or IMAGE data is then sent using the extra connection established with the A_SYB_EXTRA_PROC variable. If the first connection is inside a transaction, the second connection is locked out of the row that is added or updated. The result is that it is not possible to WRITE or REWRITE records containing TEXT or IMAGE data while inside a transaction.

Note: Large columns cannot be used in conjunction with transaction management, because any column larger than 255 bytes (254 characters) is automatically converted to a TEXT or IMAGE column when the table is created. If you have TEXT or IMAGE columns in your table, WRITEs and REWRITEs fail unless this variable is set to "On".

See also

Section E.5, "Record and Table Locking"

A_SYB_FAST_ACCESS

A_SYB_FAST_ACCESS is a configuration variable that is set from your COBOL program. Files opened while this variable is set to a nonzero value are optimized for forward sequential access. While the default value is "0" (off, false, no), this configuration variable can also take values of "On" (true, yes).

We implemented this option to substantially improve the READ NEXT performance in some cases. For example, testing the benchmark program **iobench.cbl** in three different ways, yielded the following results for the READ AND SKIP operation:

No FAST_ACCESS:	72.76 seconds
FAST_ACCESS, ROWCOUNT 0:	148.88 seconds
FAST_ACCESS, ROWCOUNT 10:	8.09 seconds

("10" is the perfect value for ROWCOUNT in this benchmark, because the program does a START, 10 READ NEXT operations, and then does it again.)

For certain reporting programs, this option can dramatically improve performance. However, please note the following restrictions.

Files must be open INPUT or open IO with MASS-UPDATE. FAST_ACCESS gives a performance boost only when no locking is required. In files that allow locking, a record must be reread after being locked; this prevents an uninterrupted forward sequential traversal.

Files opened with FAST_ACCESS use a dedicated connection for reading from the file. Since connections are memory-intensive (both on the client and on the server), the number of files opened with FAST_ACCESS should be kept to a minimum. In the event that opening a connection fails, the file open still continues, but FAST_ACCESS mode is disabled, with the following message appearing in the trace file:

```
FAST_ACCESS mode not available.
```

Also, a new connection technically uses a new concurrent Sybase license from Sybase.

Files open with FAST_ACCESS will not participate in transactions, and may even cause the runtime to hang if transactions are used, especially if the FAST_ACCESS file is updated within the transaction. We suggest that if you use transactions, you don't use FAST_ACCESS. At the minimum, if you use transactions, we suggest that you use FAST_ACCESS only for files open INPUT.

Files open with FAST_ACCESS cannot be read backwards. In other words, READ PREVIOUS does not work with FAST_ACCESS files. In fact, if you try to READ PREVIOUS on a file opened with FAST_ACCESS, you get an error 9D,20.

The ANSI standard states that READ NEXT after a READ will return the next record. Some applications depend on this, and some applications just want to read dynamically from a file, and don't use the positioning facility. Because of this ambiguity, files that are open with FAST_ACCESS cannot be READ dynamically. If you try to READ on a file open with FAST_ACCESS, you get an error 9D,20.

There are no restrictions on WRITE, REWRITE, and DELETE. However, these operations use the cursor-based connection, not the dedicated connection. This is the reason transactions may hang.

A_SYB_FORCED_INDEX

A_SYB_FORCED_INDEX causes the Acu4GL product to attempt to use an actual index that matches the key used with the table. For this variable to have any effect, the index matching the key must be a *unique* index.

While the default value is "0" (off, false, no), this configuration variable can also take the value of "Yes" (on, true). If you don't use the forced-index feature, the Acu4GL product adds an ORDER BY clause to the SELECT, which can add to processing time.

We have discovered that Sybase does not guarantee that an index will be used if this variable is set to "1", and so the order of records returned to the COBOL program is not guaranteed to be correct in that case. Because of this, we highly discourages setting this variable to "On".

A_SYB_LOCK_DB

A_SYB_LOCK_DB specifies the name of the database that holds the lock table.

See also

Section E.5, "Record and Table Locking"

A_SYB_LOGIN

A_SYB_LOGIN indicates the user name under which you want to connect to the database system.

Example

To connect to the database with the user name MYNAME, specify:

```
A_SYB_LOGIN  MYNAME
```

in the runtime configuration file.

If A_SYB_LOGIN is *not set*, on Windows the runtime uses the value of your USER environment variable as your Sybase login name. On UNIX systems, the runtime uses your UNIX login name as your Sybase login name. For this automatic login to succeed, you must have set up a user with the same name as your computer login name.

See also

A_SYB_PASSWD runtime configuration file variable

Section E.2.5, "Setting Up a User Account"

A_SYB_MAX_CHARACTERS

A_SYB_MAX_CHARACTERS indicates the maximum number of bytes the Acu4GL product will allow in a table row.

Sybase places a limit on the number of bytes per table row. The Acu4GL product adheres to this limit, but sometimes it cannot accurately count how many bytes a particular row contains (because of overhead bytes that Sybase adds). This variable enables the developer to set the Acu4GL product's upper bound.

You might want to try reducing it if you discover that a row cannot be added to a table. By reducing the upper bound, you may be able to prevent this problem.

If Sybase increases the maximum number of bytes allowed in a row (in a future release of the Sybase product), you can increase the value of this variable to take advantage of the new limit.

The A_SYB_MAX_CHARACTERS variable has a default value of "1962".

A_SYB_MAX_COLUMNS

A_SYB_MAX_COLUMNS indicates the maximum number of columns the Acu4GL product will allow in a table.

Sybase places a limit on the number of columns per table. The Acu4GL product adheres to this limit, but sometimes it cannot accurately count how many columns a table contains (because a column has been added to a table without the Acu4GL product's knowledge). This variable enables the developer to set the Acu4GL product's upper bound. You might want to try reducing the upper bound if you discover that a table cannot be created for some reason. By reducing the upper bound, you allow for uncountable columns and thus may be able to prevent this problem.

If Sybase increases the maximum number of columns allowed per table (in a future release of the Sybase product), you can increase the value of this variable to take advantage of the new limit.

The A_SYB_MAX_COLUMNS variable has a default value of "250".

A_SYB_NATIVE_LOCK_TIMEOUT

This is one of two locking methods available with Acu4GL for Sybase. The methods are accessed via two configuration variables: A_SYB_VISION_LOCKS_FILE and A_SYB_NATIVE_LOCK_TIMEOUT. The lock method used is determined as follows: If A_SYB_VISION_LOCKS_FILE is set to the name of a Vision file that can be open I/O and has the correct structure, the Vision file is used to hold lock information. If A_SYB_NATIVE_LOCK_TIMEOUT is set to a positive value, native locking is used. Otherwise, the AcuLocks table is used to hold locks.

This locking method enables you to use Sybase native locks if you explicitly code transactions in your COBOL program. You can access this method by setting the configuration variable A_SYB_NATIVE_LOCK_TIMEOUT to a positive value. This value will be the number of seconds that a connection will wait for a timeout to occur. When such a timeout occurs (for any reason), the Acu4GL product assumes that the timeout was caused by a locked record and will return error 99 (record locked). If you set this variable but do not explicitly code transactions in your COBOL program, record locking does not occur. Note that the Acu4GL product will wait the number of seconds specified, and your application may seem to hang if the timeout is too long. On the other hand, if the timeout is too short, you may get record locked errors when the network is slow.

Sybase uses a page locking mechanism, and so this method of locking records may cause your application to return spurious record locked errors because of a record being locked on the same page as the record you are trying to access. If a future version of Sybase implements row-level locking, this locking method may be the preferred method.

See also

A_SYB_VISION_LOCKS_FILE configuration variable

Section E.5, "Record and Table Locking"

A_SYB_NO_COUNT_CHECK

When doing a REWRITE, the interface checks to see that a record was actually updated. If the record was not updated, the interface returns an error 23. Setting this variable "On" causes that check not to happen. This improves performance on REWRITE, at the risk of missing an error. While the default value is "Off" (false, no), this configuration variable can also take the value of "On" (true, yes).

A_SYB_NO_DBCLOSE

When the runtime is shutting down, the interface closes all the open connections, as per the Microsoft documentation. Sometimes, with some network drivers, this takes a significant amount of time. Setting this variable "On" causes the interface not to close the connections. Though this can speed up the runtime (during shutdown), we don't recommended it, because the Microsoft documentation is not being followed. While the default value is "Off" (false, no), this configuration variable can also take the value of "On" (true, yes).

A_SYB_NO_DBID

The interface stores the Database ID in the AcuLocks and AcuOpenTables tables, to distinguish different tables in different databases. Sometimes this causes problems. Setting this variable "ON" causes the interface to use a Database ID of "0", instead of the actual ID of the database. While the default value is "Off" (false, no), this configuration variable can also take the value of "On" (true, yes).

A_SYB_NO_RECORD_LOCKS

Setting this variable "ON" causes all READS to be treated as READ NO LOCK, which can improve performance (but has the obvious consequences). While the default value is "Off" (false, no), this configuration variable can also take the value of "On" (true, yes).

A_SYB_NO_TABLE_LOCKS

Setting this variable "ON" causes the interface not to use the AcuOpenTables table, which causes all table locking to be disabled. This can improve performance on OPEN and CLOSE statements. While the default value is "Off" (false, no), this configuration variable can also take the value of "On" (true, yes).

A_SYB_NO_23_ON_START

When A_SYB_NO_23_ON_START is set to a nonzero value, START does not detect whether records actually exist. Because it does not detect the existence of records, it is possible, when using this variable, to do a START without error, and for the next READ NEXT to return END_OF_FILE, contrary to the ANSI standard.

Example

```
A_SYB_NO_23_ON_START number
```

where *number* can be a zero or nonzero value.

While the default value is “0”, (off, false, no) this configuration variable can also take values of the value of “On” (yes, true).

A_SYB_PACKETSIZE

A_SYB_PACKETSIZE sets the size of network packets. Setting this variable can affect performance, since fewer and larger network calls can improve performance.

This variable must be set in the configuration file. It has no effect if it is set in a COBOL program via SET CONFIGURATION or SET ENVIRONMENT. The value of this variable is the largest size that the transport layer uses for network packets (although the underlying library may reduce the size specified; this is out of the control of the interface.) The largest value that can be specified is “32767”. The default depends on which version of the client libraries is linked into the runtime, although “512” is the most common default.

Use this configuration variable to tune your database performance. To set the packet size to “8192” use:

```
A_SYB_PACKETSIZE 8192
```

Setting this variable to “0” or to a negative value causes the Acu4GL product to use the default value.

A_SYB_PASSWD

The variable A_SYB_PASSWD should be set to the password assigned to the database account associated with the user name specified by **A_SYB_LOGIN**.

Examples

If the account with the user name in A_SYB_LOGIN has the associated password “CW021535”, you would specify:

```
A_SYB_PASSWD CW021535
```

in the runtime configuration file.

For better security, you can accept a password from the user during program execution; set the A_SYB_PASSWD variable based on the response:

```
ACCEPT RESPONSE NO-ECHO.
SET ENVIRONMENT "A_SYB_PASSWD" TO RESPONSE.
```

Note: If the user has been set up without a password, this variable need not be set.

A_SYB_ROWCOUNT

This variable has an effect only if you are reading on a key that does not allow duplicates, or if you have added an Identity column to the table. A_SYB_ROWCOUNT determines how many rows are returned by a SELECT statement sent to the server.

This variable can be used to speed up the Acu4GL product. For example, if you know you will be reading only one record at a time and reading from a unique key, you can set A_SYB_ROWCOUNT to "1", thus speeding up the processing.

If you know you are going to be reading records ten rows at a time, set A_SYB_ROWCOUNT to "10". If you don't have any information about how many rows are going to be requested, set this variable to "0", which is the default.

Note: Setting this variable to a non-optional value can actually degrade performance, since the interface may be forced to issue more SELECT statements once the rowcount has been determined. Use caution when setting this variable.

See also

A_SYB_ADD_IDENTITY runtime configuration file variable

A_SYB_SELECT_KEY_ONLY

This variable directs the interface to select key columns only when searching for records. Its use can improve READ performance on large tables with many rows.

When set to "True", the default value, A_SYB_SELECT_KEY_ONLY causes the interface to select only key columns when searching for records, and then select the entire row of the single record that must be returned to the COBOL program.

Setting A_SYB_SELECT_KEY_ONLY to "False" does not affect how the select is created for files open I/O (since the record must be locked, and then the rest of the data fetched), but it causes the interface to select all the columns of the table for files open INPUT. While the default value is "On" (true, yes), this configuration variable can also take values of "Off" (false, no).

A_SYB_SKIP_ALTERNATE_KEYS

A_SYB_SKIP_ALTERNATE_KEYS determines whether alternate keys are used to form indexes during table creation. The default value of this variable is "0", which means it's okay to use alternate keys.

If you set the variable to a nonzero value (such as "1"), alternate keys are *not* used to form indexes, which speeds up processing if many writes or rewrites are being performed. (Note that a value of "1" may slow processing if the application is reading sequentially using an alternate key.) While the default value is "0" (off, false, no), this configuration variable can also take the value of "On" (true, yes).

You can load a great deal of data into a table rapidly with this variable set to "1", and then create the missing indexes using **isql**.

See also

The Sybase documentation on **isql**

A_SYB_TRANSLATE_TO_ANSI

Setting this variable to "On" (true, yes), causes the Acu4GL interface to call the same translation function used by the Windows runtime to translate characters going to the server into the OEM character set, and to translate characters coming from the server to ANSI.

The default is "Off" (false, no), which indicates that the Acu4GL interface does not call the translation function, but passes the data as is to the library.

This variable applies only to the Windows version of the Acu4GL for Sybase interface.

A_SYB_UNLOCK_ON_EXECUTE

Setting this variable "ON" causes all invocations of I$IO using the EXECUTE opcode to unlock all records. Normally records are unlocked when a transaction finishes. But if users perform their own transaction management using **sql.acu** (which calls I$IO using the EXECUTE opcode), the interface never knows to unlock records, because it doesn't check the text sent to the database to see if it associated with a transaction. While the default value is "Off" (false, no), this configuration variable can also take the value of "On" (true, yes).

A_SYB_USE_DROPDOWN_QUERIES

Setting A_SYB_USE_DROPDOWN_QUERIES to a nonzero value causes select queries sent to the database to be of the drop-down variety, instead of a single large query. This variable is accessed only during a positioning operation, so you can set it at different times for different tables. For example, if you have a file with three fields in the primary key (keyseg1, keyseg2, keyseg3), and your COBOL program performs a START, the following query is sent to the database:

```
select (columns) from (table) where
((keyseg1 = value1 and keyseg2 = value2 and
keyseg3 > value3) or (keyseg1 = value1 and
keyseg2 > value2) or (keyseg1 > value1))
order by keyseg1, keyseg2, keyseg3
```

If you use drop-down queries, the following collection of queries is sent instead:

```
select (columns) from (table) where (keyseg1
= value1 and keyseg2 = value2 and keyseg3 >
value3) order by keyseg1, keyseg2, keyseg3
```

When that set is finished, the interface sends:

```
select (columns) from (table) where (keyseg1
= value1 and keyseg2 > value2) order by
keyseg1, keyseg2, keyseg3
```

And when that set is finished, the interface sends:

```
select (columns) from (table) where (keyseg1
> value1) order by keyseg1, keyseg2, keyseg3
```

There are advantages and disadvantages to each method. If you use the A4GL_WHERE_CONSTRAINT variable, you should probably set A_SYB_USE_DROPDOWN_QUERIES to "0", because the WHERE constraint limits the result set sufficiently that the larger query will be more efficient. See **Section 9.1.2, "The WHERE Constraint,"** for additional information.

If you usually START files and read to the end, you should set A_SYB_USE_DROPDOWN_QUERIES to "0", because a fewer number of queries need to be sent to the database. On the other hand, if you START files and stop reading after some condition, but haven't used the WHERE constraint, then you may get more efficient access by setting this variable to "1" and using the drop-down style of query. In either case, we recommend that you run some tests to see which value of this variable makes the most sense for your application. While the default value is "0" (off, false, no), this configuration variable can also take the value of "On" (true, yes).

Example

```
A_SYB_USE_DROPDOWN_QUERIES number
```

where *number* is a zero or nonzero value.

A_SYB_VISION_LOCKS_FILE

This is one of two locking methods available with Acu4GL. The methods are accessed via two configuration variables (A_SYB_VISION_LOCKS_FILE and **A_SYB_NATIVE_LOCK_TIMEOUT**). The lock method used is determined as follows: if A_SYB_VISION_LOCKS_FILE is set to the name of a Vision file that can be open I/O and has the correct structure, this method is used; if A_SYB_NATIVE_LOCK_TIMEOUT is set to a positive value, then this method is used. Otherwise, the AcuLocks table is used to hold locks.

A_SYB_VISION_LOCKS_FILE causes the lock table (AcuLocks) to be a Vision file instead of an SQL table. This can be accessed via the configuration variable A_SYB_VISION_LOCKS_FILE, which must be set to the name of the Vision file that will hold the lock information. Note that you must also set a configuration variable that specifies this file as a Vision file (using a _host variable). This file must be accessible to all users accessing the Sybase SQL server, either through a common directory or through AcuServer.

Also included with the Acu4GL for Sybase product is a small COBOL program that will manage this Vision file ("lockmgr"). This program should be run with the same runtime that you normally use to access Sybase tables, and with the same configuration variables set. The program detects whether Acu4GL for Sybase is available, and detects the A_SYB_VISION_LOCKS_FILE variable to determine which file to manage. This program displays all the records in the lockfile, and provides options for removing single records (by highlighting the desired record to remove) or removing all shown records, and also for restricting the shown records by PID, Table, and Database. You can refresh the display by clicking the Restrict button and then clicking **OK** without restricting the display further.

If everything is working correctly, there should be no records in this table when there are no users accessing the Sybase SQL server through Acu4GL. The source for this program is in the sample/acu4gl directory.

See also

Section E.5, "Record and Table Locking"

E.5 Record and Table Locking

By default, Sybase doesn't support the type of record and table locking that COBOL expects. For this reason, the Acu4GL for Sybase product implements its own locking method. This is accomplished with the addition of two tables to a database. You choose which database will hold these tables during installation of the Acu4GL for Sybase product.

Before using the locking tables, you need to execute the included "syb_inst.sql" script. See the installation instructions you used from this manual for the exact procedure. If you don't perform this step, the first time you try to execute a COBOL program that opens a Sybase table, you receive error 9D,11, "ACUCOBOL Lock Table Incorrect".

The first locking table is called AcuLocks; it holds the record locks. The columns in this table are the DBID, the Table ID, the Process ID of the process holding the lock, and the primary key of the record that is locked. There is a unique index on the DBID, the Table ID, and the Key Value, so that inserts into this table are automatically rejected if another user holds a lock on the row in question. This also provides the Database Administrator the information needed to determine who has locks set, and whether the user in question still has a connection to the server.

The second locking table is called AcuOpenTables; it holds information about open tables. The columns in this table are:

- the DBID
- the Table ID
- the process ID (PID) of the process that has the table open
- the open mode (input, output, I/O, or extend)
- whether multiple records can be locked
- whether the file can be open for I/O by any other users
- whether the file can be open at all by any other users
- whether Mass Update was specified in the open

There are no indices on this table, but there is a trigger, which automatically rejects opens that are not allowed based on other users' open modes.

By using these lock tables, the Acu4GL for Sybase product is able to support all the types of locking ordinarily supported by ACUCOBOL-GT. No special runtime configuration variables are required.

This method of locking is all that is needed if no applications other than COBOL programs are going to be using the Acu4GL for Sybase product. But if your site has other applications that access the Sybase databases, you must use a method of locking that is inherent to Sybase.

The only method of locking that Sybase supports internally is the result of time stamping and the use of BROWSE MODE (see the discussion of BROWSE MODE in the *Sybase Commands Reference Manual*). If a table has a time stamp column, the Acu4GL for Sybase product uses browse mode. This allows the server to detect whether another application has modified a record while an ACUCOBOL-GT application has had it locked.

E.6 Stored Procedures

A stored procedure is a collection of SQL statements residing on the server, stored as text in a table in the database. Stored procedures provide an efficient environment for Acu4GL because, once they are executed on the server, they do not need to be parsed and optimized each time they are executed. (If the server goes down, however, the stored procedure is parsed and optimized again the first time the procedure is called after the database restarts.)

If you run a set of stored procedures for the database in which data is manipulated (your production database), this database must be the setting for **A_SYB_LOCK_DB**. If you run the stored procedures against both the lock database and the production database, the lock database can be the setting for A_SYB_LOCK_DB. Note that if you run one version of stored procedures against the lock database and another version against the production database, the stored procedures in the lock database override those in the production database. Therefore, we recommend that you always update your

stored procedures with each new installation of Acu4GL, so that these procedures are consistent when you run them against the tables in your database.

This section discusses two types of stored procedures:

- Procedures that you may want to add to Acu4GL for Sybase
- Procedures provided in Acu4GL for Sybase that you, as the developer or administrator, may find useful

---

Note: If you are upgrading from an earlier version of Acu4GL, be sure to install the new stored procedures. We always upgrade stored procedures in such a way that they will be compatible with older versions of the product, so installing new stored procedures over old ones does not affect your ability to run with an older version of the interface software.

E.6.1 Developer- or Site-supplied Stored Procedures

This section provides information on stored procedures you may want to create. It also supplies some example code.

The Acu4GL for Sybase interface checks for these stored procedures when opening a file and will use them in certain circumstances, such as when A_SYB_NO_23_ON_START is set to " No."

These stored procedures are

- *tablename*_insert (where *tablename* is the name of the table being accessed)
- *tablename*_delete
- *tablename*_read
- *tablename*_update
- *tablename*_start*nnn* (where *nnn* is the key number to start on)

Note: The Acu4GL for Sybase product does not create these stored procedures or check their accuracy. Thus, it is possible to create stored procedures in such a way as to make the Acu4GL for Sybase product completely inoperable.

The Acu4GL product uses these stored procedures for performance reasons only.

Sample XFD

Sample code for developer-supplied stored procedures is based on the following example of an XFD:

```
XFD,03,FTEST2-FILE,FTESTDAT
# ftestdat.xfd - generated by ACUCOBOL-GT v4.2 Alpha 1
   (8/22/99)
# Generated Sun Aug  22 07:54:28 1999
00031,00031,003
01,0,004,00000
01
FTEST2-KEY
01,1,004,00004
02
FTEST2-KEY1-SEG1
FTEST2-KEY1-SEG2
01,0,004,00008
01
FTEST2-ALTKEY2
000
0006,00006
00000,00004,16,00004,+00,000,000,FTEST2-KEY
00004,00002,16,00002,+00,000,000,FTEST2-KEY1-SEG1
00006,00002,16,00002,+00,000,000,FTEST2-KEY1-SEG2
00008,00004,16,00004,+00,000,000,FTEST2-ALTKEY2
00012,00009,00,00009,+00,000,000,FTEST2-NUMBER
00021,00010,16,00010,+00,000,000,FTEST2-INFO
```

*tablename*_insert

*tablename*_insert is used to WRITE a record to the file. The parameters passed to the stored procedure are the values of all the columns in the row, in the order of the columns in the database. The timestamp column and identity column (if present in the table) are not passed to the stored procedure.

Given the XFD above, you might want to create the following stored procedure for writing records to a file:

```
create procedure ftestdat_insert
@ft2_key char(4),
@ft2_key1_seg1 char(2),
@ft2_key1_seg2 char(2),
@ft2_altkey2 char(4),
@ft2_number char(9),
@ft2_info char(10)
as
insert into ftestdat (ftest2_key, ftest2_key1_seg1,
ftest2_key1_seg2, ftest2_altkey2, ftest2_number,
ftest2_info) values (@ft2_key, @ft2_key1_seg1,
@ft2_key1_seg2, @ft2_altkey2, @ft2_number, @ft2_info)

grant execute on ftestdat_insert to public
```

*tablename*_delete

*tablename*_delete is used to DELETE a record from the file. The parameters passed to the stored procedure are the values of the primary key, in the order they are listed in the XFD.

Based on the sample XFD, you might want to create the following stored procedure for deleting records from a file:

```
create procedure ftestdat_delete
@ft2_key char(4)
as
delete from ftestdat where ftest2_key = @ft2_key

grant execute on ftestdat_delete to public
```

*tablename*_read

*tablename*_read is used to read a random record (READ, not READ NEXT or READ PREVIOUS). The parameters passed to the stored procedure are the values of the primary key, in the order they are listed in the XFD. The expected rowset is the columns in the first table (if secondary tables are used) or the columns of the table (if secondary tables are not necessary).

This stored procedure is very similar to *tablename*_start.

*tablename*_update

*tablename*_update is used to REWRITE a record from the file. The parameters passed to the stored procedure are the values of all the columns in the row, in the order of the columns in the database. The timestamp column and identity column (if present in the table) are not passed to the stored procedure.

For example, based on the sample XFD, you might want to create the following stored procedure for rewriting a record:

```
create procedure ftestdat_update
@ft2_key char(4),
@ft2_key1_seg1 char(2),
@ft2_key1_seg2 char(2),
@ft2_altkey2 char(4),
@ft2_number char(9),
@ft2_info char(10)
as
update ftestdat set
ftest2_key = @ft2_key,
ftest2_key1_seg1 = @ft2_key1_seg1,
ftest2_key1_seg2 = @ft2_key1_seg2,
ftest2_altkey2 = @ft2_altkey2,
ftest2_number = @ft2_number,
ftest2_info = @ft2_info
where ftest2_key = @ft2_key

grant execute on ftestdat_update to public
```

*tablename*_start*nnn*

*tablename*_start*nnn* is used to START a file. The *nnn* value is the key number to start on, and will be 0 filled. For example, the start procedure for the primary key for table mytab will be "mytab_start000".

Note: If A_SYB_NO_23_ON_START is set to "Yes," the start stored procedure is disabled.

Because there can be up to 119 alternate keys, the Acu4GL product does not search for a start procedure unless, or until, it is used. The parameters passed to the stored procedure are a 2-char mode [it is a varchar(2) field], with one of the following values: >, >=, =, <=, or <. The rest of the parameters are the columns of the key used to start. Because the ANSI specification for START includes information about the size of the key being used (and in particular allows partial keys), the start procedure is used only if an entire key is given to the start verb. This procedure is also special in that it does not return data, but needs to raise an error condition if the start fails. The way to raise the error condition from within the stored procedure is to include code similar to the following:

```
raiserror 22006 "Record not found"
```

The code 22006 is very important. It is the code searched for in setting the error condition from within the Acu4GL product. If you use a different number, your starts may succeed when they should actually fail.

For example, based on the sample XFD, you might want to create the following stored procedure to start a file:

```
create procedure ftestdat_start001
@mode varchar(2),
@ft2_key1_seg1 char(2),
@ft2_key1_seg2 char(2)
as

if exists (select 1 from ftestdat where

(ftest2_key1_seg1 = @ft2_key1_seg1 and
((@mode = ">=" and ftest2_key1_seg2 >=@ft2_key1_seg2) or
(@mode = ">" and ftest2_key1_seg2 > @ft2_key1_seg2) or
(@mode = "=" and ftest2_key1_seg2 = @ft2_key1_seg2) or
```

```
(@mode = "<" and ftest2_key1_seg2 < @ft2_key1_seg2) or
(@mode = "<=" and ftest2_key1_seg2 <= @ft2_key1_seg2))))
return
if exists (select 1 from ftestdat where
(((@mode = ">=" or @mode = ">") and
ftest2_key1_seg1 > @ft2_key1_seg1) or
((@mode = "<=" or @mode = "<") and
ftest2_key1_seg1 < @ft2_key1_seg1)))
return
raiserror 22006 "Record not found"

grant execute on ftestdat_start001 to public
```

E.6.2 Built-in Stored Procedures

Several stored procedures come with Acu4GL for Sybase. One, **sp_AcuInit**, provides a means for customized initialization. The others return information based on the AcuOpenTables and AcuLocks tables.

Note: You will see that the names of several stored procedures end in "_1", indicating the first version of the stored procedure. Whenever a stored procedure is updated, the extension is updated by one. This is why you can install new stored procedures without overwriting older ones. Be sure to install all stored procedures when you install Acu4GL for Sybase.

---

sp_AcuInit

If the stored procedure **sp_AcuInit** exists in the master database, it is executed when the connection is made to the server. This is a procedure you can set up to perform customized initialization. This stored procedure does not take any parameters and does not return any results. This optional stored procedure is executed for all connections by the Acu4GL interface to the database, not just the primary connection.

As an example of customized initialization, you can use this stored procedure to remove stale locks by calling **sp_AcuRemoveUnusedLocks_1** (which is installed when all the other Acu4GL stored procedures are installed) or to limit access to the database by certain users during certain hours. If **sp_AcuInit** returns an error, the connection is denied and the error is reported

to the COBOL program. The method for returning an error is to execute the Transact-SQL statement "raiserror". See your database documentation for information about Transact-SQL and stored procedures.

sp_AcuRemoveUnusedLocks_1

Use this stored procedure to determine who is logged in and to remove Process IDs that are no longer active on the system. You can call this from **sp_AcuInit** each time a user connects to the database to ensure that the lock table contains only active locks. This stored procedure must reside in the same database as the AcuOpenTables and AcuLocks tables; it is placed there automatically when these tables are created during installation.

sp_AcuTableReport_1

Use this stored procedure to learn who is using the tables in the database at the time this procedure is run. Run this procedure before running the **sp_AcuZeroUserCount** stored procedure, so that you can contact users to inform them that the database will be closing. This stored procedure must reside in the same database as the AcuOpenTables and AcuLocks tables; it is placed there automatically when these tables are created during installation.

sp_AcuUserCount_1

You can run **sp_AcuUserCount** from the query analyzer or ISQL to learn how many users have a particular table open. You can use this to track table and database activity to ensure that your database is running as efficiently as possible. This stored procedure must reside in the same database as the AcuOpenTables and AcuLocks tables; it is placed there automatically when these tables are created during installation.

sp_AcuZeroUserCount_1

Use **sp_AcuZeroUserCount** to remove all locks on a table and close it. Be sure to run this stored procedure on all tables in the database if you will be shutting down the database for any reason. This stored procedure must reside in the same database as the AcuOpenTables and AcuLocks tables; it is placed there automatically when these tables are created during installation.

E.7 Limits and Ranges

The following limits exist for the Sybase file system:

Maximum number of columns per key: 16
Maximum number of columns: 250
Maximum number of bytes in a single row when using Acu4GL for Sybase: 1962

To achieve the same sort or retrieval sequence under Sybase as under the Vision file system, key fields that contain signed numeric data must be preceded by a **BINARY** directive.

Acu4GL for Sybase supports the data types shown below. When it's creating tables, the following conversion rules are used, in the sequence shown:

COBOL	**Sybase**
DATE directive	DATETIME
BINARY directive	VARBINARY(*n*) (if SIZE < 255) IMAGE (if SIZE ≥ 255)
VAR_LENGTH directive	VARCHAR(*n*) (if SIZE < 255)
Usage FLOAT	REAL (if SIZE = 4) FLOAT (if SIZE = 8)

Any other numeric usage:

PIC 9(*n*)V9(*m*)	SMALLINT (if *m* = 0 and *n* < 5) INT (if *m* = 0 and *n* < 10) DECIMAL(*n* + *m*, *m*)(otherwise)

Any other usage:

PIC X(*n*)	CHAR(*n*) (if *n* < 255) TEXT (if *n* ≥ 255)

Other limits are described in Appendix B in Book 4, *Appendices*, of the ACUCOBOL-GT compiler manual.

E.8 Runtime Errors

This section lists the Acu4GL error messages that could occur during execution of your program. **Chapter 9** provides information on compile-time errors and also provides several methods for retrieving runtime errors.

An explanation and a recommended recovery procedure follow each message.

Runtime errors have this format:

9D,xx

The 9D indicates a file system error and is reported in your FILE STATUS variable. The *xx* is a secondary error code. These are the secondary errors reported directly from Acu4GL:

9D,01 Read error on dictionary file

An error occurred while reading the XFD file; this probably means the XFD is corrupt. Recreate the XFD file.

9D,02 Corrupt dictionary file

The dictionary file for one of your COBOL files is corrupt and cannot be read. Recompile with "-Fx" to re-create the dictionary. See **section 8.1** for additional information on compiler options.

9D,03 Dictionary (.xfd) file not found

The dictionary file for one of your COBOL files cannot be located. Be sure you have specified the correct directory via your **XFD_PREFIX** runtime configuration file variable. You may need to recompile with "-Fx" to create the dictionary.

9D,04 Too many fields in the key

There are more than 16 fields in a key. Check your key definitions and restructure the key that is illegal, and then recompile with "-Fx".

9D,05 (no message associated with this error)

A date given to the Acu4GL interface is invalid and cannot be converted.

9D,06 Mismatched dictionary file

The dictionary file (.xfd) for one of your files conflicts with the COBOL description of the file FD. The *xx* indicates a tertiary error code that is defined by the host file system. You can determine the exact nature of the mismatch by referring to the host system's error values.

The tertiary error code may have any of these values:

01 – mismatch found but exact cause unknown (this status is returned by the host file system)
02 – mismatch found in file's maximum record size
03 – mismatch found in file's minimum record size
04 – mismatch found in the number of keys in the file
05 – mismatch found in primary key description
06 – mismatch found in first alternate key description
07 – mismatch found in second alternate key description

The list continues in this manner for each alternate key.

9D,11 ACUCOBOL-GT stored procedures not found
or
ACUCOBOL-GT lock table missing

The installation of Acu4GL for Sybase creates a number of stored procedures and tables. At least one of these was not found.

9D,12 A column of a key is of data type TEXT or IMAGE, which is illegal

Columns that are part of an index may not be of type TEXT or type IMAGE. Check your key definition.

9D,13 Internal error

Multiple records were found with the same index (this is a Sybase error).

9D,14 DB library function returned an unexpected errordbinit

(Sybase) failed. An error message from Sybase is displayed on the terminal.

9D,16 Trying to rename a table across databases

RENAME works only within the same database.

9D,17 Cache error (internal error)

The internal process cache has been corrupted. Please contact Technical Services.

9D,18 Primary Key error

This means there was an error when creating the primary key for a secondary table.

9D,19 Table Size Error

The table is larger than Sybase will accept, either in the number of columns or in the number of bytes in the row. Use the **SECONDARY_TABLE** directive to get around this.

9D,20 (no message associated with this error)

This error tells you that you are trying to do something with a FAST_ACCESS table that is not allowed. See **A_SYB_FAST_ACCESS** to learn the restrictions for tables opened this way.

9D,21 (no message associated with this error)

There was a problem accessing one or more Vision locks files and processing of this request cannot continue. The tertiary error is an error code returned from Vision.

E.9 Common Questions and Answers

This section contains some questions and answers specific to Acu4GL for Sybase. Refer to **Chapter 10** for additional questions and answers that pertain to the Acu4GL family of products.

Question: What files do I need to link my C routines into Acu4GL?

Answer: Relinking is not necessary on Windows platforms. To add your own C routines to the runtime, relink the runtime as discussed in the runtime manual. You do not need to add anything special to use Acu4GL for Sybase.

(UNIX only)
All files in the lib subdirectory of your ACUCOBOL-GT distribution.

sub.c
sub85.c
filetbl.c
config85.c
sub.h

From your Acu4GL media you need:

sybase.o

From Sybase you need the platform-specific Open Client DB-Library/C product.

Instructions for linking are given in the installation portion of this appendix.

Question: When I try to open a file for output, I get the error 9D,2714. There is already an object named * in the database. Why?

Answer: One of your record's data items probably has the same name as a Sybase reserved word. Locate the column by comparing a file trace of the CREATE TABLE to Sybase's list of reserved words. Apply the **NAME** directive to the field in the FD that is associated with the invalid column, then recompile the program to create a new XFD file.

Question: Can I open tables in different databases?

Answer: Yes. Use a file name like:

```
database.owner.tablename
```

Note that, because Acu4GL for Sybase automatically determines an owner, you can also specify a file name like "database..tablename". The two dots are mandatory in this case.

Question: Can I use multiple servers (on the same machine or on different machines) on my network?

Answer: Yes. Each server has a unique name. To see if the necessary setup has been performed, enter:

```
isql -S servername
```

If this connects you to the server you want, you can open tables on that server by giving them a name like *servername.database.owner.tablename*. Note that this naming can be done in your runtime configuration file.

Question: I'm getting an error 9D,11 ACUCOBOL-GT lock table missing. I know that I added the lock table during installation.

Answer: This is probably a permissions problem. All users must have READ, WRITE, UPDATE, and DELETE access to AcuLocks (and therefore to the database that contains it). Be sure to check your permissions.

Question: I keep receiving an error message saying that my login is invalid. But I'm sure I'm using the correct username and password.

Answer: All usernames, passwords, and database names are case-sensitive. Be sure that you are typing the names exactly as they are set up.

Question: The runtime system can't seem to locate my XFD file, but when I check the XFD directory, the file is definitely there.

Answer: Check the case of the XFD filename. The Acu4GL product may be looking for uppercase on the base name.

Question: Are there any ACUCOBOL-GT library routines that do not work with or would not make sense to use with Acu4GL for Sybase?

Answer: Yes. There are two ACUCOBOL-GT library routines that either don't work with or do not make sense to use with Acu4GL for Sybase: C$COPY and C$RECOVER.

F Acu4GL for DB2 Information

Key Topics

F.1 DB2 Concepts Overview

A quick overview of some basic design concepts underlying the DB2 Database Management System will help you interface your COBOL program to it.

Transactions

The DB2 RDBMS is a transaction-based system. All of the work that you perform while using DB2 must occur within a transaction, whether that work is being done through Acu4GL for DB2 or another 4GL application**.** When you use Acu4GL® for DB2, a transaction is implicitly started for you by the database engine itself with the first file I/O operation performed on a file associated with DB2. Because all operations with Acu4GL for DB2 occur within a transaction, any record locked during processing remains locked until either a COMMIT WORK or ROLLBACK WORK is issued. This action results in behavior similar to the LOCK ON MULTIPLE RECORDS clause in COBOL.

The benefits of a transaction management system are best illustrated by an example. A COBOL application that handles order entry might perform these steps to accept an order:

1. Write an invoice record.
2. Update a customer record.
3. Write a payroll record for sales commissions.
4. Update an inventory record.

This series of four file operations is a logical unit. If the program were interrupted, and completed only some of the four file operations, then the files would be in an inconsistent state. For example, if the program terminated unexpectedly after it updated the customer record, but before it updated the inventory record, then a subsequent run might access non-existent inventory.

The solution to this problem is to provide a method for the programmer to define a set of operations that should either all occur or all not occur. Then, if the program encounters an error or terminates, the files are left in a consistent state.

All file operations that are part of a transaction are logged. Once logged, they can be either committed or rolled back (undone) by the program.

If a program dies, or the system fails, the log file can be used to reconstruct complete transactions, returning all files to a consistent state. Transaction logging thus offers these two facilities:

- It provides the programmer with the ability to define transactions and the ability to commit them or "undo" them (usually in response to an error condition). This "undo" facility is called a "rollback."
- It provides the ability to reconstruct files into a consistent state after a program dies or system failure occurs. This operation is called "recovery."

Note that transaction management facilities are available in ACUCOBOL-GT® Version 2.4.0 and later.

Record-locking issues in transactions

Applications that are written for transaction management systems, or that perform work in small "operation-based" logical units, benefit greatly from DB2's transaction management systems. Applications that are not written for transaction management encounter difficulty with record locking when operating against a system that enforces transaction management.

The difficulty can occur with an application that is performing more than one logical task at a time. Any operation that modifies or reads data in an I/O mode without the WITH NO LOCK phrase causes a lock to be placed in the database system. As a result, the application may have many more record locks present than would be expected by the normal rule of COBOL file locking. The application would act in a manner similar to when the LOCKS ON MULTIPLE RECORDS clause in COBOL is used. This can best be illustrated by an example:

1. The user is entering a customer's order.
2. As each line item is entered into the order, the inventory file is modified to reflect that items have been removed from the stock on hand.

3. The user must switch to a different part of the application to perform a different task, perhaps as a result of a phone call from a new customer.

4. All of the records that were locked, or modified, by the application before the switch remain locked because the first order is not complete. No COMMIT or ROLLBACK has been issued to complete the transaction. All of the records locked by the transaction remain locked until the application ends the transaction.

5. Because one order is open and not yet committed, other applications may be locked out of certain order items if they are still locked by the processing of the first order. The second order entry may be held up until the first order is completed.

6. Note that the first application is not locked out. A process can read its own locked records.

Acu4GL and record locking

Your DB2 database may be set up for wait-level locks. The DB2 universal database products have the ability to time-out a lock and send an error return code to the waiting application. See your database administrator for details. (This SQL code would be placed in ERROR_MAP_FILE.)

Acu4GL provides semi-automated ways to handle transaction logging based on the setting of the **4GL_COMMIT_COUNT** variable. You can also directly alter your source code to deal with this issue. Individual users determine how much work they wish to do to conform to DB2's transaction management system by choosing the method that best fits their needs and resources. The following methods are listed in order of increasing amount of work:

4GL_COMMIT_COUNT = 0 (Default)

When you set this variable to zero, the runtime tracks the number of logical locks that are currently in effect. When the number of logical locks reaches zero, the runtime assumes that a transaction is complete and issues a COMMIT statement.

4GL_COMMIT_COUNT = *n*

When you set this variable to a nonzero value, the runtime tracks the number of WRITE, REWRITE, and DELETE operations, until the value of 4GL_COMMIT_COUNT is reached, at which time the runtime issues a COMMIT statement. The READ, START, and READ NEXT operations do not count toward this total because the runtime is tracking data-altering operations rather than logical record locks. The disadvantage of this method is that when a COMMIT is issued, any record locks held by the runtime are released.

4GL_COMMIT_COUNT = –1

No commit is issued by the Acu4GL product. When 4GL_COMMIT_COUNT is set to "-1", two alternate ways to perform a commit or rollback are available:

1. Call **sql.acu** with COMMIT WORK or ROLLBACK WORK.
2. Use the COBOL verbs COMMIT and ROLLBACK, available in ACUCOBOL-GT.

4GL_COMMIT_COUNT is set to "–1" automatically when you use the transaction management facilities available in the ACUCOBOL-GT compiler. A COMMIT WORK is, however, issued on exit from the runtime (for example, on execution of a STOP RUN).

COMMIT VERB IN COBOL

This method forces a COMMIT to be sent to DB2. It can be used in conjunction with other modes of COMMIT handling. For non-DB2 files, this is equivalent to the UNLOCK ALL verb.

EXPLICITLY CODED TRANSACTIONS

This method provides the greatest flexibility in that transactions are specifically tailored for the user's application. This method also requires the most work for traditional COBOL programs in which transaction modules may not be clearly defined.

See **section F.6, "Record and Table Locking,"** for additional information.

F.2 Installation and Setup

The following topics list the steps you must perform before you begin using Acu4GL for DB2 on a new system.

- **Windows Installation**
- **UNIX Installation Steps**
- **Sample Configuration File**
- **Setting Up the User Environment**
- **Designating the Host File System**
- **Designating the Host Data Source**

F.2.1 Windows Installation

The DB2 RDBMS, version 8.2.0 or higher, must be installed and configured *prior* to the installation of Acu4GL for DB2. Consult your DB2 documentation if you have any questions regarding this step.

You should have the client software installed and configured, along with any Database Aliases, *before* installing Acu4GL for DB2.

Note: Acu4GL for DB2 requires DB2 client software version 6.01 or higher on the client machine. The "bin" directory of your DB2 client software must be in your PATH.

CD-ROM installation

Instructions for installing your Acu4GL product from the ACUCOBOL-GT Development Suite CD-ROM are contained on the *Quick Start* card that accompanied the product. Please refer to it for installing the ACUCOBOL-GT Development Suite, which includes Acu4GL.

Once the installation is complete, please refer to this appendix for setting up your Acu4GL product.

Relinking for Windows users

Relinking is *not* required for Windows users (but it is required for UNIX users and is explained in the next section). Acu4GL for DB2 is a windows ".DLL" ("a4db232.dll") that is loaded at run time from the "bin" directory of your ACUCOBOL-GT product distribution.

F.2.2 UNIX Installation Steps

The Acu4GL for DB2 product on UNIX is an add-on module that must be linked with the ACUCOBOL-GT runtime system. For this reason, you'll need a C compiler to install the Acu4GL product. To interface, you must use the ACUCOBOL-GT compiler and runtime, and the version of the runtime must match the version of Acu4GL.

The Acu4GL product is shipped using either TAR or CPIO format, depending on the type of machine you have. The label on the medium shipped to you tells you which format has been used.

From your Acucorp directory, choose where you want to install the Acu4GL product (or create a new directory for it) and then enter one of the following commands:

```
tar xfv device
```

or

```
cpio -icvBd < device
```

This will copy the files from the distribution medium to your ACUCOBOL-GT directory structure. *device* is the appropriate hardware device name (for example, /dev/rdiskette or /dev/rmt0). Sites using Texas Instruments System 1500 should add an uppercase "T" to the **cpio** options (-icvBdT).

Contents of the medium

Note that each Acu4GL product has its own *license file*, which must be located in the same directory as the ACUCOBOL-GT runtime. For DB2, the license file is distributed with the name "runcbl.klc".

Step 1: Install DB2

The DB2 RDBMS, version 6.01 must be installed and configured *prior* to the installation of Acu4GL for DB2.

Step 2: Create a new runtime system

Complete steps 2a and 2b to create a new runtime that includes the Acu4GL for DB2 product.

Note: In the following directions, the term "runtime system" refers to the runtime shared object on systems where the ACUCOBOL-GT runtime is a shared object and to **runcbl** on other systems, where the runtime is static. The runtime is a shared object on the following systems: AIX 5.1 and later, HP-UX 11 and later, and Solaris 7 and later. To check, look at the contents of the "lib" subdirectory of your ACUCOBOL-GT installation. If the files "libruncbl.so" or "libruncbl.sl" reside in that directory, the runtime is a shared object on your system.

2a. Link the runtime system.

Edit the Makefile in the "lib" subdirectory. Change the CFLAGS file to read as follows:

```
CFLAGS = $(ACUSERVER_FLAGS) $(ACUCONNECT_FLAGS)
   $(ACUSQL_FLAGS) -DACU4GL -DUSE_DB2
```

Note: Be sure to include the first word: "CFLAGS".

Edit the FSI_SUBS line to read as follows:

```
FSI_SUBS = a4db2.o
```

Edit the FSI_LIBS line to read as follows:

```
FSI_LIBS = $(ACU_LIBDIR)/libexpat.a <ibm DB2
  directory>/lib/libdb2.so
```

Make sure you are in the directory containing the ACUCOBOL-GT runtime system. Then, at the UNIX prompt, enter the following command:

```
make
```

This compiles “sub.c” and “filetbl.c”, and then links the runtime system.

Note: Make sure you have set all of the flags specific to your platform correctly before you relink your library and object files.

If the make fails because of an out-of-date symbol table, execute the following command:

```
ranlib *.a
```

and then execute the make again. If the make fails for any other reason, call Technical Services.

2b. Verify the link.

Enter the following command:

```
./runcbl -vv
```

to verify the link. This returns version information on all of the products linked into your runtime system. Make sure it reports the version of Acu4GL for DB2.

Shared libraries

If you have relinked the ACUCOBOL-GT runtime and receive an error message of this type when you try to execute it:

```
"Could not load library; no such file or directory"

"Can't open shared library . . . "
```

this may mean that your operating system is using shared libraries and cannot find them. This can occur even if the shared libraries reside in the same directory in which you are currently located.

Set the environment variable LD_LIBRARY_PATH to find these shared libraries.

F.2.3 Sample Configuration File

The "cblconfi.db2" file is a sample configuration file. This file demonstrates setting the DEFAULT_HOST to DB2, designating an ERROR_MAP_FILE used to map DB2's errors into COBOL errors, and setting the configuration variables required for the Acu4GL for DB2 interface.

Modify "cblconfi.db2" to configure your individual login, password, and database (or database alias). These configuration variables are:

A_DB2_LOGIN
A_DB2_PASSWD
A_DB2_DATASOURCE

The sample error mapping file is placed in the /etc subdirectory of your Acu4GL for DB2 distribution. See **section F.5, "Configuration File Variables,"** for additional information.

Note: Be sure to set the PATH to find ERROR_MAP_FILE.

F.2.4 Setting Up the User Environment

You must use the sample configuration file, described in **section F.2.3**, and error map file as your template. These files contain important settings for DB2:

- **A_DB2_USE_CHAR_FOR_BINARY**
- error mappings

In addition to setting the variables required for DB2, you will need to do the following:

1. Ensure that your execution path contains the name of the directory in which you placed the runtime executable that includes Acu4GL for DB2.
2. Set the **A_DB2_LOGIN** and **A_DB2_PASSWD** variables, either in your environment or in the ACUCOBOL-GT runtime configuration file. Also, consider whether or not setting the **USER_PATH** variable

would be useful in your set up situation. For security reasons, it is best to set the password variable from your COBOL program by asking the user to enter a password and then executing:

```
SET ENVIRONMENT "A_DB2_PASSWD" TO user-entry
```

3. You may want to make and use a personalized copy of the configuration file to avoid impacting other users. The ACUCOBOL-GT documentation describes how to use the A_CONFIG environment variable or the "-c" runtime option to identify a personal configuration file.

4. Define any other configuration file variables you require for your unique environment. Possible variables define error map file location, locking method, commit count, commit timing, and BINARY/CHAR type conversion.

For detailed information on ACUCOBOL-GT configuration file variables, see **section F.5, "Configuration File Variables."**

F.2.5 Designating the Host File System

When your COBOL application opens an *existing* file, most file systems linked into the runtime will be searched for the named file. However, each time the COBOL application creates a *new* file, it needs to know which file system to use. You provide the name of the file system with one of two runtime configuration file variables. The first is:

```
DEFAULT_HOST filesystem
```

This will designate the file system to be used for newly created files that are not individually assigned. For example,

```
DEFAULT_HOST DB2
```

means that all new files will be DB2 files unless otherwise specified by the second configuration variable, which is:

```
filename_HOST filesystem
```

where *filename* is the file name, without any extension, named in the ASSIGN TO clause of your SELECT statement. This configuration variable is used to assign an individual data file to a different file system. Any file so assigned will use the designated file system, and not the one specified by DEFAULT_HOST. For example,

```
myfile_HOST VISION
```

means that *myfile* will be under the Vision file system. The ability to designate a different file system for certain files enables you to tailor your application to a specific customer's needs or to implement an incremental conversion for a customer. With relational databases, this is particularly useful in that it allows you to tune an application for processing speed and resource requirements.

You can use these runtime configuration file variables in combination to assign your new files in a default with exceptions manner; for example, this set of entries:

```
DEFAULT_HOST VISION
afile_HOST DB2
bfile_HOST DB2
```

means that all new files except *afile* and *bfile* will be assigned to Vision, and those two files will be assigned to DB2.

You can also change the values of these variables during program execution by including in your code:

```
SET ENVIRONMENT "filename_HOST" TO filesystem
```

or

```
SET ENVIRONMENT "DEFAULT_HOST" TO filesystem
```

This enables you to change file systems during the execution of your program. This is not the typical way to specify a file system; normally it is designated in the runtime configuration file and is not changed in the COBOL program.

Note: The Acu4GL for DB2 product allows you to create a DB2 table with an OPEN OUTPUT statement, just as you can create Vision indexed files. The DB2 equivalent of a Vision file is a table, not a database. You must create a database for your DB2 tables before you run the COBOL program that creates the tables, just as you must create a directory for your files before you run a COBOL program that creates Vision files.

F.2.6 Designating the Host Data Source

You must tell the runtime system which data source to use for your DB2 file. You accomplish this by setting the configuration variable **A_DB2_DATASOURCE**. You can set this variable in your COBOL configuration file if you will be using only one data source. This variable is described in section F.4. "Acu4GL for DB2 Configuration File Variables."

If you do not know the data source name in advance, or if you intend to use more than one data source, you may set the data source name dynamically at run time. In your COBOL program, you would add code similar to this prior to the statement that opens the file:

```
SET ENVIRONMENT "A_DB2_DATASOURCE" TO "DB2 Database"
```

You are now ready to use the **sql.acu** program, as outlined in **section 2.4, "Using the "sql.acu" Program."**

After you learn about and use this utility, you next find out about preparing and compiling your COBOL program, followed by learning to use the demonstration program. See **section 6.1, "Preparing and Compiling Your COBOL Program,"** and **section 2.5, "The Demonstration Program,"** for additional information.

F.3 Filename Translation

As you prepare to work with Acu4GL for DB2, you may find it helpful to understand the rules around filename interpretation and to understand how the names of tables and XFD files are formed and work together.

When the ACUCOBOL-GT compiler generates XFD files, it uses lowercase letters to name the XFD file. In addition, the compiler changes hyphens to underscores when naming the XFD file.

Through configuration variables, the runtime translates the name in the COBOL program into the filename that is passed to the **open()** function in the runtime. The **open()** function determines which file system to pass the request to, but does not change the name of the file.

Acu4GL for DB2 translates the name to uppercase letters and changes hyphens to underscores. File extensions will be stripped, based on the setting of **4GL_IGNORED_SUFFIX_LIST**. This "new" name is the one that Acu4GL for DB2 will use in the future for references to database tables.

F.4 Decimal Points

Acu4GL for DB2 reads the decimal point character from the environment variable DECIMAL_POINT. If DECIMAL_POINT is set, Acu4GL uses that character. If the variable is not set, Acu4GL uses the decimal character that is encoded in the XFD file.

The two most common decimal indicators are the period "."and the comma "," characters. The comma is used often in European code and is often indicated in COBOL programs by the "DECIMAL POINT IS COMMA" clause.

F.5 Configuration File Variables

This section lists the runtime configuration file variables that are specific to Acu4GL for DB2. Configuration file variables that are generally applicable to any RDBMS with which Acu4GL communicates are discussed in **section 8.2, "Runtime Configuration Variables."**

A_DB2_CATALOG
A_DB2_TABLE_TYPES
A_DB2_USE_CATALOG
USER_PATH

A_DB2_ALTERNATE_COMMIT_LOGIC

The primary purpose of the configuration variable 4GL_COMMIT_COUNT is to provide for applications that must communicate with a transaction-oriented database but have not explicitly coded transactions into their COBOL application.

The configuration variable A_DB2_ALTERNATE_COMMIT_LOGIC determines how the interface will respond to the setting of 4GL_COMMIT_COUNT. When A_DB2_ALTERNATE_COMMIT_LOGIC is set to "0" or "FALSE" (the default), the value of 4GL_COMMIT_COUNT is checked only at startup. The interface will issue a commit when the criteria set by 4GL_COMMIT_COUNT is met, regardless of the current transaction state.

When A_DB2_ALTERNATE_COMMIT_LOGIC is set to "1", the value of 4GL_COMMIT_COUNT is checked after each WRITE, REWRITE, DELETE, or UNLOCK operation. A commit is issued, however, only if the runtime is not currently in a transaction.

Example

```
4GL_COMMIT_COUNT=1
A_DB2_ALTERNATE_COMMIT_LOGIC=0
```

Each WRITE operation is committed to the database immediately. This prevents an application from being able to rollback a WRITE.

See Also

4GL_COMMIT_COUNT runtime configuration variable

A_ODBC_ALTERNATE_COMMIT_LOGIC configuration variable

A_DB2_CATALOG

This variable indicates the catalog name to be used when Acu4GL searches for objects in the database. Note that not all data sources will support a catalog. Using A_DB2_CATALOG with other variables, such as USER_PATH and A_DB2_TABLE_TYPES, can speed up the finding of

tables in large databases. It can also prevent an error 9D,14: "More than one table with the same name," thereby enabling access to tables with identical names, but with different catalogs.

For example, If the USER_PATH and A_DB2_CATALOG are used, the form of the SQL statement will be modified from:

```
select COL1, ... from TABLENAME ...
```

to:

```
SELECT COL1, ... FROM [catalog.][username.]TABLENAME ...
```

where *catalog* and *username* will be filled in if provided.

See Also

USER_PATH configuration variable

A_DB2_TABLE_TYPES configuration variable

A_DB2_COMMIT_ON_BEGIN

DB2 has no START TRANSACTION method. Everything since the last COMMIT or ROLLBACK is considered part of the current transaction.

To ensure that the previous transaction has been ended before a new one begins, set A_DB2_COMMIT_ON_BEGIN to a nonzero value. This causes each COBOL START TRANSACTION to first issue a COMMIT. This ensures that the previous transaction has been ended before the new one starts.

If this variable is not set, or is set to "0", a COBOL ROLLBACK may affect file I/O that occurred before the most recent COBOL START TRANSACTION. While the default value is "0" (off, false, no), this configuration variable can also take values of "On" (true, yes).

A_DB2_DATASOURCE

Set A_DB2_DATASOURCE to the exact name of the host data source. You can set this variable in your COBOL configuration file if you will be using only one data source.

Example

```
A_DB2_DATASOURCE stores
```

If you do not know the data source name in advance, or if you intend to use more than one data source, you may set the data source name dynamically at run time. In your COBOL program, you would add code similar to this prior to the statement that opens the file:

```
SET ENVIRONMENT "A_DB2_DATASOURCE" TO "data source name"
```

There is no default value for this variable.

A_DB2_ERROR_MAP_FILE

Set A_DB2_ERROR_MAP_FILE to the name of the file that maps database-specific errors to COBOL errors. This configuration file variable allows you to map errors using a text file to supplement the default method of providing errors. Create the file using the guidelines described on the following page, and then use the configuration file variable, A_DB2_ERROR_MAP_FILE, to indicate the name and location of the file you created.

Example

If the file used for mapping is called MAP, and this file is located in the directory C:\DB2, you would specify:

```
A_DB2_ERROR_MAP_FILE  c:\DB2\MAP
```

in the runtime configuration file. There is no default value for this variable.

Guidelines for creating a map file

Although you can check your data source documentation for error code information, the easiest way to determine what error codes need to be mapped to more appropriate COBOL codes is through trial and error. As users use Acu4GL for DB2, they may report receiving error messages that don’t make sense based on their situation. Research these errors and try to determine a more appropriate COBOL error response.

When you create your error map file, use the following guidelines:

- Begin comment lines with "#". Blank lines are also considered comments.

- Break the rest of the file into sections, with each section header comprising all the information enclosed in brackets from the data source error function.

  The following example leads to the entry in the sample DB2 error file:

```
  OdbcOneInfo: State: 23505, Native Error: -803
  '[IBM][CLI Driver][DB2/NT]SQL0803N One or more values
in the INSERT statement, UPDATE statement, or foreign
key update caused by a DELETE statement are not valid
because they would produce duplicate rows for a table
with a primary key, unique constraint, or unique index.
SQLSTATE=23505'
  Setting f_errno = 19
Leaving, error
***File system error value = -803 ***
>>>file status = 9D,-803
```

- Include two fields in each line in the section: the internal error number, and an ACUCOBOL-GT mapping string.

  Using the same example, you would include this line in the error_map file:

```
-803 E_DUPLICATE
```

  The valid values for the second field are as follows:

```
E_SYS_ERR
E_PARAM_ERR
E_TOO_MANY_FILES
E_MODE_CLASH
E_REC_LOCKED
E_BROKEN
E_DUPLICATE
E_NOT_FOUND
E_UNDEF_RECORD
E_DISK_FULL
E_FILE_LOCKED
E_REC_CHANGED
```

```
E_MISMATCH
E_NO_MEMORY
E_MISSING_FILE
E_PERMISSION
E_NO_SUPPORT
E_NO_LOCKS
```

The file “DB2.err” contains some initial file error mappings.

A_DB2_ISOLATION_LEVEL

Use the A_DB2_ISOLATION_LEVEL configuration variable to set the isolation level. The default ordering of isolation levels is:

SQL_TXN_READ_COMMITTED
SQL_TXN_READ_UNCOMMITTED
SQL_TXN_REPEATABLE_READ
SQL_TXN_SERIALIZABLE

You can change the isolation level by setting the configuration variable A_DB2_ISOLATION_LEVEL to an integer in the configuration file. The settings are:

Isolation level	**Setting in configuration file**
SQL_TXN_READ_UNCOMMITTED	1
SQL_TXN_READ_COMMITTED	2
SQL_TXN_REPEATABLE_READ	3
SQL_TXN_SERIALIZABLE	4

For example, to set the isolation level to SQL_TXN_READ_COMMITTED, add the following entry to the configuration file:

```
A_DB2_ISOLATION_LEVEL 2
```

If the user-set isolation level is not supported by the driver, the default method of selecting a level is used.

A_DB2_LOCK_METHOD

Use this variable to specify the locking method that Acu4GL for DB2 should use when accessing your data source. Possible values are:

none
SETPOS
SETSTMTOPTION
UPDATECOLUMN

The default is “UPDATECOLUMN”.

Example:

```
A_DB2_LOCK_METHOD UPDATECOLUMN
```

SETPOS

Specify SETPOS as your lock method to tell Acu4GL for DB2 to perform the following locking sequence:

When setting up a statement handle for accessing the DB2 data source, the Acu4GL product calls a function called SQLSetScrollOptions, with values (SQL-CONCUR-LOCK, SQL-SCROLL-KEYSET-DRIVEN, 1).

When fetching rows from the data source, the Acu4GL product calls SQLExtendedFetch instead of SQLFetch.

Finally, it calls SQLSetPos, with the values (1, SQL-POSITION, SQL-LOCK-EXCLUSIVE).

If any of these functions do not exist, A_DB2_LOCK_METHOD reverts to none.

Note: Even if these functions do exist, there is no guarantee that any rows will be locked using this sequence of calls. Contact your database vendor to determine whether or not this sequence will be effective for your data source.

SETSTMTOPTION

This option is similar to SETPOS, except a function called SQLSetStmtOption performs the task of SQLSetScrollOptions and has more functionality.

In particular, when using this method of locking, the Acu4GL product will call SQLSetStmtOption a number of times, with values (SQL-CONCURRENCY, SQL-CONCUR-LOCK), (SQL-CURSOR-TYPES, SQL-CURSOR-KEYSET-DRIVEN), and (SQL_KEYSET-SIZE, 1). Again, if the function does not exist, A_DB2_LOCK_METHOD reverts to none.

UPDATECOLUMN

Specifying UPDATECOLUMN as your lock method performs an entirely different type of locking. Instead of trying to lock a row while reading it, this method creates a new statement handle for the data source. Then, after fetching the data from the data source, it resubmits an SQL query to select the same row (based on the primary key) and adds an UPDATE clause. Last, it fetches the data from the data source. This method has the most overhead, but it is much more likely to succeed in locking records.

A_DB2_LOGIN

A_DB2_LOGIN indicates the user name under which you want to connect to the database system. Not all data sources require a user login name. Those that do may have case requirements. Check your data source documentation to determine if login is case-sensitive.

Example

To connect to the database with the user name MYNAME, you could specify:

```
A_DB2_LOGIN  MYNAME
```

or

```
A_DB2_LOGIN  myname
```

in the runtime configuration file. There is no default value for this variable. If no login is specified, none will be used.

See also

A_DB2_PASSWD runtime configuration file variable

A_DB2_PASSWD

The variable A_DB2_PASSWD should be set to the password assigned to the database account associated with the user name specified by **A_DB2_LOGIN**. Not all data sources require a password. Those that do may have case requirements. Check your data source documentation to determine if password is case-sensitive.

Examples

If the account with the user name in A_DB2_LOGIN has the associated password CW021535, you would specify:

```
A_DB2_PASSWD  CW021535
```

in the runtime configuration file.

For better security, you can accept a password from the user during program execution; set the A_DB2_PASSWD variable based on the response:

```
ACCEPT RESPONSE NO-ECHO.
SET ENVIRONMENT "A_DB2_PASSWD" TO RESPONSE.
```

Note: If the user has been set up without a password, this variable need not be set. There is no default value for this variable. If no password is specified, none will be used.

A_DB2_STRICT_EQUAL

When Acu4GL for DB2 executes a START … KEY EQUAL *keyval*, the interface generates a SELECT statement to select all rows with a key equal to the key columns in *keyval*. When fetching the data, Acu4GL eventually

detects that there is no more data. If A_DB2_STRICT_EQUAL is set to "True" at the time the START is executed, the interface returns the error END OF FILE (error 10) at this point. If A_DB2_STRICT_EQUAL is set to "False" (the default), Acu4GL for DB2 then generates a new SELECT statement to continue reading records until the end of the file rather than the subset of records represented by *keyval*, and these records may be discarded by the program. (Note that this is program-dependent behavior.) Setting A_DB2_STRICT_EQUAL to "True" causes the interface to stop reading records earlier, which can make SELECT statements generated by Acu4GL for DB2 more efficient.

Note the following conditions:

- A_DB2_STRICT_EQUAL must be set to "True" when the START is executed.
- The START must be a KEY EQUAL start.
- There must be no COMMITs between the START and the end of the file. Because DB2 closes all cursors on COMMIT and ROLLBACK (according to the ANSI 92 SQL standard), the interface must regenerate a SELECT the first time it attempts to execute a READ NEXT after a COMMIT or ROLLBACK. This means that the **4GL_COMMIT_COUNT** configuration variable should be set to "-1", so that no COMMITs are issued by Acu4GL for DB2. See "**4GL_COMMIT_COUNT**" in Chapter 8 for additional information.
- A_DB2_STRICT_EQUAL is most useful with alternate keys that allow duplicates.

Note: This same behavior can be accomplished with the **4GL_WHERE_CONSTRAINT**, but using A_DB2_STRICT_EQUAL requires less programming.

While the default value is "False" (off, no, 0), this configuration variable can also take the value "True" (on, yes, 1).

A_DB2_TABLE_TYPES

This variable is used to specify a table type (TABLE, VIEW, and so forth) that should be looked for when selecting a database table. Using A_DB2_TABLE_TYPES with other variables, such as USER_PATH and A_DB2_CATALOG, can speed up the finding of tables in large databases.

This information will be passed to the API function call SQLTables() as the TableType parameter. Please refer to your driver's documentation for supported types.

Example

```
A_DB2_TABLE_TYPES  TABLE,VIEW
```

See also

USER_PATH configuration variable

A_DB2_ALTERNATE_COMMIT_LOGIC configuration variable

A_DB2_USE_CATALOG

Not all data sources will support a catalog, or the data sources may return a catalog name that is not useful. The default behavior of the runtime is to not use the catalog in the actual SQL queries. This configuration variable enables you to modify this behavior.

The default value is "FALSE", which will cause the catalog portion of the table name to be treated as blank in SQL queries. Setting this variable to "TRUE" will cause the value of the catalog returned by SQLTables ODBC function to be used in subsequent SQL queries.

See also

USER_PATH configuration variable

A_DB2_CATALOG configuration variable

A_DB2_USE_CHAR_FOR_BINARY

Some data sources have restrictions on the number of binary large objects (BLOBs) that can be placed into a single table. If your data source has such restrictions, specify A_DB2_USE_CHAR_FOR_BINARY in the configuration file. The possible values for this variable are:

0 = use BINARY type
1 = use CHAR type

A nonzero value for A_DB2_USE_CHAR_FOR_BINARY lets you store data that uses the BINARY directive as hexadecimal encoded CHAR types. This allows you to work around your data source restriction. While the default value is "0" (off, false, no), this configuration variable can also take values of "On" (true, yes).

Because of current limitations of the interface, you must set this variable to "1", or the database returns an error on file creation. This has been set for you in the sample configuration file "cblconfig.db2". Do not change this setting.

Example

```
A_DB2_USE_CHAR_FOR_BINARY 1
```

A_DB2_USE_SQLCOLUMNS

For some large databases, the API function SQLColumns(), which is called to get a description of the columns in a table, sometimes takes a long time to execute. If you are experiencing such problems, you can use the API function call SQLDescribeCol() instead, which can improve performance for large databases.

To turn on this new functionality, set the A_DB2_USE_SQLCOLUMNS configuration variable to FALSE.

The default value of this variable is TRUE.

A_DB2_USE_SQLTABLES

For some large databases, the API function SQLTables(), which is called to get a list of the tables and information about them, sometimes takes a long time to execute. If you are experiencing such problems, you can instruct the interface to build a test SQL query and use the API function call SQLNumResultTables() to determine if the table exists. This may improve performance for large databases.

To turn on this new functionality, set the A_DB2_USE_SQLTABLES configuration variable to FALSE.

The default value of this variable is TRUE, which means the interface will use the SQLTables() API function.

USER_PATH

USER_PATH indicates the user name or names (schemas) to be used when Acu4GL searches for files. The order of the names is significant. The syntax for this variable is:

```
USER_PATH  user1 [user2]...
```

where the *user* argument may be either the name of a user (schema) on the system, or a period (“.”), which indicates the files owned by yourself.

The type of OPEN being issued for the file determines the effects of this setting.

Examples

If an OPEN INPUT or OPEN I/O is issued, and a USER_PATH variable is defined in the runtime configuration file, Acu4GL searches for a user of the named file in the list of users in USER_PATH. The first valid file is opened.

If USER_PATH is defined and the current user is the owner of the file, the current user must be included as one of the users, as indicated by a “.” or the setting of your login schema (A_DB2_LOGIN) in the USER_PATH. If this is not the case, even though the current user has created the table, it will not

be found and a file error "35" will result. This circumstance can occur if the file was created with the OPEN OUTPUT phrase, and "." is not an element in USER_PATH. When the table is created, the current user will be the owner of that table. When the runtime attempts to open the table, the runtime will not look for tables owned by the current user unless USER_PATH is not set, or unless "." or the user's current schema is part of the USER_PATH setting.

If an OPEN INPUT or OPEN I/O is issued, and *no* USER_PATH variable is in the runtime configuration file, Acu4GL searches for a user of the named file in the user named for login (A_DB2_LOGIN). Acu4GL opens the first file that has a valid combination of user and file name.

If an OPEN OUTPUT is issued (whether USER_PATH is present or not), a new table is created with the owner being the name specified in A_DB2_LOGIN.

Using USER_PATH with other variables such as A_DB2_CATALOG can speed up the finding of tables in large databases. It can also prevent an error 9D,14: "More than one table with the same name" thereby enabling access to tables with identical names, but with different schemes.

For example, If the USER_PATH and A_DB2_CATALOG are used, the form of the SQL statement will be modified from:

```
select COL1, ... from TABLENAME ...
```

to:

```
SELECT COL1, ... FROM [catalog.][username.]TABLENAME ...
```

where *catalog* and *username* will be filled in if provided.

See Also

A_DB2_LOGIN configuration variable

A_DB2_ALTERNATE_COMMIT_LOGIC configuration variable

F.6 Record and Table Locking

With Acu4GL for DB2, records are locked in a Wait mode. An application trying to read a locked record will hang until the record becomes available.

DB2 Universal Database products have the ability to time-out a lock and send an error return code to waiting applications. See your database administrator for additional information.

F.7 Limits and Ranges

The following limits exist for the DB2 protocol:

Maximum number of columns per key: 16
Maximum number of columns: 500
Maximum index size: 255
Maximum record size: 4005

Note that these limits may be further constrained by your database configuration. For example, the size of your page affects the maximum row size. Some limits are given here, but please refer to your database documentation for additional information.

Page Size	Maximum Row Size	Maximum Column Count	Maximum Index Size
4K	4,005	500	255
8K	8,101	1,012	
16K	16,293	1,012	
32K	32,677	1,012	

To achieve the same sort or retrieval sequence under DB2 as under the Vision file system, place a **BINARY** directive immediately before each key field that contains signed numeric data. High values and low values can cause problems in key fields. If you want data that uses the BINARY directive to be stored as hexadecimal encoded CHAR types, you can specify **A_DB2_USE_CHAR_FOR_BINARY** in the configuration file.

Other limits are described in Appendix B in Book 4, *Appendices*, of the ACUCOBOL-GT documentation.

F.8 Data Type Mapping

DB2 data types must be mapped to COBOL data types. In the table below, please note that directives have been applied to the COBOL data types. See **Chapter 4** for additional information on directives.

COBOL Data Type	**Acu4GL for DB2**
Date (**DATE** directive applied)	date
binary (**BINARY** directive applied)	char (2 * x)
varchar (**VAR_LENGTH** directive applied)	varchar
float	float
pic 9(4)	smallint
pic 9(9)	integer
pic 9(10)	decimal(10,0)
pic 9(x)v9(y)	decimal (x + y, y)
pic x(254)	char
pic x(255)	or greater must apply VAR_LENGTH directive

Mapping DB2 data types to COBOL data types

Sometimes developers are in a situation where they need to create a COBOL File Description based on an existing data source table. The most important thing to understand in this situation is that there is almost nothing that you can do wrong! When the Acu4GL product opens a data source table, the only thing it checks is that the column names match the COBOL data names.

When the Acu4GL product reads data from the data source, it essentially does a COBOL-style MOVE from the native data type to the COBOL data type, whatever it is. And since most types have a CHAR representation (in other words, you can actually display most data types, using a standard DB2-capable tool), using PIC X(nn) for each column will work perfectly well.

A better general rule is to use a COBOL type that closely matches the data source data type, but don't worry about getting an exact fit. So you can use PIC 9(9) whenever the data source has an INTEGER type.

If you have more information about the data source type, you might be able to use a different COBOL representation. For example, if you know that a particular column in an DB2 data source has values only in the range 0–999, you could use PIC 999 for your COBOL data. The COMP-type you use is really determined by your own preferences and should have little bearing on the COBOL data type you choose.

If you want to somehow choose your COBOL data types so that there is a best fit, you can use the following mapping:

Acu4GL for DB2	**COBOL Data Type**
date	Date (**DATE** directive applied)
varchar	varchar (**VAR_LENGTH** directive applied)
float	float
smallint	pic 9(4)
integer	pic 9(9)
decimal(10,0)	pic 9(10)
decimal (x + y, y)	pic 9(x)v9(y)
char	pic x(254)
pic x(255) or greater must apply VAR_LENGTH directive	pic x(255)

F.9 Runtime Errors

This section lists the Acu4GL error messages that could occur during execution of your program. **Chapter 9** provides information on compile-time errors and also provides several methods for retrieving runtime errors.

An explanation and a recommended recovery procedure follow each message.

Runtime errors will have this format:

9D,xx

The 9D indicates a file system error and is reported in your FILE STATUS variable. The *xx* is a secondary error code. These are the secondary errors reported directly from Acu4GL:

9D,01 Read error on dictionary file

An error occurred while reading the XFD file; this probably means the XFD is corrupt. Recompile with "-Fx" to re-create the dictionary. See **section 8.1** for information on compiler options.

9D,02 Corrupt dictionary file

The dictionary file for one of your COBOL files is corrupt and cannot be read. Recompile with "-Fx" to re-create the dictionary.

9D,03 Dictionary (.xfd) file not found

The dictionary file for one of your COBOL files cannot be located. Be sure you have specified the correct directory via your **XFD_PREFIX** runtime configuration file variable. You may need to recompile with "-Fx" to create the dictionary.

9D,04 Too many fields in the key

There are more than 16 fields in a key. Check your key definitions and restructure the key that is illegal, and then recompile with "-Fx".

9D,06 Mismatched dictionary file

The dictionary file (.xfd) for one of your files conflicts with the COBOL description of the file FD. The *xx* indicates a tertiary error code that is defined by the host file system. You can determine the exact nature of the mismatch by referring to the host system's error values.

The tertiary error code may have any of these values:

01 – mismatch found but exact cause unknown (this status is returned by the host file system)
02 – mismatch found in file's maximum record size
03 – mismatch found in file's minimum record size
04 – mismatch found in the number of keys in the file
05 – mismatch found in primary key description
06 – mismatch found in first alternate key description
07 – mismatch found in second alternate key description

The list continues in this manner for each alternate key.

9D,12 DB2 library function returned an unexpected error

One of the DB2 library functions returned an error that was not expected.

9D,13 Illegal size or type of data for variable xxx

An elementary data item in your FD was larger than 255 bytes, or there is no DB2 type that matches the current data type.

9D,14 More than one table with the same name

When tables were listed, more than one table was found with the same name. Consider whether or not setting the USER_PATH configuration variable will resolve the issue. This variable will enable tables with identical names, but with different ownership (located in a different schema) to be found and accessed.

F.10 Common Questions and Answers

This section contains some questions and answers specific to Acu4GL for DB2. Refer to **Chapter 10** for additional questions and answers that pertain to the Acu4GL family of products.

Question: I'm noticing some performance degradation when accessing my DB2 data source. What is the cause of this?

Answer: You may notice some performance impact if you were previously accessing Vision indexed files directly. This is because DB2 adds a software layer between your applications and your data sources. In return for minor performance impact, you can reap the benefits of database independence and enhanced portability. Overall performance depends on several factors, including your network configuration and your specific data source.

Question: Are there any ACUCOBOL-GT library routines that do not work with or would not make sense to use with Acu4GL for DB2?

Answer: Yes. The C$COPY and C$RENAME ACUCOBOL-GT library routines do not work with DB2.

Glossary of Terms

ANSI (American National Standards Institute)

American National Standards Institute. ANSI, along with the International Organization for Standards (ISO), standardized the C programming language, and continues to promote many other important software standards.

API (Application Programming Interface)

Application Programming Interface. The interface by which an application program accesses operating system and other services. An API is defined at source code level and provides a level of abstraction between the application and the kernel (or other privileged utilities) to ensure the portability of the code. An API can also provide an interface between a high level language and lower level utilities and services which were written without consideration for the calling conventions supported by compiled languages.

batch file

A text file containing operating system commands which are executed automatically by the command line interpreter. A batch file is called a "shell script" in UNIX, since it is the UNIX shell which includes the command line interpreter. Batch files can be used as a simple way to combine existing commands into new commands.

Binary Large Object (BLOB)

A large block of data stored in a database, such as an image or sound file. A BLOB has a structure that cannot be interpreted by a database management system but is known only by its size and location.

buffer

An internal memory area used for temporary storage of data records during input or output operations. Most programs keep track of changes in the buffer and then copy the buffer to a disk. Buffers are also used for printing.

client

A computer or program that can download files for manipulation from a server. *Contrast* server. *See also* **network**.

CPIO format (Copy In and Out format)

Copy In and Out format. A UNIX utility.

cursor

A prepared query, saved in a parameterized format.

data dictionaries

Extended file descriptors that map your application's SQL commands to COBOL records and vice versa. Based on the standard COBOL file descriptors (FDs). *See also* **file descriptor** *and* **XFD file**.

directives

Optional comments that you can place into a file descriptor in your COBOL source code to control how the data dictionary is built. *See also* **file descriptor**.

DSN (Data Source Name)

Data Source Name. An arbitrary name for every group of indexed or relative data files that your Windows application needs to access.

Extended File Descriptor

See **file descriptor** *and* **XFD file**.

field

An element of a database record in which one piece of information is stored. *Contrast* **record**.

file descriptor

An integer that identifies an open file within a process. This number is obtained as a result of opening a file. Operations that read, write, or close a file would take the file descriptor as an input parameter.

In many operating system implementations, file descriptors are small integers that index a table of open files. In UNIX, file descriptors "0", "1", and "2" correspond to the standard input, standard output, and standard error files respectively. *See also* **data dictionaries** *and* **XFD file**.

Gregorian calendar

The calendar as reformed by Pope Gregory XIII in 1582. This calendar adjusts the leap years to harmonize the civil year with the solar, as well as synchronize the time of Easter with the movable feasts by means of epochs. Every year divisible by 4, except those divisible by 100 and not by 400, has 366 days; all other years have 365 days. *Contrast* **Julian calendar**.

indexed file

In database design, a list of keys (or keywords), each of which identifies a unique record. Indexed files make it faster to find specific records and to sort records by the index field (the field used to identify each record). *See also* **key**.

I/O (Input/Output)

Input/Output. Communication between a computer and its users, its storage devices, other computers using a network or the outside world. The devices the computer uses to do this are called "peripherals."

Julian base date

The earliest Julian date to be used in defining Julian date formats. *See also* Julian calendar.

Julian calendar

The solar calendar introduced by Julius Caesar in Rome in 46 B.C., having a year of 12 months and 365 days and a leap year of 366 days every fourth year. It was eventually replaced by the Gregorian calendar. *Contrast* **Gregorian calendar**.

key

A value used to identify a record in a database, derived by applying some fixed function to the record. The key is often one of the fields (a column if the database is considered as a table with records being rows). Alternatively the key may be obtained by applying some function, for example, a hash function, to one or more of the fields. The set of keys for all records forms an index. Multiple indexes may be built for one database depending on how it is to be searched. *See also* **indexed file**.

network

Hardware and software data communication systems. Networks are often also classified according to their geographical extent: local area network (LAN), metropolitan area network (MAN), wide area network (WAN), and also according to the protocols used.

ODBC (Open Database Connectivity)

Open Database Connectivity. Released in 1992, a library of standardized data access functions that gives programmers a way to access and manipulate data in a variety of data sources. Interfaces include Visual Basic, Visual C++, SQL, and the ODBC driver pack contains drivers for the Access, Paradox, dBase, Text, Excel, and Btrieve databases. An application can submit statements to ODBC using the ODBC flavor of SQL, and ODBC then translates these to whatever flavor the database understands. ODBC is based on Call-Level Interface and was defined by the SQL Access Group.

query

A user's (or agent's) request for information, generally as a formal request to a database or search engine. SQL is the most common database query language. *See also* **SQL (Structured Query Language)**.

record

A group of related fields, or a single field, treated as a unit and comprising part of a file or data set, for purposes of input, processing, output, or storage by a computer. *Contrast* **field**.

relational database

A database system in which any database file can be a component of more than one of the database's tables. Relational database management systems may be referred to as RDBMSs.

server

A computer that makes services, as access to data files, programs, and peripheral devices, available to workstations on a network. *Contrast* client. *See also* **network**.

SQL (Structured Query Language)

Structured Query Language. According to specification, the language by which ODBC programs retrieve data. *See also* **query**.

TAR format (Tape Archive format)

A general purpose archive utility and file format used in UNIX environments. TAR was originally intended for use with magnetic tape but, though it has several command line options related to tape, it is now used more often for packaging files together on other media, for example, for distribution on the Internet. The resulting archive, a "tar file" is often compressed, using gzip or some other form of compression.

TCP/IP (Transfer Control Protocol/Internet Protocol)

A protocol for communication between computers, used as a standard for transmitting data over networks and as the basis for standard Internet protocols.

XFD file

Extended File Descriptors. Data dictionaries are also called extended file descriptors (XFDs) because they're based on the standard COBOL file descriptors (FDs). *See also* **data dictionaries** *and* **file descriptor**.

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X