Preface

The INFORMIX-SQL Reference and its companion volume, the INFORMIX-SQL User Guide, are your primary sources for learning to make the best use of the INFORMIX-SQL (I-SQL) relational database management system.

The INFORMIX-SQL Reference is a complete reference to the facilities that comprise I-SQL. It contains information about everything you can do with I-SQL, and it is organized by facility name. Once you have used the INFORMIX-SQL User Guide and are familiar with I-SQL basics, you can use the INFORMIX-SQL Reference to learn about advanced features and to quickly locate specific information.

Summary of Chapters

The INFORMIX-SQL Reference includes the following chapters and appendixes:

- This Preface provides general information about the manual and lists additional references to help you understand I-SQL concepts.
- The Introduction describes how I-SQL fits into the Informix family of products and manuals, explains the conventions used in the manual, and gives a brief introduction to the demonstration database and example files. The Introduction also lists the new features in I-SQL.
- Chapter 1, “The INFORMIX-SQL Main Menu,” explains how to use the I-SQL Main menu and describes what each option on the menu does.
- Chapter 2, “The FORMBUILD Transaction Form Generator,” focuses on FORMBUILD, supplying the information needed to build a screen form.
• Chapter 3, “The PERFORM Screen Transaction Processor,” considers each PERFORM menu option in detail, explaining how to enter, modify, remove, and retrieve data.
• Chapter 4, “The ACE Report Writer,” lists the formatting features you can use with ACE to prepare custom reports.
• Chapter 5, “User-Menu,” describes the User-menu option and provides the information needed to build a menu.
• Chapter 6, “C Functions in ACE and PERFORM,” shows you how to call C functions from ACE reports and PERFORM forms.
• Appendix A describes the stores demonstration database and provides the example forms and reports used in the INFORMIX-SQL User Guide and the INFORMIX-SQL Reference.
• Appendix B describes how to use environment variables to customize I-SQL features.
• Appendix C describes Native Language Support (NLS), and how to use it to customize I-SQL for a non-US-English environment.
• Appendix D discusses how to modify termcap and terminfo files in order to use special graphics characters in the FORMBUILD transaction processor.
• Appendix E is an ASCII chart.
• Appendix F lists reserved words for all Informix products.
• Appendix G lists the structure of the system catalogs.
• Appendix H demonstrates how to access each I-SQL program from the command line.
• Appendix I explains the special consideration you should be aware of when accessing an INFORMIX-OnLine Dynamic Server database with I-SQL.
• The Index includes entries for both the INFORMIX-SQL User Guide and the INFORMIX-SQL Reference.
Informix Welcomes Your Comments

A reader-response card is provided with this manual. Please use this card to tell us what you like or dislike about this manual. To help us with future versions of this manual, please tell us about any corrections or clarifications that you would find useful. Return this card to:

Informix Software, Inc.
Technical Publications Department
4100 Bohannon Drive
Menlo Park, CA 94025

If you prefer to share your comments on-line, address your e-mail to:
doc@informix.com

Related Reading

If you have no prior experience with database management, you should refer to the Informix Guide to SQL: Tutorial, Version 4.1. This manual is provided with all Informix database servers.

For additional technical information on database management, consult the following texts by C. J. Date:


This guide assumes that you are familiar with the UNIX operating system. If you have limited UNIX experience, you might want to look at your operating system manual or a good introductory text before you read this manual.

Some suggested texts about UNIX systems follow:

- *UNIX for People* by P. Birns, P. Brown, and J. Muster (Prentice-Hall, 1985)
If you are interested in learning more about the SQL language, consider the following text:

Table of Contents

Introduction
Documentation Included with INFORMIX-SQL 4
Other Useful Documentation 4
Conventions of this Manual 5
  Typographical Conventions 5
  Syntax Conventions 5
  Useful On-Line Files 9
On-Line Error Messages 10
The STORES Demonstration Application and Database 11
New Features in INFORMIX-SQL 11
  NLS Support 11
  Improved Performance 12
  Improved Quality 12
Compliance with Industry Standards 12

Chapter 1 The INFORMIX-SQL Main Menu
Accessing INFORMIX-SQL 1-3
The INFORMIX-SQL Screens 1-3
  Menu Screens 1-3
  Text-Entry Screens 1-4
Maps of the Menu Structure 1-5
The INFORMIX-SQL Main Menu Options 1-7

Chapter 2 The FORMBUILD Transaction Form Generator
Chapter Overview 2-3
  PERFORM Error Messages 2-3
  Demonstration Database Sample Forms 2-3
Creating and Compiling a Custom Form 2-4
  Using the Menu System to Create a Form 2-4
  Using the Operating System to Create a Form 2-5
Chapter 3  The PERFORM Screen Transaction Processor

Structure of a Form Specification File  2-6
DATABASE Section  2-9
SCREEN Section  2-10
TABLES Section  2-18
ATTRIBUTES Section  2-20
   Display Field Order  2-20
   Table Order  2-21
   Display-Only Fields  2-24
   Joining Columns  2-26
   ATTRIBUTES Syntax  2-28
   Related Attribute  2-37
INSTRUCTIONS Section  2-56
   BEFORE  2-63
   AFTER  2-64
The SAMPLE Form Specification File  2-82
   The CUSTOMER INFORMATION Screen  2-84
   The ORDER INFORMATION Screen  2-84
The PERFORM Screen  3-5
   The Information Lines  3-6
   The Screen Form  3-8
   Status Lines  3-9
   Running Operating System Commands from PERFORM  3-9
Entering Data  3-9
Data Types  3-9
Special Functions  3-12
Positioning the Cursor  3-12
Field Editing  3-13
Using the Multiline Editor  3-14
Display Field Order  3-15
Data Checking  3-15
   User Access Privileges  3-16
The Current List  3-18
Menu Options  3-18
<table>
<thead>
<tr>
<th>Chapter 4</th>
<th>The ACE Report Writer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter Overview</td>
<td>4-5</td>
</tr>
<tr>
<td>Creating and Compiling a Custom Report</td>
<td>4-5</td>
</tr>
<tr>
<td>Using the Menus to Create a Report</td>
<td>4-5</td>
</tr>
<tr>
<td>Creating a Report from the Command Line</td>
<td>4-7</td>
</tr>
<tr>
<td>Information About ACE</td>
<td>4-9</td>
</tr>
<tr>
<td>ACE Filename Conventions</td>
<td>4-9</td>
</tr>
<tr>
<td>Owner Naming</td>
<td>4-9</td>
</tr>
<tr>
<td>Using Expressions in a Report Specification</td>
<td>4-10</td>
</tr>
<tr>
<td>ACE Error Messages</td>
<td>4-11</td>
</tr>
<tr>
<td>Demonstration Database Sample Reports</td>
<td>4-12</td>
</tr>
<tr>
<td>Structure of a Report Specification File</td>
<td>4-12</td>
</tr>
<tr>
<td>DATABASE Section</td>
<td>4-13</td>
</tr>
<tr>
<td>DEFINE Section</td>
<td>4-14</td>
</tr>
<tr>
<td>INPUT Section</td>
<td>4-20</td>
</tr>
<tr>
<td>OUTPUT Section</td>
<td>4-22</td>
</tr>
<tr>
<td>SELECT Section</td>
<td>4-30</td>
</tr>
<tr>
<td>READ Section</td>
<td>4-33</td>
</tr>
<tr>
<td>FORMAT Section</td>
<td>4-36</td>
</tr>
<tr>
<td>Control Blocks</td>
<td>4-41</td>
</tr>
<tr>
<td>Statements</td>
<td>4-55</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 5</th>
<th>User-Menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter Overview</td>
<td>5-3</td>
</tr>
<tr>
<td>Accessing a Menu</td>
<td>5-4</td>
</tr>
<tr>
<td>Using a Menu Within INFORMIX-SQL</td>
<td>5-4</td>
</tr>
<tr>
<td>Designing a Menu</td>
<td>5-5</td>
</tr>
<tr>
<td>Creating a Menu</td>
<td>5-8</td>
</tr>
<tr>
<td>Accessing PERFORM with the menuform Form</td>
<td>5-9</td>
</tr>
<tr>
<td>Entering Menu Data</td>
<td>5-10</td>
</tr>
<tr>
<td>Steps for Entering Your Own Data</td>
<td>5-13</td>
</tr>
<tr>
<td>Modifying a Menu</td>
<td>5-14</td>
</tr>
<tr>
<td>Menu Display Fields</td>
<td>5-15</td>
</tr>
<tr>
<td>Creating a Script Menu</td>
<td>5-24</td>
</tr>
</tbody>
</table>
Chapter 6  C Functions in ACE and PERFORM

Chapter Overview  6-3
Calling C Functions from ACE  6-4
   Calling C Functions  6-6
   Compiling the Report Specification  6-8
Calling C Functions from PERFORM  6-9
   Calling C Functions in the INSTRUCTIONS Section  6-9
   ON BEGINNING and ON ENDING Control Blocks  6-12
   Compiling the Form Specification  6-14
Writing the C Program  6-14
   Organizing the C Program  6-15
   Passing Values to a C Function  6-18
   Returning Values to ACE and PERFORM  6-21
PERFORM Library Functions  6-22
Compiling, Linking, and Running Reports and Forms  6-32
   Syntax of the cace and cperf programs  6-32
   Use of cace and cperf  6-33
Examples  6-33
   ACE Example 1  6-33
   ACE Example 2  6-35
   PERFORM Example  6-37

Appendix A  The Demonstration Database and Application
Appendix B  Environment Variables
Appendix C  Native Language Support Within INFORMIX-SQL
Appendix D  Modifying termcap and terminfo
Appendix E  The ASCII Character Set
Appendix F  Reserved Words
Appendix G  System Catalogs
Appendix H  Accessing Programs from the Operating System
Appendix I  Using the INFORMIX-OnLine Dynamic Server

Index
Introduction

Documentation Included with INFORMIX-SQL  4
Other Useful Documentation  4
Conventions of this Manual  5
  Typographical Conventions  5
  Syntax Conventions  5
  Useful On-Line Files  9
On-Line Error Messages  10
The STORES Demonstration Application and Database  11
New Features in INFORMIX-SQL  11
  NLS Support  11
  Improved Performance  12
  Improved Quality  12
Compliance with Industry Standards  12
Introduction
INFORMIX-SQL (I-SQL) is a computer-based record-keeping system. As a database management system, I-SQL consists of useful programs or modules that perform data management tasks. I-SQL can substantially reduce the amount of time required to organize, store, and retrieve information. It can summarize, group, and format information in a variety of helpful ways. With I-SQL, you can perform these database management tasks:

- Create, modify, and drop databases and tables
- Load data from operating system files
- Run queries using an interactive query language
- Insert, delete, update, and query on data in the database
- Create and drop privileges and indexes
- Create and compile custom forms or reports
- Create and run custom menus
Documentation Included with INFORMIX-SQL

The I-SQL documentation set includes the following manuals:

<table>
<thead>
<tr>
<th>Manual</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFORMIX-SQL Reference</td>
<td>A complete reference to the programs that make up I-SQL. It contains information about everything you can do with I-SQL and is organized by program name. Once you have used the INFORMIX-SQL User Guide and are familiar with I-SQL basics, you can use the INFORMIX-SQL Reference to learn about advanced features and to quickly locate specific information.</td>
</tr>
<tr>
<td>INFORMIX-SQL User Guide</td>
<td>Introduces I-SQL and provides the context needed to understand the other manuals in the documentation set. You do not need database management experience or familiarity with basic database management concepts to use this manual. It includes general information about database systems and leads you through the steps necessary to create a database, enter and access database information, and produce printed reports.</td>
</tr>
<tr>
<td>Informix Guide to SQL: Tutorial, Version 4.1</td>
<td>Provides a tutorial on SQL as it is implemented by Informix products, and describes the fundamental ideas and terminology that are used when planning and implementing a relational database. It also describes how to retrieve information from a database, and how to modify a database.</td>
</tr>
<tr>
<td>Informix Guide to SQL: Reference, Version 4.1</td>
<td>Provides full information on the structure and contents of the demonstration database that is provided with I-SQL. It includes details of the Informix system catalog tables, describes Informix and common environment variables that should be set, and describes the column data types that are supported by Informix database engines. It also provides a detailed description of all of the SQL statements that Informix products support.</td>
</tr>
<tr>
<td>Informix Guide to SQL: Syntax, Version 6.0</td>
<td>Contains syntax diagrams for all of the SQL statements and statement segments that are supported by the 6.0 server.</td>
</tr>
<tr>
<td>Informix Error Messages, Version 6.0</td>
<td>Provides error messages organized by error number. When an error occurs you can look it up by number and learn its cause and solution.</td>
</tr>
</tbody>
</table>

Other Useful Documentation

Depending on the database server that you are using, you or your system administrator need either the INFORMIX-OnLine Dynamic Server Administrator’s Guide, Version 6.0 or the INFORMIX-SE Administrator’s Guide, Version 6.0.
Conventions of this Manual

This manual assumes that you are using I-SQL with INFORMIX-SE. Features and behavior specific to INFORMIX-OnLine Dynamic Server are noted throughout the manual with the icon and discussed fully in Appendix I. You should encounter no differences from the default behavior described in this manual when working with a remote INFORMIX-SE database server.

Typographical Conventions

Informix product manuals use a standard set of conventions to introduce new terms, illustrate screen displays, describe command syntax, and so forth. The following typographical conventions are used throughout this manual:

- **KEYWORD** All keywords appear in UPPERCASE letters. (You can in fact enter keywords in either uppercase or lowercase letters.)
- **italics** New terms and emphasized words are printed in italics. Italics also mark syntax terms for which you must specify some valid identifier, expression, keyword, or statement.
- **boldface** Database names, table names, column names, utilities, filenames, pathnames, command-line specifications, and other similar terms are printed in boldface.
- **monospace** Information that I-SQL displays and information that you enter are printed in this typeface.

Additionally, when you are instructed to “enter” or “execute” text, immediately press RETURN after the entry. When you are instructed to “type” the text, no RETURN is required. Note that the word RETURN is used throughout this document set rather than ENTER, although your keyboard may perform this function with a key labeled ENTER.

Syntax Conventions

SQL statement syntax is described in the Informix Guide to SQL: Syntax, Version 6.0. The syntax of form specification files is described in Chapter 2 of this manual. The syntax of report specification files is described in Chapter 4. The syntax of various command line options is documented in various appendices.
Syntax diagram conventions are described in this section. Each diagram displays the sequences of required and optional keywords, terms, and symbols that are valid in a given statement, command line, or other specification, as in the following diagram of the DATABASE statement of ACE.

The following are the three most important rules to remember regarding terms that appear in the syntax diagrams of this book:

- For ease of identification, all I-SQL keywords (like DATABASE) are shown in UPPERCASE characters, even though you can enter them in lowercase.
- Terms for which you must supply specific values or names are in italics. In this example, database-name must be replaced by an identifier.
- All punctuation and other non-alphabetic characters are literal symbols.

Syntax elements in a path represent terms, keywords, symbols, and segments that can appear in your statement. Except for separators in loops, which the path approaches counter-clockwise from the right, the path always approaches elements from the left, and continues to the right. Unless otherwise noted, at least one blank character separates syntax elements.

You may encounter one or more of the following elements on a path:

- **KEYWORD**: Spell any word in UPPERCASE letters exactly as shown (but you can type it in either uppercase or lowercase letters).
- **( . ; @ ± * / )**: Punctuation and other non-alphabetic characters are literal symbols that you must enter exactly as shown.
- **" "**: Double quotes must be entered as shown. If you prefer, you can replace the pair of double quotes with a pair of single quotes, but you cannot mix double and single quotes.
- **variable**: A word in italics represents a term that you must supply. An explanation below the diagram identifies what values, identifiers, or keywords you can substitute for the italicized term.

A term in a rectangle represents a subdiagram on the same page (if no page number is supplied) or on a specified page, as if the subdiagram were spliced into the diagram at this point. (Here “segment” and “subdiagram” are synonyms.) The aspect ratio is not significant. That is, the same segment...
can be represented by rectangles of different shapes, as in these symbols for the ATTRIBUTES Section segment.

A reference to SQL:R in a syntax diagram represents an SQL statement or segment that is described in the Informix Guide to SQL: Reference, Version 4.1. Imagine that the segment were spliced into the diagram at this point.

An icon is a warning that this path is valid only for some products, or only under certain conditions. Symbols on the icons indicate what products or conditions support the path.

These icons that appear in the Informix Guide to SQL: Reference, Version 4.1 can also appear in an I-SQL syntax diagram:

- **SE** This path is valid only for INFORMIX-SE.
- **OL** This path is valid only for INFORMIX-OnLine Dynamic Server.

A shaded option is the default. If you do not specify any of the available options, then by default, this option is in effect.

Syntax enclosed between a pair of arrows is a subdiagram.

The vertical line is a terminator. This only appears at the right, indicating that the syntax diagram is complete.

A branch below the main path indicates an optional path. (Any term on the main path is required, unless a branch can circumvent it.)

A set of multiple branches indicates a syntax context where a choice among more than two different paths is available.

A loop indicates a path that can be repeated. Punctuation along the top of the loop indicates the separator symbol for list items, as in this example. If no symbol appears, a blank space is the separator.

A gate (__) on a path indicates that you can only use that path the indicated number of times, even if it is part of a larger loop. Here BEFORE CONSTRUCT can be specified no more than once within this statement segment.
Icons that appear in the left margin of the text indicate that the accompanying shaded text is valid only for some products or under certain conditions. In addition to the icons described in the preceding list, you may encounter the following icon in the left margin:

- **ANSI**
  This icon indicates that the functionality described in the shaded text is valid only if your database is ANSI-compliant.

- **NLS**
  This icon indicates that the functionality described in the shaded text is valid only if NLS features are active, as described in Appendix C, “Native Language Support Within INFORMIX-SQL.”

The following diagram shows the elements of a form specification file in the syntax notation that has been described above:

---

**Figure 1** Elements of a syntax diagram

---

8 Introduction
To construct a similar form specification, start at the top left with the DATABASE section. Then follow the diagram to the right, including the elements that you want. This diagram conveys the following information:

1. You must include the DATABASE section.
2. You must include the SCREEN section at least once, but additional SCREEN sections are allowed.
3. A TABLES section must follow the SCREEN section or sections. An ATTRIBUTES section must follow the TABLES section.
4. An INSTRUCTIONS section, following the ATTRIBUTES section, is optional.
5. Each subdiagram rectangle refers to a diagram elsewhere that contains the syntax for that portion of the main diagram. The subdiagram for the ATTRIBUTES section appears below the main diagram for illustration purposes. To create an ATTRIBUTES section in the form specification, you do the following:
   a. Type in the keyword ATTRIBUTES.
   b. Type in one or more field-tag = expressions. Each of these expressions contains a value of the variable field-tag, followed by an equals sign, followed by a field description or a display-only field, and terminated with a semicolon. One field tag may be equated to multiple field descriptions and display-only fields.
   c. When you reach the segment indicator at the end of the subdiagram, you return to the main diagram.
6. When you reach the terminator, the form specification is complete.

Note: When you are instructed to “enter” characters or to “execute” a command, immediately press RETURN after the entry. When you are instructed to “type” the text or to “press” other keys, no RETURN is required.

Useful On-Line Files

In addition to the Informix set of manuals, the following on-line files, located in the $INFORMIXDIR/release directory, may supplement the information in the INFORMIX-SQL User Guide and Reference Manual:

**Documentation Notes** describe feature and performance topics not covered in the manual or which have been modified since publication. The file containing the Documentation Notes for I-SQL is called ISQLDAC_6.0.
Release Notes  describe performance differences from earlier versions of Informix products and how these differences may affect current products. The file containing the Release Notes for I-SQL and other products is called TOOLS.6.0.

Please examine these files because they contain important information about application and performance issues.

I-SQL provides on-line Help; invoke Help by pressing CONTROL-W.

On-Line Error Messages

Use the finderr script to display a particular error message or messages on your terminal screen. The script is located in the $INFORMIXDIR/bin directory.

The finderr script has the following syntax:

```
finderr [-]msg_num
```

- **msg_num** Indicates the number of the error message to display. Error messages range from -1 to -32000. Specifying the - sign is optional.

You can specify up to 16 error messages per finderr command. finderr copies all the specified messages to standard output.

For example, to display the -359 error message, you can enter either of the following:

```
finderr -359
```

or, equivalently:

```
finderr 359
```

The following example demonstrates how to specify a list of error messages. The example also pipes the output to the UNIX more command to control the display. You can also direct the output to another file so that you can save or print the error messages:

```
finderr 233 107 113 134 143 144 154 | more
```

A few messages have positive numbers. These messages are used solely within the application tools. In the unlikely event that you want to display them, you must precede the message number with the + sign.
The messages numbered -1 to -100 can be platform-dependent. If the message text for a message in this range does not apply to your platform, check the operating system’s documentation for the precise meaning of the message number.

The STOR ES Demonstration Application and Database

Your server software includes a database called the stores demonstration database that contains information about a fictitious wholesale sporting-goods distributor. The sample command files that make up a demonstration application are included as well.

Most of the examples in this manual are based on the stores demonstration database. The stores demonstrations database is described in detail and its contents are listed in Appendix A, “The Demonstration Database and Application.” The appendix also describes the procedure for creating and restoring the demonstration database, and provides a complete set of example screen forms and report specifications.

New Features in INFORMIX-SQL

Several new features have been added or modified for version 6.0 of I-SQL. This version provides support for developers working in European countries, improved performance, and improved quality.

NLS Support

Native Language Support (NLS) is supplied to meet the needs of European countries. This feature extends the ASCII character set from 128 to 256 characters. These additional characters allow you to include characters such as Ö and ç in the definition of your database and in ACE and PERFORM. NLS also provides character sorting and comparison specific to particular languages, and region-specific monetary and numeric formatting. To use NLS, you need to set environment variables, as described in Appendix C, “Native Language Support Within INFORMIX-SQL.”
Improved Performance

The removal of the relay module in the 6.0 engine results in improved speed in which data can be retrieved from and sent to a database. As a result, the performance of your I-SQL screens and reports that access a database should improve.

Improved Quality

Over 100 bug fixes have been made to this version of the product. Also, the documentation set has been completely reorganized, rewritten, and updated to include all 6.0 I-SQL features.

Compliance with Industry Standards

The American National Standards Institute (ANSI) has established a set of industry standards for SQL. Informix SQL-based products are compliant with ANSI Level 1 and ANSI Level 2 (published as ANSI X3.135-1989) on the INFORMIX-SE database server with the following two exceptions:

- No support for effective checking of unique constraints
- No support for repeatable reads
The INFORMIX-SQL Main Menu

Accessing INFORMIX-SQL 3
The INFORMIX-SQL Screens 3
  Menu Screens 3
    Selecting Options 4
    Exiting the Menu 4
    Asking for Help 4
  Text-Entry Screens 4
    Entering Text 5
    Exiting a Text-Entry Screen 5
    Asking for Help 5
Maps of the Menu Structure 5
The INFORMIX-SQL Main Menu Options 7
  DATABASE 8
  EXIT 10
  FORM 11
  QUERY-LANGUAGE 13
  REPORT 15
  TABLE 17
  USER-MENU 18
The INFORMIX-SQL Main Menu
Accessing INFORMIX-SQL

To begin working with INFORMIX-SQL (I-SQL), enter isql at the operating system prompt. At this point, I-SQL displays the Main menu:

```
INFORMIX-SQL: Form Report Query-language User-menu Database Table Exit
Run, Modify, Create, or Drop a form.
-----------------------------------------------------Press CONTROL-W for Help ----
```

The INFORMIX-SQL Screens

The I-SQL menu system uses two kinds of screens: a menu screen, like the I-SQL Main menu, and a text-entry screen.

Menu Screens

The top line of a menu screen lists your options. One option is always highlighted. The second line gives a brief description of the highlighted option. Each time you press the SPACEBAR, the highlight moves to the next option and the description changes. You can also use the [→] and [←] keys to move the highlight. The fourth line displays the name of the current database and the following message:

```
Press CONTROL-W for Help
```
Selecting Options
You can normally select menu options in two ways:

- Use the SPACEBAR to move the highlight over the option you want to choose and press RETURN.
- Type the first letter of the option you want to select. Case is not important—you can type t or T to select the Table option.

I-SQL displays the screen for the menu option you have selected.

Exiting the Menu
Each menu has an Exit option. When you want to leave a menu screen, type e for Exit. I-SQL displays the previous menu or screen.

Asking for Help
The CONTROL-W key displays the help message appropriate for each part of I-SQL. When you have finished reading the message displayed on the HELP screen, press RETURN. I-SQL redisplays the screen you were working with before you called for help.

Text-Entry Screens
The text-entry screen is the second kind of screen. It requires that you enter text instead of choosing a menu option. The top line of the screen displays the screen name, followed by double angle (>>) brackets. The second line gives directions.

The RUN FORM screen is an example of a text-entry screen. Some of the items it includes follow:

RUN FORM >> Run Form
Choose a form with the Arrow Keys, or enter a name, then press RETURN.

-------------------- stores2 ------------- Press CONTROL-W for Help -----
customer
orderform
sample
Maps of the Menu Structure

Entering Text

Whatever you type appears after the double angle brackets at the top of the screen. Press the RETURN key when you are finished typing. Some screens, like the RUN FORM screen, give you the option of selecting an item from a list on the lower part of the screen instead of typing your selection. Use the Arrow keys to position the highlight over the item you want, and then press RETURN. I-SQL displays the next screen.

Exiting a Text-Entry Screen

Text-entry screens do not have an Exit option. Press CONTROL-C and I-SQL redisplays the previous menu or screen.

Asking for Help

The CONTROL-W key works with text-entry screens exactly as it does with menu screens. When you are finished reading the Help message, press RETURN. I-SQL redisplays the screen you were working with before you called for help.

Maps of the Menu Structure

The I-SQL Main menu has seven options: Form, Report, Query-language, User-menu, Database, Table, and Exit. Each option on the Main menu calls a submenu, which displays options that allow you to work with a part of I-SQL. The I-SQL menu structure is displayed in Figure 1-1 on page 1-6 and Figure 1-2 on page 1-7.

Figure 1-1 on page 1-6 is a map of the I-SQL menu hierarchy. This figure illustrates the options on each of the submenus available from the Main menu.
Maps of the Menu Structure

Figure 1-1  The INFORMIX-SQL menu hierarchy
Figure 1-2 is a functional guide to the I-SQL menu system. Menu options are grouped according to activity or task.

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>MENU OPTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use it:</td>
<td>Run</td>
</tr>
<tr>
<td></td>
<td>Run</td>
</tr>
<tr>
<td></td>
<td>Run</td>
</tr>
<tr>
<td></td>
<td>Run</td>
</tr>
<tr>
<td></td>
<td>Select</td>
</tr>
<tr>
<td>Modify it:</td>
<td>Modify</td>
</tr>
<tr>
<td></td>
<td>Modify</td>
</tr>
<tr>
<td></td>
<td>Modify</td>
</tr>
<tr>
<td></td>
<td>Alter</td>
</tr>
<tr>
<td>Create it: Default</td>
<td>Generate</td>
</tr>
<tr>
<td></td>
<td>Generate</td>
</tr>
<tr>
<td>Custom</td>
<td>New</td>
</tr>
<tr>
<td></td>
<td>New</td>
</tr>
<tr>
<td></td>
<td>Use-editor</td>
</tr>
<tr>
<td>Compile it:</td>
<td>Compile</td>
</tr>
<tr>
<td></td>
<td>Compile</td>
</tr>
<tr>
<td>Special Tasks:</td>
<td>Info</td>
</tr>
<tr>
<td></td>
<td>Choose</td>
</tr>
<tr>
<td></td>
<td>Output</td>
</tr>
<tr>
<td></td>
<td>Save</td>
</tr>
<tr>
<td>Drop it:</td>
<td>Drop</td>
</tr>
<tr>
<td></td>
<td>Drop</td>
</tr>
<tr>
<td></td>
<td>Drop</td>
</tr>
<tr>
<td></td>
<td>Drop</td>
</tr>
<tr>
<td></td>
<td>Drop</td>
</tr>
<tr>
<td>Exit:</td>
<td>Exit</td>
</tr>
<tr>
<td></td>
<td>Exit</td>
</tr>
<tr>
<td></td>
<td>Exit</td>
</tr>
<tr>
<td></td>
<td>Exit</td>
</tr>
</tbody>
</table>

Figure 1-2  A functional guide to the menu hierarchy

The INFORMIX-SQL Main Menu Options

The following sections discuss the options on the I-SQL Main menu. The options are presented in alphabetical order.
DATABASE

Use the Database option to create a new database, make an existing database current, or drop an existing database (see Figure 1-3).

![Database Menu](image)

Figure 1-3 The DATABASE menu

Usage

The DATABASE menu displays four options:

- **Select**: Makes a database the current database.
- **Create**: Creates a new database and makes that database the current database.
- **Drop**: Removes a database from the system.
- **Exit**: Exits the DATABASE menu and returns to the I-SQL Main menu.

- When you create a database with the Create option, that database becomes the current database.
- When you use the Select option, you can type the name of an existing database rather than highlight one of the database names listed on your screen. If you do so, you must enter the name of a database located in the current directory or a directory specified in your DBPATH environment variable. If you enter the name of a nonexistent database or a database that I-SQL cannot locate, I-SQL displays the following messages:
  - 329: Database not found or no system permission.
  - 2: No such file or directory
- Be very careful when you drop a database; all data in the database is permanently discarded.
• Chapter 2, “Creating a Database,” and Chapter 9, “Database Structure and Integrity,” in the INFORMIX-SQL User Guide describe the use of options on the DATABASE menu.

• Chapter 6 of Informix Guide to SQL: Reference, Version 4.1, explains the workings of all SQL database statements supported by Informix products.

• When using the Query-Language option, you are not allowed to drop the current database. You must explicitly close it first with the CLOSE DATABASE statement. For details about the CLOSE DATABASE statement, see Chapter 6 of the Informix Guide to SQL: Reference, Version 4.1.

For information on how creating or accessing an NLS database differs from creating or accessing a non-NLS database, refer to Appendix C, “Native Language Support Within INFORMIX-SQL.”
Use the Exit option to leave the I-SQL Main menu and return to the operating system.

To exit from the Main menu:
1. From the I-SQL Main menu, type e to select the Exit option.
2. You leave the I-SQL Main menu and return to the operating system.
FORM

Use the Form option to run a screen form, create or modify a screen form, compile a screen form, or drop an existing screen form (see Figure 1-4).

Figure 1-4  The FORM menu

Usage

The FORM menu displays seven options:

- **Run** runs a previously compiled screen form.
- **Modify** modifies a screen form specification.
- **Generate** creates a default screen form.
- **New** creates a custom screen form specification.
- **Compile** compiles a screen form specification.
- **Drop** drops a screen form.
- **Exit** exits the FORM menu and returns to the I-SQL Main menu.

- After you edit a form specification file (with the New or Modify options on the FORM menu), you must compile it. (You cannot use the form in I-SQL until it has been compiled.) Menus allowing you to compile an edited form are displayed when you select the New or Modify options. You can also use the Compile option on the FORM menu to compile a form specification.

I-SQL notifies you if errors are in the form specification. Follow the directions on the screen to correct and recompile the form. You can save or discard the form after compilation. You can also save an uncompiled form to work on at a later time or discard it completely.

- Chapter 3, “Entering Data,” Chapter 4, “Querying a Database,” Chapter 5, “Using Multiple-Table Forms,” and Chapter 6, “Creating Your
Own Forms,” in the INFORMIX-SQL User Guide describe how to create and use screen forms.

- The I-SQL program that compiles a form specification is called FORMBUILD. See Chapter 2, “The FORMBUILD Transaction Form Generator,” for information about FORMBUILD.

- The I-SQL program that runs a screen form is called PERFORM. See Chapter 3, “The PERFORM Screen Transaction Processor,” for information about PERFORM.
QUERY-LANGUAGE

Select the Query-language option to use the SQL query language.

Enter new SQL statements using SQL editor.

Figure 1-5 The SQL menu

Usage

The SQL menu displays ten options:

New       allows you to enter new SQL statements using the SQL editor.
Run        executes the current SQL statement or statements.
Modify     allows you to use the SQL editor to modify the current SQL statement or statements.
Use-editor allows you to enter or edit SQL statements with a system editor.
Output     routes the output from executing the current SQL statements to a system file, a printer, or a system pipe.
Choose     allows you to select an existing command file containing SQL statements and make them your current statements. You can run or edit the current statements.
Save       saves the current SQL statements in a command file. You can use this command file later by selecting the Choose option on the SQL menu.
Info       allows you to retrieve information about the columns, indexes, privileges, and status of a table.
Drop       drops a command file from the database.
Exit       exits the SQL menu and returns to the I-SQL Main menu.
• If there is no current database, I-SQL displays the CHOOSE DATABASE screen after you select the Query-language option on the I-SQL Main menu.
• Chapter 7, “Using SQL,” in the INFORMIX-SQL User Guide describes how to use the SQL menu and how to create and run SQL statements.
• Chapter 6 of the Informix Guide to SQL: Reference, Version 4.1 explains the workings of all SQL database statements supported by Informix products.
REPORT

Use the Report option to run a report, create or modify a report, compile a report, or drop an existing report from the database (see Figure 1-6).

![The REPORT menu](image)

**Usage**

The REPORT menu displays seven options:

- **Run** runs a report.
- **Modify** modifies a report specification.
- **Generate** creates a default report specification.
- **New** creates a custom report specification.
- **Compile** compiles a report specification.
- **Drop** drops a report specification from the database.
- **Exit** exits the REPORT menu and returns to the I-SQL Main menu.

- After you edit a report specification file (with the New or Modify options on the REPORT menu), you must compile it. (You cannot use the report in I-SQL until it has been compiled.) Menus allowing you to compile an edited report are displayed when you select the New or Modify options. You can also use the Compile option on the REPORT menu to compile a report specification.

  - **I-SQL** notifies you if there are errors in the report specification. Follow the directions on the screen to correct and recompile the report. You can save or discard the report after compilation. You can also save an uncompiled report to work on later or discard the report completely.

- Chapter 9, “Creating and Printing Reports,” in the INFORMIX-SQL User Guide describes how to create and use reports.
The I-SQL program that compiles a report specification is called ACEPREP. The I-SQL program that runs a report specification is called ACEGO. See Chapter 4, “The ACE Report Writer,” for complete information about these programs.
TABLE

Use the Table option to create or modify a table, retrieve information about a table, or drop a table from the database (see Figure 1-7).

Figure 1-7 The TABLE menu

Usage

The TABLE menu displays five options:

Create allows you to use the interactive schema editor to create a new table.

Alter allows you to modify a table using the interactive schema editor.

Info retrieves information about the structure of a table.

Drop deletes a table from the database.

Exit exits the TABLE menu and returns to the I-SQL Main menu.

• If there is no current database, the CHOOSE DATABASE screen appears after you select the Table option.

• Be very careful when you drop a table; you lose all the data in the table.

• Chapter 2, “Creating a Database,” and Chapter 9, “Database Structure and Integrity,” in the INFORMIX-SQL User Guide describe the use of options on the TABLE menu.

• Chapter 6 of the Informix Guide to SQL: Reference, Version 4.1 explains the workings of all SQL database statements supported by Informix products.
USER-MENU

Use the User-menu option to run a user-created menu, create a user-menu, or modify an existing user-menu (see Figure 1-8).

Figure 1-8 The USER-MENU menu

Usage

The USER-MENU menu displays three options:

- **Run** runs the user-menu for the current database.
- **Modify** allows you to create or modify a user-menu.
- **Exit** exits the USER-MENU and returns to the I-SQL Main menu.

- After you select the User-menu option, the CHOOSE DATABASE screen appears if there is no current database.
- Use the Modify option to both create and modify a user-menu.
- If there is no user-menu for the current database, I-SQL displays a message notifying the user when the Run or Modify option is selected.
- See Chapter 5, “User-Menu,” for complete information about creating, modifying, and using a menu.
The FORMBUILD Transaction Form Generator

Chapter Overview 3
PERFORM Error Messages 3
Demonstration Database Sample Forms 3

Creating and Compiling a Custom Form 4
Using the Menu System to Create a Form 4
Using the Operating System to Create a Form 5

Structure of a Form Specification File 6
DATABASE Section 9

SCREEN Section 10
Page Layout 12
Graphics Characters in Forms 15
Required Terminal Entries 17

TABLES Section 18

ATTRIBUTES Section 20
Display Field Order 20
Table Order 21
Display-Only Fields 24
Joining Columns 26
Verify Joins 27
ATTRIBUTES Syntax 28
AUTONEXT 29
COLOR 30
COMMENTS 33
DEFAULT 34
DOWNSHIFT  35
FORMAT  36
Related Attribute  37
INCLUDE  38
INVISIBLE  40
LOOKUP  41
NOENTRY  43
NOUPDATE  44
PICTURE  45
QUERYSHIFT  51
QUERYCLEAR  47
REQUIRED  48
REVERSE  49
RIGHT  50
UPSHIFT  51
VERIFY  52
WORDWRAP  53
ZEROFILL  55

INSTRUCTIONS Section  56
COMPOSITES  57
DELMITERS  59
MASTER OF  60
Control Blocks  62
BEFORE  63
AFTER  64
EDITADD and EDITUPDATE  65
ADD  67
UPDATE  68
QUERY  69
REMOVE  70
DISPLAY  71
Action Syntax  72
ABORT  73
LET  74
NEXTFIELD  77
COMMENTS  79
IF-THEN-ELSE  80

The SAMPLE Form Specification File  82
The CUSTOMER INFORMATION Screen  84
The ORDER INFORMATION Screen  84
Chapter Overview

Before you can use PERFORM with a customized screen form (see Chapter 3, “The PERFORM Screen Transaction Processor”), you must first use FORMBUILD to compile a form specification file. The form specification file contains the screen format and the instructions to PERFORM about how to display the data.

PERFORM Error Messages

The text of all error messages, along with suggestions for corrections, is included in Informix Error Messages, Version 6.0.

Demonstration Database Sample Forms

Examples that are in the INFORMIX-SQL User Guide and the INFORMIX-SQL Reference are based on the following five sample form specifications. These form specifications illustrate a wide variety of commands available with PERFORM.

- customer.per  A simple form used to enter and retrieve customer information.
- orders.per    A simple form used to enter and retrieve order information.
- orderform.per A more complex form used to enter and retrieve information about customer orders.
- sample.per    The most sophisticated form included with the stores2 database. It is an expanded version of the ORDERFORM form.
- p_ex1.per     Illustrates calling a C function from within a form.

Appendix A of this manual contains the full text of these sample form specifications. In addition, a copy of the sample specification is included in the section entitled “The SAMPLE Form Specification File,” starting on page 2-82.
Creating and Compiling a Custom Form

You can create a form specification file in one of two ways: you can use the Form option on the INFORMIX-SQL (I-SQL) Main menu, or you can work directly with the appropriate programs from the operating system command line. Either alternative requires that you have already created the database and all the tables to which the form will refer. The following two sections describe these alternative procedures. They do not, however, describe the rules for constructing or modifying the form specification file. These rules are defined in the remaining sections of this chapter.

Using the Menu System to Create a Form

To create a customized screen form using the I-SQL menu system, follow these steps:

1. Select the Form option on the I-SQL Main menu, and then the Generate option on the FORM menu.

2. If there is no current database, the SELECT DATABASE screen appears. After you select a database, the GENERATE FORM screen displays. Enter the name you want to assign to the form (for example, NEWFORM). I-SQL asks you for the names of the tables whose columns you want in your form. The GENERATE FORM menu allows you to enter up to eight tables. (If you want to include more than eight tables in your form specification, use the New option on the FORM menu to create it from scratch.) When you have selected all the tables you want to include, FORMBUILD creates a default form specification file. The FORM menu then displays. You can now use the default screen form with PERFORM.

   The default form specification file formats the screen as a list of all the columns in the tables included in the form. It does not provide any special instructions to PERFORM about how to display the data, nor does it include instructions to perform data manipulations.

3. Select the Modify option on the FORM menu, and I-SQL displays the MODIFY FORM screen. Indicate the name of the default form specification (NEWFORM). If you have not specified an editor previously in this session or set the DBEDIT environment variable (as explained in Appendix B, “Environment Variables,”), I-SQL asks for the name of your editor. Then I-SQL calls your system editor with the file.

   Edit the default form specification file to produce your customized screen form and associated instructions. Exit from the editor.

4. The MODIFY FORM menu displays. Select the Compile option.
Creating and Compiling a Custom Form

5. If your form specification file compiles correctly, a message to that effect displays, and FORMBUILD creates a form file with the filename extension .frm (for example, newform.frm). Go to Step 7. If your form specification file contains errors, a message to that effect displays, and FORMBUILD creates a form file with the filename extension .err (for example, newform.err). Go to Step 6.

6. Select the Correct option from the COMPILE FORM menu. I-SQL calls your system editor with the form specification file marked with the compilation errors. When you correct your errors, you need not delete the error messages. I-SQL does that for you. Repeat Step 4.

7. When the compilation is successful, select the Save-and-exit option on the MODIFY FORM menu.

As an alternative to using the Generate option and creating a default form specification, you can select the New option. I-SQL calls your system editor.

The Generate option is usually a more efficient way to create a custom form because, if you use the New option, you must enter all form specification instructions into the file.

Using the Operating System to Create a Form

To create a customized screen form directly from the operating system command line, follow these steps:

1. Create a default form specification file by entering the following command at the operating system prompt:
   
   sformbld -d

   FORMBUILD asks for the name of your form specification file, the name of your database, and the names of the tables whose columns you want in your form. FORMBUILD allows you to enter up to 14 tables. When you enter a blank line for the table name, FORMBUILD assumes you have selected all the tables and creates a default form specification file. FORMBUILD appends the extension .per to the name of the file.

   Alternatively, you can create a form specification file using a system editor. You do not need to add the .per extension to the name of the form file, but you can if you wish. If you use this method, proceed to Step 3.

2. Use a system editor to modify the default form specification file to meet your specifications.
3. Enter the command
   
   sformbld newform
   
   where newform is the name of your form specification file (without the .per extension). If the compilation is successful, FORMBUILD creates a compiled form file called newform.frm, and you are finished creating your customized screen form. If your compilation is not successful, FORMBUILD creates a file named newform.err, and you must proceed to Step 4.

4. Edit the file newform.err and correct the compilation errors. You must erase the error messages. Overwrite the file newform.per with this corrected version and repeat Step 3.

You can run the compiled form specification directly from the command line by entering the following command:

   sperform newform

You can also create a customized screen form from the operating system command line using a shortened version of the I-SQL Main menu options. Appendix H, “Accessing Programs from the Operating System,” discusses this method in detail.

Structure of a Form Specification File

Form specification files consist of four required sections (DATABASE, SCREEN, TABLES, and ATTRIBUTES) and one optional section (INSTRUCTIONS) as shown in Figure 2-1.

![Diagram of form specification file structure]

**Database Section**

Each form specification file must begin with a DATABASE section that identifies the database you want to use with the form.

**Screen Section**

The SCREEN section appears next and shows the exact layout of the form as you want it to appear on the screen. If the form has several screens, this section includes the layout for each screen, one after another.
You can use graphics characters to enhance the appearance of the screen.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLES section</td>
<td>Each form specification file must contain a TABLES section following the SCREEN section. The TABLES section identifies the tables whose columns appear in the form.</td>
</tr>
<tr>
<td>ATTRIBUTES section</td>
<td>The ATTRIBUTES section describes each field on the form including, for example, appearance, acceptable input values, displayed comments, and default values.</td>
</tr>
<tr>
<td>INSTRUCTIONS section</td>
<td>The INSTRUCTIONS section is optional and specifies master-detail relationships, composite joins, alternative field delimiters, and control blocks.</td>
</tr>
</tbody>
</table>

Using the END keyword to mark the end of sections in the form specification file is optional. Some users find it helpful to indicate the close of a section with END. The forms included with the demonstration database use the END keyword.

Figure 2-2 on page 2-8 illustrates the overall structure of a form specification file.
database stores

screen

CUSTOMER INFORMATION:
Customer Number: [c1 ] Telephone: [c10 ]

SHIPPING INFORMATION:
Customer P.O.: [o20 ]
Ship Date: [o21 ] Date Paid: [o22 ]

end tables

customer orders

attributes
c1 = *customer.customer_num = orders.customer_num;
c10 = phone, picture = "###-###-####x#####";


o20 = po_num;
o21 = ship_date;
o22 = paid_date;

instructions
customer master of orders;
orders master of items;
end

Figure 2-2  A partial form specification file
DATABASE Section

The DATABASE section of a form specification file identifies the database with which the form is designed to work.

```
DATABASE database-name
```

DATABASE is a required keyword.

database-name is the name of the database.

WITHOUT NULL INPUT are optional keywords that enable you to disallow NULL values.

Use the WITHOUT NULL INPUT option only if you have elected to create and work with a database that does not have NULL values. For fields that have no other defaults, this option causes I-SQL to display zeros as default values for number and INTERVAL fields, and blanks for character fields.

The default DATE value is 12/31/1899; the default DATETIME value is 1899-12-31 23:59:59.99999.

The following DATABASE section is from the sample form specification file at the end of this chapter:

```
database stores2
```
SCREEN Section

The SCREEN section of the form specification file describes how the form appears on the screen when you use it with PERFORM. A form specification can include multiple SCREEN sections that correspond to multiple page layouts.

SCREEN is a required keyword.
SIZE is an optional keyword that tells FORMBUILD to create a screen that is a specific number of lines long and cols wide.
lines is an integer that specifies the screen length in lines. The default is 24 lines.
BY is an optional keyword.
cols is an integer that specifies the screen width in columns. The default is 80 columns.
END is an optional keyword to end the SCREEN section.

Usage

- Each page layout is preceded by the SCREEN keyword and is enclosed in braces ({}). A page layout consists of an array of display fields and textual information, such as titles, field labels, and graphics characters. Display fields are indicated by brackets ([ ]) that define the field length and by field tags that identify the field.
- The default SCREEN section is SCREEN SIZE 24 by 80. FORMBUILD prepares a screen of up to 20 lines (4 lines are reserved for system use) and up to 80 characters in a line.
- Use the SIZE keyword to indicate an alternative screen size. If you do not indicate a larger screen size, and if you include more than 20 screen lines between a pair of braces, FORMBUILD splits the page, with line 21 at the top of the second page.
- If you specify a screen size, the size must appear on the first screen. The size applies to all the screens.
• You can use command-line syntax to override either or both of the lines or dimensions of the SCREEN section by specifying:

```bash
sformbld -l lines -c cols filename
```

where `lines` and `cols` are defined as in the above syntax diagram, and `filename` is the name of the form specification file. `FORMBUILD` uses the `INFORMIXTERM` environment variable to determine whether to use `termcap` or `terminfo` at compile time to set screen characteristics. If `INFORMIXTERM` is unset, `FORMBUILD` uses `termcap`.

The following figure illustrates the use of the `SCREEN` keyword with multiple page layouts:

```
SCREEN SIZE 18 BY 75
{
  
  display field page layout
  
}

SCREEN
{
  
  display field page layout
  
}
```

The sample form included at the end of this chapter demonstrates the SCREEN sections of a multiple-page form.
Page Layout

You indicate where data is to be displayed on the screen by using brackets ([ ]) to delimit a field. Each field has an associated field tag that identifies the field in the ATTRIBUTES and INSTRUCTIONS sections.

text is the material you want displayed on the screen.
[ ] are delimiters for a field. The width of the field is the number of characters that can be placed between the brackets. In this context, the brackets do not signify an optional input.
| denotes the close of one field and the beginning of the next field.

field-tag is the field tag used to identify the display field.

Usage

- Each field must have a field tag. The field tag is from 1 to 50 characters long. The first character must be a letter; the rest of the tag can include letters, numbers, and underscores (_). The field tag must be short enough to fit within the brackets. You can use the same field tag at more than one position in the SCREEN section of the form specification if you want the same column information to appear in more than one place.

- Field tags are not the same as database column names; they are the associations used in the SCREEN section, the ATTRIBUTES section, and the INSTRUCTIONS section to tell PERFORM where to display and store information. The ATTRIBUTES section associates each field tag with a column in your database or identifies it as a display-only field.

- FORMBUILD ignores the case of a field tag; a1 and A1 are the same.

- One-character columns are given the display tags a through z. This means that a screen form can use no more than 26 one-character columns.
• When you create a default form specification file, the widths of all fields are determined by the data type of the corresponding columns in the database tables.

• The width of a field in the SCREEN section of the form specification is normally set equal to the width of the database column to which it corresponds. You can reduce or expand the screen width, but you should be careful because this can truncate the screen presentation or database storage of the data.

• Fields corresponding to numeric columns should be large enough to contain the largest number that you might display. If the field is too small to display the number you assign to it, PERFORM fills the field with asterisks.

• Fields for CHAR type data can be shorter than the defined length of the column. PERFORM fills the field from the left and truncates the right end of any longer CHAR string assigned to the field. Through subscripting, you can assign portions of a CHAR column to one or more fields. (See the “ATTRIBUTES Section” on page 2-20.)

• If you edit and modify the default form specification file or create one from scratch, you can verify that the character column field widths match the data type of the corresponding columns by using the verify (-v) option of FORMBUILD. Enter the following command in response to the system prompt:

```
sformbld -v newform
```

FORMBUILD reports any discrepancies in the file newform.err, where newform is the name of the form specification file that has been verified.

• The | bar symbol can be used to denote the close of one field and the beginning of the next field. In the following example, field-tag1 identifies the first display field; field-tag2 identifies the second display field:

```
text [field-tag1 | field-tag2 ]
```

When you use the bar symbol to denote the close of one field and the beginning of the next field, you must include a DELIMITERS statement in the INSTRUCTIONS section of the form specification. Use the same symbol as both the left and right delimiters in the statement.
The screen layout from the sample form specification file follows:

```plaintext
class screen {
  // Screen layout from the sample form specification file follows:
  // ...

  Customer Number: [c1]
  Company: [c4]
  First Name: [c2]
  Last Name: [c3]
  Address: [c5]
  City: [c7]
  State: [c8]
  Zip: [c9]
  Telephone: [c10]

  CUSTOMER NUMBER: [c1]
  COMPANY: [c4]

  ORDER INFORMATION:
  Order Number: [o11]
  Order Date: [o12]
  Stock Number: [s13]
  Manufacturer: [i16]
  Description: [s14]
  Unit: [s16]
  Quantity: [i18]
  Unitprice: [s15]
  Total Price: [i19]

  SHIPPING INFORMATION:
  Customer P.O.: [o20]
  Ship Charge: [d1]
  Backlog: [a]
  Total Order Amount: [d2]
  Ship Date: [o21]
  Date Paid: [o22]
  Instructions: [o23]
}
end
```
Graphics Characters in Forms

You can use graphics characters to place boxes and other rectangular shapes in a screen form.

You can enter command strings to invoke a simple line drawing, or graphics, mode in which standard characters produce special effects on a compiled, displayed form. The letter p, for example, becomes the upper lefthand corner of a box, while a series of hyphens becomes a solid horizontal rule, as shown in Figure 2-3.

Form Specification

```plaintext
database STORES2
screen
{
  \gp---------------------------------------------------------q
  \g\g
  \g\g Cust No \[f000\] Company \[f001 \] \g
  \g\g\g
  \g\g Name: \[f002 \] \[f003 \] \g
  \g\g\g
  \g\g Telephone Number: \[f004 \] \g
  \\
  \g\g
  \g\g CUSTOMER
  \g\g Cust No [112 ] Company [Runners & Others ]
  \g\g Name [Margaret ] [Lawson ]
  \g\g Telephone Number: [415-887-7235 ]
```

Displayed Form

Figure 2-3  Form specification and displayed form (method 1)
The following procedure outlines the steps for specifying graphics characters.

1. Create a form in the usual manner.
2. Use the following characters in the SCREEN section to indicate the borders of one or more boxes on the form:

<table>
<thead>
<tr>
<th>Character</th>
<th>Produces</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>upper left corner of box</td>
</tr>
<tr>
<td>q</td>
<td>upper right corner of box</td>
</tr>
<tr>
<td>b</td>
<td>lower left corner of box</td>
</tr>
<tr>
<td>d</td>
<td>lower right corner of box</td>
</tr>
<tr>
<td>_</td>
<td>a horizontal-line character</td>
</tr>
<tr>
<td></td>
<td>a vertical-line character</td>
</tr>
</tbody>
</table>

The meanings for these six characters are derived from the gb specification in the termcap file, or the acsc specification in the terminfo file. I-SQL substitutes the graphics characters specified in the termcap or terminfo file for these characters when you display the compiled form.

3. Once the form has the desired configuration, use the \g string to indicate when to begin graphics mode and when to end graphics mode.

   Insert the \g string before the first p, q, d, b, dash, or pipe that represents a graphics character. To leave graphics mode, insert the string \g after the p, q, d, b, dash, or pipe. Figure 2-4 shows the commands that draw a box around text.
Figure 2-4  Drawing a box around text

Do not insert the \g strings in original white space on the form. The backslash should displace the first graphics character in the row and push the remaining row to the right. Although this distorts the way the form specification looks on screen, the actual output will not be distorted.

In your form specification, you can include not only the characters used to create a box or rectangle, but also other graphics characters. However, the meaning of a character other than p, q, d, b, dash, and pipe depends on your terminal.

Required Terminal Entries

If you plan to use graphics characters, your terminal entry in the termcap or terminfo files must include the following variables:

termcap:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gs</td>
<td>the escape sequence for entering graphics mode.</td>
</tr>
<tr>
<td>ge</td>
<td>the escape sequence for leaving graphics mode.</td>
</tr>
<tr>
<td>gb</td>
<td>the concatenated, ordered list of ASCII equivalents for the six graphics characters used to draw the border.</td>
</tr>
</tbody>
</table>
terminfo:
smacs the escape sequence for entering graphics mode.
rmacs the escape sequence for leaving graphics mode.
acsc the concatenated, ordered list of ASCII equivalents for the six graphics characters used to draw the border.

For information about making changes to your termcap and terminfo files, see Appendix D, “Environment Variables,” or check the manual that comes with your terminal.

TABLES Section

The third section of the form specification file lists all the tables from which columns appear in the screen form. You need not display in the screen form every column of every table listed, but any table contributing data to the form must be included.

The TABLES section consists of the TABLES keyword, followed by a table name or a list of table names separated by spaces.

In a MODE ANSI database, a form must qualify any table name with the owner. prefix if users other than owner run the form. If you specify an owner name, you must specify a simple alias for owner.table-name in the TABLES section to reference the table in other sections of the form specification file.

TABLES is a keyword to begin the TABLES section.
alias is the table name, synonym, or alias for the name of the table in the form specification file.
owner is the username of the user who created the table.
table-name is the identifier or synonym of the table in its database.
END is an optional keyword to end the TABLES section.
**Usage**

- You cannot include a temporary table in your table list.
- You can build a form based on a view as long as the columns contributing data to the view belong to only one table. Aggregate data is not allowed.
- The number of tables that you can use in a form is machine dependent. On most UNIX systems, the maximum number of tables open at one time is 12.

The TABLES section from the *sample* form specification file at the end of this chapter is as follows:

```
tables
  customer items stock
  orders manufact
```

The following TABLES section specifies aliases for two tables:

```
tables
  tab1 = refdept.archive
  tab2 = athdept.equip
```

ATTRIBUTES Section

The ATTRIBUTES section describes the behavior and appearance of each field defined in the SCREEN section. Every field in the SCREEN section must be described in the ATTRIBUTES section. You use attributes to describe how PERFORM should display the field, to specify a default value, to limit the values that can be entered, and to set other parameters as described in the “ATTRIBUTES Syntax” section beginning on page 2-28.

The order in which the fields are described in the ATTRIBUTES section determines the default order for the cursor movement on the screen. The order in which columns are referenced determines the order in which PERFORM makes tables active (that is, available for data entry).

Fields in the SCREEN section do not have to be associated with a column. A field not associated with a column is called a display-only field. The ATTRIBUTES section contains the following two kinds of link statements: statements that link field tags to database columns and statements that link field tags to display-only fields.

ATTRIBUTES is a required keyword.

field-tag is the field tag used in the SCREEN section

Display Field Order

When you use the Query, Add, and Update options in PERFORM, the cursor advances by default from field to field in the active table according to the order in which the field tags appear in the ATTRIBUTES section of the form specification. When the cursor has advanced through all the fields in the active table on the screen, it returns to the first field. You can change the default order by using control blocks, described in the “INSTRUCTIONS Section” on page 2-56.
Table Order

When a form contains fields that correspond to several database tables, PERFORM puts the tables in an ordered list. When you use the Table option, PERFORM changes the active table by selecting the next table in the list. PERFORM assigns tables in the order in which columns in the tables are referenced in the ATTRIBUTES section. You reference a table whenever you associate a column from the table with a tag name or another column in a join.
Fields Linked to Database Columns

You can link two kinds of fields to database columns in the ATTRIBUTES section: fields that accept input and fields that do not. Fields that are linked to database columns and that do not allow input from the keyboard are called lookup fields. You specify lookup fields with the LOOKUP attribute, defined in “ATTRIBUTES Syntax” on page 2-28.

This section describes column-linked fields that allow input.

Field Description

* is optional punctuation that identifies the dominant column in a verify join.

table is the optional name or alias of a database table. You need to specify table only if the same column name occurs in more than one table in the form. (This allows PERFORM to uniquely identify a column.)

column-name is the name of a column.

attribute-specification is a FORMBUILD attribute or a list of attributes separated by commas. All of the attributes are described in “ATTRIBUTES Section” on page 2-20.

Usage

- You can display portions of CHAR-type columns in a field by using subscripting. For example, the orders table has a ship_instruct column that is a CHAR-type column of length 40. You can display it on the screen as two display fields of length 20. If the field tags for the two fields are inst1 and inst2, respectively, the ATTRIBUTES section entry is as follows:

  inst1 = ship_instruct[1,20];
  inst2 = ship_instruct[21,40];

You can also use the WORDWRAP attribute to display long CHAR fields on multiple lines.
• If you use an alias in the TABLES section, you must use the alias to refer to the table in the ATTRIBUTES section.

Display-Only Fields

Display-only fields are not associated with columns of the database and appear only on the screen. They receive their values as a result of calculations or logical decisions based on the values in other fields.

DISPLAYONLY is a required keyword that indicates that the field does not correspond to a column of a table in the database. You describe how such a field receives its value in the INSTRUCTIONS section.

ALLOWING INPUT are optional keywords that you use to allow input.

TYPE is a required keyword.

data-type is any one of the data types permitted by I-SQL except SERIAL. (See Chapter 3 of the Informix Guide to SQL: Reference, Version 4.1 for information about data types.)

NOT NULL are optional keywords that inform I-SQL that, if the field allows input, the user must give it a value.

attribute-specification is an attribute or a list of attributes separated by commas. All the attributes are described in “ATTRIBUTES Section” on page 2-20.

- Do not give a length to type CHAR; the display width determines the length.
- If you specify the precision for a DECIMAL or MONEY type, be certain that the display width can hold the value.
- When the field does not allow input, you can use only the following attributes with display-only fields:

  | DEFAULT | DOWNSHIFT |
  | FORMAT  | QUERYCLEAR |
  | REVERSE | RIGHT |
  | UPSHIFT | ZEROFILL |
• When you specify that one or more display-only fields allow input, PERFORM collects these fields into a database table named displaytable. No database table is actually created, but PERFORM behaves as though displaytable existed and as though its field tags were column names. You use displaytable in the INSTRUCTIONS section to control data entry and cursor movement in display-only fields that allow input.

• If a displaytable exists, it is always the last table in the sequence of active tables, regardless of how you order its fields among the other field tags.

• When the DATABASE section has the WITHOUT NULL INPUT clause, the NOT NULL keywords instruct I-SQL to use zero (number data type) or blanks (CHAR data type) as a default for this field.

The ATTRIBUTES section of the sample form specification file contains the following two DISPLAYONLY fields:

d1 = displayonly type money;
d2 = displayonly type money;

These fields are used to calculate the shipping charge and total order amount for each order. This information is not stored in any columns in the database.

Joining Columns

A screen form that contains information from several database tables normally includes a display field that joins two (or more) database columns that contain the same information. While it is not required that the join columns be indexed, it is advisable since cross-table queries do not run as quickly if the underlying join columns are not indexed.

The database columns you join must be of the same data type. If they are CHAR columns, they must be the same length. Do not join two SERIAL columns to each other; join a SERIAL column only to an INTEGER column.

You join columns by equating them to the same field tag in the ATTRIBUTES section:

\[
\text{field-tag} = \text{col1} = \text{col2};
\]

An example from the sample form follows:

\[
o11 = \ast\text{orders.order_num} = \text{items.order_num};
\]

Field-tag o11 joins the order_num column of the orders table with the order_num column of the items table. (The asterisk placed before the orders.orders_num column name indicates that this is a special kind of join—a verify join. Verify joins are explained on page 2-27.)

The placement of attributes determines when they take effect. If you want an attribute to apply regardless of which table in the join is active, place the column names on the same line and the attribute after the last column name:

\[
\text{field-tag} = \text{col1} = \text{col2}, \text{attr};
\]

If you want different attributes to apply for each of the columns in the join, place the column names on separate lines:

\[
\text{field-tag} = \text{col1}, \text{attr1};
\]
\[
= \text{col2}, \text{attr2};
\]

attr1 is effective when the table containing col1 is active, and attr2 is effective when the table containing column2 is active.

Here is an example from the sample form:

\[
i13 = \text{items.stock_num};
\]
\[
= \ast\text{stock.stock_num}, \text{noentry},
\]
\[
\text{noupdate, queryclear};
\]
The attributes NOENTRY, NOUPDATE, and QUERYCLEAR (explained in “ATTRIBUTES Syntax” on page 2-28) are effective only when the stock table is active.

The FORMAT and REVERSE attributes, also described in the “ATTRIBUTES Syntax” section, always take effect, regardless of their placement.

Verify Joins

You can verify that the value you enter into a field that corresponds to a column in one table already exists in another column (the dominant column) in another table. You do this through a verify join. You indicate the verify join by placing an asterisk in front of the dominant column name, as follows:

\[
\text{field-tag} = \text{col1} = *\text{col2};
\]

PERFORM prevents entry of any value into field-tag that does not already occur in col2. (This applies for noncomposite conditions.)

For example, when you assign orders to customers, you want to ensure that the customer number entered for a store is a valid customer number in the customer table. The following statement in the ATTRIBUTES section of the sample form does just this:

\[
c1 = *\text{customer.customer_num} = \text{orders.customer_num};
\]

In the previous statement, the customer.customer_num column is the dominant column in a verify join; when the orders table is active, you cannot enter a value into field c1 that does not already exist in the customer_num column of the customer table.

A third kind of join, described under the LOOKUP attribute on page page 2-41, allows you to display or verify data from a table that is not active.
ATTRIBUTES Syntax

PERFORM recognizes the following attributes. The syntax for each attribute is detailed in the following sections.

- AUTONEXT
- NOUPDATE
- COLOR
- PICTURE
- COMMENTS
- QUERYCLEAR
- DEFAULT
- REQUIRE
- DOWNSHIFT
- REVERSE
- FORMAT
- RIGHT
- INCLUDE
- UPSHIFT
- INVISIBLE
- VERIFY
- LOOKUP
- WORDWRAP
- NOENTRY
- ZEROFILL

AUTONEXT

Use the AUTONEXT attribute to cause the cursor to advance automatically to the next field when the current field is full.

Usage

- AUTONEXT is particularly useful for entering text into a CHAR type database column that is split among two or more display fields with the use of subscripts.
- Another use of AUTONEXT is with CHAR fields in which the input data is of a standard length (for example, the abbreviation for a state name is always two digits) or when the CHAR field has a length of one (only one keystroke is required to enter the data and to move to the next field).

The sample form specification file uses the AUTONEXT attribute to display the state and zipcode columns from the customer table, as shown:

```plaintext
c8 = state, upshift, autonext;
c9 = zipcode, autonext;
```

When two characters are entered into the c8 field (and the field is full), the cursor moves automatically to the beginning of the next field (the c9 field). When five characters are entered into the c9 field (and the field is full), the cursor moves automatically to the beginning of the next field.
COLOR

Use COLOR to display field text in one of eight colors, either alone or combined with one or more of four intensities.

```
COLOR = display mode

COLOR is a required keyword.
display mode is one or more display attributes selected from the list of available colors and intensities.
WHERE is an optional keyword.
where condition is a situation under which you want specified display attributes to be in effect.
```

Usage

The display mode consists of zero attributes or one attribute from the color list, and zero or more attributes from the intensity list, as follows:

<table>
<thead>
<tr>
<th>Color</th>
<th>Intensity</th>
<th>Displayed As</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHITE</td>
<td></td>
<td>White</td>
</tr>
<tr>
<td>YELLOW</td>
<td></td>
<td>Yellow</td>
</tr>
<tr>
<td>MAGENTA</td>
<td></td>
<td>Magenta</td>
</tr>
<tr>
<td>RED</td>
<td></td>
<td>Red</td>
</tr>
<tr>
<td>CYAN</td>
<td></td>
<td>Cyan</td>
</tr>
<tr>
<td>GREEN</td>
<td></td>
<td>Green</td>
</tr>
<tr>
<td>BLUE</td>
<td></td>
<td>Blue</td>
</tr>
<tr>
<td>BLACK</td>
<td></td>
<td>Black</td>
</tr>
<tr>
<td>BLINK</td>
<td></td>
<td>Blinking</td>
</tr>
<tr>
<td>UNDERLINE</td>
<td></td>
<td>Underlined</td>
</tr>
<tr>
<td>REVERSE</td>
<td></td>
<td>Reverse Video</td>
</tr>
<tr>
<td>LEFT</td>
<td></td>
<td>Left Justified (number fields)</td>
</tr>
</tbody>
</table>

- If you do not select a where condition, the display mode always applies to the field. When you select a where condition, FORMBUILD tests the condition whenever a new value enters the field.
- If the condition is true, FORMBUILD displays the field with the mode you selected. If the condition is false, FORMBUILD displays the field with default characteristics.
• You can code the where condition syntax in any one of the following ways:

expr LIKE expr
expr NOT LIKE expr
expr MATCHES expr
expr NOT MATCHES expr
expr IS NULL
expr IS NOT NULL
expr BETWEEN expr AND expr
expr NOT BETWEEN expr AND expr
expr IN (list of exprs)
expr NOT IN (list of exprs)
expr relop expr
(bool-expr)
bool-expr OR bool-expr
bool-expr AND bool-expr

where expr can represent any one of the following items:

field tag
constant
TODAY
CURRENT
agg-function OF field tag
- expr
expr [+ - * / ] expr
(expr)

and relop can be any of the following comparison operators:

= <> != >= <= < >

The following example illustrates how to specify that field text should be displayed in red type:

f000 = customer.customer_num, color=red;

The following examples illustrate conditional use of COLOR:

f002 = manufact.manu_code,
        color=red where f002="HRO";
f003 = customer.lname,
        color=red where f003 not like "Quinn";
f004 = customer.zipcode
        color=red blink where f004 > 10000;
The evaluation of MATCHES, LIKE, and BETWEEN expressions containing character arguments is dependent on the LANG and LC_COLLATE settings in an NLS database. Refer to Appendix C, “Native Language Support Within INFORMIX-SQL.”
COMMENTS

Use COMMENTS to cause PERFORM to display a message on the Comment line at the bottom of the screen. The message displays when the cursor moves to the associated field.

 COMMENTS = "message"

COMMENTS is a required keyword.
message is a character string enclosed in quotes.

Usage

• The message must appear in quotation marks on a single line of the form specification file.
• The Status line is the bottom line of the screen. The Comment line is just above the Status line.
• The most common use of the COMMENTS attribute is to give information or instructions to the user. This is particularly appropriate when the field accepts only a limited set of user-specified values.

An example from the sample form specification file follows:

c2 = fname, comments = "Please enter initial if available." ;

Related Attribute

INCLUDE

The FORMBUILD Transaction Form Generator  2-33
Use the DEFAULT attribute to assign a default value to a display field.

```
  DEFAULT = value
```

**Usage**

- If you do not use the DEFAULT attribute, display fields default to blanks.
- Enclose DATE values and CHAR values that contain spaces or special characters in quotation marks. Using quotation marks around CHAR values that contain no spaces or special characters is optional.
- PERFORM displays the default value whenever the field displays for data entry in an Add operation.
- If both the DEFAULT attribute and the REQUIRED attribute are assigned to the same field, the REQUIRED attribute is ignored.
- Use the TODAY keyword as the `value` to assign the current date as the default value of a DATE field.
- Use the CURRENT keyword to assign the current date and time as the default value of a DATETIME field.
- If you use the WITHOUT NULL INPUT option in the DATABASE section and you do not use the DEFAULT attribute, then character fields default to blanks, number and INTERVAL fields default to 0, and MONEY fields default to $0.00. The default DATE value is 12/31/1899, and the default DATETIME value is 1899-12-31 23:59:59.99999.
- If you do not use WITHOUT NULL INPUT in the DATABASE section, all fields default to NULL values unless you use the DEFAULT attribute.

Two examples from the **sample** form specification file follow:

```
c8 = state, upshift, autonext,
    default = "CA" ;

o12 = order_date, default = today,
    format = "mm/dd/yyyy" ;
```
DOWNSHIFT

Assign the DOWNSHIFT attribute to a CHAR field when you want PERFORM to convert uppercase letters to lowercase letters.

Usage

Because uppercase and lowercase letters have different ASCII values, storing character strings in one format or the other can simplify sorting and querying a database.

NLS

When NLS is active, the results of conversion between uppercase and lowercase are appropriate for the national language in use, as defined by the LC_CTYPE environment variable. Refer to Appendix C, “Native Language Support Within INFORMIX-SQL.”

Related Attribute

UPSHIFT
FORMAT

Use the FORMAT attribute with a DECIMAL, SMALLFLOAT, FLOAT, or DATE column to control the format of the display.

```
FORMAT = fstring
```

**FORMAT** is a required keyword.

**fstring** is a (format) string of characters that specifies the desired data format. You must enclose `fstring` in quotation marks.

**Usage**

- For DECIMAL, SMALLFLOAT, or FLOAT data types, `fstring` consists of pound signs (#) that represent digits and a decimal point. For example, `###.##` produces up to three places to the left of the decimal point and exactly two places to the right.
- If the actual displayed number is shorter than the `fstring`, PERFORM right-justifies it and pads the left with blanks.
- If the `fstring` is smaller than the display width, FORMBUILD gives a warning, but the form is usable.
- If necessary, PERFORM rounds numbers before displaying them.
- For DATE data types, PERFORM recognizes the following symbols as special in the string `fstring`:
  - `mm` produces the two-digit representation of the month.
  - `mmm` produces a three-letter abbreviation of the month, for example, Jan, Feb, and so on.
  - `dd` produces the two-digit representation of the day.
  - `ddd` produces a three-letter abbreviation of the day of the week, for example, Mon, Tue, and so on.
  - `yy` produces the two-digit representation of the year.
  - `yyyy` produces a four-digit year.
  For dates, FORMBUILD interprets any other characters as literals and displays them wherever you place them within `fstring`.
- You cannot use the FORMAT attribute with DATETIME or INTERVAL data types.
The following table lists example FORMAT attributes for DATE fields:

<table>
<thead>
<tr>
<th>Input</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>no FORMAT attribute</td>
<td>09/15/1994</td>
</tr>
<tr>
<td>FORMAT = &quot;mm/dd/yy&quot;</td>
<td>09/15/94</td>
</tr>
<tr>
<td>FORMAT = &quot;mmm dd, yyyy&quot;</td>
<td>Sep 15, 1994</td>
</tr>
<tr>
<td>FORMAT = &quot;yymmd&quot;</td>
<td>940915</td>
</tr>
<tr>
<td>FORMAT = &quot;dd-mm-yyyy&quot;</td>
<td>15-09-94</td>
</tr>
<tr>
<td>FORMAT = &quot;(ddd.) mmm. dd, yyyy&quot;</td>
<td>(Thu.) Sep. 15, 1994</td>
</tr>
</tbody>
</table>

Two examples from the sample form specification file follow:

```c
o12 = order_date,
    default = today,
    format = "mm/dd/yyyy";

o22 = paid_date,
    format = "mm/dd/yyyy";
```

When NLS is active, the setting in the NLS environment variables LC_MONETARY, LC_NUMERIC, and LANG affect the way the format string in the FORMAT attribute is interpreted for numeric and monetary data. In the format string, the period symbol (.) is not a literal character but a placeholder for the decimal separator specified by environment variables. Likewise, the comma symbol (,) is a placeholder for the thousands separator specified by environment variables. The $ symbol is a placeholder for the leading currency symbol. The @ symbol is a placeholder for the trailing currency symbol. Thus, the format string $$#,###.## will format the value 1234.56 as £1,234.56 in a British locale but as f1,234.56 in a French locale. Note that setting either DBFORMAT or DBMONEY overrides any settings in LC_MONETARY or LC_NUMERIC. Refer to Appendix C, “Native Language Support Within INFORMIX-SQL.”

The mmm and ddd specifiers in a format string can display language-specific month name and day name abbreviations on the form. This requires the installation of message files in a subdirectory of $INFORMIXDIR/msg and subsequent reference to that subdirectory by way of the environment variable DBLANG. For example, the ddd specifier in a Spanish locale translates the day Saturday into the day name abbreviation Sab, which stands for “Sabado” (the Spanish word for Saturday). Refer to the section entitled “DBLANG” on page B-18.

Related Attribute

PICTURE
Use the INCLUDE attribute to specify acceptable values for a field and to cause PERFORM to check input before accepting it.

- INCLUDE is a required keyword.
- value is either a list of individual values (value1, value2, ...), a range of values (value1 TO value2), or a combination of individual values and ranges, all separated by commas.
- NULL is an optional keyword that specifies a NULL value.

Usage

- When you specify a range of values, the lower value must appear first.
- For ranges of character values, PERFORM uses dictionary ordering with the printable ASCII character set. (See Appendix E, “The ASCII Character Set,” for an ordered list of the ASCII character set.) In a number field, the range (5 to 10) is acceptable. In a CHAR field, it is incorrect. The character string 10 is less than the string 5, since 1 comes before 5 in the ASCII character set.
- If you include a character string that contains a blank space, a comma, or any special characters, you must enclose the entire string in quotation marks. It is advisable to enclose character strings in quotation marks at all times.
- Before PERFORM accepts a new row, you must enter an acceptable value in each display field with the INCLUDE attribute. Use the keyword NULL to indicate that a NULL value is acceptable. If a field has both the DEFAULT and INCLUDE attributes, then the DEFAULT value must appear in the INCLUDE list. Otherwise the form does not compile.
- Including COMMENTS that indicate acceptable values makes data entry easier.
An example from the sample form specification file follows:

```
ii8 = items.quantity,
    include = (1 to 50),
    comments = "Acceptable values are 1 through 50";
```

When NLS is active, settings in the environment variables LC_COLLATE and LANG determine the results of evaluation of character data in INCLUDE ranges. A given character will be contained in an INCLUDE range or not depending on where it collates relative to the INCLUDE values. Refer to Appendix C, “Native Language Support Within INFORMIX-SQL.”

**Related Attributes**

- COMMENTS
- REQUIRE
INVISIBLE

If a field is defined as INVISIBLE, I-SQL does not display the value assigned to the field or the value the user is entering in the field.

Usage

- If you assign both INVISIBLE and COLOR attributes to a field, I-SQL ignores the COLOR attribute, unless you specify COLOR=REVERSE. In this case, I-SQL displays the field in reverse video and maintains the invisibility of the field’s contents.
- If you assign both INVISIBLE and PICTURE attributes to a field, I-SQL does not display the picture pattern.
LOOKUP

Use the LOOKUP attribute to display data from another table while entering data into or querying the active table. You can also use it to prevent data from being entered into the active table if the value does not exist in another table.

LOOKUP field-tag1 = table2.col1

JOINING field-tag2 = table2.col2

LOOKUP is the keyword that specifies a join.
field-tag1 is the field tag of a field that displays a value from the LOOKUP table.
table2.col1 is a column in table2 whose value displays in field-tag1.
JOINING is the keyword that identifies the joined column.
* is optional punctuation that identifies the dominant column in a verify join. (See the “Verify Joins” section on page 2-27.)
table2.col is the name of a column that belongs to table2 and is joined to table1.col.
field-tag2 is the field tag of a field that displays a value from the LOOKUP table.
table2.col2 is a column in table2 whose value displays in field-tag2.

Usage

• If you use an alias in the TABLES section, you must use the alias to refer to the table in the ATTRIBUTES section.
• The optional asterisk placed in front of table2.col tells PERFORM to accept a value for table1.col only if the same value already exists in table2.col.
• The optional list of field tags with column names following the LOOKUP attribute directs PERFORM to display these values whenever there is a successful join between table1.col and table2.col. You cannot enter values into these fields from the keyboard.
• If the join columns in a LOOKUP are not indexed, the LOOKUP does not run as quickly.

An example of the LOOKUP join from the sample form specification file follows:

```plaintext
ll6 = items.manu_code,
    lookup m17 = manufact.manu_name
    joining *manufact.manu_code;
```

In this example, the entry of the item manufacturer code number is checked against the list of manufacturer code numbers in the manufact table. If the same value is found there, the manufacturer’s name is extracted from the manufact table and displays in field m17.
NOENTRY

Use the NOENTRY attribute to prevent data entry when a new row is created during an Add operation.

Usage

• The NOENTRY attribute does not prevent you from modifying the field during an Update operation.

• The NOENTRY attribute is unnecessary with a SERIAL column.

Two examples from the sample form specification file follow:

```plaintext
i13 = items.stock_num;
   = *stock.stock_num, noentry,
      noupdate, queryclear;

s14 = stock.description, noentry,
      noupdate;
```

When the stock table is active, the columns i13 and s14 (corresponding to the columns stock.stock_num and stock.description, respectively) cannot have values added. (The inclusion of the NOUPDATE attribute prevents data entry during an Update operation.)

Related Attribute

NOUPDATE
NOUPDATE

Use the NOUPDATE attribute to prevent data entry when a row is modified during an Update operation.

Usage

The NOUPDATE attribute does not prevent you from entering data into the field during an Add operation.

Two examples from the sample form specification file follow:

\[
\begin{align*}
\text{s15} &= \text{stock.unit_price}, \text{noentry}, \text{noupdate}; \\
\text{s16} &= \text{stock.unit_descr}, \text{noentry}, \text{noupdate};
\end{align*}
\]

When the stock table is active, the fields s15 and s16 (corresponding to the columns stock.unit_price and stock.unit_descr, respectively) cannot receive values during an Update operation. (The inclusion of the NOENTRY attribute prevents data entry during an Add operation.)

Related Attribute

NOENTRY
PICTURE

Use the PICTURE attribute to specify the character pattern for data entry to a non-number field.

```plaintext
PICTURE = "pstring"
```

- **PICTURE** is a required keyword.
- **pstring** is a (picture) string of characters that specifies the desired character pattern.

**Usage**

- **pstring** is a combination of three special symbols:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Any letter</td>
</tr>
<tr>
<td>#</td>
<td>Any digit</td>
</tr>
<tr>
<td>X</td>
<td>Any character</td>
</tr>
</tbody>
</table>

  Any other character in the pstring is treated as a literal and occurs, during data entry, in the exact location indicated.

- If you attempt to enter a character that does not conform with the pstring, you hear a beep and PERFORM does not echo the character on the screen.
- The PICTURE attribute does not require the entry of the entire field; it only requires that what you enter conforms to pstring. Note that the length of pstring must equal the length of the corresponding display field.
- PERFORM reminds data entry operators of the required pattern by displaying the literal characters in the display field and leaving blanks elsewhere.

The following examples are from the sample form specification file:

```plaintext
c10 = phone,
    picture = "###-###-####x####";
```

produces the following display field before data entry:

```plaintext
[ - - ]
```
As another example, if you specify a field for part numbers like this,

\[ f1 = \text{part\_no}, \text{picture} = "\text{AA#####-AA(X)}"; \]

\textbf{PERFORM} would accept any of the following inputs:

- LF49367-BB(*)
- TG38524-AS(3)
- YG67491-ZZ(D)

\textbf{NLS} The PICTURE attribute is not affected by the NLS environment variables because PICTURE only formats character information.

\textbf{Related Attribute}

\textbf{FORMAT}
QUERYCLEAR

Use the QUERYCLEAR attribute to clear a joining field on the screen when you enter a Query operation.

**Usage**

- When you enter the Query option, PERFORM normally clears all fields except joining and display-only fields.
- QUERYCLEAR does not apply to display-only fields. You must give explicit instructions in the INSTRUCTIONS section to clear display-only fields.

An example from the sample form specification file follows:

```plaintext
i13 = items.stock_num;
    = *stock.stock_num, noentry,
      noupdate, queryclear;
```

Here the *items* table and the *stock* table are joined through the stock number. When the *stock* table is the active table and a query is made, the *stock_num* field is cleared. When *items* is the active table, however, the *stock_num* field is not cleared when a query is made.
REQUIRED

Use the REQUIRED attribute to force data entry into a particular field during an Add operation.

Usage

- The REQUIRED attribute has no effect during a PERFORM Update operation. You are free to erase values from REQUIRED fields when you use an Update operation.
- There is no default value for a REQUIRED field. If you assign both the REQUIRED attribute and the DEFAULT attribute to the same field, the REQUIRED attribute is ignored.

An example from the sample form follows:

```plaintext
o20 = po_num, required;
```

FORMBUILD requires the entry of a purchase order value when adding a new order to the database.
Assign the REVERSE attribute to fields you want **PERFORM** to display in reverse video.

**Usage**

On machines that do not support reverse video, fields that have the REVERSE attribute are enclosed in angle brackets (< >).
Assign the RIGHT attribute to fields in which you want the data to be right-justified.

Usage

- **PERFORM** right-justifies data you enter during an Add or Update operation.
- To search for a right-adjusted CHAR field of value “string” during a Query operation, use the wildcard search pattern “*string*” to account for potential leading blanks.
UPSHIFT

Assign the UPSHIFT attribute to a CHAR field when you want PERFORM to convert lowercase letters to uppercase letters.

Usage

Because uppercase letters and lowercase letters have different ASCII values, storing character strings in one format or the other can simplify sorting and querying of a database.

An example from the sample form follows:

```sql
C8 = state, upshift, autonext,
    include = ("CA", "OR", "NV", "WA"),
    default = "CA" ;
```

Because of the UPSHIFT attribute, PERFORM enters uppercase characters in the state field regardless of the case used to enter them.

Related Attribute

DOWNSHIFT
Use the VERIFY attribute when you want PERFORM to require users to enter data twice for a particular field to reduce the probability of erroneous data entry.

**Usage**

Since some data is critical, the VERIFY attribute supplies an additional step in data entry to ensure the integrity of the data in your database. After you enter a value into a VERIFY field and press RETURN, PERFORM erases the field and requests that you reenter the value. You must enter exactly the same data each time, character for character: 15000 is not exactly the same as 15000.00.

If you specify the following field for salary information:

```plaintext
s10 = salary, verify;
```

PERFORM requires the entry of exactly the same data twice.
**WORDWRAP**

Use the WORDWRAP attribute when you want PERFORM to wrap a long character string to the next field that has the same field tag.

- WORDWRAP is the required keyword that instructs PERFORM to wrap long character strings to the next successive field.
- COMPRESS is the optional keyword that tells PERFORM to discard any spaces that you did not type and that are not part of the data.

**Usage**

- The keyword WORDWRAP enables the multiline editor. When you enter text from the keyboard and reach the end of a line, the editor brings the current word down to the next line, moving text to subsequent lines as necessary. When you delete text, the editor pulls words up from lower lines whenever it can.
  
  If you do not use the WORDWRAP attribute, words do not flow from one line in the field to the next, and you must edit text by using the arrow keys or the RETURN key to move from field to field.

- The editor distinguishes between *intentional* blanks (blanks that you typed or that are part of the data) and *editor* blanks (blanks that the editor inserts at the ends of lines to make text wrap around to the next line). Intentional blanks are retained as part of the data. Editor blanks are inserted and deleted automatically as required for word wrapping.

- The COMPRESS attribute tells PERFORM to discard editor blanks. If you do not use the COMPRESS attribute, and the sum of the segment lengths exceeds the column size, PERFORM might truncate some trailing words.

- When you design a multiline field, allow room for editor blanks. You can expect the average number of editor blanks per line to be half the length of an average word.

- The editor breaks lines between words whenever possible. Ordinarily, the field is as long as, or longer than, the column size, and PERFORM displays all text.

- If the column data is longer than the field, the editor fills the field and discards the excess data. You lose data if you use a truncated display to update a database.
The following example shows the SCREEN and ATTRIBUTES sections of a form specification file that specifies a multiple-line field:

database...
screen
{
Enter text:  [mlf ]
    [mlf ]
    [mlf ]
    [mlf ]
}
tables...
attributes
   mlf = charcolm, wordwrap compress;

Since the screen field whose tag is mlf appears in four physical segments in the screen layout and has the WORDWRAP attribute, it is a multiple-line field. Its value is composed of the physical segments taken in top-to-bottom, left-to-right order. The field should ordinarily be as long as or longer than the column so that it can display all of the text. It is not necessary that the segments be the same size, as they are in the example.

In the field description in the ATTRIBUTES section, the keyword WORDWRAP enables the multiline editor. If you omit it, words cannot flow from one segment to the next.

ZEROFILL

Assign the ZEROFILL attribute to fields that you want to be right-justified and padded with leading zeros.

Usage

This attribute is most useful with numeric fields. If the number entered into the field is shorter than the field itself, PERFORM right-justifies it and fills the leading blanks with zeros.
INSTRUCTIONS Section

The final section of the form specification file is the optional INSTRUCTIONS section. This section is used for the following tasks:

- Establishing composite joins
- Specifying alternative field delimiters
- Creating master/detail relationships
- Defining control blocks

You can also call C functions from within the INSTRUCTIONS section. See Chapter 6, "C Functions in ACE and PERFORM," for details.

The INSTRUCTIONS keyword begins the INSTRUCTIONS section as shown in the following diagram:
Establish a COMPOSITE JOIN between two tables when you must specify the values of more than one column in a table to specify a row uniquely.

COMPOSITES is the keyword indicating that the following sets of column names enclosed in angle brackets (<>) are to be treated as composite columns that are joined to each other.

- \( \text{table1.col}J \) (where \( J = 1, 2, 3, \ldots \)) is a column in \( \text{table1} \).
- \( \text{table2.col}J \) (where \( J = 1, 2, 3, \ldots \)) is a column in \( \text{table2} \).
- * indicates that the join is a verify join—that is, unless the marked composite exists in \( \text{table2} \), PERFORM does not allow the corresponding row to be written to \( \text{table1} \).

Usage

- If you use an alias in the TABLES section, you must use that alias to refer to the table in the composite join.
- Each column included in a composite join must also be individually joined in the ATTRIBUTES section of the form specification. This means that \( \text{table1.col1} \) must be joined individually to \( \text{table2.col1} \) in the ATTRIBUTES section, as must \( \text{table1.col2} \) to \( \text{table2.col2} \), and so on.
- There can be no additional joins between columns of the two tables that are not included in the composite join.
- If the columns in a composite join are not individually and jointly indexed, cross-table queries do not run as quickly.

An example from the sample form specification file follows:

```plaintext
composites <items.stock_num, items.manu_code>
   * <stock.stock_num, stock.manu_code>
```

The \textit{stock_num} and \textit{manu_code} fields in the \textit{items} and \textit{stock} table are included in a composite join. This is a composite verify join. When the \textit{items} table is active, values entered in the \textit{stock_num} and \textit{manu_code} fields are
compared with values existing in those two columns in the stock table. 
**PERFORM** notifies the user if there is not a match and rejects the entry. This precludes the entry of stock numbers and manufacturer codes that individually exist in the database but, as a composite, do not correspond to a unique row in the stock table.

To specify a unique row in the stock table requires both the **stock_num** and **manu_code**. For example, the stock table contains three rows with the stock number 1, and four rows with the manufacturer code HRO. (See Appendix A, “customer The Demonstration Database and Application,” for a list of data included in the sample database.) Knowing the stock number or manufacturer code alone does not allow you to locate a unique row. You need both the stock number (1) and the manufacturer code (HRO) to specify a unique row (baseball gloves produced by Hero) in the table.
DELIMITERS

You may change the delimiters that PERFORM uses to enclose the fields when the form appears on the screen. The default delimiters are brackets ([ ]), but you can substitute any other printable character, including blank spaces.

Usage

• The DELIMITERS instruction tells PERFORM the symbol to use as a delimiter when it displays the fields on the screen.
• Each delimiter is a single character only.
• FORMBUILD still requires that you use brackets in the form specification file.
• If your form has columns from more than one database table, you might not want to use blank spaces as delimiters. If you use blank spaces, you have no visual indication on the screen of which fields correspond to columns in the active table.
• You can use the | bar symbol to denote both a closing delimiter and an opening delimiter. For example,

| Name | tag1 | tag2 |

`tag1` identifies the first display field; `tag2` identifies the second display field. If you use the bar symbol in the SCREEN section, you must include a DELIMITERS statement in the INSTRUCTIONS section of the form specification. Use two identical symbols for the left and right delimiter in the DELIMITERS statement.
MASTER OF

Create a master/detail relationship between two tables when a row of one table (master) is associated with several rows of another table (detail).

**Table Diagram:**

```
  table1 ---- MASTER OF ---- table2
```

- `table1` is a table in the database that is designated as the master table.
- `table2` is a table in the database that is designated as the detail table.
- `MASTER OF` are required keywords.

**Usage**

- If you have used an alias in the TABLES section, you must use that alias to refer to the table in the master/detail relationship.
- You cannot include a temporary table in your table list.
- The master/detail relationship simplifies cross-table queries, especially when one row of `table1` is associated with several rows of `table2`.
- Master/detail relationships can be defined in both directions.
- If no explicit master/detail relationship exists, `PERFORM` displays an error message when you use the Master or Detail option.

Two examples from the sample form specification file follow:

```plaintext
customer master of orders;
orders master of items;
```

These master/detail relationships are useful because each customer can have many orders, and each order can have many items.
Additional examples are displayed in the following table:

<table>
<thead>
<tr>
<th>Master</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>projects</td>
<td>personnel</td>
</tr>
<tr>
<td>orders</td>
<td>items</td>
</tr>
<tr>
<td>agents</td>
<td>clients</td>
</tr>
<tr>
<td>parents</td>
<td>children</td>
</tr>
</tbody>
</table>

These master/detail relationships are useful where several staff members (personnel) work on the same project (projects), each purchase order (orders) contains more than one item (items), or a single agent (agents) has many clients (clients).

With a master/detail relationship defined in both directions, you can explore the database in the following way. Suppose you have a database consisting of personnel and projects tables. Each person is assigned to a single project, and each project has several people working on it. The screen form includes an INSTRUCTIONS section stating:

```
personnel master of projects;
projects master of personnel;
```

Assume that you want to query the database to find all employees who work with a particular employee, but you do not know on which project they work. When you identify and bring the particular employee to the screen (the personnel table is active) and select the Detail option, PERFORM moves to the PROJECT INFORMATION screen (the projects table is active) and displays the information on the employee’s project. If you then choose the Detail option, PERFORM selects all employees on that project and shifts to the PERSONNEL INFORMATION screen (the personnel table is active).
Control Blocks

Use control blocks to perform these functions:

- Control the cursor movement when you add or update a row.
- Check the value of data you enter against criteria that depend on other data that has already been entered.
- Modify the data in fields after Add, Update, and Query operations.
- Perform calculations on field values and enter the results into another field.
- Display aggregate information like averages and totals on columns in the current list. (The current list is the set of rows that results from a Query as modified by subsequent Add or Remove actions.)
- Call C functions from PERFORM. See Chapter 6 for details.

Each control block is either a BEFORE block or an AFTER block. Screen control actions can be taken either before or after PERFORM operations are completed. You can use BEFORE blocks with the Add, Update, and Remove operations. You can use AFTER blocks with the Add, Update, Query, Remove, and Display operations.

![Control Block Section Diagram]
BEFORE

Use a BEFORE control block to cause PERFORM to take a series of actions before it executes an operation.

BEFORE is a required keyword.
OF is a required keyword.
$table.column$ is a list of up to 16 names or aliases of database tables and/or names of columns. Depending on your operating system, the limit on the number of table names may be lower.

- The EDITADD and EDITUPDATE keywords refer to the act of editing during an Add and an Update, respectively. For EDITADD and EDITUPDATE, the actions are taken before PERFORM writes the row to the table.
- You can use the BEFORE REMOVE operation with the ABORT keyword (described on page 2-73) to prevent a user from removing the last row from a detail table.
AFTER

Use an AFTER control block to cause PERFORM to take a series of actions after it executes an option.

AFTER is a required keyword.

OF is a required keyword.

table.column is a list of up to 16 names or aliases of database tables and/or names of columns. Depending on your operating system, the limit on the number of table names may be lower.

- The ADD, UPDATE, QUERY, and REMOVE keywords correspond to the PERFORM options Add, Update, Query, and Remove, respectively. See Chapter 3, “The PERFORM Screen Transaction Processor,” for more information about the PERFORM options.
- The DISPLAY keyword refers to the display of fields after PERFORM executes Next, Previous, Query, or other options.
- EDITADD and EDITUPDATE differ from ADD and UPDATE. For EDITADD and EDITUPDATE, the actions are taken before PERFORM writes the row to a table. For ADD and UPDATE, they are taken after PERFORM writes the row to the table.
- You can list only table names or aliases, including displaytable in table.column following the ADD, UPDATE, QUERY, REMOVE, and DISPLAY keywords.
EDITADD and EDITUPDATE

The EDITADD and EDITUPDATE keywords give you the ability to perform one or more actions before or after you enter data into a field during an Add and an Update operation, respectively. The action occurs before the row is written to the table.

Usage

- If you are using EDITADD or EDITUPDATE in a BEFORE control block and the table.column contains the names of columns only, the BEFORE keyword instructs PERFORM to execute the actions when the cursor moves to the corresponding field, before you enter data.

- If you are using EDITADD or EDITUPDATE in an AFTER control block and the table.column contains the names of columns only, the AFTER keyword instructs PERFORM to execute the actions when you enter data into the corresponding field and press RETURN. PERFORM makes all the attribute-specified checks (such as INCLUDE, VERIFY, and so on) before executing the actions.

- When you specify a database table or alias instead of a column in a BEFORE block, PERFORM executes the actions before you enter any data into the form. Using this feature, you can make PERFORM enter defaults into fields and display comments depending on the active table.

- When you specify a database table or alias instead of a column in an AFTER block, PERFORM executes the actions after you enter all the data and press ESCAPE to complete the transaction, before the row is written to the database. Using this technique, you can make consistency checks of all the data entered and return to data entry if you find inconsistencies.

- In a BEFORE block, when you refer to a CHAR column that is split into more than one field, PERFORM executes the actions before each section of the displayed field. If you want these actions executed only after the first section of a split field, replace the BEFORE block of the split field with an AFTER block of the immediately preceding field.

- If you want the actions executed only after the last section of a split field, replace the AFTER block of the split field with a BEFORE block of the immediately succeeding field.
The following examples are taken from the sample form specification file at the end of this chapter. The syntax of the action statements used in these examples is described in “Action Syntax” on page 2-72.

after editadd editupdate of quantity
  let i19 = i18 * s15
  nextfield = o11

After you enter a value in the quantity field (using the Add or Update options), PERFORM calculates and places the value in the i19 (Total Price) column, and places the cursor in the o11 (Order Number) field.

before editadd editupdate of orders
  nextfield = o20

In this example, as soon as you indicate that you want either the Add or Update options when orders is the active table, PERFORM is instructed to move the cursor to the o20 (Customer P.O.) field. Without this instruction, the cursor would go first to the o11 (Order Number) field because it is the first orders field to appear in the ATTRIBUTES section of the form.
ADD

Use the ADD keyword to cause PERFORM to execute actions after the Add operation. The action occurs after the row is written to the table.

Usage

The main use of the ADD keyword involves keeping track of the number of rows written, computing statistics on the values entered into particular fields, and other bookkeeping operations.

The following example is from the sample form. The action statements used in this example are described in “Action Syntax” on page 2-72.

```after add update query of items
  if (total of i19) <= 100 then
    let d1 = 7.50
  else
    let d1 = (total of i19) * .04
  let d2 = (total of i19) + d1
```

Once you press ESCAPE following an Add, Update, or Query of the items table, PERFORM calculates values for the d1 and d2 fields and displays the values on the screen.
Use the UPDATE keyword to cause PERFORM to execute actions after the Update operation.

Usage

The main use of the UPDATE keyword involves keeping track of the number of rows written, computing statistics on the values entered into particular fields, and other bookkeeping operations.

The following example is from the sample form specification file. The action statements used in this example are described in “Action Syntax” on page 2-72.

```plaintext
after add update query of items
  if (total of i19) <= 100 then
    let d1 = 7.50
  else
    let d1 = (total of i19) * .04
  let d2 = (total of i19) + d1
```

Once you press ESCAPE following an Add, Update, or Query of the items table, PERFORM calculates values for the d1 and d2 fields and displays the values on the screen.
Use the QUERY keyword to cause PERFORM to execute actions after the Query operation.

### Usage

The main use of the QUERY keyword involves keeping track of the number of rows written, computing statistics on the values entered into particular fields, and other bookkeeping operations.

The following example is from the sample form specification file. The action statements used in this example are described in “Action Syntax” on page 2-72.

```plaintext
after add update query of items
  if (total of i19) <= 100 then
    let d1 = 7.50
  else
    let d1 = (total of i19) * .04
  let d2 = (total of i19) + d1
```

Once you press ESCAPE following an Add, Update, or Query of the items table, PERFORM calculates values for the d1 and d2 fields and displays the values on the screen.
Use the REMOVE keyword to cause PERFORM to execute actions before or after the Remove operation.

Usage

- The main use of the AFTER REMOVE operation involves keeping track of the number of rows removed, computing statistics on the values entered into particular fields, and other bookkeeping operations.
- Use the BEFORE REMOVE operation to cause PERFORM to take one or more actions before removing a row from a database table.

The following statement prints a message on the screen whenever a user selects the Remove option:

```
instructions
before remove of customer
    comments reverse
        "Remember to send a notice to the sales department"
```

You can use the BEFORE REMOVE operation with the ABORT keyword (described on page 2-73) to prevent a user from removing the last row from a detail table.
DISPLAY

Use the DISPLAY keyword to cause PERFORM to execute actions after any of the PERFORM operations that cause data to be displayed on the screen.

Usage

The following example is taken from the sample form specification file at the end of this chapter.

<table>
<thead>
<tr>
<th>after display of orders</th>
</tr>
</thead>
<tbody>
<tr>
<td>let d1 = 0</td>
</tr>
<tr>
<td>let d2 = 0</td>
</tr>
</tbody>
</table>

As soon as the data displays when the orders table is active, this control block instructs PERFORM to set the values in the d1 (Ship Charge) and d2 (Total Order Amount) fields to zero.
Action Syntax

This section provides the syntax of the following actions:

- **ABORT** exits to the **PERFORM** menu without making a change to the database.
- **COMMENTS** displays a message on the Status line.
- **IF-THEN-ELSE** performs other actions based on conditions among the values in the fields.
- **LET** assigns values to fields.
- **NEXTFIELD** moves the cursor to a specific field or exits to the **PERFORM** menu.

For these actions to compile properly, you must include them in a BEFORE or AFTER control block.
ABORT

Use the ABORT keyword in the INSTRUCTIONS section of a form specification to end a current Add, Update, or Remove action without altering the database and return to the PERFORM menu.

Usage

• The ABORT action compares to the NEXTFIELD EXITNOW action in the following respects:
  o With NEXTFIELD EXITNOW, PERFORM executes an Update, Remove, or Add, and then exits to the PERFORM menu. NEXTFIELD EXITNOW is equivalent to pressing the ESCAPE key.
  o With ABORT, PERFORM exits to the PERFORM menu without executing an Update, Remove, or Add. ABORT is equivalent to pressing the CONTROL-C key.

• You can use the ABORT keyword with the EDITADD, EDITUPDATE, and REMOVE options.

For example, suppose you maintain a master table with employee information and a detail table with information about employee projects (joined to the master table by employee number). Projects are added and deleted on a regular basis, and you want to ensure that all employees have projects. (It is an administrative or clerical error to remove the last detail row, thereby leaving the employee with no project.) You can use the ABORT keyword with the BEFORE REMOVE operation to call a C function that checks the number of rows in the detail table. If the current row is the last detail row, the operation aborts. For information about calling C functions from PERFORM, see Chapter 6, “C Functions in ACE and PERFORM.”
**LET**

Use the LET action to attach a value to a field tag for display on the form.

```plaintext
LET Option

LET  field-tag = expression

LET is a required keyword.
field-tag is the field tag of a display-only field, of a column named in the table.column list in the control block, or of a column belonging to each of the tables named in the table.column list.
expression is an expression as defined below.

**Usage**

- **FORMBUILD** gives an error if field-tag does not satisfy the preceding conditions.
- You can assign values only to fields corresponding to columns in the active table or to display-only fields.
- An expression is:
  - A field tag
  - A constant value
  - One of the aggregate functions followed by the phrase OF tagname, where tagname is the field tag of a database column and not the name of a display-only field. The aggregate function values are computed over the current list.

The aggregate functions are as follows:
COUNT the number of rows
TOTAL the arithmetic sum of the values of tagname
AVERAGE the average of the values of tagname (AVERAGE can also be written as AVG)
MAX the maximum value of tagname
MIN the minimum value of tagname
o The keyword TODAY that returns today’s date
o The keyword CURRENT that returns the current date and time
o Any combination of the preceding functions, combined with the arithmetic operators +, -, *, and /

For more information about aggregate functions, see Chapter 6 of the Informix Guide to SQL: Reference, Version 4.1.

• An expression can contain parentheses to make explicit the precedence of the arithmetic operators.

The following example is from the sample form:

```plaintext
after add update query of items
    if (total of i19) <= 100 then
        let d1 = 7.50
    else
        let d1 = (total of i19) * .04
    let d2 = (total of i19) + d1
```

Once you press ESCAPE following an Add, Update, or Query of the items table, PERFORM calculates values for the d1 (Ship Charge) and d2 (Total Order Amount) fields. If the value of the i19 (Total Price) field (all items in the order) is less than or equal to 100, then the value of the d1 field (Ship Charge) is set to 7.50; otherwise the value is set to the sum of the i19 (Total Price) field times .04.

The value of the d2 (Total Order Amount) field is set to the sum of the i19 (Total Price) field plus the value in the d1 (Ship Charge) field.

Additional examples of the uses of the LET statement follow:

```plaintext
let f1 = f2 * 1.065
let s2 = "default string"
let f3 = (f5 + f8) * f7
let ftax = 0.065 * f_price
let f9 = average of f_price
let yr = (today - hdate)/365
```
When NLS is active, the conversion of numeric or monetary values to character strings through the LET statement is influenced by environment variables LC_NUMERIC and LC_MONETARY. Both the default conversion and the conversion with a USING clause insert locale-specific separator and currency symbols into the created strings, not US English symbols.
**NEXTFIELD**

When you use the EDITADD or EDITUPDATE options, use the NEXTFIELD action to direct the movement of the cursor. The NEXTFIELD action overrides the default progression as determined by the ATTRIBUTES section of the form specification file.

```
NEXTFIELD = field-tag
```

**field-tag** is the field tag that corresponds to a database column in the active table.

**EXITNOW** is a keyword value for NEXTFIELD that ends the current editing.

**Usage**

- You cannot change the active table by using the NEXTFIELD action to move the cursor to the field of a column in a new table.
- The NEXTFIELD EXITNOW action compares to the ABORT action in the following respects:
  - With NEXTFIELD EXITNOW, **PERFORM** executes an Update, Remove, or Add, and then exits to the **PERFORM** menu. NEXTFIELD EXITNOW is equivalent to pressing the ESCAPE key.
  - With ABORT, **PERFORM** exits to the **PERFORM** menu without executing an Update, Remove, or Add.
- Since the NEXTFIELD action controls the movement of the cursor, it is effective only after the EDITADD and EDITUPDATE options.

An example from the sample form follows:

```plaintext
before editadd editupdate of orders
  nextfield = o20
```
In this example, as soon as you indicate that you want either the Add or Update options when **orders** is the active table, **PERFORM** is instructed to move the cursor to the **o20** column (Customer P.O.). Without this instruction, the cursor would go first to the **o11** field (Order Number) because it is the first **orders** field to appear in the ATTRIBUTES section of the form.
COMMENTS

Use the COMMENTS action to display a message on the Status line of the screen. This use of COMMENTS contrasts with the COMMENTS attribute included in the ATTRIBUTES section that writes a message on the Comment line.

**COMMENTS** is the keyword that directs PERFORM to write a message on the Status line.

**BELL** is the keyword that directs PERFORM to ring the bell as it writes the message.

**REVERSE** is the keyword that directs PERFORM to write the message in reverse video. The default is normal video.

**mstring** is the message (string) and must be enclosed in quotation marks. It must fit on one screen line and on one line of the form specification file.

**Usage**

- If you use the REVERSE keyword, you must take care on some monitors to account for the space required at the beginning of the line for the control characters.

- The message is cleared at the next keystroke. Since PERFORM writes a message whenever a row is written, an Update or an Add is aborted, or a Query or Remove is made, this action is useful only for the EDITADD and EDITUPDATE keywords.
IF-THEN-ELSE

Use the IF-THEN-ELSE action to take actions that depend on the values in the displayed fields.

IF is a required keyword.

boolean-expression is a Boolean expression involving field tags that can take on the values true and false.

THEN is a required keyword.

t-action is the action or actions to be taken if boolean-expression is true.

ELSE is a keyword.

f-action is the action or actions to be taken if boolean-expression is false.

Usage

- A Boolean expression is a combination of logical comparisons (=, <>, >, <, >=, <=) and logical operations (AND, OR, NOT) among expressions as previously defined. You can also use the operators IS NULL and IS NOT NULL.

- For CHAR type fields only, a Boolean expression may be also of the form
  
  field-tag MATCHES "string"
  
  where string must be enclosed within quotation marks and can include wildcard characters as defined in Chapter 6, “Syntax” of the Informix Guide to SQL: Reference, Version 4.1.

- t-action and f-action are either single actions as defined in this section or more than one such action between the keywords BEGIN and END.

The following example illustrates a simple IF-THEN-ELSE with one true and one false action:

```sql
if (f1 * f2 > 200) then LET f5 = -f4
else LET f5 = -5
```
A more complex example from the sample form specification file follows:

```sql
after add update query of items
  if (total of i19) <= 100 then
    let d1 = 7.50
  else
    let d1 = (total of i19) * .04
  let d2 = (total of i19) + d1
```

Once you press ESCAPE following an Add, Update, or Query of the items table, PERFORM calculates values for the d1 (Ship Charge) and d2 (Total Order Amount) fields. If the value of the Total Price field (all items in the order) is less than or equal to 100, then the value of the d1 field (Ship Charge) is set to 7.50; otherwise, the value is set to the sum of the Total Price field times .04.

The value of the d2 (Total Order Amount) field is set to the sum of the Total Price field (all items in the order) plus the value in the Ship Charge field.

When NLS is active, the results of character comparisons and of MATCHES, LIKE and BETWEEN expressions containing character arguments are dependent on the LC_COLLATE setting. Refer to Appendix C, “Native Language Support Within INFORMIX-SQL.”
The SAMPLE Form Specification File

The sample form specification file was designed for entering and maintaining data on customers and orders listed in the stores2 database. The first screen displays information from the customer table and is labeled CUSTOMER INFORMATION. The second screen is used to enter and retrieve information about customer orders.

database stores2
screen

================================================================================
CUSTOMER INFORMATION:

Customer Number: [c1]
Company: [c4]
First Name: [c2] Last Name: [c3]
Address: [c5]
[c6]
City: [c7] State: [c8] Zip: [c9]
Telephone: [c10]

================================================================================

screen

================================================================================
CUSTOMER NUMBER: [c1] COMPANY: [c4]

ORDER INFORMATION:
Order Number: [o11] Order Date: [o12]
Stock Number: [i13] Manufacturer: [i16]
Description: [s14]
Unit: [s16]
Quantity: [i18]
Unitprice: [s15]
Total Price: [i19]

SHIPPING INFORMATION:
Customer P.O.: [o20] Ship Charge: [d1]
Backlog: [a] Total Order Amount: [d2]
Ship Date: [o21]
Date Paid: [o22]
Instructions: [o23]

end
tables
customer items stock
orders manufact

attributes
c1 = *customer.customer_num = orders.customer_num;
c2 = fname,
   comments = "Please enter initial if available";
c3 = lname;
c4 = company, reverse;
c5 = address1;
c6 = address2;
c7 = city;
c8 = state, upshift, autonext,
   include = ("CA","OR","NV","WA"),
   default = "CA";
c9 = zipcode, autonext;
c10 = phone, picture = "###-###-####x####";
o11 = *orders.order_num = items.order_num;
o12 = order_date, default = today, format = "mm/dd/yyyy";
i13 = items.stock_num;
   = *stock.stock_num, noentry, noupdate, queryclear;
i18 = items.manu_code, lookup m17 = manufact.manu_name
   joining *manufact.manu_code, upshift, autonext;
   = *stock.manu_code, noentry, noupdate, upshift, autonext, queryclear;
s14 = stock.description, noentry, noupdate;
s16 = stock.unit_descr, noentry, noupdate;
s15 = stock.unit_price, noentry, noupdate;
i18 = items.quantity, include = (1 to 50),
   comments = "Acceptable values are 1 through 50";
o20 = po_num, required,
   comments = "If no P.O. Number enter name of caller";
a = backlog, autonext;
o21 = ship_date, default = today, format = "mm/dd/yyyy";
o22 = paid_date, format = "mm/dd/yyyy";
o23 = ship_instruct;
d1 = displayonly type money;
d2 = displayonly type money;

instructions
customer master of orders;
orders master of items;
composites <items.stock_num, items.manu_code>
   *<stock.stock_num, stock.manu_code>
   before editadd editupdate of orders
nextfield = o20
   before editadd editupdate of items
nextfield = i13
after editadd editupdate of quantity
let i19 = i18 * s15
nextfield = o11
The FORMBUILD Transaction Form Generator

after add update query of items
  if (total of i19) <= 100 then
    let d1 = 7.50
  else
    let d1 = (total of i19) * .04
  let d2 = (total of i19) + d1
after display of orders
  let d1 = 0
  let d2 = 0
end

The CUSTOMER INFORMATION Screen

The CUSTOMER INFORMATION screen contains fields for the entry and display of all the columns in the customer table. You can use this screen to add or remove a customer from the database.

Note the following points about this screen:

• The customer table is joined with the orders table at the customer_num column. The customer table is the dominant table in this join.
• A Comment line appears when the cursor moves into the c2 (First Name) field.
• The c4 (Company) field appears in reverse video.
• PERFORM automatically enters uppercase letters into the c8 (State) field. The default value for the field is CA, and only abbreviations for four states are allowed.
• A character pattern is specified for the c10 (Telephone) field.

The ORDER INFORMATION Screen

The second screen contains fields drawn from the orders, items, stock, and manufact tables. This screen is used to enter information about a customer’s order. Shipping information (purchase order number, instructions, date sent, and so on), and order information (order number, date, items included in the order, total prices on each item, and so on) are entered on this screen.
Note the following points about this screen:

- The `c1` (Customer Number) and `c4` (Company) fields are repeated from the CUSTOMER INFORMATION screen.
  The verify join between the `customer.customer_num` column and the `orders.customer_num` column prevents the assignment of an order to a nonexisting customer. When the `orders` table is active, no value can be entered into a field that does not already exist in the `customer` table.
- The `orders` and `items` tables are joined at the `order_num` column. This verify join prevents the assignment of an item to a nonexisting order number. When the `items` table is active, no value can be entered into the field that does not already exist in the `orders` table.
- The `o12` (Order Date) field has an assigned format and defaults to the current date.
- The `stock_num` column in the `items` table is joined with the `stock_num` column in the `stock` table. This is a verify join.
- The `manu_code` column in the `items` table is joined with the `manu_code` column in the `stock` table. This is a verify join.
- The `i13` (Stock Number) and `i16` (Manufacturer) fields are members of a composite join between the `items` and `stock` tables. This is a composite verify join, so no values can be entered in the `stock_num` and `manu_code` fields (when the `items` table is active) that do not already exist in those two columns in the `stock` table. This precludes entry of stock numbers and manufacturer codes that individually exist in the database but, as a composite, do not correspond to a unique row in the `stock` table.
  To specify a unique row in the `stock` table requires both the `stock_num` and `manu_code`. For example, the `stock` table contains three rows with the stock number 1 and four rows with the manufacturer code HRO.
  Knowing the stock number or manufacturer code alone does not allow you to locate a unique row. You need both the stock number (1) and the manufacturer code (HRO) to specify a unique row (baseball gloves produced by Hero) in the table.
- Once the `i13` (Stock Number) and `i16` (Manufacturer) fields are filled, `PERFORM` can locate the corresponding unique row in the `stock` table. The `s14` (Description), `s16` (Unit), and `s15` (Unitprice) fields automatically display this information.
- The `i16` (Manufacturer) field is involved in a lookup join that locates the appropriate manufacturer name in the `manufact` table and places this information in the `m17` field.
• The i18 (Quantity) field allows the entry of values 1 through 50 only, and it displays a comment when the cursor moves into the field. This helps to prevent mistaken entries of an extra digit (for example, 100 in place of 10).

• The cursor does not visit the o11 (Order Number) field when the orders table is the active table because the order_num column in the orders table is a SERIAL data type.

• The following entry in the INSTRUCTIONS section tells PERFORM that when the orders table is the active table, the cursor first goes to the o20 (Customer P.O.) field, rather than the o12 (Order Date) field:

```plaintext
before editadd editupdate of orders
    nextfield = o20
```

• The default value for field o21 (Ship Date) is set to today.

• The d1 and d2 fields do not correspond to any database columns. One of these is the Ship Charge field and is the total shipping charge for the entire order. The second is the Total Order Amount field and is the total charge for the entire order, including all items and the shipping charge. PERFORM calculates each field automatically. The following entry in the INSTRUCTIONS section tells PERFORM that if the total of all values in the field i19 for this order is less than or equal to 100, then set the value in the field d1 to 7.50. If the total of all values in the field i19 for this order is greater than 100, then set the value in field d1 to the product of this total times .04.

```plaintext
after add update query of items
    if (total of i19) <= 100 then
        let d1 = 7.50
    else
        let d1 = (total of i19) * .04
    let d2 = (total of i19) + d1
```

The Total Order Amount (the d2 field) is the sum of all values in Total Price (the i19 field) for the order plus the Shipping Charge (the d1 field).

The total price of an individual item in a customer order is calculated automatically by FORMBUILD. This field is filled in by PERFORM as soon as you supply information for the fields i13 (the Stock Number of the item), i16 (the Manufacturer of the item), and i18 (the Quantity of the item). PERFORM can do this because, in the INSTRUCTIONS section, PERFORM is told to calculate the value of the Total Price (the i19 field).
based on the values entered into the \textit{i18} (the Quantity) and \textit{s15} (the Unit Price) fields.

- The \textbf{customer} table is the master of the \textbf{orders} table. You can easily query the \textbf{orders} table (and locate all orders placed by a customer) based on the current row in the \textbf{customer} table by selecting the Detail option.

- The \textbf{orders} table is the master of the \textbf{items} table. You can easily query the \textbf{items} table (and locate all items contained in each order) based on the current row in the \textbf{orders} table by selecting the Detail option.
Chapter 3

The PERFORM Screen Transaction Processor

Chapter Overview 3

Running PERFORM 3
  Accessing PERFORM from the Main Menu 4

The PERFORM Screen 5
  The Information Lines 6
  The Screen Form 8
  Status Lines 9
  Running Operating System Commands from PERFORM 9

Entering Data 9
Data Types 9
Special Functions 12
Positioning the Cursor 12
Field Editing 13
Using the Multiline Editor 14
Display Field Order 15

Data Checking 15
  User Access Privileges 16

The Current List 18
Menu Options 18
ADD 19
CURRENT 20
DETAIL 21
EXIT 22
MASTER 23
NEXT 24
OUTPUT 25
PREVIOUS 28
QUERY 29
REMOVE 32
SCREEN 33
TABLE 34
UPDATE 35
VIEW 36
Chapter Overview

PERFORM is an INFORMIX-SQL (I-SQL) program designed to streamline data entry and retrieval. After you create a screen form with FORMBUILD, you can use the form with PERFORM to query and modify the data in a database. See Chapter 2, “The FORMBUILD Transaction Form Generator,” for information about designing and building screen forms.

This chapter is divided into two parts. The first part describes the following PERFORM procedures:

- Accessing PERFORM from the Main menu
- Running operating system commands while using PERFORM
- Using special keys to position the cursor and edit text
- Entering and editing data on the screen
- Data checking with PERFORM
- Controlling user privileges in PERFORM
- Using the current list

The second part of this chapter discusses each PERFORM option. The options are listed in alphabetical order.

Running PERFORM

PERFORM uses the file that FORMBUILD generates when you compile a form specification file. This file must be in your working directory or a directory included in your DBPATH environment variable.

You can use PERFORM from the I-SQL Main menu or directly from the operating system. See Appendix H, “Accessing Programs from the Operating System,” for information about command-line usage.
Accessing PERFORM from the Main Menu

Select the Form option on the I-SQL Main menu. The FORM menu is displayed.

FORM: Run Modify Generate New Compile Drop Exit
Use a form to enter data or query a database.
------------------------------------------------ Press CONTROL-W for Help ----

Select the Run option on the FORM menu. The RUN FORM screen is displayed, with a list of available screen forms.

RUN FORM >>
Choose a form with Arrow Keys, or enter a name, then press Enter.
------------------------------------------------ Press CONTROL-W for Help ----
customer
orderform
sample
Type the name of the form you want to use, or use the Arrow keys to highlight your choice on the screen. Press RETURN. The form you select appears on the screen with the PERFORM menu, as shown in the following figure:

The PERFORM Screen

The PERFORM screen is divided into three sections:

- The first two lines of the screen (the Information lines) display the PERFORM menu options, a message describing the highlighted option, and the number and name or alias of the active table.
- The middle section of the screen (the screen form) displays the form you selected.
- The bottom two lines of the screen (the Comment line and Status line) display PERFORM messages, as well as comments specified in the form file.
The Information Lines

The PERFORM menu is two pages long. The first Information line displays a list of menu options; the second Information line describes the current option and indicates the number and name or alias of the active database table. The next two screens illustrate the Information lines on the two-page PERFORM menu.

The ellipsis on the first menu page indicates that additional menu items are available on the second menu page. The ellipsis on the second menu page indicates that additional menu items are available on the previous menu page.

Note: The number of options that appears on the first menu page depends on the character capacity of your screen. The two-page screens displayed here demonstrate a terminal or monitor with an 80-character screen. Terminals with a larger character capacity show more options on the first menu page.

Use the SPACEBAR or the Arrow keys to move the highlight onto the menu options. When you move the highlight past the first or last menu option on a page, the alternate menu page appears; the menu does not scroll. The highlight never rests on the ellipses; when you move the highlight past the last or first option on each screen page, the next PERFORM menu page appears.

PERFORM is a menu-driven program; to work with the data on the screen, you select one of the menu options. You select a menu option on the first Information line by using the Arrow keys or the SPACEBAR to position the
highlight on a menu option and then pressing RETURN, or by typing the first letter of the menu option. PERFORM immediately displays the screen for the selected option. If you want to return to the menu without making any entries, press the Interrupt key. This key is DEL on most systems.

The PERFORM screen has 14 menu options:

- **Query** retrieves rows from the database based on search values you enter on the form and stores the rows in the current list.
- **Next** displays the next row in the current list.
- **Previous** displays the previous row in the current list.
- **View** displays the contents of a field of data type TEXT or BYTE (applies to INFORMIX-OnLine Dynamic Server only). See Appendix I, “Using the INFORMIX-OnLine Dynamic Server,” for more information on using this feature.
- **Add** adds data to the database.
- **Update** modifies data in the database.
- **Remove** deletes a row from a database table.
- **Table** displays a different table in the form.
- **Screen** displays a different screen page of the form.
- **Current** restores the base current list in multitable queries and displays the most up-to-date version of the displayed row in multiuser environments.
- **Master** displays the master table of the active table.
- **Detail** displays the detail table of the active table.
- **Output** writes the selected row or rows to an operating system file in either Screen or Unload format.
- **Exit** leaves PERFORM.

The Information lines also indicate the number and name or alias of the active table. Every table included in the screen form has a table number assigned according to the order in which display field tags (including joins) for the table first appear in the ATTRIBUTES section of the form specification file. This number appears next to the table name in the right-hand corner of the second Information line when the table is active. The table number is useful for nonsequential moves to another table using the Detail and Table options.
The Screen Form

The screen form consists of one or more display fields in which PERFORM displays—and you enter—data. Each display field on a screen form corresponds to one or more of the database columns or to a display-only field specified in the form file. Unless you specify alternative delimiters in the form specification file, active display fields are surrounded on the screen by brackets ( [ ] ). Fields with no delimiters are not active; values may appear in them if they are LOOKUP fields or display-only fields, but you cannot enter data into them.

A screen form may be one page or several pages long and can contain columns from several tables. All tables included in a form must be part of the same database.

Here is how the PERFORM screen looks when you use the customer form included with the demonstration database:
The PERFORM Screen

Status Lines

PERFORM uses the last two lines of the screen to display PERFORM error messages, as well as any messages generated by the form itself.

The two entries were not the same - please try again.

Running Operating System Commands from PERFORM

To run operating system commands from the PERFORM menu, press the exclamation point ( ! ) key. PERFORM displays the exclamation point at the bottom of the screen. Enter an operating system command and press RETURN. PERFORM displays the results of your command and then the following message:

Press RETURN to continue

Press the RETURN key to leave the operating system and return to PERFORM.

Entering Data

Use the Add and Update options to enter data directly into the database from the screen form. You must enter data of the type specified when the table was created—dates in DATE fields, money in MONEY fields, and so on. If you make a mistake entering data, you can use the field-editing keys to correct it. (See the discussion of field editing on page 3-13.)

Data Types

The following list discusses the kind of data to enter for each data type. If you enter data of the wrong type, PERFORM displays the following message on the Status line:

Error in field
Enter an acceptable value or press the Interrupt key to cancel the option you are using. You can use the Info options on the SQL or TABLE menu to find out the data type for each column in a table. See Chapter 9, “Database Structure and Integrity,” in the INFORMIX-SQL User Guide for more information about data types.

**CHAR[n]** Enter letters, numbers, and symbols. During an Add or Update, the character data string may be as long as the display field.

**CHARACTER** is a synonym for CHAR.

**SMALLINT** Enter a whole number from -32,767 to +32,767.

**INTEGER** Enter a whole number from -2,147,483,647 to +2,147,483,647.

**INT** is a synonym for INTEGER.

**SERIAL** `PERFORM` assigns SERIAL values automatically, so you never add data to a SERIAL field or update it. However, you can enter search values in SERIAL fields when you use the Query option.

**DECIMAL[m,n]** Enter decimal numbers. The format of the number (number of places to the right and left of the decimal point) depends on the format you specified when you created the database table. If you enter a number with too many spaces after the decimal point, `PERFORM` rounds it off.

**DEC** is a synonym for DECIMAL.

**NUMERIC** is a synonym for DECIMAL.

**MONEY** Enter dollars-and-cents amounts without dollar signs and commas (for example, 4254.30 not $4,254.30). When you press RETURN, `PERFORM` automatically adds a dollar sign.

**SMALLFLOAT** Enter floating point numbers with up to 7 significant digits. `PERFORM` sometimes introduces a slight discrepancy when you type numbers into SMALLFLOAT fields. The entry 1.1, for example, might display as 1.11000001 after you press RETURN or ESCAPE. This occurs because of the way a computer stores numbers internally, and only affects you when a precision of more than 7 digits is required.

**REAL** is a synonym for SMALLFLOAT.
FLOAT[(\(n\)]) Enter floating-point numbers with up to 14 significant digits. The discrepancies mentioned for SMALLFLOAT data also apply to this data type if you require a precision of more than 14 digits.

DATE Enter dates in the form \([mm]\text{m} [d]\text{d} [yy]\text{y}\), with any nonnumeric characters as optional dividers (for May 2, 1985, you could enter May 2 85, 08/02/85, 8.2.85, or 08 02 1985).

DATETIME Enter DATETIME values in the form 
\([yyyy] y-[m]\text{m}-(d)\text{d} [h]\text{h}:[m]\text{m}:[s]\text{s}.[ffff].\) You must separate the fields as follows: YEAR-MONTH, MONTH-DAY, DAY(space)HOUR, HOUR:MINUTE, MINUTE:SECOND, SECOND.FRACTION.

INTERVAL Enter INTERVAL values in the form 
\([yyyy] y-[m]\text{m} or [d]d [h]\text{h}:[m]\text{m}:[s]\text{s}.[ffff].\) You must separate the fields as follows: YEAR-MONTH, DAY(space)HOUR, HOUR:MINUTE, MINUTE:SECOND, SECOND.FRACTION.


In an NLS environment, the settings in LC_NUMERIC, LC_MONETARY, and LANG determine the numeric and decimal separators. These settings change the separators displayed on the screen in a numeric or monetary field. For example, 1234.56 will display as 1234,56 in a French or German locale. Also, in the French or German locale values input by the user will be expected to contain commas, not periods, as decimal separators.

The installation of message files in a subdirectory of $INFORMIXDIR/msg and subsequent reference to that subdirectory by way of the environment variable DBLANG causes DATETIME and DATE values to display locale-specific month name abbreviations on the form. Similarly, month name values are expected to be valid locale specific names when input. For example, the month name June in a French locale would have to be input as the month name abbreviation Jui, which stands for Juin (the French word for June), rather than Jun. If you are unsure about the correct month name, specify months numerically.
Special Functions

As you enter data or a query, three special functions are available by using selected keys.

<table>
<thead>
<tr>
<th>Function</th>
<th>Key Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Help</td>
<td>The CONTROL-W key displays a HELP screen that contains a short summary of special keys, control keys, editing keys, and other information about PERFORM.</td>
</tr>
<tr>
<td>Execute</td>
<td>The ESCAPE key runs the option you select. To add a new row, type a to select the Add option, enter the information for the row, and press ESCAPE to add the row to the database.</td>
</tr>
<tr>
<td>Interrupt</td>
<td>On most systems the DELETE or CONTROL-C key interrupts or cancels the option you are using. For example, if you select Add when you really want Query, press CONTROL-C, and then select the Query option.</td>
</tr>
</tbody>
</table>

Positioning the Cursor

You can use the following keys to position the cursor on the screen:

<table>
<thead>
<tr>
<th>Movement</th>
<th>Key Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next Field</td>
<td>The RETURN and [ ↓ ] keys move the cursor to the next field.</td>
</tr>
<tr>
<td>Backspace</td>
<td>The BACKSPACE and [ ← ] keys move the cursor backward one character at a time without erasing any text. Pressing either key at the beginning of a field moves the cursor to the previous field.</td>
</tr>
<tr>
<td>Forward</td>
<td>The [ → ] key moves the cursor forward one character at a time without erasing any text. Pressing the [ → ] key at the end of a field moves the cursor to the next field.</td>
</tr>
<tr>
<td>Fast Forward</td>
<td>The CONTROL-F key moves the cursor down the screen rapidly, stopping in the first field on each line. Use CONTROL-F to move quickly to the bottom of a form that contains many fields.</td>
</tr>
<tr>
<td>Fast</td>
<td>The CONTROL-B key moves the cursor up the screen rapidly, stopping at Backspace the last field on each line. Use CONTROL-B to move quickly to the top of a long form.</td>
</tr>
</tbody>
</table>
Field Editing

If you make a mistake entering data in a field, you can correct it by backspacing and retyping. However, you might find it faster to use the PERFORM field-editing feature. You can use two editing modes to enter data into a field:

- In typeover mode, the characters you type replace existing data. For example, you could use typeover mode to change “Sports ‘R Us” to “Abe’s Sporting Goods.”
- In insert mode, the characters you type push existing data to the right. For example, you could use insert mode to add an i to Richard.

Whenever the cursor enters a field, PERFORM is in typeover mode; you must press the Insert key to activate the insert mode. Press the Insert or CONTROL-A key a second time to return to typeover mode. Move the cursor into a new field, and you are automatically in typeover mode.

Use the following keys to edit data that appears in a field:

<table>
<thead>
<tr>
<th>Function</th>
<th>Key Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backspace</td>
<td>The BACKSPACE and [ ← ] keys move the cursor back one character at a time without erasing any text. If you press either key at the beginning of a field, the cursor moves back to the previous field.</td>
</tr>
<tr>
<td>Forward</td>
<td>The [ → ] key moves the cursor forward one character at a time without erasing any text. If you press [ → ] at the end of a field, the cursor moves forward to the next field.</td>
</tr>
<tr>
<td>Delete a Character</td>
<td>Delete or CONTROL-X deletes the character beneath the cursor. The cursor remains in place, and text shifts over to fill the space that was occupied by the deleted character.</td>
</tr>
<tr>
<td>Change Mode</td>
<td>Insert or CONTROL-A shifts between insert and typeover mode. When you access PERFORM, you are in typeover mode.</td>
</tr>
<tr>
<td>Delete Forward</td>
<td>CONTROL-D deletes everything from the current cursor position to the end of the field.</td>
</tr>
<tr>
<td>Repeat Data</td>
<td>CONTROL-P enters the most recently displayed value in a field. When you use the Add option to enter several rows in which one or more fields contain the same data, you can avoid retyping the data by pressing CONTROL-P. When you use the Update option, CONTROL-P restores the value that appeared in a field before you modified the field.</td>
</tr>
<tr>
<td>Clear Screen</td>
<td>CONTROL-C clears any search criteria you have entered with the Query option.</td>
</tr>
</tbody>
</table>
Using the Multiline Editor

A multiline editor is available for editing long character fields, also called multiline fields. A multiline field has more than one physical field, as shown in the following form specification file:

```plaintext
database reference
screen
{
   TITLE: [b001 ]
   AUTHOR: [b002 ]
   SYNOPSIS: [b003 ]
   [b003 ]
   [b003 ]
   [b003 ]
}
tables
booktab
attributes
b001 = refdpt.booktab.title
b002 = refdpt.booktab.author
b003 = refdpt.booktab.synopsis,WORDWRAP COMPRESS
```

You invoke the multiline editor by using the WORDWRAP attribute (see “WORDWRAP” on page 2-53 for detailed information). Most keys function the same in multiline editing as they do in normal field editing, with a few exceptions:

<table>
<thead>
<tr>
<th>Key</th>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RETURN</td>
<td>The RETURN key causes the cursor to leave the current multiline field and move to the first position in the next field.</td>
</tr>
<tr>
<td>Up Arrow</td>
<td>The [↑] key moves the cursor one line up within the same multiline field. The cursor moves to the left if necessary to avoid editor blanks (see page 2-53). If the cursor is on the top line of a multiline field, the [↑] key moves the cursor to the first character position in the preceding field.</td>
</tr>
<tr>
<td>Down Arrow</td>
<td>The [↓] key moves the cursor one line down within the same multiline field. The cursor moves to the left if necessary to avoid editor blanks. If the cursor is on the bottom line of a multiline field, the [↓] key moves the cursor to the first character position in the following field.</td>
</tr>
<tr>
<td>TAB</td>
<td>If you are in typeover mode, the TAB key moves the cursor to the next field.</td>
</tr>
</tbody>
</table>
CONTROL-N inserts a newline character, causing subsequent text to move to the first position in the following line of the same multiline field. This could cause text to ripple down toward the bottom of the field, and you might lose the text that was in the last line of the field.

Display Field Order

The cursor ordinarily moves through the display fields in the order in which their field tags are listed in the ATTRIBUTES section of the form file. You can modify this order by using a NEXTFIELD statement in the INSTRUCTIONS section of the form file.

Data Checking

The attributes and instructions in the form file can affect data entry, data storage, data display, and cursor movement when you use the Add, Update, and Query options. If you get undesired displays or cursor movement, you can modify the form file. The effects of some attributes and instructions are listed here, followed by the relevant options. (See Chapter 2, “The FORMBUILD Transaction Form Generator,” for details about attributes and instructions.)

- The case of the character data on the screen is different from what you type. Check for UPSHIFT and DOWNSHIFT attributes (Add, Update, Query).
- SMALLFLOAT or FLOAT data on the screen is different from what you type. Check for a FORMAT attribute that causes rounding off by specifying the number of places to the right and left of the decimal point (Add, Update).
- PERFORM displays the following message:

  This value is not among the valid possibilities.

  Check for an INCLUDE attribute that specifies acceptable values and ranges of values (Add, Update).
- The terminal beeps and does not echo your entry on the screen. Check for a PICTURE attribute that limits data entry to a specified pattern of variables and literals (Add, Update).
- The cursor skips over a bracketed display field. If the field is not a SERIAL field, check for a NOENTRY attribute (for Add only), a NOUPDATE attribute (for Update only), or a NEXTFIELD action (both Add and Update).
The PERFORM Screen Transaction Processor

- **PERFORM** displays the following message:
  
  This field requires an entered value.

  Check for a REQUIRED attribute (Add). You must explicitly enter all values for a field with the REQUIRED attribute unless you have specified a value with DEFAULT.

- **PERFORM** displays the following message:
  
  Please type again for verification.

  Check for a VERIFY attribute (Add, Update).

- Data you enter appears on the screen justified to the right. Check for a RIGHT attribute (Add).

- Number data appears on the screen justified to the right and padded with leading zeros. Check for a ZEROFILL attribute (Add).

- A value you did not enter appears in a field. Check for a DEFAULT attribute, a PICTURE attribute with literals (Add, Query), a joined field (Add, sometimes Query), or a LET action (Add).

- The cursor moves automatically to the next field after this field is full. Check for an AUTONEXT attribute (Add, Update).

- A line of text appears on the screen. Check for a COMMENTS attribute (Add, Update, Query).

- A line of text appears on the screen, in regular or reverse video, and/or the terminal bell rings. Check for a COMMENTS action (Add, Update, Query).

- Data stores automatically before you press ESCAPE. Check for a NEXTFIELD EXITNOW action (Add, Update).

- **PERFORM** displays the following message:

  This is an invalid value--it does not exist in tablename.

  Check for a verify join (Add, Update).

### User Access Privileges

Access privileges controlled by the GRANT and REVOKE statements can affect your ability to display, enter, modify, and remove data for a table or a display field. A message like the following means that you do not have access privileges:

Permission not granted to allow update
Use the Info option on the SQL or TABLE menu to find out the access privileges for a particular table.

The Current List

The current list is a temporary storage area where PERFORM stores the results of a query. It can hold from one row to all the rows in a database. Whenever you select the Query option, PERFORM erases the existing current list to make room for new query results.

The Query, Next, Previous, Remove, and Update options all involve the current list. The Query option finds all rows that satisfy the search conditions and puts them in the current list. The Next and Previous options step through the rows in the current list in sequential order. The Update and Remove options can only work with rows in the current list.

Menu Options

The menu options you can use with PERFORM are described in detail on the following pages. They are listed in alphabetical order, rather than menu order, for easy reference.
ADD

Use the Add option to create new rows in the active table. You can type data on a screen form, review it, edit it, and store it in the database.

Use the following procedure for the ADD option:

1. Type a to select the Add option. PERFORM clears all data from the screen form except joined fields and some display-only fields, displays spaces or default values in the fields, and positions the cursor in the first field.
2. Fill in the form with the values you want to enter. If you enter data inappropriate to the data type of the display field, PERFORM displays the following message:
   
   Error in field
   
   You cannot move on to the next display field until you correct the entry.
3. Press ESCAPE to store the row, or the program Interrupt key to cancel the addition and redisplay the PERFORM menu options.

Usage

- If PERFORM displays this message it does not store a row when you press ESCAPE:

   Could not insert new row - duplicate value in a unique index column

   You are trying to enter a duplicate value where it is not permitted. Use the Info option on the SQL or TABLE menu (or execute an INFO statement) to check for unique indexes.

- Tables may be fully or partly unavailable to you because another user has invoked the LOCK TABLE statement or because another user is updating a row that you attempt to update or remove. In such a case, PERFORM displays an error message.

- PERFORM sometimes introduces a slight discrepancy when you enter numbers in SMALLFLOAT or FLOAT fields. The entry 1.11, for example, might display as 1.11000001 after you press RETURN or ESCAPE. This discrepancy occurs because of the way a computer stores numbers internally.

Related Options

Update, Remove
CURRENT

The Current option rereads and redisplays the current row in the current list for the active table.

Use the following procedure for the Current option:

1. Type c to select the Current option.
2. PERFORM displays the most up-to-date version of the screen you were looking at before you moved to another table.

Usage

The Current option is useful in two situations:

- In a LAN environment, another user can modify the information corresponding to a display field on your screen. When you use the Current option, PERFORM rereads the row, displaying the most recent information.
- When a form includes a join field, each table represented on the screen form has its own current list. Looking at the information in the active table might make you “lose your place” in one or more of the other current lists. The Current option returns you to your original position in the current list of the active table.
DETAIL

The INSTRUCTIONS section of the form file may include one or more master-detail table relationships for tables with join fields to simplify multitable queries. The Detail option automatically selects, displays, and queries the detail table of the active table.

Use the following procedure for the Detail option:

1. Enter d to select the Detail option.
2. If the active table has no detail table, PERFORM displays an error message. If the active table has one or more detail tables specified, PERFORM locates and displays the first detail table of the active table, no matter what the absolute table number of the detail table is.
3. PERFORM automatically runs a query on the detail table, using the current values in the fields in the master table that join the fields in the detail table as search values. PERFORM then puts the rows that satisfy the query conditions in the current list of the detail table. You can use the Next and Previous options to examine the rows. If no rows are found, PERFORM displays the following message:
   There are no rows satisfying the conditions.
4. Type m to make the master table the active table again. You will see the first master-table row with join-field values that match the current values in the detail table.

Usage

- If no explicit master/detail relationship exists, PERFORM displays an error message when you use the Master option or the Detail option without a table number.
- If more than one detail table has been specified for a master table in the INSTRUCTIONS section, type d to display and query the first detail table; type d preceded by the number of another detail table to display and query the other detail tables.
- You use a table number to query any detail table that joins the active table, even if no master-detail relationship is specified. Type 4d; if table number 4 joins the active table, PERFORM queries table number 4 and it becomes the new active table. However, PERFORM displays an error message when you type d without a table number if no master-detail relationship has been specified in the INSTRUCTIONS section.

Related Option

Master
EXIT

Use the Exit option to exit from PERFORM.
Use the following procedure for the Exit option:
1. Type e to run the Exit option.
2. PERFORM returns you to your starting place, either the FORM menu or the operating system.
Use the Master option to move directly from a detail table to its master table.

Use the following procedure for the Master option:

1. Type m to select the Master option.
2. If the active table has no master table, PERFORM displays an error message. If the active table has a master table, PERFORM displays the master table with the row found joining the detail table.
3. You must declare a Master-Detail relationship in the INSTRUCTIONS section to use this option.

Related Option

Detail
Use the Next option to step forward through the rows in the current list.

Use the following procedure for the Next option:

1. Use the Query option to put the rows you want to inspect in the current list.
2. Type `n` to run the Next option. **PERFORM** displays the next sequential row in the current list.
3. Type `n` repeatedly. When you reach the last row in the current list, **PERFORM** displays the following message:

   There are no more rows in the direction you are going.

**Usage**

If you want to move forward several rows at once, enter a number before the Next option; for example, entering `10n` skips ahead 10 rows.

**Related Options**

Query, Previous
OUTPUT

You can use the Output option on the PERFORM menu to write one or all rows in the current list to a new or existing file.

You can produce an output file in which rows appear just as they do on your screen, including data, display field labels, boxes, lines, and so on. Alternatively, you can produce an output file in which rows appear just as they do when you run an UNLOAD statement. Rows retrieved using this alternative method appear in an ASCII file, one row per record, with fields separated by the default delimiter. You can use a file in this Unload format with the ACE READ statement to produce a report.

Use the following procedure for the Output option:

1. Select the Query option of the PERFORM menu to retrieve a list of the row or rows that you want to write to a file. If necessary, use the Next or Previous options to display the single row that you want to write to the file.
2. Type o to select the Output option. PERFORM prompts you for a filename:

   Enter output file (default is perform.out):

3. Press RETURN to accept the default filename. Alternatively, type the name of a file in which to store your output and press RETURN. If you wish to store your output in a different directory, make sure you include the complete pathname. The name you enter becomes the new default filename of the Output option for the rest of the session, or until you enter another, different, filename.

   PERFORM displays the FORM OUTPUT FILE menu as follows:

   FORM OUTPUT FILE
   Append
   Create
   Adds new data to an existing output file

4. Type a or press RETURN to append the information to the file that you specified in Step 3. Type c to create a new file containing this information.
Note: If you enter an existing filename in Step 3 and select the Create option, \textit{PERFORM} overwrites the old version of the file when you run the Output option. You lose any data stored in the old file.

\textit{PERFORM} displays the FORM OUTPUT FILE LIST menu as follows:

5. Type \texttt{c} or press RETURN if you want to store every row in the current list. Type \texttt{o} to store only the row that currently appears on your screen.

\textit{PERFORM} prompts you for the format of the output file:

6. Press RETURN or type \texttt{u} to store the retrieved row or rows as an ASCII file (Unload-format). Select this option if you plan to use this file in an ACE report or as input for another application.

Type \texttt{s} if you want to store the retrieved row or rows in a file formatted to look the same as your screen display (Screen-format). The file includes the data and any additional field labels, boxes, lines, or other screen items.

\textit{PERFORM} writes the rows to the file. A counter at the bottom of the screen increments as each output row is written to the file:

Output record number 1

If you select the Current-list option in Step 5, \textit{PERFORM} displays each row that it writes and updates the counter as it does so.
Usage

- If you select the Screen-format option on the OUTPUT FORMAT menu, PERFORM copies one page of a screen form for each row in the current list. To copy a row that occupies more than one screen, you must use the Output option separately with each screen.

  If you want to copy all the screens of a three-screen form, for example, perform the following operations:

  1. Type O, A, C, and S to select the Output, Append, Current-List, and Screen-format options to copy all the first screens in the current list to a file.

  2. Select the Screen option to display the second screen.

  3. Repeat the same Output options to copy the second screen.

  4. Use Screen to display the third screen, and then type O, A, C, and S to append the third screen to the file.

  If the query retrieves multiple rows, the file contains all the first screens, followed by all the second screens, and so on.

- The Unload-format option on the OUTPUT FORMAT menu copies entire rows in unload format, regardless of the number of screens in the form.

- The Unload-format option on the OUTPUT FORMAT menu copies the value of every field listed in the ATTRIBUTES section of the form specification file for each row. Fields appear in an output row in the same order in which the corresponding fields are listed in the ATTRIBUTES section of the form specification file. Look-up fields are appended to the end of the row.
Use the Previous option to display prior rows in the current list.

Use the following procedure for the Previous option:

1. Use the Query option to put the rows you want to look at in the current list.
2. Type \texttt{n} to display the next row.
3. Type \texttt{p} to use the Previous option. \texttt{PERFORM} displays the previous row (in this case, the first row) in the current list.
4. When you reach the first row in the current list, type \texttt{p}. \texttt{PERFORM} displays the following message:
   
   There are no more rows in the direction you are going.

5. You can use the Previous option whenever you want to display prior rows in the current list.

Usage

If you want to move backward several rows at once, enter a number before the Previous option; for example, entering \texttt{10p} skips back 10 rows.

Related Options

Query, Next
QUERY

Use the Query option to search for database rows and columns with specified values based on search values you enter directly into the display fields on a screen form. You can specify the search criteria with 11 different query operators, including 6 relational operators, 2 range operators, 2 wildcard operators, and highest/lowest value operators. PERFORM finds all the database rows that satisfy the conditions and puts them in the current list. You can use the Next and Previous options to view them.

Use the following procedure for the Query option:

1. Type q to select the Query option. PERFORM clears the fields in the active table of all data (except data in joined fields with no QUERYCLEAR attribute) and puts the cursor in the first field.

2. Enter search values in one or more display fields using the syntax described on page 3-30. To find all the rows in the table, do not enter any search values.

   If a display field is too short to hold the search value you enter, PERFORM creates a workspace at the bottom of the screen. When you press RETURN, PERFORM removes the workspace. The display field contains what you entered in the workspace even though you can only see the part of it that fits in the field.

3. Press ESCAPE to run the query; press the Interrupt key to cancel the query and display the PERFORM menu again. When you run the query, PERFORM searches the active table, puts all the rows that satisfy the conditions in the current list, and displays the first matching row on the screen. A Status line message reads as follows:

   # row(s) found

   where # represents the number of rows that contain the specified search value(s) in the specified display field(s). You can use the Next and Previous options to look at the rows in the current list. If no rows satisfy the conditions, the Status line message reads as follows:

   There are no rows satisfying the conditions
The following symbols can be used in the specification of queries:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Data Types</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>equal to</td>
<td>all</td>
<td>=x</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>all</td>
<td>&gt;x</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>all</td>
<td>&lt;x</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td>all</td>
<td>&gt;=x</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>all</td>
<td>&lt;=x</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>not equal to</td>
<td>all</td>
<td>&lt;&gt;x</td>
</tr>
<tr>
<td>:</td>
<td>range</td>
<td>all</td>
<td>x:y</td>
</tr>
<tr>
<td>..</td>
<td>range</td>
<td>DATETIME</td>
<td>x..y</td>
</tr>
<tr>
<td>*</td>
<td>wildcard</td>
<td>CHAR</td>
<td><em>x, x</em>, <em>x</em></td>
</tr>
<tr>
<td>?</td>
<td>single-character wildcard</td>
<td>CHAR</td>
<td>?x, x?, ?x?, x??</td>
</tr>
<tr>
<td></td>
<td>or</td>
<td>all</td>
<td>a</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>highest value</td>
<td>all</td>
<td>&gt;&gt;</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>lowest value</td>
<td>all</td>
<td>&lt;&lt;</td>
</tr>
</tbody>
</table>

= The default query operator; if you do not enter another operator, PERFORM assumes the equal sign.

Enter the equal sign by itself to search for a database row that contains a null CHAR column; enter =* to find a row that contains a column with an asterisk.

x Any search value with the appropriate data type for the search field. Enter the search value immediately after any one of the first six query operators in the previous table. Do not leave a space between the query operator and the search value.

> For CHAR data, greater than means later in ASCII order (a>A>a). For DATE or DATETIME data, greater than means after. (See Appendix E, “The ASCII Character Set,” in this manual for more information.)

< For CHAR data, less than means earlier in ASCII order (a>A>a). For DATE or DATETIME data, less than means before.

y Any search value with a value higher than x.

| The search operator that signifies or. 110 | 118 | 112 in the Customer Number field, for instance, would search for rows with the value 110, 118, or 112 in the customer_num column.

: The search operator that specifies a range. You must give the lower value before the search operator and the higher value after the operator in a range query. Queries with the range operator are inclusive. The search criterion 1 : 10 would find all rows with a value in that column from 1 through 10, inclusive.

When an odd number of colons appear in a query expression, the middle one is assumed to be a query operator. If the query expression con-
tains an even number of colons and is not a valid single DATETIME or
INTERVAL value, I-SQL returns an error.

.. An alternative search operator that specifies a range with DATETIME
and INTERVAL data types. Since DATETIME and INTERVAL constants
might include colons, use the .. search operator to avoid ambiguity.

? A wildcard character. It represents a single character. A ?ick search
value in the First Name display field of the orderform form would find
“Dick,” “Rick,” “Nick,” and so on.

* A wildcard character. It represents zero or more characters. An S*
search value in the Last Name display field of the orderform form
would find Sadler and Sipes. An *r search value would find five
names: Baxter, Jaeger, Miller, Sadler, and Vector.

>> The highest-value search operator. Enter it (with no search value) in a
display field to find the highest value for the field.

<< The lowest-value search operator. Enter it (with no search value) in a
display field to find the lowest value for the field.

Usage

• Because of the way a computer stores floating-point numbers, you might
not be able to retrieve FLOAT and SMALLFLOAT data by querying for the
exact value you entered. You can solve this problem by using a range
query. Specifying a FORMAT with a few places to the right of the decimal
point in the FORMBUILD ATTRIBUTES section might also help.

• If the RIGHT attribute is specified for a display field, you might have to
use an asterisk in front of a search value. (RIGHT does not right-justify the
search value after you enter it.)

• Although the literals in PICTURE specifications appear on the screen
when you add and update data, they do not appear on the screen when
you query. If you enter the wrong literal value, your search will not be
successful. A COMMENTS entry in the form file can help you avoid this
problem.

Related Options

Next, Previous

---

The evaluation of less than (<) and greater than (>) expressions containing
character arguments is dependent on the LANG and LC_COLLATE settings
in an NLS database. Refer to Appendix C, “Native Language Support
Within INFORMIX-SQL.”
REMOVE

Use the Remove option to delete the row on the screen from the active table.

Use the following procedure for the Remove option:

1. Use the Query, Next, and Previous options to display the row you want to delete.
2. Type r to select the Remove option.
3. PERFORM displays the following screen:

   REMOVE: Yes No
   Removes this row from the active table.

   Enter y to delete the row, or n to keep it. In either case, the PERFORM menu appears on the screen next. The following message appears at the bottom of the screen when you remove a row:

   Row deleted

Usage

You cannot remove a verify join row from one table (generally the master table, against which the join field is verified) unless you first remove all the rows that join it in other tables (generally detail tables, which are verified against the master table). For example, using the ORDERFORM form, you can remove rows in the items table. However, you cannot remove a row in the orders table without removing all rows in the items table that join that row because the Order Number display field is a verify join.

Related Options

Add, Update
SCREEN

Use the Screen option to cycle through the screen pages of the form.

Use the following procedure for the Screen option:

1. Type `s` to run the Screen option.
2. `PERFORM` displays the next screen page of the form. When you reach the last screen page, type `s` to display the first page again.
TABLE

Use the Table option when there is more than one table in the screen form and you want to select a new active table. Each table is assigned a table number assigned according to the order in which display field tags (including joins) for the table first appear in the ATTRIBUTES section of the form file. This number appears next to the table name or alias in the screen Information lines when the table is active. The Table option steps through the tables in table-number order starting with the active table.

Use the following procedure for the Table option:

1. Type t. If your screen form includes fields from two or more tables, PERFORM automatically selects and displays whichever page of the screen form contains the greatest number of fields from the new active table and surrounds those fields with delimiters (brackets); fields that belong to other tables do not have delimiters.

   If each table is on a separate screen page, PERFORM displays the screen page for the new active table.

2. Type t again. PERFORM displays the next sequential table. When you reach the last table, PERFORM displays the first table again.

Usage

If you know the number of the table you want to view next, you can go directly to that table without passing through the intervening tables. For example, suppose there are five tables in your form and you are looking at table number 4. If you want to see table number 2 next, type 2t and PERFORM displays table number 2. Tables number 5 and 1 are skipped.
UPDATE

Use the Update option to modify the data in the displayed row of the current list.

Use the following procedure for the Update option:

1. Use the Query, Next, and Previous options to display the row you want to modify.
2. Type u to run the Update option. PERFORM puts the cursor in the first active field.
3. Edit the data, modifying as many display fields as you like.
4. Press ESCAPE to store the changed row, or the Interrupt key to ignore the changes and display the menu again.

Usage

• You cannot update a field that is a verify join field for another table without first updating the relevant field in the other table.

• If you press ESCAPE after you select the Update option, PERFORM displays the following message whether or not you actually changed anything:

  This row has been changed.

• PERFORM sometimes introduces a slight discrepancy when you enter numbers in SMALLFLOAT or FLOAT fields. The entry 1.11, for example, might display as 1.11000001 after you press RETURN or ESCAPE. This discrepancy occurs because of the way a computer stores numbers internally.
The View option displays the contents of a field of data type TEXT or BYTE (applies to INFORMIX-OnLine Dynamic Server only). See Appendix I, “Using the INFORMIX-OnLine Dynamic Server,” for information on using this feature.
# The ACE Report Writer

Chapter Overview 5

Creating and Compiling a Custom Report 5
  Using the Menus to Create a Report 5
  Generating a Default Report 6
  Creating a Custom Report 6
  Creating a Report from the Command Line 7
  Command-Line Options 8

Information About ACE 9
  ACE Filename Conventions 9
  Owner Naming 9
  Using Expressions in a Report Specification 10
  ACE Error Messages 11

Demonstration Database Sample Reports 12

Structure of a Report Specification File 12

DATABASE Section 13

DEFINE Section 14
  ASCII 16
  PARAM 18
  VARIABLE 19

INPUT Section 20
  PROMPT FOR 21
OUTPUT Section  22
REPORT TO    23
LEFT MARGIN  24
RIGHT MARGIN 25
TOP MARGIN  26
BOTTOM MARGIN 27
PAGE LENGTH  28
TOP OF PAGE  29
SELECT Section  30
READ Section  33
READ  34
FORMAT Section  36
EVERY ROW  39
Control Blocks  41
AFTER GROUP OF  43
BEFORE GROUP OF 45
FIRST PAGE HEADER  47
ON EVERY ROW  49
ON LAST ROW  51
PAGE HEADER  52
PAGE TRAILER  54
Statements  55
FOR  56
IF THEN ELSE  57
LET  59
NEED  61
PAUSE  62
PRINT  63
PRINT FILE  65
SKIP  66
SKIP TO TOP OF PAGE  67
WHILE  68
Aggregates  69
ASCII  71
CLIPPED  73
COLUMN  74
CURRENT  75
DATE  76
DATE( )  77
DAY( )  78
LINENO  79
MDY() 80
MONTH() 81
PAGENO 82
SPACES 83
TIME 84
TODAY 85
USING 86
WEEKDAY() 93
WORDWRAP 94
YEAR() 95
Chapter Overview

ACE is a general-purpose relational report writer that produces reports based on the tables of a database or the data in an ASCII input file. ACE can draw information from several database tables based on relationships that you specify among the tables when you design the report.

Creating and Compiling a Custom Report

You can create a report specification file based on a database table or tables in one of two ways:

• You can use the Report option on the INFORMIX-SQL (I-SQL) Main menu.
• You can work directly with the appropriate programs from the operating system command line.

Either alternative requires that you have already created the database and all the tables from which the report will draw information. The following two sections describe these alternative procedures. They do not, however, describe the rules for constructing or modifying the report specification file. These rules are defined beginning on page 4-9.

Creating a report from the command line is also described in Appendix H, “Accessing Programs from the Operating System.” Use this option if you are retrieving data for the report from an input file.

Using the Menus to Create a Report

The procedure for creating, compiling, and running a report from the REPORT menu is described in the next two sections. “Generating a Default Report” explains the procedure used to produce a default report. “Creating a Custom Report” covers the steps involved in the production of a custom report. A more detailed description of each procedure is presented in Chapter 8, “Creating and Printing Reports,” of the INFORMIX-SQL User Guide.
Creating and Compiling a Custom Report

Generating a Default Report

To create a default report using the I-SQL menu system, follow these steps:

1. Select the Report option on the I-SQL Main menu and then the Report option on the REPORT menu.
2. If there is no current database, the CHOOSE DATABASE screen appears. After you select a database, the GENERATE REPORT screen is displayed. Enter the name you want to assign to the report (for example, newrpt). Do not use the .ace filename extension; I-SQL automatically adds the required extension.
3. I-SQL prompts you for the name of the table you want it to use to create the default report. Once you enter the table name, I-SQL automatically compiles the report specification and displays the REPORT menu. The report is now available to be used.
4. Select the Run option on the REPORT menu to run the report.

The default report specification file formats a report as a list of all columns in the table included in the report. It does not provide any special instructions to ACE about how to display the data, nor does it include instructions to perform data manipulations. Only one table contributes information to a default report.

Creating a Custom Report

To create a customized report using the I-SQL menu system, follow these steps:

1. Complete the steps described in the previous section, “Generating a Default Report.”
2. The REPORT menu should now appear on the screen. Select the Modify option on the REPORT menu.
3. The MODIFY REPORT screen appears. Enter the name of the default report (newrpt) just created.
4. If you have not specified an editor previously in the session or set the DBEDIT environment variable as described in Appendix B, “Environment Variables,” I-SQL asks you to select the editor with which you want to work. Press RETURN if you want to select the editor whose name is displayed on the top line of the screen. If you want to work with a different editor, enter the name of the editor. I-SQL calls the editor with the default report specification file.

Modify the specification to include the data you need and the appearance you desire. Exit from the editor.
5. The MODIFY REPORT menu is displayed. Select the Compile option.

6. If your report specification file compiles correctly, a message to that effect is displayed, and ACE creates a report file with the filename extension .arc (for example, newrpt.arc). Go to Step 8. If your report specification file contains errors, a message to that effect is displayed, and ACE creates a report file with the filename extension .err (for example, newrpt.err). Go to Step 7.

7. Select the Correct option from the COMPILE REPORT menu. I-SQL calls your system editor and the report specification file with the compiler errors. When you correct the errors, you need not delete the error messages. I-SQL does that for you. Return to Step 5.

8. When the compilation is successful, select the Save-and-exit option on the MODIFY REPORT menu. The REPORT menu is displayed. The report is now available for use.

9. Select the Run option on the REPORT menu and run the report.

As an alternative to using the Generate option and creating a default report specification, you can select the New option. I-SQL calls your system editor, and you enter all the report specification instructions.

Creating a Report from the Command Line

To create a customized report specification directly from the operating system command line, follow these steps:

1. Use the system editor to create a report specification file. Append the extension .ace to the filename.

2. Compile the specification with the ACEPREP program. Call ACEPREP as saceprep. You can omit the .ace filename extension when you call ACEPREP.

   For example, use this command line to compile the newrpt.ace specification file:

   ```
saceprep newrpt
   ```

3. If the compilation is successful, ACE creates a compiled report file called newrpt.arc and you are finished creating your customized report. Go to Step 5. If errors are detected in the report specification, a newrpt.err file is created. Go to Step 4.

4. Use the system editor to edit this specification. Remove all error comments from the specification file. Overwrite the file newrpt.ace with this corrected version. Go to Step 2.
5. To run the newrpt.arc report, use the ACEGO program. Call ACEGO as sacego. Do not include the .arc filename extension when you call ACEGO.

For example, use this command line to run the newrpt.arc report:

sacego newrpt

Creating a report from the command line is also described in Appendix H, “Accessing Programs from the Operating System.”

Command-Line Options

The following four command-line options are available for use with ACE:

- **-o**  
  Use the -o (output) option, followed by the pathname of a directory, to indicate the directory where ACEPREP places its output file. If you do not use this option, ACEPREP puts the file in your working (current) directory.

  For example, to instruct ACEPREP to place the output file from compiling the NEWRPT specification in the OUTPUT directory, use the following command line:

  saceprep -o output newrpt

- **-s**  
  Use the -s (silent) option with both ACEGO and ACEPREP to suppress all nonessential screen messages. For example, use this command line to suppress program banners in ACEPREP:

  saceprep -s newrpt

- **-ansi**  
  When you use -ansi to compile a report, ACEPREP generates a warning whenever it encounters an Informix extension to the SELECT statement. ACEPREP places the warnings in a name.err file. When you invoke I-SQL with -ansi, ACEPREP automatically checks for non-ANSI syntax. Use the following command:

  saceprep -ansi newrpt

- **-d**  
  Use the -d (database) option with ACEGO, followed by the name of a database, to override the database that is named in the report specification. For example, to substitute the sales database for the database included in the NEWRPT specification, use this command line:

  sacego -d sales newrpt
Do not use -ansi with ACEGO.

You can also check for non-ANSI syntax by setting the DBANSIWARN environment variable. See Appendix B, “Environment Variables,” for more information about using DBANSIWARN.

Information About ACE

The report specification file, the compiled report specification, and database files used in the report must be in your working directory or in a directory named in the DBPATH environment variable. You must refer to an input file by its full pathname if it is not in your current directory.

ACE Filename Conventions

ACE uses the following file-naming conventions:

- A report specification filename can be up to ten characters long. The filename must have an .ace filename extension. Without the .ace filename extension, the ACE compiler does not recognize the file.
  When you use the New or Generate options on the REPORT menu, I-SQL automatically adds the .ace extension to the filename; you must not include it in the filename you choose.
- When you compile a report specification, you can omit the filename extension.
- The extension that the ACE compiler gives the output file depends on whether the compile is successful. If the compile is successful, the extension .arc is appended to the filename. If the compile is unsuccessful, the extension .err is appended to the filename.
  An .err file is a text file that contains the original specification file along with error messages that describe and point to the problem ACE found.

Owner Naming

In an ANSI-compliant database, the prefix owner. must precede the table name if the report will be run by users other than the owner. The prefix owner. is optional in a database that is not ANSI-compliant. I-SQL does check the accuracy of owner. if you include it in the statement, however.
Using Expressions in a Report Specification

An expression can be anything from a simple number or alphabetic constant to a more complex series of column values, functions, quoted strings, operators, and keywords. ACE evaluates expressions when it generates a report. It can display the result of the evaluation, assign it to a variable, or use it in a calculation.

When ACE evaluates an expression, it combines elements of the expressions that are separated from each other by operators. It combines elements in the order shown in Figure 4-1. You can use parentheses to override this order.

When this manual refers to a number expression (num-expr), you can supply any type of expression, including character, as long as ACE can evaluate it as a number. The character string “123” is a valid number expression, while “m23” is not.

Similarly, date-expr is an expression that ACE can evaluate as a date. You can use a quoted string (“’01012010’” or “’1-1-2010’”) or an INTEGER that evaluates to a legal date.

A quoted string is any string of characters in quotation marks. You can use a quoted string anywhere ACE requires a type CHAR expression.
Exponents are treated as integers and not as decimals. If a decimal is provided as an exponent, ACE truncates the number to an integer. For example, the expression $4^{3.4}$ is truncated to $4^3$ before it is evaluated.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Function</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>equality for strings</td>
<td>5</td>
</tr>
<tr>
<td>!= or &lt;&gt;</td>
<td>not equal</td>
<td>5</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>5</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>5</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal</td>
<td>5</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal</td>
<td>5</td>
</tr>
<tr>
<td>not</td>
<td>not</td>
<td>6</td>
</tr>
<tr>
<td>and</td>
<td>and</td>
<td>7</td>
</tr>
<tr>
<td>or</td>
<td>or</td>
<td>8</td>
</tr>
</tbody>
</table>

Precedence of operators: 1 is highest, 8 is lowest.

**Figure 4-1  Operator precedence**

A unary minus indicates or changes the algebraic sign of a value (from positive to negative or from negative to positive). It operates on a single operand.

**ACE Error Messages**

The text of all I-SQL error messages, along with suggestions for corrections, is included in *Informix Error Messages*, Version 6.0.
Demonstration Database Sample Reports

The sample reports in this chapter are taken from the following list. Additional reports, included with the database, are available for further study. These reports illustrate a wide variety of the commands available with ACE.

mail1.ace  A simple report that generates mailing labels
mail2.ace  A more sophisticated report that produces one column of mailing labels
mail3.ace  An interactive report that generates one to three columns of mailing labels
clist1.ace A report that lists customer information
clist2.ace An interactive customer report
ord1.ace   A custom report of orders placed with the store
ord2.ace   A second customer order report
ord3.ace   An interactive report that lists daily orders

"Appendix A, "customerThe Demonstration Database and Application," contains the full text of each sample report specification.

Structure of a Report Specification File

A report specification file contains the instructions that specify what data a report includes and how that data appears. A report specification consists of three required sections (DATABASE, SELECT, or READ, and FORMAT) and three optional sections (DEFINE, INPUT, and OUTPUT). The following diagram and list define the required and optional sections of a report specification.

```
<table>
<thead>
<tr>
<th>DATABASE</th>
<th>DEFINE</th>
<th>INPUT</th>
<th>OUTPUT</th>
<th>SELECT</th>
<th>FORMAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section</td>
<td>Section</td>
<td>Section</td>
<td>Section</td>
<td>Section</td>
<td>Section</td>
</tr>
<tr>
<td>p. 4-14</td>
<td>p. 4-15</td>
<td>p. 4-20</td>
<td>p. 4-22</td>
<td>p. 4-31</td>
<td>p. 4-37</td>
</tr>
</tbody>
</table>
```
Note that the ACE report specification sections must be kept in the following general order:

1. DATABASE section: Each report specification must begin with a DATABASE section that identifies the database you want the report to use.
   
2. DEFINE section: The optional DEFINE section is used to declare variables that are used by the report as well as parameters that the report can accept from the command line. This section is also used to specify the field names and data types of values in an ASCII input file.
   
3. INPUT section: The INPUT section is optional. It is used to pass parameters to the report.
   
4. OUTPUT section: The OUTPUT section is optional. It is used to control page length and margin width, and to direct the output from the report to a file, a system printer, or a pipe.

5. SELECT or READ section:
   - SELECT section: If you retrieve data from a database table, the SELECT section specifies the columns and tables on which the report is based.
   - READ section: If you retrieve data from an ASCII file, the READ section specifies the name of the input file on which the report is based.

6. FORMAT section: The FORMAT section appears next. It includes commands that determine the appearance of the data in the report.

You can include comments anywhere in an ACE report specification. Simply enclose comments within a set of braces ({}) or precede them with the pound sign (#) or double dash (--).

**DATABASE Section**

Every report specification must have a DATABASE section. The DATABASE section specifies the database ACE uses as the basis of the report. You can override the database that you specify in this section with the -d command-line option. See the section “Command-Line Options” on page 4-8 for more information.

The DATABASE section must be the first section in an ACE report specification. It begins with the DATABASE keyword, followed by the name of the database, and ends with the END keyword.

If you want to retrieve data from an ASCII file using the READ statement, you still must specify a database in the DATABASE section even though a report based on an ASCII file is not related to a database. You can either specify the name of an existing database or use the ASCII keyword.

*The ACE Report Writer 4-13*
The following diagram shows the structure of the DATABASE section:

```
**DEFINE Section**

DATABASE

**database-name**

END

**ASCII**
```

`database-name` is the name of the database accessed.

The following example DATABASE section is from the `clist1.ace` report:

```
database {use stores2 database}
stores2
end
```

The following example illustrates the use of the ASCII keyword.

```
database {using the READ statement}
ascii
end
```

**DEFINE Section**

An ACE report specification can optionally contain a DEFINE section. The DEFINE section is used to declare variables used in the report and parameters the report can accept from the command line. If you are retrieving values from an input file using the READ statement, you must use the ASCII keyword in the DEFINE section to specify the field names and data types for the data in that file.

The DEFINE section begins with the DEFINE keyword and ends with the corresponding END keyword. The variable definition list appears between these keywords and is composed of one or more PARAM or VARIABLE statements or a combination of both. You can use a single ASCII keyword and field list between the DEFINE and END keywords.
The following diagram shows the structure of the DEFINE section:

```
DEFINE Section

DEFINE database-name

ASCII Statement p.4-16

PARAM Statement p. 4-18

VARIABLE Statement p. 4-19

FUNCTION Statement p. 6-5

END
```

database-name is the name of the database accessed.

The next three sections describe the ASCII, PARAM, and VARIABLE statements. The FUNCTION statement is described in Chapter 6.
### ASCII

You use an ASCII statement in a DEFINE section to specify the field names and data types of the records in an ASCII input file. The ACE report writer accesses this file in a READ statement.

**ASCII** is a required keyword that specifies ASCII input.

- **field-name** is a required identifier for a field in the file.
- **data-type** is a valid SQL data type.

#### Usage

- You must include an ASCII statement in the DEFINE section if you use a READ statement in the READ section.
- You cannot use a SELECT statement to access an ASCII file, nor can you use a READ statement to access a database table.
- Although a report based on ASCII data is not related to a database, you must specify a database in the DATABASE section. Either specify the name of an existing database, or use the ASCII keyword.
- The number of fields in the ASCII statement must match the number of fields in the ASCII file.
- Each field-name must be followed by a data type specification. ACE does not check the accuracy of data types, so run-time errors can occur if a data type has been specified incorrectly.
- No further specification of the MONEY data type is permitted beyond the keyword MONEY.
The following ASCII statement defines a record from an ASCII file in unload format:

```
define ascii stock_num smallint, manu_code char(3),
       description char(15), unit_price money,
       unit char(4), unit_descr char(15)
end
```

In this instance, the field names happen to have the same names and sequence as the column names in the `stock` table of the `stores2` database. Like the variable names of a PARAM or VARIABLE statement, the field names do not need to match the column names of any table. The number, order, and data types of the field names must be consistent with the fields in the ASCII file.


**Related Commands**

READ, UNLOAD
PARAM

This statement allows you to use arguments specified on the command line at the time you run an ACE report. It declares a variable whose initial value is that of a command-line argument. To use PARAM, you must call ACE from a custom user menu (see Chapter 5, “User-Menu”) or the command line (see the section “Creating and Compiling a Custom Report” on page 4-5).

PARAM is a required keyword.

- \texttt{int} is a required integer that specifies the position of the argument on the command line. The first argument is number 1.
- \texttt{var-name} is the name of the variable that you are declaring—it will initially have the value of a command-line argument when you run the report.
- \texttt{data-type} is a valid SQL data type.

Usage

- You can define a total of 100 variables using PARAM and VARIABLE statements in an ACE report specification.
- If a report specification uses a PARAM statement and you fail to provide arguments on the command line when you run the report, ACE gives an error message.
- If you want to use a variable defined by a PARAM statement in the SELECT section, you must precede the variable name with a dollar sign. Refer to the “SELECT Section” on page 4-30 for more information.

\textbf{INFORMIX-OnLine Dynamic Server} accepts the VARCHAR data type with the VARIABLE statement. Refer to Appendix I, “Using the INFORMIX-OnLine Dynamic Server,” for more information regarding the use of VARCHAR data types in ACE reports.
VARIABLE

This statement declares a variable that you can use in an ACE report specification.

```
VARIABLE var-name data-type
```

- **VARIABLE** is a required keyword.
- **var-name** is the name of the variable that you are defining.
- **data-type** is a valid SQL data type.

**Usage**

- You can define a total of 100 variables using PARAM and VARIABLE statements in an ACE report specification.
- If you want to use a variable that you declare in a PARAM or VARIABLE statement in the SELECT section, you must precede the variable name with a dollar sign. (Refer to the “SELECT Section” on page 4-30 for more information.)
- No further specification of the MONEY data type is permitted beyond the keyword MONEY.

The following example is from the **ord3.ace** report:

```
define
   variable begin_date date
   variable end_date date
end
```

The user enters the values for the two variables when the program runs. See the “INPUT Section” on page 4-20 for a description of this process.

**INFORMIX-OnLine Dynamic Server** accepts the VARCHAR data type with the VARIABLE statement. Refer to Appendix I, “Using the INFORMIX-OnLine Dynamic Server,” for more information.

---

_The ACE Report Writer_ 4-19
INPUT Section

An ACE report specification optionally can contain an INPUT section. The INPUT section allows you to produce an interactive ACE report by prompting for and accepting input while ACE is running a report.

The INPUT section consists of the keywords INPUT and END with one or more PROMPT FOR statements in between. The following diagram shows the structure of the INPUT section:
PROMPT FOR

This statement prompts you while ACE is running a report and assigns the value you enter to a variable.

```
PROMPT FOR  var-name  USING  "string"
```

PROMPT FOR are required keywords.

var-name is the name of the variable that receives your input. You must declare this variable in the DEFINE section of the ACE report specification.

USING is a required keyword.

string is the string of characters that ACE uses as a prompt. You must enclose this string in quotation marks.

Usage

- You cannot prompt for, or accept, a database name using the PROMPT FOR statement. Refer to the DATABASE section and to the discussion of the -d option in “Command-Line Options” on page 4-8.
- You cannot prompt for, or accept, an output filename using the PROMPT FOR statement.

The following example is from the ord3.ace report:

```
input
  prompt for begin_date
  using "Enter beginning date for report: 

  prompt for end_date
  using "Enter ending date for report: 

end
```

The two character strings "Enter beginning date for report:" and "Enter ending date for report:" appear as prompts on the screen when the ord3.ace report runs. The response to the first prompt is entered as the value to the begin_date variable; the response to the second prompt is entered as the value to the end_date variable. These two variables are used at several points in the ord3.ace report.
OUTPUT Section

An ACE report specification can optionally contain an OUTPUT section. The OUTPUT section controls the width of the margins and the length of the page. The OUTPUT section also allows you to direct the output from the ACE report to a file or a printer.

The OUTPUT section begins with the OUTPUT keyword and ends with the corresponding END keyword, with one or more statements in between. The following diagram shows the structure of the OUTPUT section:
REPORT TO

This statement directs the output of the ACE report to a file or a printer.

```
REPORT TO
  "filename"

PRINTER "program"
```

REPORT TO are required keywords.

- `filename` is the name of a system file that receives the report. You must enclose the filename in quotation marks.
- `PRINTER` is the keyword that sends the report to the printer.
- `program` is the name of a system command.

Usage

- When you do not use one of the REPORT TO statements, ACE sends the report to your screen.
- You cannot use more than one REPORT TO statement in a report specification.
- The TO PRINTER keywords cause ACE to send the report to the program named by the DBPRINT environment variable. If you do not define this environment variable, ACE sends the report to the `lp` program.
- If you want to send the report to a printer other than the system printer, you can use the REPORT TO `filename` statement to send the output to a file and then send the file to the printer of your choice.
- If the REPORT TO `filename` statement writes to an existing filename, the file is replaced with the new output. You can also use the REPORT TO PIPE statement to direct the output to a program that will send the output to the appropriate printer.

The following example directs the output to the `labels` file:

```
output
  report to "labels"
end
```
LEFT MARGIN

This statement sets a left margin for a report.

```
LEFT MARGIN Statement

  LEFT MARGIN     integer

LEFT MARGIN  are required keywords.
integer       is an integer that specifies the width of the left margin in spaces.
```

Usage

The default left margin is five spaces.

The following example is from the `mail2.ace` report. ACE prints the left side of the report as far to the left as possible.

```
output
  top margin 0
  bottom margin 0
  left margin 0
  page length 9
  report to "labels"
end
```
RIGHT MARGIN

This statement sets a right margin for a report.

\[
\text{RIGHT MARGIN} \hspace{1cm} \text{integer}
\]

RIGHT MARGIN are required keywords.

\textit{integer} is an integer that specifies the width of the text on the page in characters.

Usage

- The RIGHT MARGIN determines the right margin by specifying the width of the page in characters. It does not depend on the LEFT MARGIN, but always starts its count from the left edge of the page (space 0).
- The RIGHT MARGIN is only effective when the FORMAT section contains an EVERY ROW statement.
- The default right margin is 132 characters.

The following example report specification demonstrates the use of the RIGHT MARGIN statement. ACE sets the right margin for the report at 70 characters.

\begin{verbatim}
database
  stores2
end

output
  right margin 70
end

select *
  from customer
end

format
  every row
end
\end{verbatim}
This statement sets a top margin for a report.

**TOP MARGIN**

This statement sets a top margin for a report. The default top margin is three lines. The top margin appears above any page header you specify.

**Usage**

- The default top margin is three lines.
- The top margin appears above any page header you specify.

**Example**

The following example is from the `mail2.ace` report. ACE begins printing at the top of each page.

```plaintext
output
  top margin 0
  bottom margin 0
  left margin 0
  page length 9
  report to "labels"
end
```
BOTTOM MARGIN

This statement sets a bottom margin for a report.

```
BOTTOM MARGIN  integer
```

BOTTOM MARGIN are required keywords.

`integer` is an integer that specifies the number of blank lines that ACE is to leave at the bottom of each page.

**Usage**

- The default bottom margin is three lines.
- The bottom margin appears below any page trailer.

In the following example, the printing continues to the bottom of each page:

```
output
top margin 0
  bottom margin 0
end
```
PAGE LENGTH

This statement sets the number of lines on each page of a report.

```
PAGE LENGTH integer
```

PAGE LENGTH are required keywords.

`integer` is an integer that specifies the length of the page in lines.

Usage

- The default page length is 66 lines.
- The PAGE LENGTH includes the TOP MARGIN and BOTTOM MARGIN.

The following example demonstrates the use of the PAGE LENGTH statement. ACE prints each page with 22 lines.

```
output
  (This length works on std 24-line crt)
  page length 22
  top margin 0
  bottom margin 0
end
```
TOP OF PAGE

This statement specifies the character string that causes your printer to eject a page.

```
TOP OF PAGE
Statement
```

`TOP OF PAGE` are required keywords.

`char-string` is a one- or two-character string that causes your printer to eject a page.

**Usage**

- On most printers, `char-string` is "^L", the ASCII form-feed character. **ACE** uses the first character of the string as the TOP OF PAGE character unless it is the ^ character. If the first character is the ^ character, **ACE** decodes the second character as a control character. (If you are unsure of the character string to specify for your printer, refer to the documentation provided with your printer.)

- **ACE** places the character string in the report to advance to the next page whenever the program causes a new page to be set up. Any of the following items can initiate a new page:
  - The next print line meets the bottom margin.
  - A `SKIP TO TOP OF PAGE` statement is executed.
  - A `SKIP n LINES` statement skips more lines than are available on the current page.
  - A `NEEDS` statement specifies more lines than are available on the current page.

- If you specify the TOP OF PAGE statement, **ACE** uses the specified page-eject character to set up new pages instead of using line feeds.

- If you omit the TOP OF PAGE statement, **ACE** fills the remaining lines of the current page with line feeds when a new page is set up.
SELECT Section

The following example sets CONTROL-L as the page-eject character:

```sql
output
top of page "^L"
report to "r_out"
end

select * from customer
end

format
  on every row
end
```

**SELECT Section**

Every report specification must have a SELECT section or a READ section. The SELECT section specifies the columns or tables or both that the report is based on if you retrieve data from a database. The READ section specifies the input file the report is based on if you retrieve data from an ASCII file. See the “READ Section” on page 4-33 for details on how to use ASCII files in ACE reports.

You can use the SELECT section to specify criteria for selecting and ordering rows based on the contents of specific columns. The FORMAT section can group rows in the report based on the order you specify in the SELECT section.

The SELECT section contains one or more SELECT statements. These statements are identical to the SELECT statements described in Chapter 6 of the Informix Guide to SQL: Reference, Version 4.1. This chapter does not define the SELECT statement but shows how to incorporate it in an ACE report specification.

The SELECT section begins with the SELECT keyword. This keyword introduces both the SELECT section and the first SELECT statement. Other SELECT statements can follow the first—each must begin with the SELECT keyword. All SELECT statements, except for the last, must end with a semicolon. (If there is only one SELECT statement, it does not require a semicolon.) The SELECT section ends with the END keyword. All but the last SELECT statement must have an INTO TEMP clause.

If you use an ORDER BY clause in the SELECT section, you cannot use an integer or a column with a table prefix (table.column) to indicate the column to sort by. If you cannot use the column name alone because it is not unique or
because it is an expression, define a display label in the select list and use it in both the ORDER BY clause and the FORMAT section in the AFTER and BEFORE GROUP OF control blocks. The second example in this section demonstrates the use of a display label.

If you use an ACE variable in the SELECT section, you must precede the variable name with a dollar sign. The third example in this section demonstrates the use of a variable name.

The following diagram shows the structure of the SELECT section:

The following example is from the \texttt{mail1.ace} report. ACE selects all rows from the \texttt{customer} table and orders the rows first by zip code and then by last name.

\begin{verbatim}
select *
   from customer
   order by zipcode, lname
end
\end{verbatim}

The following example is from the \texttt{ord1.ace} report:

\begin{verbatim}
select
   orders.order_num number,
   order_date, customer_num,
   po_num, ship_date, ship_charge,
   paid_date,
   items.order_num, stock_num, manu_code,
   quantity, total_price
from orders, items
where orders.order_num = items.order_num
order by number
end
\end{verbatim}
ACE selects the indicated columns from the orders and items tables. The order_num column in the orders table is given the display label number and is joined to the order_num column in the items table. ACE orders the rows by the values in the number column.

The following example is from the clist2.ace report:

```sql
select
    customer_num,
    fname,
    lname,
    company,
    city,
    state,
    zipcode,
    phone
from
    customer
where
    state matches $thisstate
order by
    zipcode,
    lname
end
```

ACE selects the indicated columns from the customer table. The WHERE clause tells ACE to select only those rows where the value in the state column matches the value in the variable thisstate. ACE orders the rows by the values in the zipcode column first and then by lname.

READ Section

As an alternative to the SELECT section, you can include a READ section containing a READ statement. Unlike the SELECT statement, which queries the database for rows, the READ statement retrieves rows from an ASCII input file. Every report specification must have either a READ section or a SELECT section.

The READ statement allows you to retrieve data from ASCII files produced by the UNLOAD statement of SQL or the Output option of PERFORM. In addition, you can produce reports from ASCII files created or edited by other software products.

The following conditions must be satisfied before you can read data from an ASCII file:

- You must know the complete pathname of the ASCII data file. You must also know the number of fields, the delimiter symbol, and the order and data type of each field of a record in the file.
- You must use the ASCII statement in the DEFINE section of a report specification to indicate the format of a record in the ASCII file. The ASCII keyword is followed by an ordered list of the field names and data types of the ASCII file. See the “DEFINE Section” on page 4-14 for details on using the ASCII statement.
- Although a report based on ASCII data is not related to a database, you must include a DATABASE section in the report specification. You can either specify an existing database or use the ASCII keyword. See the “DATABASE Section” on page 4-13 for details on using the ASCII keyword.

The following diagram shows the structure of the READ section:
READ

You use the READ statement in the READ section to retrieve data from an ASCII input file in unload format. The READ section specifies the name of the input file, any nondefault delimiter, and optional sorting specifications. You use the ASCII statement in the DEFINE section to specify the fields of each record in the input file.

READ is a required keyword that specifies ASCII input.

filename is the name of the ASCII file enclosed in quotes. If the file is not located in the current directory, you must include the complete pathname.

DELIMITER is an optional keyword that is used if the field separator symbol is not the default delimiter ( | )

symbol is the character, enclosed in quotation marks, that is used between fields of the records in filename.

ORDER BY is an optional keyword that is required if records from the input file are to be sorted in the report according to the values in one or more fields.

EXTERNAL is an optional keyword indicating that the records in the input file are already sorted.

fieldname is the name of a field in an input file record, as defined in the ASCII statement of the DEFINE section, that is used as a sorting key.

ASC is an optional keyword specifying that the values in the fieldname field are used to sort records in ascending order (smallest values first).

DESC is an optional keyword specifying that the values in the fieldname field are used to sort records in descending order (largest values first).
Usage

- The READ statement requires an ASCII statement in the DEFINE section. You cannot use a SELECT statement to access an ASCII file, nor can you use a READ statement to access a database table.
- The default delimiter is a vertical bar (| = ASCII 124). See Appendix B, “Environment Variables,” for information on how to specify a different default delimiter with the DBDELIMITER environment variable.
- ACE uses the delimiter specified in the READ statement as the field separator, regardless of whether you have set the DBDELIMITER environment variable.
- An ORDER BY clause in a READ statement can list up to eight field names as sorting keys. These names must match the field names that are specified in the ASCII statement. Specify an existing database, rather than use the ASCII keyword, if you are using an ORDER BY clause.
- If more than one sorting key is specified in an ORDER BY clause, the primary key is the first field named in the ORDER BY list, the secondary sorting key is the second field named in that list, and so on.
- If the ASCII file named in a READ statement is already sorted, and the FORMAT section contains BEFORE GROUP OF or AFTER GROUP OF control blocks on two or more fields, then an ORDER EXTERNAL BY clause must specify the hierarchy of the order.
- ACE does not allow the use of the space or double quotation mark (" ) as a delimiter in the ASCII file.

The following READ statement specifies an ASCII file (in unload format) that corresponds to the stock table of the stores2 database:

```
read "stock1" delimiter ":"  
   order by unit_price desc, description 
end
```

This READ statement tells ACE to read the records in an ASCII file called STOCK1 that uses the colon (:) as a delimiter, and to sort the records in descending order according to the values in the field unit_price. Since the description field is a secondary sorting key, records that have the same unit_price value appear in ascending alphabetical order according to the label in their description field.
The next example reverses the previous order of the sorting keys and sorts unit prices in default (ascending) order:

```
read "stock1" delimiter ":"
  order external by description, unit_price
end
... format
... after group of description
... after group of unit_price
... ...
```

The ORDER EXTERNAL BY clause specifies that the `stock1` file is already sorted. Totals or subtotals specified in the AFTER GROUP OF control blocks are printed after the groups of rows have been printed according to the sorting instructions in the ORDER BY clause.

**Related Commands**

ASCII, ORDER BY (option of SELECT), OUTPUT (option of PERFORM), UNLOAD (SQL statement)

**FORMAT Section**

An ACE report specification must contain a FORMAT section. The FORMAT section determines the appearance of the report. It works with the data that is qualified by the last (or only) SELECT statement in the SELECT section, or with the contents of an ASCII file referenced by the READ statement in the
READ section. The FORMAT section begins with the FORMAT keyword and ends with the corresponding END keyword as shown in the following diagram.

The simplest FORMAT section contains only an EVERY ROW statement between the FORMAT and END keywords. If you use an EVERY ROW statement, you cannot use any other statements or control blocks in the FORMAT section. The following example shows the structure of this type of FORMAT section.

```
FORMAT
  EVERY ROW statement
END
```
More complex FORMAT sections can contain control blocks such as ON EVERY ROW and BEFORE GROUP OF. Each of these control blocks must contain at least one statement such as PRINT or SKIP n LINES. In the FORMAT section, you cannot refer to a column using its table name \( \text{table.column} \) as a prefix. If you do not use an EVERY ROW statement, you can combine control blocks as required. You can place control blocks in any order within the FORMAT section.
EVERY ROW

The EVERY ROW statement causes ACE to output every row that the SELECT or READ section retrieves. It uses a default format.

Usage

- This statement is useful when you want to develop a report quickly using a default format. The report uses as column headings the column names you assigned when you created the table or the field names that you assigned in the ASCII statement. Because the EVERY ROW statement cannot contain any control blocks or other statements, you cannot alter the default format to create a custom report.
- The EVERY ROW statement stands by itself—you cannot modify it with any of the statements listed in the “Statements” section of this chapter.
- When you use the EVERY ROW statement, you cannot use any control blocks in the FORMAT section.
- A report generated by an EVERY ROW statement uses the column names you assigned when you created the table.
- If the columns that you specify in the SELECT section, or the values that you retrieve in the READ section, fit on one line, ACE produces a report with column or field names across the top of each page; otherwise, ACE produces a report with the column or field names down the left side of the page.
- You can use the RIGHT MARGIN statement in the OUTPUT section to control the width of a report that uses the EVERY ROW statement.
- Use the ON EVERY ROW control block if you want to display every row in a format other than the default format. (See the discussion of ON EVERY ROW on page 4-49.)
The following example shows a minimal ACE report specification using the EVERY ROW statement:

```plaintext
database
  stores2
end

select *
  from customer
end

format
  every row
end
```

The following example shows a portion of the output from the preceding specification:

```
customer_num    101
fname           Ludwig
lname           Pauli
company         All Sports Supplies
address1        213 Erstwild Court
address2
city            Sunnyvale
state           CA
zipcode         94086
phone           408-791-8075

```
```
customer_num    102
fname           Carole
lname           Sadler
company         Sports Spot
address1        785 Geary St
address2
city            San Francisco
state           CA
zipcode         94117
phone           415-822-1291

```
```
customer_num    103
fname           Philip
lname           Currie

```
```
The following example shows another example of a brief report specification that uses the EVERY ROW statement:

```
database
   stores2
end

select order_num, customer_num,
   order_date
 from orders
end

format
   every row
end
```

The following example shows the output from the preceding specification:

```
<table>
<thead>
<tr>
<th>order_num</th>
<th>customer_num</th>
<th>order_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>104</td>
<td>01/20/1991</td>
</tr>
<tr>
<td>1002</td>
<td>101</td>
<td>06/01/1991</td>
</tr>
<tr>
<td>1003</td>
<td>104</td>
<td>10/12/1991</td>
</tr>
<tr>
<td>1004</td>
<td>106</td>
<td>04/12/1991</td>
</tr>
<tr>
<td>1005</td>
<td>116</td>
<td>12/04/1991</td>
</tr>
<tr>
<td>1006</td>
<td>112</td>
<td>09/19/1991</td>
</tr>
<tr>
<td>1007</td>
<td>117</td>
<td>03/25/1991</td>
</tr>
<tr>
<td>1008</td>
<td>110</td>
<td>11/17/1991</td>
</tr>
<tr>
<td>1009</td>
<td>111</td>
<td>02/14/1991</td>
</tr>
<tr>
<td>1010</td>
<td>115</td>
<td>05/29/1991</td>
</tr>
<tr>
<td>1011</td>
<td>104</td>
<td>03/23/1991</td>
</tr>
<tr>
<td>1012</td>
<td>117</td>
<td>06/05/1991</td>
</tr>
<tr>
<td>1013</td>
<td>104</td>
<td>09/01/1991</td>
</tr>
<tr>
<td>1014</td>
<td>106</td>
<td>05/01/1991</td>
</tr>
<tr>
<td>1015</td>
<td>110</td>
<td>07/10/1991</td>
</tr>
</tbody>
</table>
```

### Control Blocks

Control blocks provide the structure for a custom report. The control blocks that you can use in a FORMAT section follow:

- AFTER GROUP OF
- BEFORE GROUP OF
- FIRST PAGE HEADER
- ON EVERY ROW
ON LAST ROW
PAGE HEADER
PAGE TRAILER

Each control block is optional, but if you do not use the EVERY ROW statement, you must include at least one control block in a report specification.

Each control block must include at least one statement. (See the “Statements” section beginning on page 4-55). If you have INFORMIX-ESQL/C, you can also call C functions from within a control block. See the INFORMIX-ESQL/C Programmer’s Manual and Chapter 6 of this manual for details.

When you use an ORDER BY clause in the SELECT or READ section of an ACE report specification, you can use BEFORE GROUP OF and AFTER GROUP OF control blocks in the FORMAT section. When you use the BEFORE GROUP OF, AFTER GROUP OF, and ON EVERY ROW control blocks in a single report specification, ACE processes the control blocks in the order shown in Figure 4-2. (The figure assumes that the SELECT or READ section orders by columns a, b, and c.)

```
before group of a
  before group of b
    before group of c
      on every row
    after group of c
  after group of b
after group of a
```

**Figure 4-2** Order of group processing
AFTER GROUP OF

The AFTER GROUP OF control block specifies what action ACE takes after it processes a group of rows. Grouping is determined by the ORDER BY clause of the SELECT or READ section.

**Usage**

- A *group* of rows is all the rows that contain the same value for a given column. ACE automatically groups rows when you use an ORDER BY clause in the SELECT or READ section of a report specification (that is, groups come together when you order a list).

When you specify more than one column in an ORDER BY clause, ACE orders the rows first by the first column you specify (most significant), second by the second column you specify, and so on, until the last column you specify (least significant).

ACE processes the statements in an AFTER GROUP OF control block each time the specified column changes value, each time a more significant column changes value, and at the end of a report. (See Figure 4-2 on page 4-42.)

- Each column specified in the ORDER BY clause in the SELECT or READ section can contain one AFTER GROUP OF control block.
• If you have more than one AFTER GROUP OF control block, their order within the FORMAT section is not significant. ACE processes the AFTER GROUP OF control blocks in the reverse order specified in the ORDER BY clause in the SELECT or READ section. (See Figure 4-2 on page 4-42.)

• When ACE finishes generating a report, it executes all of the statements in the AFTER GROUP OF control blocks before it executes those in the ON LAST ROW control block.

• You cannot reference the column-name in the AFTER GROUP OF clause using the name of a table (the table.column structure is not allowed). If you cannot use the column name alone because it is not unique or because it is an expression, define a display label in the select list of the SELECT section and use it in the AFTER GROUP OF clause.

• You cannot use an integer to indicate by which column of the select list the grouping is to occur.

• You can only use group aggregates in AFTER GROUP OF control blocks. You cannot use group aggregates in any other control blocks.

• If you specify a substring of a CHAR column in an ORDER BY clause in the SELECT or READ section, you must use the same substring specification as the column-name in an AFTER GROUP OF control block.

• You can use a SKIP TO TOP OF PAGE statement in an AFTER GROUP OF control block to start a new page after each group.

• When ACE processes the statements in an AFTER GROUP OF control block, the columns that the report is processing still have the values from the last row of the group. From this perspective, the AFTER GROUP OF control block could be called the “on last row of group” control block.

The following example is from the ord2.ace report:

```ace
after group of number
    skip 1 line
    print 4 spaces, "Shipping charges for the order: ",
        ship_charge using "$$$.&&"
    skip 1 line

    print 5 spaces, "Total amount for the order: ",
        ship_charge + group total of total_price
        using "$$, $$$, $$$.&&"
    skip 3 lines

after group of custnum
    skip 2 lines
```
BEFORE GROUP OF

The BEFORE GROUP OF control block specifies what action ACE is to take before it processes a group of rows. Grouping is determined by the ORDER BY clause of the SELECT or READ section.

BEFORE GROUP OF are required keywords.

OF

column-name is the name of one of the columns or identifiers specified in the ORDER BY clause of the SELECT or READ section.

int1,int2 are optional subscripts that you can use with a type CHAR column to refer to a subset of the column beginning at character position int1 and ending at character position int2. If int2 is not specified, the end of the column is used.

statement is a list of one or more statements or a compound statement.

Usage

• A group of rows is all the rows that contain the same value for a given column. ACE automatically groups rows when you use an ORDER BY clause in the SELECT or READ section of a report specification (that is, groups come together when you order a list).

When you specify more than one column in an ORDER BY clause, ACE orders the rows first, by the first column you specify (most significant), second, by the second column you specify, and so on, until the last column you specify (least significant).

ACE processes the statements in a BEFORE GROUP OF control block at the start of a report, each time the specified column changes value, and each time a more significant column changes value. (See Figure 4-2 on page 4-42.)

• Each column specified in the ORDER BY clause in the SELECT or READ section can contain one BEFORE GROUP OF control block.
• If you have more than one BEFORE GROUP OF control block, their order within the FORMAT section is not significant. ACE processes the BEFORE GROUP OF control blocks in the reverse order specified in the ORDER BY clause in the SELECT or READ section. (See Figure 4-2 on page 4-42.)

• When ACE starts to generate a report, it executes all of the statements in the BEFORE GROUP OF control blocks before it executes those in the ON EVERY ROW control block.

• You cannot reference the column-name in the BEFORE GROUP OF clause using the name of a table (the table.column structure is not allowed). If you cannot use the column name alone because it is not unique or because it is an expression, define a display label in the select list of the SELECT section and use it in the BEFORE GROUP OF clause.

• You cannot use an integer to indicate by which column of the select list the grouping is to occur.

• If you specify a substring of a CHAR column in an ORDER BY clause in the SELECT or READ section, you must use the same substring specification as the column-name in a BEFORE GROUP OF control block.

• You can use a SKIP TO TOP OF PAGE statement in a BEFORE GROUP OF control block to start a new page after each group.

• When ACE processes the statements in a BEFORE GROUP OF control block, the columns that the report is processing have the values from the first row of the row group. From this perspective, the BEFORE GROUP OF control block could be called the “on first row of group” control block.

The following example is from the ord1.ace report:

```plaintext
before group of ordnum

print "Order number: ", ordnum using "####",
   " for customer number: ", customer_num
      using "#####"
print "Customer P.O. : ", po_num,
   " Date ordered: ", order_date
skip 1 line
print "Stockno", column 20,
   "Mfcode", column 28, "Qty", column 38, "Price"
```
FIRST PAGE HEADER

The FIRST PAGE HEADER control block specifies what information appears at the top of the first page of the report.

**FIRST PAGE HEADER**

- *first_page_header* are required keywords.
- *statement* is a list of one or more statements or a compound statement.

**Usage**

- The vertical size of the first page header is equal to the number of lines that you specify in the FIRST PAGE HEADER control block. The TOP MARGIN (in the OUTPUT section) affects how close to the top of the page ACE displays the page header.
- A FIRST PAGE HEADER control block overrides a PAGE HEADER control block on the first page of a report.
- You cannot use the SKIP TO TOP OF PAGE statement in a FIRST PAGE HEADER control block.
- If you use an IF THEN ELSE statement in a FIRST PAGE HEADER control block, the number of lines displayed by the PRINT and SKIP statements following the THEN keyword must be equal to the number of lines displayed by the PRINT and SKIP statements following the ELSE keyword.
- You cannot use the PRINT FILE statement to read and display text from a file in a FIRST PAGE HEADER control block.
- You can use a FIRST PAGE HEADER control block to produce a title page as well as column headings.
The following example is from the `mail3.ace` report:

```
first page header
let i = 1
let l_size = 72/count1
let white = 8/count1
```

This FIRST PAGE HEADER does not display any information. Because ACE executes the FIRST PAGE HEADER control block *before* it generates any output, you can use this control block (as demonstrated in the example) to initialize variables that you use in the FORMAT section.
ON EVERY ROW

The ON EVERY ROW control block specifies what action ACE takes after the SELECT section qualifies a row, or after the READ section retrieves a row.

- **statement** is a list of one or more statements or a compound statement.

**Usage**

- ACE processes the statements in an ON EVERY ROW control block as each new row is formatted.
- If a BEFORE GROUP OF control block is triggered by a change in column value, all BEFORE GROUP OF control blocks are executed (in the order of their significance) before the ON EVERY ROW control block is executed.
- If an AFTER GROUP OF control block is triggered by a change in column value, all AFTER GROUP OF control blocks are executed (in the order of their significance) after the ON EVERY ROW control block is executed.

The following example is from the clist1.ace report:

```plaintext
on every row
  print customer_num using "####",
  column 9, fname clipped, 1 space, lname clipped,
  column 32, city clipped, ", " , state,
  column 54, zipcode,
  column 62, phone
```
The following example is from the `mail2.ace` report:

```
on every row
  if (city is not null) and
  (state is not null) then
  begin
    print fname clipped, 1 space, lname
    print company
    print address1
    if (address2 is not null) then
      print address2
    print city clipped, ", ", state,
    2 spaces, zipcode
    skip to top of page
  end
```
ON LAST ROW

The ON LAST ROW control block specifies the action ACE takes after processing the last row qualified by the SELECT section, or the last row retrieved by the READ section.

```
on last row
  skip 1 line
  print "TOTAL NUMBER OF CUSTOMERS:",
        column 30, count using "##"
on last row
```
PAGE HEADER

The PAGE HEADER control block specifies what information will appear at the top of each page of the report.

The PAGE HEADER control block is a list of one or more statements or a compound statement.

PAGE HEADER are required keywords.

statement is a list of one or more statements or a compound statement.

Usage

- The vertical size of the page header is equal to the number of lines that you specify in the PAGE HEADER control block. The TOP MARGIN (in the OUTPUT section) affects how close to the top of the page ACE displays the page header.
- A FIRST PAGE HEADER control block overrides a PAGE HEADER control block on the first page of a report.
- You cannot use the SKIP TO TOP OF PAGE statement in a PAGE HEADER control block.
- If you use an IF THEN ELSE statement in a PAGE HEADER control block, the number of lines displayed by the PRINT and SKIP statements following the THEN keyword must be equal to the number of lines displayed by the PRINT and SKIP statements following the ELSE keyword.
- If you use a FOR or WHILE statement that contains a PRINT statement in a PAGE HEADER control block, you must terminate the PRINT statement with a semicolon. The semicolon suppresses any NEWLINE (RETURN) characters in the loop, keeping the number of lines in the header constant from page to page.
- You cannot use a PRINT FILE statement to read and display text from a file in a PAGE HEADER control block.
- You can use a PAGE HEADER control block to display column headings in a report.
- You can use the PAGENO expression in a PRINT statement within a PAGE HEADER control block to display the page number automatically at the top of every page.
• **ACE** delays the processing of the PAGE HEADER control block until the first PRINT, SKIP, or NEED statement to guarantee that any group columns printed in the PAGE HEADER control block have the same values as the columns printed in the ON EVERY ROW control block.

The following example is from the `clist1.ace` report:

```ace
page header
  print "NUMBER",
    column 9, "NAME",
    column 32, "LOCATION",
    column 54, "ZIP",
    column 62, "PHONE"
  skip 1 line
```
The PAGE TRAILER control block specifies what information will appear at the bottom of each page of the report.

**PAGE TRAILER**

The vertical size of the page trailer is equal to the number of lines that you specify in the PAGE TRAILER control block. The BOTTOM MARGIN (in the OUTPUT section) affects how close to the bottom of the page ACE displays the page trailer.

- You cannot use the SKIP TO TOP OF PAGE statement in a PAGE TRAILER control block.
- If you use an IF THEN ELSE statement in a PAGE TRAILER control block, the number of lines displayed by the PRINT and SKIP statements following the THEN keyword must be equal to the number of lines displayed by the PRINT and SKIP statements following the ELSE keyword.
- If you use a FOR or WHILE statement that contains a PRINT statement in a PAGE TRAILER control block, you must terminate the PRINT statement with a semicolon. The semicolon suppresses any NEWLINE (RETURN) characters in the loop, keeping the number of lines in the trailer constant from page to page.
- You cannot use the PRINT FILE statement to read and display text from a file in a PAGE TRAILER control block.
- You can use the PAGENO expression in a PRINT statement within a PAGE TRAILER control block to display the page number automatically at the bottom of every page.

The following example is from the **ord3.ace** report:

```
page trailer
  print column 28, pageno using "page <<<<"
```
Statements

The format control blocks determine *when* ACE takes an action, while the statements determine *what* action ACE takes.

Statements are composed of keywords and expressions, as explained under each of the specific statements.

Any statement can be a single statement or a compound statement. A compound statement is one or more statements, including other compound statements, preceded by the BEGIN keyword and followed by an END keyword.
The FOR statement defines a loop. It repeatedly executes a simple or compound statement, incrementing the loop index before each pass through the loop. Control passes to the first statement following the end of the loop when the termination condition is satisfied.

FOR loop-index = expr1 TO expr2 [STEP expr3] DO statement

- **FOR** is a required keyword.
- **loop-index** is the name of a variable declared in the DEFINE section. The FOR statement uses this variable as the loop index, changing its value each time through the loop.
- **=** is a required keyword.
- **expr1** is a required expression that specifies the starting value of the loop index.
- **TO** is a required keyword.
- **expr2** is a required expression that specifies the ending value of the loop index. This expression specifies the termination condition for the loop.
- **STEP** is an optional keyword.
- **expr3** is an optional expression that specifies the amount the FOR statement increments the loop index each time through the loop. If you do not specify STEP and expr3, the FOR statement assumes an increment value of 1.
- **DO** is a required keyword.
- **statement** is a single statement or a compound statement.

**Usage**

- You cannot have a decrementing loop—the value of expr3 must be positive.
- If a compound statement follows the DO keyword, you must precede the compound statement with a BEGIN keyword and follow it with END.
IF THEN ELSE

This statement defines a conditional branch. It evaluates an expression and executes specific statements based on the result of the evaluation.

\[
\text{IF} \quad \text{expr} \quad \text{THEN} \quad \text{statement1} \quad \text{ELSE} \quad \text{statement2}
\]

IF is a required keyword.
expr is a required expression that determines which, if any, of the statements IF executes.
THEN is a required keyword.
statement1 is a required single statement or compound statement that IF executes if \( expr \) evaluates as true (not equal to zero).
ELSE is an optional keyword.
statement2 is an optional single statement or compound statement that IF executes if \( expr \) evaluates as false (equal to zero).

Usage

- If a compound statement follows the THEN or ELSE keyword, you must precede the compound statement with the BEGIN keyword and follow it with END.
- You can nest IF THEN ELSE statements to 128 levels.
The following example is from the `mail3.ace` report:

```plaintext
if i = count1 then
    begin
        print array1 clipped
        print array2 clipped
        print array3 clipped
        skip 1 line
        let array1 = " "
        let array2 = " "
        let array3 = " 
        let i = 1
    end
else
    let i = i + 1
```
LET

The LET statement assigns a value to a declared variable.

Let

variable

is a required variable name; that is, a variable declared in a previous DEFINE section.

num-expr

is an optional number expression or list of one or two number expressions. They specify a substring of the CHAR variable to which LET is to assign a value. If one num-expr is specified, the substring is the beginning character within the variable through the end of the string. If two num-expr are present, the substring is from the first num-expr through the second one.

You can only use these substring operations with a CHAR variable. You must enclose subscripts in brackets ( [ ] ).

= is a required keyword.

eval

is a required list of one or more expressions separated by commas. LET assigns the value of this list to the variable.

If the list contains more than one expression, the variable must be of type CHAR. LET assigns the value of the string generated by the concatenation of all expressions in the expression list to the variable. The result that ACE assigns to the variable takes the same form as if you had displayed the same expression list with a PRINT statement. LET accepts all the expressions that PRINT accepts including USING, CLIPPED, ASCII, COLUMN, and subscripted character expressions.
Usage

- If you assign a value with a fractional part to an INTEGER or SMALLINT variable, ACE truncates the fractional part.
- Refer to the descriptions of expressions beginning on page 4-71 for more information about type conversion.

The following example is from the FORMAT section of the `mail3.ace` report:

```plaintext
let i = 1
let l_size = 72/count1
let white = 8/count1
```

When NLS is active, conversion of a monetary or numeric value to a character string using the LET statement results in a string containing locale-specific formatting. Monetary values take on separators and currency symbols specified by LC_MONETARY. Numeric values take on separators specified by LC_NUMERIC. This is true for both the default conversion and the conversion with a USING clause. However, if DBFORMAT or DBMONEY is set, these settings will override settings in LC variables. Refer to Appendix C, “Native Language Support Within INFORMIX-SQL.”
NEED

This statement causes subsequent display to start on the next page if the specified number of lines cannot be placed on the current page.

\[ \text{NEED} \quad \text{num-expr} \quad \text{LINES} \]

NEED is a required keyword.
num-expr is an expression that evaluates to a number specifying the number of lines needed.
LINES is a required keyword.

Usage

Use the NEED statement to prevent ACE from separating parts of the report that you want to keep together on a single page.
PAUSE

This statement causes output to the terminal to pause until you press RETURN.

```
PAUSE

"string"
```

PAUSE is a required keyword.

`string` is an optional message that PAUSE displays. If you do not supply a message, PAUSE does not display a message.

Usage

The PAUSE statement has no effect if you use a REPORT TO `filename` or a REPORT TO PRINTER statement in the OUTPUT section.

The following example causes ACE to pause while running the report:

```
after group of item_num

skip to top of page
pause "Press RETURN to continue"
```
PRINT

This statement displays information on the screen or as specified in the OUTPUT section.

PRINT is a required keyword.

expr is an optional list of one or more expressions, separated by commas.

; is an optional keyword that suppresses a NEWLINE (RETURN) at the end of the line.

Usage

- Unless you use the optional WORDWRAP keyword, one PRINT statement displays its output on one line, no matter how many lines the statement occupies in the report specification.
- Unless you use the keyword CLIPPED or USING following an expression, ACE displays an expression so that it occupies a predetermined number of spaces.

<table>
<thead>
<tr>
<th>Column</th>
<th>Default Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>declared size</td>
</tr>
<tr>
<td>DATE</td>
<td>10</td>
</tr>
<tr>
<td>FLOAT</td>
<td>14 (including sign and decimal point)</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>6 (including sign)</td>
</tr>
<tr>
<td>INTEGER</td>
<td>11 (including sign)</td>
</tr>
<tr>
<td>SMALLFLOAT</td>
<td>14 (including sign and decimal point)</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>number of digits plus 2 (including sign and decimal point)</td>
</tr>
<tr>
<td>SERIAL</td>
<td>11</td>
</tr>
<tr>
<td>MONEY</td>
<td>number of digits plus 3 (including sign, decimal point, and dollar sign)</td>
</tr>
<tr>
<td>DATETIME</td>
<td>depends on precision</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>depends on precision</td>
</tr>
</tbody>
</table>

Figure 4-3 Default display widths
The following example is from the clist2.ace report:

```
first page header
print column 32, "CUSTOMER LIST"
print column 32, "----------"
skip 2 lines
print "Listings for the State of ", thisstate
skip 2 lines
print "NUMBER",
    column 9, "NAME",
    column 32, "LOCATION",
    column 54, "ZIP",
    column 62, "PHONE"
skip 1 line
page header
print "NUMBER",
    column 9, "NAME",
    column 32, "LOCATION",
    column 54, "ZIP",
    column 62, "PHONE"
skip 1 line
on every row
print customer_num using "####",
    column 9, fname clipped, 1 space, lname clipped,
    column 32, city clipped, ", ", state,
    column 54, zipcode,
    column 62, phone
```
PRINT FILE

This statement displays the contents of a text file in a report.

```
PRINT FILE "filename"
```

PRINT FILE are required keywords.
filename is a required filename that can be a pathname. You must enclose the filename in quotation marks.

Usage

You can use the PRINT FILE statement to include the body of a form letter in a report that generates custom letters.
**SKIP**

This statement skips lines in a report.

```
  SKIP    int    LINES
```

- **SKIP** is a required keyword.
- **int** is an integer specifying the number of lines to skip.
- **LINES** is a required keyword. You can use the keyword **LINE** in place of **LINES** if you like.

The following example is from the `mail1.ace` report:

```
format
  on every row
    print fname, lname
    print company
    print address1
    print address2
    print city, ", " , state,
    2 spaces, zipcode
    skip 2 lines
```
This statement causes subsequent printing to begin at the top of the next page.

Usage

You cannot use a SKIP TO TOP OF PAGE statement in a FIRST PAGE HEADER, PAGE HEADER, or PAGE TRAILER control block.

The following example is from the `mail2.ace` report:

```plaintext
format
on every row
  if (city is not null) and
    (state is not null) then
    begin
      print fname clipped, 1 space, lname
      print company
      print address1
      if (address2 is not null) then
        print address2
      print city clipped, ", ", state,
        2 spaces, zipcode
      skip to top of page
    end
  end
end
```
The WHILE statement defines a loop that repeatedly executes a simple or compound statement while the expression is true. Control passes to the first statement following the loop when the expression evaluates as false.

WHILE expression DO statement

**WHILE**
is a required keyword.

**expression**
is a required expression. While this expression evaluates as true, WHILE executes the loop. When this expression evaluates as false, control passes to the first statement following the loop.

**DO**
is a required keyword.

**statement**
is a single statement or a compound statement.

**Usage**

If more than one statement follows the DO keyword, you must precede them with the BEGIN keyword and follow them with END to create a compound statement.
Aggregates

Aggregates allow you to summarize information in a report.

- **GROUP** is an optional keyword that causes the aggregate to reflect information for a specific group only. The group must have been designated by an ORDER BY clause in the SELECT or READ section of the report specification. You can only use this keyword in an AFTER GROUP OF control block.

- **COUNT** is a keyword that is evaluated as the total number of rows qualified by the SELECT section or retrieved by the READ section, and qualified by the optional WHERE expr2 expression that can appear in an aggregate statement.

- **PERCENT** is the keyword that evaluates COUNT as a percent of the total number of rows qualified by the SELECT section or retrieved by the READ section, and qualified by the optional WHERE expr2 expression that can appear in an aggregate statement.

- **TOTAL** evaluates as the total of expr1 in the rows qualified by the SELECT section or retrieved by the READ section, and qualified by the optional WHERE expr2 expression that can appear in an aggregate statement.

- **AVERAGE**/AVG evaluates as the average of expr1 in the rows qualified by the SELECT section or retrieved by the READ section, and qualified by the optional WHERE expr2 expression that can appear in an aggregate statement.

- **MIN** evaluates as the minimum of expr1 in the rows qualified by the SELECT section or retrieved by the READ section, and qualified by the optional WHERE expr2 expression that can appear in an aggregate statement.
MAX evaluates as the maximum of expr1 in the rows qualified by the SELECT section or retrieved by the READ section, and qualified by the optional WHERE expr2 expression that can appear in an aggregate statement.

OF is a keyword.

expr1 is the expression that TOTAL, AVERAGE, MIN, and MAX evaluate. It is typically a column or a number expression that includes a number column.

WHERE is a keyword.

expr2 is a logical expression that qualifies the aggregate.

Usage

- The WHERE part of an aggregate statement further qualifies rows that the SELECT section already qualified or that the READ statement already retrieved. WHERE cannot select rows that were not qualified by the SELECT section or retrieved by the READ section.
- Aggregates produce unpredictable results when expr1 or expr2 contains user-defined variables. (See “PARAM” on page 4-18 and “VARIABLE” on page 4-19.)

The following example is from the ord2.ace report:

```plaintext
on every row
  print snum using "###", column 10, manu_code, column 18, description clipped, column 38, quantity using "###", column 43, unit_price using "$$$$&&", column 55, total_price using "$$,$$,$$$&&"

after group of number
  skip 1 line
  print 4 spaces, "Shipping charges for the order: ", ship_charge using "$$$.&&"
  skip 1 line
  print 5 spaces, "Total amount for the order: ", ship_charge + group total of total_price using "$$,$$,$$$&&"
```
ASCII

ACE evaluates this expression as a value that you can use as a character. You can use it to display control characters.

\[
\text{ASCII} \quad num-expr
\]

ASCII is a required keyword.

num-expr is a number expression.

Usage

Do not confuse this keyword with the ASCII keyword used in the DEFINE section to specify the identifiers and data values of an input file.

The following PRINT statement rings the bell (ASCII value of 7) of your computer.

\[
\text{print ascii 7}
\]

The next report specification segments show how to implement special printer or computer functions. They assume that when the printer receives the sequence of ASCII characters 9, 11, and 1, it starts printing in red, and when it receives 9, 11, and 0, it reverts to black printing. The values used in the example are hypothetical; refer to your printer or terminal manual for information on your printer or terminal.
This specification uses the FIRST PAGE HEADER control block to initialize variables that are used in other control blocks.

```
define
  variable red_on char(3)
  variable red_off char(3)
end

format
  first page header
    let red_on =
      ascii 9, ascii 11, ascii 1
    let red_off =
      ascii 9, ascii 11, ascii 0
  on every row
    print red_on,
      "Your bill is overdue.",
    red_off
```

**Note:** ACE cannot distinguish printable and nonprintable ASCII characters. Be sure to account for the nonprinting characters when you use the COLUMN expression to format your page. Since various devices print spaces with control characters differently, you might have to use trial and error to line up columns when you print control characters.
CLIPPED

This expression displays the character field that precedes it without any trailing blanks.

\[ \text{char-expr} \quad \text{CLIPPED} \]

\textit{char-expr} is a required character expression.

\textit{CLIPPED} is a required keyword.

Usage

You normally use CLIPPED following a \textit{column-name} in a PRINT statement.

The following example is from the \texttt{mail2.ace} report:

\begin{verbatim}
format
  on every row
  if (city is not null) and
    (state is not null) then
  begin
    print fname clipped, 1 space, lname
    print company
    print address1
    if (address2 is not null) then
      print address2
    print city clipped, ", " , state,
    2 spaces, zipcode
    skip to top of page
  end
end
\end{verbatim}
COLUMNN

This expression evaluates to a string of spaces long enough to position the
next item in the designated column.

\[ \text{COLUMN } \text{num-expr} \]

COLUMN is a required keyword.
num-expr is a required number expression that specifies the column
position for the next item to be printed.

Usage

- ACE calculates the column number by adding the number to the left mar-
gin you set in the OUTPUT section.
- If ACE has already printed past the column specified by \text{num-expr}, ACE
ignores the COLUMN expression.

The following example is from the \text{clist2.ace} report:

\begin{verbatim}
page header
print "NUMBER",
    column 9, "NAME",
    column 32, "LOCATION",
    column 54, "ZIP",
    column 62, "PHONE"
skip 1 line
\end{verbatim}
CURRENT

This expression evaluates as a character string with the value of the current date and time as supplied by the operating system.

CURRENT is a required keyword.

first is an optional qualifier that specifies the first field to be returned.

TO is a required keyword if you include first and last qualifiers.

last is an optional qualifier that specifies the last field to be returned.

The following example prints the current date and time to a precision of MINUTE:

print current year to minute
DATE

This expression evaluates as a character string with a value of today’s date in the form “Thu Feb 17 1993.”

Because DATE evaluates as type CHAR, you can use it with subscripts to express a day, month, date, or year. The following example:

```
print "Today is ", date[1,3]
```

displays the three-letter abbreviation for the day of the week.

Today is Mon

The installation of message files in a subdirectory of $INFORMIXDIR/msg and subsequent reference to that subdirectory by way of the environment variable DBLANG causes month and day portions of the character string returned by DATE to contain language-specific month name and day name abbreviations. For example, in a Spanish locale the day Saturday is translated into the day name abbreviation Sab, which stands for Sabado (the Spanish word for Saturday). Refer to “DBLANG” on page B-18.
DATE()

The DATE function converts the expression with which you call it to type DATE.

```markdown
DATE ( date-expr )
```

- **DATE** is a required keyword.
- **date-expr** is a required expression of any type that evaluates to a type DATE value.

**Usage**

- The DATE function is typically used to convert date strings to type DATE.
- A properly formatted date string is required. The default format is *mm/dd/yy*, but this may be changed by way of the environment variable DBDATE.
DAY(

The DAY function returns the day of the month when you call it with a type DATE or DATETIME expression.

DAY (date-expr)

DAY is a required keyword.

date-expr is a required expression that evaluates to a type DATE or DATETIME value.
LINENO

This expression has the value of the line number of the line that ACE is currently displaying. ACE computes the line number by calculating the number of lines from the top of the page.

Usage

Do not use LINENO within a page header or trailer. LINENO works on the first page header but does not work on any of the subsequent pages.
The MDY function returns a type DATE value when you call it with three expressions that evaluate to integers representing the month, date, and year.

\[
\text{MDY} \left( \text{num-expr1}, \text{num-expr2}, \text{num-expr3} \right)
\]

- **MDY** is a required keyword.
- **num-expr1** is an expression that evaluates to an integer that represents the number of the month (1-12).
- **num-expr2** is an expression that evaluates to an integer that represents the number of the day of the month (1-28, 29, 30, or 31, depending on the month).
- **num-expr3** is an expression that evaluates to a four-digit integer that represents the year.
MONTH()

The MONTH function returns an integer that corresponds to the month (1-12) of its type DATE or DATETIME argument.

MONTH ( date-expr )

MONTH is a required keyword.
date-expr is a required expression that evaluates to a type DATE or DATETIME value.
PAGENO

This expression has the value of the page number of the page that ACE is currently displaying.

Usage

Use PAGENO in a PRINT statement in the PAGE HEADER or PAGE TRAILER control block to number the pages of a report. (You can also use PAGENO in other control blocks.)

The following example is from the ord3.ace report:

```
page trailer
  print column 28, pageno using "page <<<<"
```
This expression evaluates as a string of spaces. It is identical to a quoted string of spaces.

```
  num-expr  SPACE  SPACES
```

`num-expr` is a number expression that designates how many spaces.

`SPACE[S]` is a required keyword. You can use either the keyword `SPACE` or `SPACES`.

The following example is from the `mail1.ace` report:

```
format
  on every row
    print fname, lname
    print company
    print address1
    print address2
    print city, "", state,
    2 spaces, zipcode
    skip 2 lines
  end
```
This expression evaluates as a character string with a value of the current time in the form \textit{hh:mm:ss}.
This expression evaluates as type DATE with a value of the current date as supplied by the operating system.

The following example is from the `ord3.ace` report:

```verbatim
skip 1 line
print column 15, "FROM: ", begin_date
   using "mm/dd/yy",
column 35, "TO: ", end_date
   using "mm/dd/yy"
print column 15, "Report run date: ",
   today using "mmm dd, yyyy"
skip 2 lines
print column 2, "ORDER DATE", column 15,
    "COMPANY", column 35, "NAME",
    column 57, "NUMBER", column 65, "AMOUNT"
```
This expression allows you to format a number or date expression. With a number expression, you can use USING to line up decimal points, right- or left-justify numbers, put negative numbers in parentheses, and perform other formatting functions. With a date expression, USING converts the date to a variety of formats.

\[ expr1 \quad \text{USING} \quad expr2 \]

- **expr1** is the required expression that specifies what USING is to format.
- **expr2** is the required format string that specifies how USING is to format **expr1**.

**Usage**

- The format string must appear between quotation marks.
- Although USING is generally used as part of a PRINT statement, you can also use it with LET.
- If you attempt to display a number that is too large for a display field, ACE fills the field with asterisks to indicate an overflow.

**Formatting Number Expressions**

The format string consists of combinations of the following characters: * & # < . - + ( ) $. The following characters float: - + ( ) $. When a character floats, ACE displays multiple leading occurrences of the character as a single character as far to the right as possible, without interfering with the number that is being displayed. Refer to the following list:

- **\*** This character fills with asterisks any positions in the display field that would otherwise be blank.
- **&** This character fills with zeros positions in the display field that would otherwise be blank.
- **#** This character does not change any blank positions in the display field. Use this character to specify a maximum width for a field.
- **<** This character causes the numbers in the display field to be left-justified.
The ACE Report Writer  4-87

,  This character is a placeholder for the thousands separator. The default thousands separator is a comma. Environment variables DBFORMAT, DBMONEY, and LANG determine the thousands separator.
.
This character is a placeholder for the decimal separator. The default decimal separator is a period. Environment variables DBFORMAT, DBMONEY, and LANG determine the decimal separator.
-
This character is a literal; USING displays it as a minus sign when expr1 is less than zero. When you group several in a row, a single minus sign floats to the rightmost position without interfering with the number being printed.
+
This character is a literal; USING displays it as a plus sign when expr1 is greater than or equal to zero and as a minus sign when it is less than zero. When you group several in a row, a single plus sign floats to the rightmost position without interfering with the number being printed.
(  This character is a literal; USING displays it as a left parenthesis before a negative number. It is the accounting parenthesis that is used in place of a minus sign to indicate a negative number. When you group several in a row, a single left parenthesis floats to the rightmost position without interfering with the number being printed.
)  This is the accounting parenthesis that is used in place of a minus sign to indicate a negative number. A single one of these characters generally closes a format string that begins with a left parenthesis.
$  This character is a placeholder for the leading currency symbol. The default leading currency symbol is a dollar sign. Environment variables DBFORMAT, DBMONEY, and LANG determine the leading currency symbol. When you group several in a row, a single dollar sign (or leading specified currency symbol) floats to the rightmost position without interfering with the number being printed.
@  This character is a placeholder for the trailing currency symbol. The default trailing currency symbol defaults to NULL. Environment variables DBFORMAT, DBMONEY, and LANG determine the trailing currency symbol. When you group several in a row, a single at sign (or NLS locale specified trailing currency symbol) floats to the leftmost position without interfering with the number being printed.

Refer to page 4-90 for examples of formatting number expressions.
Formatting Date Expressions

The format string consists of combinations of the characters \( m, d, \) and \( y \) shown in Figure 4-4.

<table>
<thead>
<tr>
<th>Format Substring</th>
<th>Formatted Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>( dd )</td>
<td>Day of the month as a two-digit number (01-31)</td>
</tr>
<tr>
<td>( ddd )</td>
<td>Day of the week as a three-letter abbreviation (Sun through Sat)</td>
</tr>
<tr>
<td>( mm )</td>
<td>Month as a two-digit number (01-12)</td>
</tr>
<tr>
<td>( mmm )</td>
<td>Month as a three-letter abbreviation (Jan through Dec)</td>
</tr>
<tr>
<td>( yy )</td>
<td>Year as a two-digit number in the 1900s (00-99)</td>
</tr>
<tr>
<td>( yyyy )</td>
<td>Year as a four-digit number (0001-9999)</td>
</tr>
</tbody>
</table>

**Figure 4-4** Combinations of date format characters

The following example shows some sample conversions for December 25, 1994:

<table>
<thead>
<tr>
<th>Format String</th>
<th>Formatted Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>“mmddyy”</td>
<td>122594</td>
</tr>
<tr>
<td>“ddmmyy”</td>
<td>251294</td>
</tr>
<tr>
<td>“yymmdd”</td>
<td>941225</td>
</tr>
<tr>
<td>“yy/mm/dd”</td>
<td>94/12/25</td>
</tr>
<tr>
<td>“yy mm dd”</td>
<td>94 12 25</td>
</tr>
<tr>
<td>“yy-mm-dd”</td>
<td>94-12-25</td>
</tr>
<tr>
<td>“mm.m. dd, yyyy”</td>
<td>Dec. 25, 1994</td>
</tr>
<tr>
<td>“mmm dd yyyy”</td>
<td>Dec. 25, 1994</td>
</tr>
<tr>
<td>“yyyy dd mm”</td>
<td>1994 25 12</td>
</tr>
<tr>
<td>“mmmm dd yyyy”</td>
<td>Dec 25 1994</td>
</tr>
<tr>
<td>“ddd, mmm. dd, yyyy”</td>
<td>Tue, Dec. 25, 1994</td>
</tr>
<tr>
<td>“(ddd) mmm. dd, yyyy”</td>
<td>(Tue) Dec. 25, 1994</td>
</tr>
</tbody>
</table>

**Figure 4-5** Results of date format strings
The setting in the NLS environment variables LC_MONETARY, LC_NUMERIC, and LANG affect the way the format string in the USING expression is interpreted for numeric and monetary data. In the format string, the period symbol (.) is not a literal character but a placeholder for the decimal separator specified by environment variables. Likewise, the comma symbol (,) is a placeholder for the thousands separator specified by environment variables. The $ symbol is a placeholder for the leading currency symbol. The @ symbol is a placeholder for the trailing currency symbol. Thus, the format string $$#,###,##&.&& will format the value 1234.56 as $1,234.56 in a US English locale but as DM1.234,56 in a German locale. Note that setting the DBFORMAT or DBMONEY environment variables override settings in LC variables. Refer to Appendix C, “Native Language Support Within INFORMIX-SQL.”

The installation of locale files in a subdirectory of $INFORMIXDIR/msg and subsequent reference to that subdirectory by way of the environment variable DBLANG causes mmm and ddd specifiers in a format string to display locale-specific month name and day name abbreviations on the form. For example, the ddd specifier in a Spanish locale translates the day Saturday into the day name abbreviation Sab., which stands for Sábado (the Spanish word for Saturday). Refer to the section entitled “DBLANG” on page B-18.

The following example prints the balance field using a format string that allows up to $9,999,999.99 to be formatted correctly:

```sql
print "The current balance is ",
   23485.23 using "$#,###,##&.&&"
```

The result of executing this PRINT statement with the value 23,485.23 follows:

```
The current balance is $ 23,485.23
```

This example fixes the dollar sign. If dollar signs had been used instead of # characters, the dollar sign would have floated with the size of the number. It also uses a mix of # and & fill characters. The # character provides blank fill for unused character positions, while the & character provides zero filling. This format ensures that even if the number is zero, the positions marked with & characters appear as zeros, not blanks.

The tables on the following pages illustrate the results of various combinations of data and USING format strings.
### Sample Format Strings

<table>
<thead>
<tr>
<th>Format String</th>
<th>Data Value</th>
<th>Formatted Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;####&quot;</td>
<td>0</td>
<td>bbbbb</td>
</tr>
<tr>
<td>&quot;&amp;&amp;&amp;&amp;&quot;</td>
<td>0</td>
<td>00000</td>
</tr>
<tr>
<td>&quot;$$$$&quot;</td>
<td>0</td>
<td>bbbb$</td>
</tr>
<tr>
<td>&quot;*****&quot;</td>
<td>0</td>
<td>*****</td>
</tr>
<tr>
<td>&quot;&lt;&lt;&lt;&lt;&quot;</td>
<td>12345</td>
<td>12,345</td>
</tr>
<tr>
<td>&quot;&lt;&lt;&lt;&lt;&quot;</td>
<td>1234</td>
<td>1,234</td>
</tr>
<tr>
<td>&quot;&lt;&lt;&lt;&lt;&quot;</td>
<td>123</td>
<td>123</td>
</tr>
<tr>
<td>&quot;&lt;&lt;&lt;&lt;&quot;</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>&quot;###,###&quot;</td>
<td>12345</td>
<td>12,345</td>
</tr>
<tr>
<td>&quot;###,###&quot;</td>
<td>1234</td>
<td>b1,234</td>
</tr>
<tr>
<td>&quot;###,###&quot;</td>
<td>123</td>
<td>bbb123</td>
</tr>
<tr>
<td>&quot;###,###&quot;</td>
<td>12</td>
<td>bbb12</td>
</tr>
<tr>
<td>&quot;###,###&quot;</td>
<td>1</td>
<td>bbb1</td>
</tr>
<tr>
<td>&quot;###,###&quot;</td>
<td>0</td>
<td>bbb</td>
</tr>
<tr>
<td>&quot;&lt;,&gt;&lt;&quot;</td>
<td>12345</td>
<td>12,345</td>
</tr>
<tr>
<td>&quot;&lt;,&gt;&lt;&quot;</td>
<td>1234</td>
<td>1,234</td>
</tr>
<tr>
<td>&quot;&lt;,&gt;&lt;&quot;</td>
<td>123</td>
<td>123</td>
</tr>
<tr>
<td>&quot;&lt;,&gt;&lt;&quot;</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>&quot;##,###&quot;</td>
<td>12345</td>
<td>12,345.67</td>
</tr>
<tr>
<td>&quot;##,###&quot;</td>
<td>1234</td>
<td>1234.56</td>
</tr>
<tr>
<td>&quot;##,###&quot;</td>
<td>123</td>
<td>00123.45</td>
</tr>
<tr>
<td>&quot;##,###&quot;</td>
<td>12</td>
<td>0000123.45</td>
</tr>
<tr>
<td>&quot;##,###&quot;</td>
<td>1</td>
<td>000001.01</td>
</tr>
<tr>
<td>&quot;##,###&quot;</td>
<td>0</td>
<td>000000.01</td>
</tr>
<tr>
<td>&quot;$$, $$&quot;</td>
<td>12345</td>
<td>*****</td>
</tr>
<tr>
<td>&quot;$$, $$&quot;</td>
<td>1234</td>
<td>$1,234</td>
</tr>
<tr>
<td>&quot;$$, $$&quot;</td>
<td>123</td>
<td>bb$123</td>
</tr>
<tr>
<td>&quot;$$, $$&quot;</td>
<td>12</td>
<td>bbb$12</td>
</tr>
<tr>
<td>&quot;$$, $$&quot;</td>
<td>1</td>
<td>bbb$b1</td>
</tr>
<tr>
<td>&quot;$$, $$&quot;</td>
<td>0</td>
<td>bbb$</td>
</tr>
<tr>
<td>&quot;<strong>,</strong>*&quot;</td>
<td>12345</td>
<td>12,345</td>
</tr>
<tr>
<td>&quot;<strong>,</strong>*&quot;</td>
<td>1234</td>
<td>*1,234</td>
</tr>
<tr>
<td>&quot;<strong>,</strong>*&quot;</td>
<td>123</td>
<td>***123</td>
</tr>
<tr>
<td>&quot;<strong>,</strong>*&quot;</td>
<td>12</td>
<td>***123</td>
</tr>
<tr>
<td>&quot;<strong>,</strong>*&quot;</td>
<td>1</td>
<td>*****1</td>
</tr>
<tr>
<td>&quot;<strong>,</strong>*&quot;</td>
<td>0</td>
<td>*****</td>
</tr>
</tbody>
</table>

Here the character b represents a blank or space.
## The ACE Report Writer

<table>
<thead>
<tr>
<th>Format String</th>
<th>Data Value</th>
<th>Formatted Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;##,###.##&quot;</td>
<td>12345.67</td>
<td>12,345.67</td>
</tr>
<tr>
<td>&quot;##,###.##&quot;</td>
<td>1234.56</td>
<td>b1,234.56</td>
</tr>
<tr>
<td>&quot;##,###.##&quot;</td>
<td>123.45</td>
<td>bbb123.45</td>
</tr>
<tr>
<td>&quot;##,###.##&quot;</td>
<td>12.34</td>
<td>bbb12.34</td>
</tr>
<tr>
<td>&quot;##,###.##&quot;</td>
<td>1.23</td>
<td>bbb1.23</td>
</tr>
<tr>
<td>&quot;##,###.##&quot;</td>
<td>0.12</td>
<td>bbbb0.12</td>
</tr>
<tr>
<td>&quot;##,###.##&quot;</td>
<td>0.01</td>
<td>bbbbbb.01</td>
</tr>
<tr>
<td>&quot;##,###.##&quot;</td>
<td>-0.01</td>
<td>bbbbbb.01</td>
</tr>
<tr>
<td>&quot;##,###.##&quot;</td>
<td>-1</td>
<td>bbbbbb1.00</td>
</tr>
<tr>
<td>&quot;$$,###.##&quot;</td>
<td>12345.67</td>
<td>********* (overflow)</td>
</tr>
<tr>
<td>&quot;$$,###.##&quot;</td>
<td>1234.56</td>
<td>$1,234.56</td>
</tr>
<tr>
<td>&quot;$$,###.##&quot;</td>
<td>1234.00</td>
<td>$1,234.00</td>
</tr>
<tr>
<td>&quot;$$,###.##&quot;</td>
<td>0.00</td>
<td>$.00</td>
</tr>
<tr>
<td>&quot;$$,###.##&quot;</td>
<td>0.00</td>
<td>$.00</td>
</tr>
</tbody>
</table>

Here the character `b` represents a blank or space.
<table>
<thead>
<tr>
<th>Format String</th>
<th>Data Value</th>
<th>Formatted Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;----,--$.&amp;&amp;&quot;</td>
<td>-12345.67</td>
<td>-$12,345.67</td>
</tr>
<tr>
<td>&quot;----,--$.&amp;&amp;&quot;</td>
<td>-1234.56</td>
<td>-$1,234.56</td>
</tr>
<tr>
<td>&quot;----,--$.&amp;&amp;&quot;</td>
<td>-123.45</td>
<td>-$123.45</td>
</tr>
<tr>
<td>&quot;----,--$.&amp;&amp;&quot;</td>
<td>-12.34</td>
<td>-$12.34</td>
</tr>
<tr>
<td>&quot;----,--$.&amp;&amp;&quot;</td>
<td>-1.23</td>
<td>-$1.23</td>
</tr>
<tr>
<td>&quot;----,--$.&amp;&amp;&quot;</td>
<td>-.12</td>
<td>-.12</td>
</tr>
<tr>
<td>&quot;$$<em>,</em>**.&amp;&amp;&quot;</td>
<td>12345.67</td>
<td>$12,345.67</td>
</tr>
<tr>
<td>&quot;$$<em>,</em>**.&amp;&amp;&quot;</td>
<td>1234.56</td>
<td>$1,234.56</td>
</tr>
<tr>
<td>&quot;$$<em>,</em>**.&amp;&amp;&quot;</td>
<td>123.45</td>
<td>$123.45</td>
</tr>
<tr>
<td>&quot;$$<em>,</em>**.&amp;&amp;&quot;</td>
<td>12.34</td>
<td>$12.34</td>
</tr>
<tr>
<td>&quot;$$<em>,</em>**.&amp;&amp;&quot;</td>
<td>1.23</td>
<td>$1.23</td>
</tr>
<tr>
<td>&quot;$$<em>,</em>**.&amp;&amp;&quot;</td>
<td>.12</td>
<td>$.12</td>
</tr>
<tr>
<td>&quot;($$$,$$$.&amp;&amp;)&quot;</td>
<td>-12345.67</td>
<td>($12,345.67)</td>
</tr>
<tr>
<td>&quot;($$$,$$$.&amp;&amp;)&quot;</td>
<td>-1234.56</td>
<td>($1,234.56)</td>
</tr>
<tr>
<td>&quot;($$$,$$$.&amp;&amp;)&quot;</td>
<td>-123.45</td>
<td>($123.45)</td>
</tr>
<tr>
<td>&quot;($$$,$$$.&amp;&amp;)&quot;</td>
<td>-12.34</td>
<td>($12.34)</td>
</tr>
<tr>
<td>&quot;($$$,$$$.&amp;&amp;)&quot;</td>
<td>-1.23</td>
<td>($1.23)</td>
</tr>
<tr>
<td>&quot;($$$,$$$.&amp;&amp;)&quot;</td>
<td>-.12</td>
<td>($0.12)</td>
</tr>
<tr>
<td>&quot;((((,,(()$.&amp;&amp;)&quot;</td>
<td>-12345.67</td>
<td>($12,345.67)</td>
</tr>
<tr>
<td>&quot;((((,,(()$.&amp;&amp;)&quot;</td>
<td>-1234.56</td>
<td>($1,234.56)</td>
</tr>
<tr>
<td>&quot;((((,,(()$.&amp;&amp;)&quot;</td>
<td>-123.45</td>
<td>($123.45)</td>
</tr>
<tr>
<td>&quot;((((,,(()$.&amp;&amp;)&quot;</td>
<td>-12.34</td>
<td>($12.34)</td>
</tr>
<tr>
<td>&quot;((((,,(()$.&amp;&amp;)&quot;</td>
<td>-1.23</td>
<td>($1.23)</td>
</tr>
<tr>
<td>&quot;((((,,(()$.&amp;&amp;)&quot;</td>
<td>-.12</td>
<td>($0.12)</td>
</tr>
<tr>
<td>&quot;($$$, $$$$.&amp;&amp;)&quot;</td>
<td>12345.67</td>
<td>$12,345.67</td>
</tr>
<tr>
<td>&quot;($$$, $$$$.&amp;&amp;)&quot;</td>
<td>1234.56</td>
<td>$1,234.56</td>
</tr>
<tr>
<td>&quot;($$$, $$$$.&amp;&amp;)&quot;</td>
<td>123.45</td>
<td>$123.45</td>
</tr>
<tr>
<td>&quot;($$$, $$$$.&amp;&amp;)&quot;</td>
<td>12.34</td>
<td>$12.34</td>
</tr>
<tr>
<td>&quot;($$$, $$$$.&amp;&amp;)&quot;</td>
<td>1.23</td>
<td>$1.23</td>
</tr>
<tr>
<td>&quot;($$$, $$$$.&amp;&amp;)&quot;</td>
<td>.12</td>
<td>$.12</td>
</tr>
</tbody>
</table>

Here the character b represents a blank or space.
WEEKDAY( )

The WEEKDAY function returns an integer that represents the day of the week when you call it with a type DATE or DATETIME expression.

\[
\text{WEEKDAY} (\text{date-expr})
\]

WEEKDAY is a required keyword.

\text{date-expr} is a required expression that evaluates to a type DATE or DATETIME value.

WEEKDAY returns an integer in the range 0-6. 0 represents Sunday, 1 represents Monday, and so forth.
WORDWRAP

This expression displays the character field that precedes it on multiple lines with lines broken between words at temporary left and right margins.

\[ \text{char-expr} \quad \text{WORDWRAP} \quad \text{RIGHT MARGIN} \quad column \]

- char-expr is an expression with a CHAR value.
- WORDWRAP is a required keyword.
- RIGHT introduces a temporary right margin that overrides the right margin of the report.
- column specifies the column number of the temporary right margin.

Usage

- The temporary left margin is the current printing column. The contents of char-expr are displayed on as many lines as necessary between the temporary left and right margins.
- Line breaks are positioned to avoid breaking words where possible.
- A line break is forced where the data contains a line feed (ASCII 10), a return (ASCII 13), or a combination of the two.

The following example invokes WORDWRAP with a temporary right margin at column 70:

\[
\text{print column 10, textcol wordwrap right margin 70}
\]

YEAR( )

YEAR is a function that returns an integer that represents the year when you call it with a type DATE or DATETIME expression.

YEAR( )

YEAR is a required keyword.
date-exp is a required expression that evaluates to a type DATE or DATETIME value.
User-Menu

Chapter Overview 3

Accessing a Menu 4
  Using a Menu Within INFORMIX-SQL 4

Designing a Menu 5

Creating a Menu 8
  Accessing PERFORM with the menuform Form 9
  Entering Menu Data 10
  Steps for Entering Your Own Data 13

Modifying a Menu 14

Menu Display Fields 15
  MENU NAME 16
  MENU TITLE 17
  SELECTION NUMBER 18
  SELECTION TYPE 19
  SELECTION TEXT 21
  SELECTION ACTION 22

Creating a Script Menu 24
Chapter Overview

The User-menu option allows you to create and run custom menus. The options on a user-menu can call the following items:

- Submenus
- INFORMIX-SQL (I-SQL) programs (PERFORM, for example)
- Other programs or sets of programs in your software library
- Operating system utilities
- Forms or reports

You use a special PERFORM screen form to create or alter a menu structure. Two special tables hold the menus, text, and command references for each menu option in the menu structure.

You can create one user-menu for each database. You cannot create a user-menu without specifying a database. The options in a user-menu, however, do not have to refer to any particular database.

A user-menu cannot exist separately from a database. If you want to keep the user-menu apart from your working databases, create a database that contains only menu data.

This chapter contains six sections:

- Accessing a Menu
- Designing a Menu
- Creating a Menu
- Modifying a Menu
- Menu Display Fields
- Creating a Script Menu
Accessing a Menu

You can access a menu from the I-SQL Main menu or from the operating system command line. The next section describes the use of the User-menu option on the I-SQL Main menu. Appendix H explains how to access a user-menu directly from the operating system command line.

Using a Menu Within INFORMIX-SQL

Follow these steps to access the user-menu included with the stores demonstration database:

1. Use the Database option on the I-SQL Main menu to make the stores2 database your current database.
2. Select the User-menu option on the I-SQL Main menu. The USER-MENU menu displays (see Figure 5-6).

```
USER-MENU: Run Modify Exit
Run the user-menu for the current database.
------------------ stores2 ------------------ Press CONTROL-W for Help -----
```

Figure 5-6 The USER-MENU menu
3. Select the Run option on the USER-MENU menu. The Main menu of the user-menu for the stores2 database displays (see Figure 5-7).

4. Select a menu option by typing the number to the left of the desired option or positioning the highlight on the option with the Arrow keys. Press the RETURN key. I-SQL executes the option you select.

5. Type e to exit a menu. If you type e from the demonstration database menu, the I-SQL USER-MENU menu displays.

Designing a Menu

You can have up to 19 levels of menus in the user-menu structure and up to 28 options on each menu.

The total number of options in a single menu depends on two factors:

- The number of lines your screen can hold
- The length of the menu titles

Most screens can accommodate 14 single-spaced menu lines. Each menu line can display 2 options of up to 33 characters of text in each option. If the text for each option on a menu does not exceed 33 characters, you can display up to 14 double-spaced menu options (2 options per line) or 28 single-spaced
menu options (2 options per line). If the text for an option is longer than 33 characters, it requires an entire line, reducing the total number of options available for that menu.

As an example of a simple menu, consider the user-menu for the stores2 demonstration database. It has five options on the top or main level:

- FORMS
- REPORTS
- QUERIES
- TABLE DEFINITIONS
- UTILITIES
Each option on the Main menu calls a menu, and each option on a menu performs an action. A design outline for the user-menu included with the stores2 database might look like this:

WEST COAST WHOLESALERS, INC.

1. USE DATA ENTRY FORMS
   1. CUSTOMER ENTRY/QUERY FORM
   2. ORDER ENTRY/QUERY FORM
   3. DISPLAY CUSTOMER FORM SPECIFICATION
   4. DISPLAY ORDER FORM SPECIFICATION

2. RUN REPORTS
   1. RUN CUSTOMER REPORT
   2. RUN REPORT ON CUSTOMER BY DESIGNATED STATE
   3. RUN CUSTOMER MAILING LABELS
   4. RUN MATRIX REPORT ON MONTHLY SALES
   5. DISPLAY REPORT 1 SPECIFICATION
   6. DISPLAY REPORT 2 SPECIFICATION
   7. DISPLAY REPORT 3 SPECIFICATION
   8. DISPLAY REPORT 4 SPECIFICATION

3. EXECUTE DATABASE QUERIES
   1. DISPLAY CUSTOMER INFORMATION BASED ON PARTIAL NAME MATCH
   2. INSERT, UPDATE, SELECT, AND DELETE NEW CUSTOMER ROW
   3. DISPLAY ALL CURRENTLY UNPAID ORDERS
   4. DISPLAY INFORMATION BASED ON A VIEW
   5. DISPLAY CUSTOMERS PLUS OUTSTANDING ORDERS (NO OUTER JOIN)

4. DISPLAY DATABASE TABLE DEFINITIONS
   1. TABLE "CUSTOMER"
   2. TABLE "ITEMS"
   3. TABLE "MANUFACT"
   4. TABLE "ORDERS"
   5. TABLE "STOCK"
   6. TABLE "STATE"
   7. TABLE "CUST_CALLS"

5. UTILITY MENU
   1. DISPLAY DATE AND TIME
   2. CHECK CUSTOMER TABLE
   3. UNLOAD CUSTOMER TABLE
Creating a Menu

The first step in creating a menu is to access the PERFORM program with the menuform screen form. The second step is to enter data through this screen form.

The menuform form is a special PERFORM screen form that you use only to create or to modify a user-menu. You can enter, change, and remove menu information with the form, but you cannot change the appearance of the screen form itself, and you cannot run FORMBUILD on it.
Accessing PERFORM with the menuform Form

Select the User-menu option on the I-SQL Main menu. If there is no current database, the CHOOSE DATABASE screen appears. After you select a database, the USER-MENU menu displays. Select the Modify option and I-SQL displays the PERFORM menu with the menuform form (see Figure 5-8).

Two tables are accessed in the menuform form:

- The sysmenus table stores information about each menu in the user-menu. This information includes the Menu name (used by I-SQL to identify the menu), and the Menu title (text that you want to appear when the menu displays).

- The sysmenuitems table stores information about the options on each menu. This information includes the following:
  - The option number
  - The option type (program, report, form, I-SQL command file, script menu, or menu)
  - The option title (text that appears on the screen)
  - The action the option specifies (execute a program, report, form, or script menu, or call an I-SQL command file or a menu). A script menu allows the user to run multiple actions in sequence for a single item. Script menus are described in “Creating a Script Menu” on page 5-24.
The `sysmenus` table is the master of the `sysmenuitems` table.

The first person to press Modify from the USER-MENU menu creates both tables. The creator of the `sysmenus` and `sysmenuitems` tables is also the owner of those tables. Ownership is important when an ANSI-compliant database is created because, though anyone can run the user-menu, only the owner of the tables can modify the menu items. Any other user who tries to modify the menu items receives an error message.

### Entering Menu Data

You enter menu data in two steps. First you enter data for a menu in the two fields at the top of the screen form (the fields associated with the `sysmenus` table). For example, the entry into the `sysmenus` table for the Main menu in the `stores2` database is shown in this screen:

```
PERFORM: Query Next Previous View Add Update Remove Table . . .
Searches the active database table. ** 2: sysmenus table**

-------------------------------MENU ENTRY FORM-------------------------------
Menu Name: [main ]
Menu Title: [WEST COAST WHOLESALERS, INC. ]

--------------------SELECTION SECTION----------------------------
Selection Number: Selection Type:
Selection Text:
Selection Action:
```

The following table shows the information for the Main menu and its options, which are stored in the `sysmenus` table in the `stores2` database:

<table>
<thead>
<tr>
<th>Menu Name</th>
<th>Menu Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>WEST COAST WHOLESALERS, INC.</td>
</tr>
<tr>
<td>forms</td>
<td>FORMS</td>
</tr>
<tr>
<td>reports</td>
<td>REPORTS</td>
</tr>
<tr>
<td>queries</td>
<td>DATABASE QUERIES</td>
</tr>
<tr>
<td>tables</td>
<td>DATABASE TABLE DEFINITIONS</td>
</tr>
<tr>
<td>utilities</td>
<td>UTILITIES</td>
</tr>
</tbody>
</table>
Second, you enter data for each option of each menu in the fields on the lower half of the screen (the fields associated with the sysmenuitems table).

You must make an entry in the sysmenus table before you make entries for a menu option in the sysmenuitems table. This step is necessary because I-SQL checks when you enter data in the sysmenuitems table to make sure that menus cited in that table exist in the sysmenus table.

For example, the complete entry for option 1 on the Main menu is shown in this screen:

```
PERFORM: Query Next Previous View Add Update Remove Table . . .
Searches the active database table. ** 2: sysmenuitems table**

==========================MENU ENTRY FORM=================================
Menu Name: [main ]
Menu Title: WEST COAST WHOLESALERS, INC.
-------------------------SELECTION SECTION--------------------------------
Selection Number: [1 ] Selection Type: [M]
Selection Text: [FORMS ]
Selection Action: [forms ]
```

In this instance, the Selection Action (forms) is the name of a menu (corresponding to Selection Type M). Information about the FORMS menu must be entered into the sysmenus table before you complete the entry for option 1.
The information stored in the **sysmenuitems** table in the **stores2** database appears in Figure 5-9:

<table>
<thead>
<tr>
<th>Menu Name</th>
<th>Option</th>
<th>Selection Text</th>
<th>Type</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>1</td>
<td>FORMS</td>
<td>M</td>
<td>forms</td>
</tr>
<tr>
<td>main</td>
<td>2</td>
<td>REPORTS</td>
<td>M</td>
<td>reports</td>
</tr>
<tr>
<td>main</td>
<td>3</td>
<td>QUERIES</td>
<td>M</td>
<td>queries</td>
</tr>
<tr>
<td>main</td>
<td>4</td>
<td>TABLE DEFINITIONS</td>
<td>M</td>
<td>tables</td>
</tr>
<tr>
<td>main</td>
<td>5</td>
<td>UTILITIES</td>
<td>M</td>
<td>utilities</td>
</tr>
<tr>
<td>forms</td>
<td>1</td>
<td>CUSTOMER ENTRY/QUERY FORM</td>
<td>F</td>
<td>customer</td>
</tr>
<tr>
<td>forms</td>
<td>2</td>
<td>ORDER ENTRY/QUERY FORM</td>
<td>F</td>
<td>orders</td>
</tr>
<tr>
<td>forms</td>
<td>3</td>
<td>DISPLAY CUSTOMER FORM SPECIFICATION</td>
<td>P</td>
<td>type customer.per</td>
</tr>
<tr>
<td>forms</td>
<td>4</td>
<td>DISPLAY ORDER FORM SPECIFICATION</td>
<td>P</td>
<td>type orders.per</td>
</tr>
<tr>
<td>reports</td>
<td>1</td>
<td>RUN CUSTOMER REPORT</td>
<td>R</td>
<td>clist1</td>
</tr>
<tr>
<td>reports</td>
<td>2</td>
<td>RUN REPORT ON CUSTOMER BY DESIGNATED STATE</td>
<td>R</td>
<td>clist2</td>
</tr>
<tr>
<td>reports</td>
<td>3</td>
<td>RUN CUSTOMER MAILING LABELS</td>
<td>S</td>
<td>mailinglabels</td>
</tr>
<tr>
<td>reports</td>
<td>4</td>
<td>RUN MATRIX REPORT ON MONTHLY SALES</td>
<td>R</td>
<td>months</td>
</tr>
<tr>
<td>reports</td>
<td>5</td>
<td>DISPLAY REPORT1 SPECIFICATION</td>
<td>P</td>
<td>type clist1.ace</td>
</tr>
<tr>
<td>reports</td>
<td>6</td>
<td>DISPLAY REPORT2 SPECIFICATION</td>
<td>P</td>
<td>type clist2.ace</td>
</tr>
<tr>
<td>reports</td>
<td>7</td>
<td>DISPLAY REPORT3 SPECIFICATION</td>
<td>P</td>
<td>type mail.ace</td>
</tr>
<tr>
<td>reports</td>
<td>8</td>
<td>DISPLAY REPORT4 SPECIFICATION</td>
<td>P</td>
<td>type months.ace</td>
</tr>
<tr>
<td>mailinglabels</td>
<td>1</td>
<td>run mailing labels report</td>
<td>R</td>
<td>mail</td>
</tr>
<tr>
<td>mailinglabels</td>
<td>2</td>
<td>display output file from mailing labels report</td>
<td>P</td>
<td>type mail.out</td>
</tr>
<tr>
<td>queries</td>
<td>1</td>
<td>DISPLAY CUSTOMER INFORMATION BASED ON PARTIAL NAME MATCH</td>
<td>S</td>
<td>query1</td>
</tr>
<tr>
<td>queries</td>
<td>2</td>
<td>INSERT, UPDATE, SELECT, AND DELETE NEW CUSTOMER ROW</td>
<td>S</td>
<td>query2</td>
</tr>
<tr>
<td>queries</td>
<td>3</td>
<td>DISPLAY ALL CURRENTLY UNPAID ORDERS</td>
<td>S</td>
<td>query3</td>
</tr>
<tr>
<td>queries</td>
<td>4</td>
<td>DISPLAY INFORMATION BASED ON A VIEW</td>
<td>S</td>
<td>query4</td>
</tr>
<tr>
<td>queries</td>
<td>5</td>
<td>DISPLAY CUSTOMERS PLUS OUTSTANDING ORDERS</td>
<td>S</td>
<td>query5</td>
</tr>
<tr>
<td>query1</td>
<td>1</td>
<td>display SQL syntax for query menu choice 1</td>
<td>P</td>
<td>type cust_nme.sql</td>
</tr>
<tr>
<td>query1</td>
<td>2</td>
<td>run query menu choice 1</td>
<td>Q</td>
<td>cust_nme</td>
</tr>
<tr>
<td>query2</td>
<td>1</td>
<td>display SQL syntax for query menu choice 2</td>
<td>P</td>
<td>type cust_row.sql</td>
</tr>
<tr>
<td>query2</td>
<td>2</td>
<td>run query menu choice 2</td>
<td>Q</td>
<td>cust_row</td>
</tr>
<tr>
<td>query3</td>
<td>1</td>
<td>display SQL syntax for query menu choice 3</td>
<td>P</td>
<td>type unpaid.sql</td>
</tr>
<tr>
<td>query3</td>
<td>2</td>
<td>run query menu choice 3</td>
<td>Q</td>
<td>unpaid</td>
</tr>
<tr>
<td>query4</td>
<td>1</td>
<td>display SQL syntax for query menu choice 4</td>
<td>P</td>
<td>type view_c.sql ; type view_s.sql ; type view_d.sql</td>
</tr>
<tr>
<td>query4</td>
<td>2</td>
<td>run query menu choice 4</td>
<td>Q</td>
<td>view_s</td>
</tr>
</tbody>
</table>
Creating a Menu

Steps for Entering Your Own Data

The following steps describe the procedure for entering your own menu and option data. See the list of display fields in “Menu Display Fields” on page 5-15 for information on the kind of data to enter for each display field.

1. Select the User-menu option on the I-SQL Main menu.
2. Select the Modify option on the USER-MENU Menu. The MENUFORM screen form displays.
3. Type a to select the Add option.
4. Enter a menu name and menu title for the first menu. Note that main must be the first menu name. Press ESCAPE when you finish.
5. Enter data in the same way for the second menu. Press ESCAPE when you finish.
6. When you have entered the menu name and menu title data for all menus in the user-menu, you are ready to enter data into the SELECTION SECTION of the form. You should now enter information about all options on
the Main menu. Select the Query option, enter main in the Menu Name field, and press ESCAPE.

7. Type d to make the detail table (sysmenuitems) active. I-SQL displays the following message:

There are no rows satisfying the conditions.

The sysmenuitems table contains no rows joining the main entry and the Menu Name field.

8. Select the Add option. Enter the Selection Number, Selection Type, Selection Text, and Selection Action data for the first option on the Main Menu. Press ESCAPE when you finish entering data about the first menu option.

If there is a second option to the Main Menu, select Add and enter data about that option. Press ESCAPE when you finish.

Repeat this step until you have entered data for each option on the Main Menu.

9. Type m to call the master table again. Use the Query option to locate and display the Menu Name and Menu Title data for your next menu. Type d to display the detail table joined to the current row of the master table.

10. Enter the Selection Number, Selection Type, Selection Text, and Selection Action data for the first option on this menu. Press ESCAPE when you finish entering data. Repeat this step for each option in this menu.

11. Repeat Steps 9 and 10 to enter data for the remaining menu options. When you have entered data for all the options in each menu, the menu is complete. Select the Exit option to leave PERFORM and return to the USER-MENU menu.

12. Select the Run option on the USER-MENU menu to run the new menu.

Modifying a Menu

You change a user-created menu in the same way you create one. Select the User-menu option from the I-SQL Main menu. Then select the Modify option on the USER-MENU menu. Use the PERFORM options to modify the menu entries in the MENUFORM screen form.

See Chapter 3, “The PERFORM Screen Transaction Processor,” for information about PERFORM.
Menu Display Fields
This section discusses the kinds of information you can enter for each field in the MENUFORM screen.
I-SQL uses the entry in the Menu Name field to find the menu you want when you make a selection that calls another menu. The menu name is used only by I-SQL and never displays on a screen.

**Usage**

- The menu name must follow the standard rules for identifiers. It can be from 1 to 18 characters long; the first character must be a letter; and you can use numbers, letters, and underscores (_) for the rest of the name.
- The top-level menu must be named `main` in all lowercase letters.

The Menu Name entry for the Main menu must be `main`, as shown in the following screen:
MENU TITLE

Use this field to enter the text I-SQL displays at the top of the menu.

Usage

The brackets on the screen show the maximum length of the text. It can contain any number of words that fit within the brackets.

The Menu Title entry for the Main menu follows:

---

PERFORM: Query Next Previous View Add Update Remove Table ...
Searches the active database table. **1: sysmenus table**

-----------------------------MENU ENTRY FORM---------------------------------

Menu Name: [main ]
Menu Title: [WEST COAST WHOLESALERS, INC. ]

-------------------------------SELECTION SECTION-------------------------------

Selection Number: Selection Type:
Selection
Text:
Selection
Action:
SELECTION NUMBER

Use the Selection Number field to enter the option number you want to appear to the left of each menu item on the screen.

Usage

• I-SQL displays the menu items in numbered order. The user selects items by number.

• The total number of options you can have in one menu depends on two factors: the number of lines your screen can hold and the length of the menu titles you enter. Most screens can accommodate 14 single-spaced menu lines, and each menu line can display 2 options of up to 33 characters. If the text for each option on a menu does not exceed 33 characters, you can display up to 14 double-spaced menu options (2 options per line) or 28 single-spaced menu options (2 options per line). If the text for an option is longer than 33 characters, it requires an entire line, reducing the total number of options available for that menu.

The Selection Number entry for an option on the REPORTS menu follows:

---

PERFORM: Query Next Previous View Add Update Remove Table . . .
Searches the active database table. ** 2: sysmenuitems table**

-----------------------------MENU ENTRY FORM----------------------------------
Menu Name: [reports ]
Menu Title: REPORTS
-----------------------------SELECTION SECTION-------------------------------
Selection Number: [3 ] Selection Type: [S]
Selection Text: [RUN CUSTOMER MAILING LABELS ]
Selection Action: [mailinglabels ]
---
SELECTION TYPE

Use the Selection Type field to specify the type of action an option performs. You can indicate that an option runs a form or report; calls a menu; executes an I-SQL command file, a program, or an operating system command; or invokes a script menu.

The following options are available for the Selection Type field:

<table>
<thead>
<tr>
<th>Option</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>Runs a form</td>
</tr>
<tr>
<td>R</td>
<td>Runs a report</td>
</tr>
<tr>
<td>M</td>
<td>Calls a menu</td>
</tr>
<tr>
<td>Q</td>
<td>Executes an I-SQL command file</td>
</tr>
<tr>
<td>P</td>
<td>Executes a program or an operating system command</td>
</tr>
<tr>
<td>S</td>
<td>Executes a script menu</td>
</tr>
</tbody>
</table>

Usage

- The entry in the Selection Type field must agree with the entry in the Selection Action field. For example, when the Selection Type is R, the Selection Action must be the name of a compiled report.
- You can enter the Selection Type option in either an uppercase or lowercase letter. I-SQL automatically displays it as an uppercase letter on the screen.
The Selection Type entry for running the clist2 report follows:

```plaintext
PERFORM: Query Next Previous View Add Update Remove Table . . .
Searches the active database table ** 2: sysmenuitems table**

------------------------------------MENU ENTRY FORM------------------------------------
Menu Name: [reports ]
Menu Title: REPORTS
------------------------------------SELECTION SECTION------------------------------------
Selection Number: [2 ] Selection Type: [R]
Selection Text: [RUN REPORT ON CUSTOMER BY DESIGNATED STATE ]
Selection Action: [clist2 ]
```

The Selection Type entry for running the customer entry form follows:

```plaintext
PERFORM: Query Next Previous View Add Update Remove Table . . .
Searches the active database table ** 2: sysmenuitems table**

------------------------------------MENU ENTRY FORM------------------------------------
Menu Name: [forms ]
Menu Title: FORMS
------------------------------------SELECTION SECTION------------------------------------
Selection Number: [1 ] Selection Type: [F]
Selection Text: [CUSTOMER ENTRY/QUERY FORM ]
Selection Action: [customer ]
```
SELECTION TEXT

Use the Selection Text field to enter the text you want to appear to the right of the option number on the screen.

Usage

- The brackets on the screen show the maximum length of the text allowed in this field.
- The length of the selection text affects the total number of options you can include in a single menu. See the “Usage” section on page 5-18 for more information.

The Selection Text entry for an option on the REPORTS menu follows:

```
PERFORM: Query Next Previous View Add Update Remove Table . . .
Searches the active database table. ** 2: sysmenutens table**

-------------------------------MENU ENTRY FORM-------------------------------
Menu Name: [reports ]
Menu Title: REPORTS
------------------------SELECTION SECTION-----------------------------
Selection Number: [3 ] Selection Type: [S]
Selection Text: [RUN CUSTOMER MAILING LABELS ]
Selection Action: [mailinglabels ]
```
SELECTION ACTION

Use the Selection Action field to specify the name of the action executed when the user selects the option indicated in the Selection Type field. You can enter a compiled form or report specification, an I-SQL command file, a menu, an operating system command, a program, or a script menu.

Usage

- The entry in the Selection Action field must agree with the entry in the Selection Type field, as shown in the following table:

<table>
<thead>
<tr>
<th>Selection Type</th>
<th>Selection Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>Enter the menu name (not the menu title) of the menu. You cannot enter a menu name that does not exist in the Menu Name field in the <code>sysmenus</code> table. Example: <code>reports</code></td>
</tr>
<tr>
<td>P</td>
<td>Enter a program name or an operating system command. Example: <code>date</code></td>
</tr>
<tr>
<td>F</td>
<td>Enter a form name. It is not necessary to add the <code>.frm</code> extension. Example: <code>customer</code></td>
</tr>
<tr>
<td>R</td>
<td>Enter a report name. It is not necessary to add the <code>.arc</code> extension. Example: <code>clist1</code></td>
</tr>
<tr>
<td>Q</td>
<td>Enter an SQL command filename. It is not necessary to add the <code>.sql</code> extension. Example: <code>cust_city</code></td>
</tr>
<tr>
<td>S</td>
<td>Enter a script menu name. Example: <code>mailinglabels</code></td>
</tr>
</tbody>
</table>

- You can enter Q in the Selection Type field and nothing in the Selection Action field. When the user selects that option, I-SQL calls the Query-language option on the I-SQL Main menu. The user can then enter one or more SQL statements.

- You can enter an R or F in the Selection Type field and enter nothing in the Selection Action field. When the user selects this option, I-SQL calls the Report or Form options, respectively, on the I-SQL Main menu.

- When you finish using the Query-language Report or Form option, choose the E option to exit. I-SQL then returns you to the USER-MENU menu.
The Selection Action entry used in the demonstration database for running the `clist1` report follows:

PERFORM: Query Next Previous View Add Update Remove Table . . .
Searches the active database table.
Creating a Script Menu

A script menu includes more than one action for a single menu item. By selecting the appropriate menu option, these actions are run in sequence without the necessity of displaying the menu on the screen. An S in the Selection Type field requires the entry of a script menu name in the Selection Action field.

The following list describes the procedure used to create the `mailinglabels` script included in the demonstration user-menu. This script runs and displays customer mailing labels. It is Selection Number 3 on the REPORTS menu. When the user selects option 3 on the REPORTS menu, the mailing labels report runs, and the output file displays on the screen.

1. Select the Modify option on the USER-MENU menu. (See the section “Creating a Menu” on page 5-8.)
2. Type a to select the Add option.
3. Enter `reports` in the Menu Name field. The script becomes an option on the REPORTS Menu.
4. Enter `REPORTS` in the Menu Title field.
5. Press ESCAPE when you finish.

The preceding steps show you how to enter the necessary information in the `sysmenus` table. The following display shows how the PERFORM screen appears at this point:

```
PERFORM: Query Next Previous View [Add] Update Remove Table . . .
 Added a row to the active database table. ** 1: sysmenus table**

==============================================================================
MENU ENTRY FORM=================================================================

Menu Name: [reports   ]
Menu Title: [REPORTS    ]

-----------------------------------------------------SELECTION SECTION---------------------

Selection Number:        Selection Type:
Selection
Text:
Selection
Action:

Row Added
```
The following steps show you how to enter data into the lower half of the screen (the sysmenuitems table fields):

1. Type d to make the detail table active.
2. Type a to Add.
3. Enter 3 in the Selection Number field. The script becomes the third choice on the REPORTS menu.
4. Enter s in the Selection Type field. This indicates you will run a script menu.
5. Enter the following text in the Selection Text field:
   
   RUN CUSTOMER MAILING LABELS
   
   This is the text that appears to the right of the option number on the screen.
6. Enter mailinglabels in the Selection Action field. This is the name of the script menu you want to run.
7. Press ESCAPE when you finish.

The following screen is the PERFORM screen as it appears at this point:

PERFORM: Query Next Previous View Add Update Remove Table . . .
Added a row to the active database table. ** 2: sysmenuitems table**

================================ MENU ENTRY FORM ===========================
Menu Name: [reports ]
Menu Title: REPORTS
-------------------------SELECTION SECTION--------------------------------
Selection Number: [3 ] Selection Type: [s]
Selection Text: [RUN CUSTOMER MAILING LABELS]
Selection Action: [mailinglabels ]

Row added
Creating a Script Menu

You must now enter the actions you want the script to perform and the order in which you want them to be performed.

1. Type m for Master to make the sysmenus table active.
2. Type a to select the Add option.
3. Enter mailinglabels in the Menu Name field. This is the name of the script menu.
4. Enter the following text in the Menu Title field:
   
   run report menu selection 3 and display the output file
   
   Unlike all other user-menu menus, the entry in the Menu Title field for a script menu does not display on the screen. You can use it as a Comment line to list the series of actions that comprise the script menu.
5. Press ESCAPE when you finish.

The preceding steps show you how to enter the necessary information in the sysmenus table. The following screen shows how the PERFORM screen appears at this point:

The following steps show you how to enter data for each option on the menu into the lower half of the screen (the sysmenuitems table fields):

1. Type d to make the detail table active.
2. Select the Add option.
3. Enter 1 in the Selection Number field. I-SQL executes this action first.
4. Enter R in the Selection Type field. This runs a report.
5. Enter the following text in the Selection Text field:
   
   run mailing labels report

   Unlike other user-menu menus, the entry in the Selection Text field for a
   script menu does not display on the screen. You can use it as a Comment
   Line to describe the action specified by the entry in the Selection Type
   field.

   6. Enter mail in the Selection Action field. This is the name of the compiled
   report specified by the action.

   7. Press ESCAPE when you finish.

   The following screen is the PERFORM screen as it appears at this point:

   PERFORM: Query Next Previous View Add Update Remove Table . . .
   Adds a row to the active database table. " 3: sysmenus table"

   ===========================MENU ENTRY FORM==========================
   Menu Name: [mailinglabels ]
   Menu Title: run report menu selection 3 and display the output file
   ---------------SELECTION SECTION-------------------
   Selection Number: [1 ] Selection Type: [R]
   Selection Text: [run mailing labels report ]
   Selection Action: [mail ]

   Row added

   You enter the second action for the mailinglabels script in a similar fashion
   to what you did when you set up the initial action.

   1. Type a for Add.
   2. Enter 2 in the Selection Number field. 1-SQL executes this action second.
   3. Enter P in the Selection Type field. This executes a program.
   4. Enter the following text in the Selection Text field:

   display output file from mailing labels report
Creating a Script Menu

5. Enter `type mail.out` in the Selection Action field. This is the name of the operating system program you want to run.

6. Press ESCAPE when you finish.

The following screen is the completed PERFORM screen as it appears at this point:

Using the Selection Type `S` when you want to run more than one action for a single menu item.

The two actions of running the report and displaying the output file are now entered as details of the `mailinglabels` script menu. When the user selects option 3 on the REPORTS menu, the `mailinglabels` script menu is selected, the mailing labels report is run, and the output file displays on the screen.
C Functions in ACE and PERFORM

Chapter Overview 3
Calling C Functions from ACE 4
  FUNCTION 5
  Calling C Functions 6
  CALL (in ACE) 7
  Compiling the Report Specification 8
Calling C Functions from PERFORM 9
  Calling C Functions in the INSTRUCTIONS Section 9
  CALL (in PERFORM) 10
ON BEGINNING and ON ENDING Control
  Blocks 12
ON BEGINNING and ON ENDING 13
Compiling the Form Specification 14
Writing the C Program 14
  Organizing the C Program 15
Passing Values to a C Function 18
    Testing for the Data Type 18
    Converting the Data Type 20
Returning Values to ACE and PERFORM 21
PERFORM Library Functions 22
  PF_GETTYPE 23
  PF_GETVAL 25
  PF_PUTVAL 27
  PF_NXFIELD 29
  PF_MSG 31
Chapter Overview

This chapter discusses calling C functions from ACE and PERFORM. While both ACE and PERFORM usually can handle all your database report and screen needs without modification, occasionally you might find it necessary to add a feature that is not present. For example, a C function called from ACE might make statistical computations on the data presented in a report and add these to the report. PERFORM might call C functions to check on the validity of data, to record the date, time, and name of the person updating the records, or to update the database.

The C functions can contain the following:

- Math functions or other C subroutines described in your system’s C development manuals.
- If used with PERFORM, the functions described beginning on page 6-22.
- If used with INFORMIX-ESQL/C, the library routines and the INFORMIX-ESQL/C statements described in the INFORMIX-ESQL/C Programmer’s Manual.

This chapter discusses the following topics:

- Calling C functions from ACE
- Calling C functions from PERFORM
- Writing the C program
- Using the PERFORM library functions
- Compiling, linking, and running customized versions of ACE and PERFORM

The chapter concludes with several examples.
Calling C Functions from ACE

The general format of an ACE report specification-file includes the following seven sections:

- DATABASE section (required)
- DEFINE section
- INPUT section
- OUTPUT section
- SELECT section (SELECT or READ required)
- READ section (SELECT or READ required)
- FORMAT section (required)

You call a C function from within a report specification file by declaring the function name in the DEFINE section and by using the function in the FORMAT section. Then use ACEPREP to compile the report specification file.
FUNCTION

You declare a C function in the DEFINE section of the report-specification file using the FUNCTION statement.

```
FUNCTION userfunc
```

FUNCTION is a required keyword.
userfunc is the name by which the C function is referenced in the specification file. userfunc must satisfy the conditions for an ACE identifier.

**Usage**

You can declare several functions at the same time by repeating the keyword FUNCTION followed by the next function name. Do not include parentheses after the function name.

You can have PARAM, VARIABLE, and ASCII statements within the DEFINE section in addition to the FUNCTION statement.
Calling C Functions

The FORMAT section of the report specification file contains one or more of the following control blocks that determine when ACE takes an action:

- PAGE HEADER control block
- PAGE TRAILER control block
- FIRST PAGE HEADER control block
- ON EVERY ROW control block
- ON LAST ROW control block
- BEFORE GROUP OF control block
- AFTER GROUP OF control block

Each control block contains one or more statements that tell ACE what action to take. For complete information on the statements that ACE allows, see Chapter 4 of this manual. In addition to the statements described in Chapter 4, you can use a C function call with the syntax shown on page 6-7.
CALL (in ACE)

CALL is an optional keyword that you must use when the C function does not return a value. If you omit the CALL keyword, `userfunc` must return a value.

`userfunc` is the name of a C function that you have previously declared in the DEFINE section.

`expression` is 1 to 10 expressions, separated by commas.

Usage

An expression can include the following items:

- Numeric or character constants
- Column names
- ACE variables
- ACE parameters
- ACE functions (such as group aggregates and date functions)
- Quoted strings
- Arithmetic and logical operators
- Keywords

ACE statements are composed of keywords and expressions. You can use a C function in an expression wherever you can use a constant. When you use a function in this way you need not use the CALL keyword, but you must make sure the function returns a value.

If you are connecting to an INFORMIX-OnLine Dynamic Server database server, you can pass columns of type VARCHAR, TEXT, or BYTE. You cannot, however, return TEXT or BYTE values from a C function.
CALL (in ACE)

The following control block calls a C function `stat` that calculates statistics on the data in the rows that correspond to the `order_num` order:

```plaintext
after group of order_num
  call stat(order_num)
```

The following control block prints the order number and a value intended to correlate the total price of each order with the period of time the order has been outstanding. It calls a C function that computes the logarithm:

```plaintext
on every row
  print order_num,
  logarithm((total of total_price)/(today - order_date))
```

The following control block is taken from “ACE Example 1” on page 6-33. It prints the system date and time at the top of the first page of the report. The function `to_unix` sends its string argument to UNIX:

```plaintext
first page header
  call to_unix("date")
```

### Compiling the Report Specification

Use `ACEPREP` to compile the report specification that includes calls to C function calls, just as you compile a report with no calls. When you name the file that contains the report specification file, assign it the `.ace` extension. For example, you could name the file `specfile.ace`. To invoke `ACEPREP` for this file, enter the following statement on the command line:

```
saceprep specfile
```

The `.ace` extension is optional when you identify the report specification file for the `saceprep` command.

For more information on compiling report specification files, see Chapter 4, “The ACE Report Writer.”
Calling C Functions from PERFORM

The general format of a PERFORM form specification file includes the following five sections:

- DATABASE section (required)
- SCREEN section (required)
- TABLES section (required)
- ATTRIBUTES section (required)
- INSTRUCTIONS section (optional)

You can use C functions in the control blocks in the INSTRUCTIONS section of a form specification file. You can also use the ON BEGINNING and ON ENDING control blocks with a function call within the INSTRUCTIONS section.

Calling C Functions in the INSTRUCTIONS Section

In control blocks, you can use C functions anywhere you can use an expression, or the function can stand alone. Use the CALL statement for expressions or for simple function calls. The syntax of the CALL statement is described on page 6-10.
CALL (in PERFORM)

Use the CALL statement in PERFORM to execute a C function or to retrieve values from a C function.

\[
\text{CALL} \quad \text{userfunc} \quad ( \quad \text{expression} \quad )
\]

**CALL** is an optional keyword that you must use when the C function does not return a value. If you omit the keyword CALL, *userfunc* must return a value.

**userfunc** is the name by which the C function is referenced in the PERFORM specification.

**expression** is a list of 1 to 10 expressions. An expression is defined as follows:
- A field tag
- A constant value
- An aggregate value
- A C function
- The keyword TODAY
- The keyword CURRENT
- Any combination of the preceding items, combined by using the arithmetic operators +, -, *, and /.

If you are connecting to an INFORMIX-OnLine Dynamic Server database server, you can pass columns of type VARCHAR, TEXT, or BYTE. You cannot, however, return TEXT or BYTE values from a C function.
The following examples demonstrate several methods of calling C functions:

```plaintext
after editadd of proj_num
    let f001 = userfunc(f002)

before editupdate of paid_date
    if boolfunc(f003) then
        let f004 = 15
    else
        let f004 = 10

after add update remove of customer
    call userfunc()
```
ON BEGINNING and ON ENDING Control Blocks

You can use the ON BEGINNING and ON ENDING control blocks only with calls to C functions. Specify these control blocks in the INSTRUCTIONS section of the form specification file. ON BEGINNING executes immediately after invoking PERFORM, and ON ENDING executes immediately after the EXIT command.

For example, use ON BEGINNING to do one of the following:
- Give instructions
- Request a special password
- Initialize a temporary work file in which to keep a batch of transaction records

For example, use ON ENDING to do one of the following:
- Perform calculations to summarize the changes made in the database during the PERFORM session that just concluded
- Print summaries of the records added
- Erase work files

You can include multiple ON BEGINNING and ON ENDING control blocks in the INSTRUCTIONS section. However, you can include only one CALL statement in each control block.
ON BEGINNING and ON ENDING

Use ON BEGINNING to designate that a C function call occurs immediately after you invoke PERFORM. Use ON ENDING to designate that a C function call occurs immediately after the EXIT command.

ON BEGINNING

Block

| ON BEGINNING — CALL — userfunc — (expression) |

ON ENDING

Block

| ON ENDING — CALL — userfunc — (expression) |

ON

BEGINNING

are required keywords that begin the ON BEGINNING control block.

ON ENDING

are required keywords that begin the ON ENDING control block.

CALL

is a required keyword.

userfunc

is the name by which the C function is referenced in the PERFORM specification.
expression is a list of 1 to 10 expressions. An expression is defined as one of the following items:

- A field tag
- A constant value
- An aggregate value
- A C function
- The keyword TODAY
- The keyword CURRENT
- Any combination of the preceding items, combined by using the arithmetic operators +, -, *, and /

Compiling the Form Specification

Use FORMBUILD to compile the form specification file that includes calls to C functions just as you compile a report with no calls to C. When you name the form specification file, assign it the .per extension. For example, you could name the file specfile.per. To invoke FORMBUILD with this file, enter the following statement on the command line:

```
  sformbld specfile
```

The .per extension is optional when you identify the form specification file for the sformbld command.

For more information on compiling form specification files, see Chapter 2, “The FORMBUILD Transaction Form Generator.”

Writing the C Program

The C program must include the appropriate header files and structure declarations, as well as the C functions. This section describes the following topics:

- Organizing the C program
- Passing values to the C program
- Returning values to ACE and PERFORM
Organizing the C Program

To create a custom version of ACE or PERFORM that includes your functions, you must write a C program that contains the appropriate declarations. Your program can have one or more functions, and you can define other functions to use internally in your program. The following example illustrates the general structure of a C program that includes two user-defined functions:

```c
#include "ctools.h"
/* add other includes as desired */

valueptr funct1();
valueptr funct2();

struct ufunc userfuncs[] =
{
  "myfunct1", funct1,
  "myfunct2", funct2,
  0,0
};

/* add other global declarations */

valueptr funct1()
{
  /* funct1 takes no arguments
     and returns a character string */
  .
  .
  strreturn(s, len);
}

valueptr funct2(arg1, arg2)
valueptr arg1, arg2;
{
  /* funct2 takes two arguments
     and returns no value */
  .
  .
  .
}
```
The following steps describe how to organize your C program:

1. Place the `ctools.h` header file at the top of the program:

   ```c
   #include "ctools.h"
   ```

   You may want to include other header files, such as `math.h` or `stdio.h`, depending on your application. If you use INFORMIX-ESQL/C, you can include `sqlca.h` and other header files.

   The `ctools.h` header file automatically includes the following additional header files:
   - `value.h`
   - `datetime.h`
   - `sqltypes.h`

2. Before you initialize the required array of `ufunc` structures, you must declare your functions. Included in `ctools.h` is the definition of the `value` structure and pointers to that structure, as shown in the following example:

   ```c
   typedef struct value *valueptr;
   typedef struct value *acevalue;
   typedef struct value *perfvalue;
   ```

   The last two pointers are included for compatibility with earlier releases of ACE and PERFORM. All of your functions must be of type `valueptr`. If `funct1()` and `funct2(arg1, arg2)` are your functions, declare them next:

   ```c
   valueptr funct1();
   valueptr funct2();
   ```
3. Make the structure declaration and initialization for `userfuncs[]` the next section of your program. This structure is required so that ACE and PERFORM can call your functions at run time:

```c
struct ufunc userfuncs[] =
{
    "myfunct1", funct1,
    "myfunct2", funct2,
    0, 0
};
```

The quoted strings, "myfunct1" and "myfunct2", must be the names of the functions as they appear in the specification file. `funct1` and `funct2` (which correspond to "myfunct1" and "myfunct2"), are pointers to the functions as defined within the C program. Note that the C functions do not need to have the same names that you used in your specification file. The purpose of the `userfuncs` array is to make the connection between these two names. The two zeros at the end of the array are required as terminators.

4. The last section of the C program is the code for your functions. As stated earlier, all the functions that you call in ACE or PERFORM must be declared as returning pointers to a `value` structure. Also, all arguments of your functions must be declared type `valueptr`.

Several macros are included that you can use to return values of type `valueptr`. These and other conversion routines are described in “Passing Values to a C Function” on page 6-18.
Passing Values to a C Function

Including the `ctools.h` header file allows you to pass values to the C functions from ACE and PERFORM. When passing values to a C function, the C function must determine the data type of the data passed. The C function has two options for determining the data type:

- Testing for the data type
- Converting the data type

Testing for the Data Type

By using the following definitions, you can test for the type of data passed to the C function. For example, if the parameter passed to the C function is `arg`, you can use the following definitions to detect the data type of `arg` and to extract the value of `arg`:

<table>
<thead>
<tr>
<th>Definition</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>arg-&gt;v_charp</code></td>
<td>pointer to string</td>
</tr>
<tr>
<td><code>arg-&gt;v_len</code></td>
<td>length of string</td>
</tr>
<tr>
<td><code>arg-&gt;v_int</code></td>
<td>integer value</td>
</tr>
<tr>
<td><code>arg-&gt;v_long</code></td>
<td>long value</td>
</tr>
<tr>
<td><code>arg-&gt;v_float</code></td>
<td>float value</td>
</tr>
<tr>
<td><code>arg-&gt;v_double</code></td>
<td>double value</td>
</tr>
<tr>
<td><code>arg-&gt;v_decimal</code></td>
<td>decimal, money, datetime, or interval value</td>
</tr>
<tr>
<td><code>arg-&gt;v_type</code></td>
<td>data type</td>
</tr>
<tr>
<td><code>arg-&gt;v_ind</code></td>
<td>null indicator</td>
</tr>
<tr>
<td><code>arg-&gt;v_prec</code></td>
<td>datetime/interval qualifier</td>
</tr>
</tbody>
</table>
You can determine the data type of `arg` by checking `arg->v_type` against a series of integer constants defined in `sqltypes.h`:

<table>
<thead>
<tr>
<th>v_type</th>
<th>SQL Type</th>
<th>C Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLCHAR</td>
<td>CHAR</td>
<td>string</td>
</tr>
<tr>
<td>SQLSMINT</td>
<td>SMALLINT</td>
<td>short</td>
</tr>
<tr>
<td>SQLINT</td>
<td>INTEGER</td>
<td>long</td>
</tr>
<tr>
<td>SQLFLOAT</td>
<td>FLOAT</td>
<td>double</td>
</tr>
<tr>
<td>SQLSMFLOAT</td>
<td>SMALLFLOAT</td>
<td>float</td>
</tr>
<tr>
<td>SQLDECIMAL</td>
<td>DECIMAL</td>
<td>long</td>
</tr>
<tr>
<td>SQLSERIAL</td>
<td>SERIAL</td>
<td>long</td>
</tr>
<tr>
<td>SQLDATE</td>
<td>DATE</td>
<td>long</td>
</tr>
<tr>
<td>SQLMONEY</td>
<td>MONEY</td>
<td>dec_t</td>
</tr>
<tr>
<td>SQLDTIME</td>
<td>DATETIME</td>
<td>dtime_t</td>
</tr>
<tr>
<td>SQLINTERVAL</td>
<td>INTERVAL</td>
<td>intrvl_t</td>
</tr>
</tbody>
</table>

If `arg->v_type` is SQLCHAR, then the pointer to the string is available in `arg->v_charp`, and the number of characters in the string (length) is available in `arg->v_len`. The string is not null-terminated.

`arg->v_ind` is set to a negative value if the value of `arg` is NULL; otherwise `arg->v_ind` is set to zero.

If you are connecting to an INFORMIX-OnLine Dynamic Server database server, you can use the VARCHAR data type as well. You can check `arg->v_type` against the following in `sqltypes.h`:

<table>
<thead>
<tr>
<th>v_type</th>
<th>SQL Type</th>
<th>C Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLVCHAR</td>
<td>VARCHAR</td>
<td>char</td>
</tr>
</tbody>
</table>

You cannot pass TEXT or BYTE values.
Converting the Data Type

The `ctools.h` header file provides an alternative to testing the type of the parameter passed from ACE or PERFORM. Several functions, listed in the following table, can force conversion of a parameter passed as a pointer to a value structure, to a C data type of your choice:

<table>
<thead>
<tr>
<th>Function</th>
<th>Returned Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>toint</td>
<td>int</td>
</tr>
<tr>
<td>tolong</td>
<td>long</td>
</tr>
<tr>
<td>tofloat</td>
<td>double</td>
</tr>
<tr>
<td>todouble</td>
<td>double</td>
</tr>
<tr>
<td>todate</td>
<td>long</td>
</tr>
<tr>
<td>todecimal</td>
<td>dec_t</td>
</tr>
<tr>
<td>todatetime</td>
<td>dtime_t</td>
</tr>
<tr>
<td>tointerval</td>
<td>intrvl_t</td>
</tr>
</tbody>
</table>

All of these functions require a pointer to a type value structure and return a value of the type indicated. The `todecimal`, `todatetime`, and `tointerval` functions each require a second argument, as shown in the following table:

<table>
<thead>
<tr>
<th>Function</th>
<th>Second Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>todecimal</td>
<td>pointer to the dec_t structure</td>
</tr>
<tr>
<td>todatetime</td>
<td>pointer to the dtime_t structure</td>
</tr>
<tr>
<td>tointerval</td>
<td>pointer to the intrvl_t structure</td>
</tr>
</tbody>
</table>

If the type conversion is not successful, the global integer `toerrno` is set to a negative value; if the conversion is successful, `toerrno` is set to zero.
Returning Values to ACE and PERFORM

If a function returns a value to ACE or PERFORM, you must insert the value in a type value structure and return a pointer to that structure. The ctools.h header file contains the following macros to perform that procedure for you:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Type Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>intreturn(i)</td>
<td>returns integer i</td>
</tr>
<tr>
<td>lngreturn(l)</td>
<td>returns long l</td>
</tr>
<tr>
<td>floreturn(f)</td>
<td>returns float f</td>
</tr>
<tr>
<td>dubreturn(d)</td>
<td>returns double d</td>
</tr>
<tr>
<td>strreturn(s,c)</td>
<td>returns string s of length c (short)</td>
</tr>
<tr>
<td>decreturn(d)</td>
<td>returns decimal d (of type dec_t)</td>
</tr>
<tr>
<td>dtimereturn(d)</td>
<td>returns datetime d (of type dtime_t)</td>
</tr>
<tr>
<td>invreturn(i)</td>
<td>returns interval i (of type intrvl_t)</td>
</tr>
</tbody>
</table>

Use the appropriate macro even when you want to return an error condition. Do not use a simple return.

Since strreturn(s,c) returns a pointer to the string s, be sure to define s as a static or external variable.

If you are connecting to an INFORMIX-OnLine Dynamic Server database server, you can use the following macro to return VARCHAR values:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Type Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>vcharreturn(s,c)</td>
<td>returns string s of length c (short)</td>
</tr>
</tbody>
</table>

You cannot return TEXT or BYTE values.
PERFORM Library Functions

The following five C functions are designed to control PERFORM screens from within C functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>pf_gettype</td>
<td>determines the type and length of a display field</td>
</tr>
<tr>
<td>pf_getval</td>
<td>reads a value from a display field</td>
</tr>
<tr>
<td>pf_putval</td>
<td>puts a value onto a display field</td>
</tr>
<tr>
<td>pf_nxfield</td>
<td>moves the cursor to a specified field</td>
</tr>
<tr>
<td>pf_msg</td>
<td>writes a message at the bottom of the screen</td>
</tr>
</tbody>
</table>

These functions are described in detail on the following pages. If these functions execute successfully, they return 0; if they are unsuccessful, they return a non-zero error code.
The pf_gettype function returns the SQL data type and the length of the display field for a specified field tag.

```c
pf_gettype(tagname, type, len)
    char *tagname;
    short *type, *len;
```

tagname is a string containing the field tag that specifies a display field.
type is a pointer to a short integer that describes the data type of the display field tagname.
len is a pointer to a short integer that is the length of the display field tagname on the PERFORM screen.

### Usage

The options for type follow:

<table>
<thead>
<tr>
<th>type</th>
<th>SQL Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLCHAR</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>SQLSMINT</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>SQLINT</td>
<td>INTEGER</td>
</tr>
<tr>
<td>SQLFLOAT</td>
<td>FLOAT</td>
</tr>
<tr>
<td>SQLSMFLOAT</td>
<td>SMALLFLOAT</td>
</tr>
<tr>
<td>SQLDECIMAL</td>
<td>DECIMAL</td>
</tr>
<tr>
<td>SQLSERIAL</td>
<td>SERIAL</td>
</tr>
<tr>
<td>SQLDATE</td>
<td>DATE</td>
</tr>
<tr>
<td>SQLMONEY</td>
<td>MONEY</td>
</tr>
<tr>
<td>SQLDTIME</td>
<td>DATETIME</td>
</tr>
<tr>
<td>SQLINTERVAL</td>
<td>INTERVAL</td>
</tr>
</tbody>
</table>

If you are connecting to an INFORMIX-OnLine Dynamic Server database server, you can also specify the following data types:

```c
<table>
<thead>
<tr>
<th>type</th>
<th>SQL Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLVARCHAR</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>SQLTEXT</td>
<td>TEXT</td>
</tr>
<tr>
<td>SQLBYTE</td>
<td>BYTE</td>
</tr>
</tbody>
</table>
```

All of these types are defined in the sqltypes.h header file.
Return Codes

0  The operation was successful; display field was found.
3759  There is no such field tag in the form.
If the display field is a character field, `pf_getval` obtains the value found in a display field and the length of the value.

**pf_getval**(tagname, retvalue, valtype, vallen)
char *tagname, *retvalue;
short valtype, vallen;

tagname is a string containing the field tag that specifies a display field.
retvalue is a pointer to the string, short, long, float, double, decimal, datetime, or interval structure returned by `pf_getval`.
valtype is a short integer indicating the type of value to which retvalue should point.
vallen is a short integer specifying the length of the string (plus 1 for the terminating null byte) returned in retvalue, when valtype is CCHARTYPE. For any other value for valtype, vallen is ignored.

**Usage**

The parameter retvalue must be a pointer to the variable that contains the value. A common programming error is to use the variable itself. This results in a run-time system error and is not detected by the compiler.

The options for valtype are as follows:

<table>
<thead>
<tr>
<th>valtype</th>
<th>SQL Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCHARTYPE</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>CFIXCHARTYPE</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>CSTRINGTYPE</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>CINTTYPE</td>
<td>INTEGER</td>
</tr>
<tr>
<td>CSHORTTYPE</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>CLONGLONG</td>
<td>INTEGER, DATE, SERIAL</td>
</tr>
<tr>
<td>CFLOATTYPE</td>
<td>SMALLFLOAT</td>
</tr>
<tr>
<td>CDOUBLETYPE</td>
<td>FLOAT</td>
</tr>
<tr>
<td>CDECIMALTYPE</td>
<td>DECIMAL, MONEY</td>
</tr>
<tr>
<td>CDATETIME</td>
<td>DATETIME</td>
</tr>
<tr>
<td>CINTERVAL</td>
<td>INTERVAL</td>
</tr>
</tbody>
</table>
The value given to the parameter \texttt{valtype} determines the type of \texttt{retvalue}. The parameter \texttt{valtype} need not correspond exactly to the data type of the display field, but both should be either a number or a character so that \texttt{PERFORM} can do the proper type conversion.

If \texttt{valtype} is a number field and the display field is a character field, \texttt{1-SQL} tries to convert the data type of \texttt{valtype}. If the conversion is unsuccessful, \texttt{retvalue} points to a zero. If \texttt{valtype} is character and the display field is a number field, a conversion to a string occurs. If the string does not fit in the length specified by \texttt{vallen}, \texttt{retvalue} contains the string, truncated to fit and null-terminated.

If you are connecting to an \texttt{INFORMIX-OnLine Dynamic Server} database server, you can also specify the following data types for \texttt{valtype}:

<table>
<thead>
<tr>
<th>\texttt{valtype}</th>
<th>SQL Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVCHARTYPE</td>
<td>SQLVCHAR</td>
</tr>
<tr>
<td>CLOCATORTYPE</td>
<td>SQLTEXT</td>
</tr>
<tr>
<td></td>
<td>SQLBYTES</td>
</tr>
</tbody>
</table>

For VARCHAR values, \texttt{vallen} must contain the number of bytes the value buffer can hold. For TEXT and BYTE values, if you point \texttt{retvalue} to a \texttt{loc_t} structure, \texttt{PERFORM} copies the internal locator of \texttt{loc_t} to your structure.

\textbf{Return Codes}

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The operation was successful; display field was found.</td>
</tr>
<tr>
<td>3700</td>
<td>The user is not permitted to read the field.</td>
</tr>
<tr>
<td>3759</td>
<td>There is no such field tag in the form.</td>
</tr>
</tbody>
</table>
The pf_putval function puts a value into a PERFORM screen in a specified display field. The user must have permission to update or to enter data into the desired destination field.

```c
pf_putval(pvalue, valtype, tagname)
char *pvalue;
short valtype;
char *tagname;
```

- **pvalue** is a pointer to a string, short, integer, long, float, double, decimal, datetime, or interval structure inserted into the display field designated by `tagname`.
- **valtype** is a short integer indicating the type of the value to which `pvalue` points.
- **tagname** is a string containing the field tag that specifies the display field where the information pointed to by `pvalue` is placed.

**Usage**

The `pvalue` parameter must be a pointer to the variable containing the value. A common programming error is to use the variable itself. This results in a run-time system error and is not detected by the compiler.

The options for **valtype** are as follows:

<table>
<thead>
<tr>
<th>valtype</th>
<th>SQL Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCHARTYPE</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>CFIXCHARTYPE</td>
<td></td>
</tr>
<tr>
<td>CSTRINGTYPE</td>
<td></td>
</tr>
<tr>
<td>CINTTYPE</td>
<td>INTEGER</td>
</tr>
<tr>
<td>CSHORTTYPE</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>CLONGLONGTYPE</td>
<td>INTEGER, DATE</td>
</tr>
<tr>
<td>CFLOATTYPE</td>
<td>SMALLFLOAT</td>
</tr>
<tr>
<td>CDOUBLETYPE</td>
<td>FLOAT</td>
</tr>
<tr>
<td>CDECIMALTYPE</td>
<td>DECIMAL, MONEY</td>
</tr>
<tr>
<td>CDATETIME</td>
<td>DATETIME</td>
</tr>
<tr>
<td>CINTERVAL</td>
<td>INTERVAL</td>
</tr>
</tbody>
</table>

If `valtype` is one of the character types and the display field is a number field, PERFORM tries to convert `valtype`. If the conversion is unsuccessful, PERFORM enters 0 in the display field.
If the type specified is a number field and the display field is character, a conversion to a string occurs. If the string does not fit in the display field, \texttt{PERFORM} truncates the display field.

If a number value does not fit in a number display field, \texttt{PERFORM} fills the field with asterisks.

\begin{tabular}{|l|l|}
\hline
\textbf{valtype} & \textbf{SQL Type} \\
\hline
CVCHARTYPE & SQLVCHAR \\
CLOCATORTYPE & SQLTEXT \\
SQLBYTES & \\
\hline
\end{tabular}

If you are connecting to an \texttt{INFORMIX-OnLine Dynamic Server} database server, you can also specify the following data types for \texttt{valtype}:

\begin{itemize}
\item CVCHARTYPE
\item CLOCATORTYPE
\item SQLBYTES
\end{itemize}

Use this function with VARCHAR values just as you use it with CHARACTER values.

If you use this function with TEXT or BYTE data types, \texttt{pvalue} must point to a \texttt{loc_t} structure. \texttt{PERFORM} requires that the \texttt{loc_t} structure contain exactly the same information as the \texttt{loc_t} structure corresponding to \texttt{tag-name}. For this reason, refrain from changing anything in your copy of the locator. You can then use the locator to change the actual value of the TEXT or BYTE data type, which \texttt{PERFORM} stores in a temporary file.

\section*{Return Codes}

\begin{itemize}
\item 0 \hspace{1cm} The operation was successful; display field was found.
\item 3710 \hspace{1cm} The user is not permitted to update the field.
\item 3720 \hspace{1cm} The user is not permitted to add to the field.
\item 3756 \hspace{1cm} The display field is not in the current table.
\item 3759 \hspace{1cm} There is no such field tag in the form.
\end{itemize}
The pf_nxfield function controls the cursor placement on a PERFORM screen when you add a new record or update an old record.

```
pf_nxfield(tagname)
char *tagname;
```

`tagname` is a string that contains the field tag for the display field on a PERFORM screen to which the cursor is sent.

**Usage**

The following list describes what happens at the different times when you call `pf_nxfield`:

- If called during a BEFORE EDITADD or a BEFORE EDITUPDATE of a table, it controls which display field is edited first.
- If called during an AFTER EDITADD or an AFTER EDITUPDATE of a table, it causes the cursor to move to the designated display field `tagname` for further editing, rather than allowing PERFORM to write the record.
- If called either BEFORE or AFTER an EDITADD or EDITUPDATE of a column, it determines the next field to be edited.
- If called either AFTER ADD or AFTER UPDATE, it is inoperative, since the record has already been written.

If `tagname` is set equal to the value EXITNOW, `pf_nxfield` causes an immediate exit from the add or update operation with the row being added or updated. This option performs the same as when you press ESCAPE to complete the transaction.
### Return Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The operation was successful; display field was found.</td>
</tr>
<tr>
<td>3710</td>
<td>The user is not permitted to update the field.</td>
</tr>
<tr>
<td>3720</td>
<td>The user is not permitted to add to the field.</td>
</tr>
<tr>
<td>3755</td>
<td>The display field is display-only.</td>
</tr>
<tr>
<td>3756</td>
<td>The display field is not in the current table.</td>
</tr>
<tr>
<td>3759</td>
<td>There is no such field tag in the form.</td>
</tr>
</tbody>
</table>
The **pf_msg** function displays a message at the bottom of the screen.

```c
pf_msg(msgstr, reverseflag, bellflag)
    char *msgstr;
    short reverseflag, bellflag;
```

- **msgstr** is a string containing the message displayed at the bottom of the screen.
- **reverseflag** is a short integer indicating whether the message is displayed in reverse video. 0 indicates normal video; 1 indicates reverse video.
- **bellflag** is a short integer indicating whether the terminal bell is rung when the message is displayed. 0 indicates not to ring the bell; 1 indicates to ring the bell.

**Usage**

If several calls to **pf_msg** are invoked at the same time in response to satisfying several conditions simultaneously, only the last message displayed is visible to the user.

In normal video display, **msgstr** can have up to 80 characters. In reverse video display, the maximum number of characters is less than 80 because the reverse video control characters require one or more spaces on some monitors.
Compiling, Linking, and Running Reports and Forms

After you have written the file containing your C functions, you must compile the files and link the necessary library functions to create a custom version of `sacego` or `sperform`.

I-SQL provides programs to simplify the compiling and linking process. You do not need to be concerned with names of special ACE or PERFORM libraries, nor with the location of the include files associated with these programs; the `cace` and `cperf` programs include these files automatically.

Syntax of the `cace` and `cperf` programs

```
cace cperf -m cprogram.c -o custprog -other-C-list
```

- **cace** is the program that creates a custom version of `sacego`.
- **cperf** is the program that creates a custom version of `sperform`.
- **cprogram** is the name of the C program that contains your functions, as described in the previous sections.
- **.c** is the extension to use if `cprogram` only contains C statements.
- **.ec** is the extension to use if you have `INFORMIX-ESQL/C` and `cprogram` includes any `INFORMIX-ESQL/C` statements.
- **-m** compiles real-mode application - medium model only
- **-pm** create protected-mode application - large model only
- **-o** specifies the output filename.
- **custprog** is the name of your custom version of `sacego` or `sperform`.
- **other-C-list** is the rest of the arguments that you want to pass to the standard `cc` program.
Use of *cace* and *cperf*

You can compile several C programs at the same time.

After you compile your custom version of *sacego* or *sperform*, you can run reports or forms with the following command line:

```
custprog specfile
```

where *custprog* is the output file of the *cace* or *cperf* command, and *specfile* is the name of the report or form specification file you compiled using ACEPREP or FORMBUILD. When using *sacego*, specify the .arc suffix for *specfile*; when using *sperform*, specify the .frm suffix for *specfile*.

Examples

This section contains examples of both ACE applications and PERFORM applications. ACE C functions can be used with PERFORM as well. These sample programs are delivered with the demonstration database.

ACE Example 1

The following specification file calls a user function to execute a system command. The program is named *a_ex1.ace* in the demonstration database:

```
database
  stores
end

define
  function to_unix
end

select * from customer
end

format
  first page header
  call to_unix("date")
  skip 1 line
  on every row
  print customer_num, 3 spaces,
    fname clipped, 1 space, lname
end
```
The function `to_unix.c` follows:

```c
#include "ctools.h"

valueptr to_unix();

struct ufunc userfuncs[] =
{
"to_unix", to_unix,
0,0
};

valueptr to_unix(string)
valueptr string;
{
char savearea[80];

/*copy bytes from string to savearea*/
bycopy(string->v_charp, savearea, string->v_len);

/*put null on end*/
savearea[string->v_len]=0;

system(savearea);
}
```

To execute this example, perform the following steps:

1. Compile the report by executing the following command:
   ```
   saceprep a_ex1.ace
   ```
2. Create a custom version of `sacego` by executing the following command:
   ```
   cace to_unix.c -o output_file
   ```
   where `output_file` is the name of the file to contain the customer version of `sacego`.
3. Run the program by executing the following command:
   ```
   output_file a_ex1.ace
   ```
   where `output_file` is the name of the file containing the customer version of `sacego`.
ACE Example 2

The following ACE program computes the average and the standard deviation of the total cost of all the orders in the stores2 database. This program is named a_ex2.ace in the demonstration database:

database
stores
end

define
function decsqroot
end

select o.order_num, sum(total_price) t_cost
  from orders o, items i
  where o.order_num = i.order_num
  group by o.order_num
end

format
on every row
  print order_num, t_cost
on last row
  skip 1 line
  print "The average total order is : ",
  (total of t_cost)/count
  using "$#####.##"
print "Standard deviation is : ",
  decsqroot((total of t_cost*t_cost)/count
             - ((total of t_cost)/count)**2)
  using "$#####.##"
end
The function **decsqrt.c** follows:

```c
#include "ctools.h"
#include <math.h>
valueptr squareroot();

struct ufunc userfuncs[] =
{
    "decsqroot", squareroot,
    0, 0
};

valueptr squareroot(pnum)
valueptr pnum;
{
    double dub;
    dec_t dec;

    /* convert decimal to double */
    dectodbl(&pnum->v_decimal, &dub);
    dub = sqrt(dub);

    /* convert double to decimal */
    deccvdbl(dub, &dec);

    /* return decimal */
    decretun(DECIMAL, dec);
}
```

To execute this example, perform the following steps:

1. Compile the report by executing the following command:
   ```bash
   saceprep a_ex2.ace
   ```

2. Create a custom version of **sacego** by executing the following command:
   ```bash
   cace decsqrt.c -o output_file -lm
   ```
   where *output_file* is the name of the file to contain the customer version of **sacego**. You must specify `-lm` to include the math libraries.

3. Run the program by executing the following command:
   ```bash
   output_file a_ex2.ace
   ```
   where *output_file* is the name of the file containing the customer version of **sacego**.
PERFORM Example

This example demonstrates accessing and displaying the following data from UNIX:

- The current user’s login
- The time the user entered some data

The sample program then displays the data on a form.

The following form specification file uses the customer table to let you enter new customers into the stores2 database. The form also includes two DISPLAYONLY fields that display the name of the entry clerk and the entry time. (To include the name of the entry clerk and the entry time in the database, you would need to add entry clerk and entry time columns to the customer table, rather than use the DISPLAYONLY fields.)

The cursor moves from the upper left down through the Customer Data by following the order of the fields listed in the ATTRIBUTES section. After the Telephone field, the cursor moves to the Owner Name field. When the entry clerk presses ESCAPE to complete the transaction, PERFORM calls the C function stamptime.

The form is in p_ex1.per in the demonstration database; stamp.c contains the function stamptime.
database stores
screen
{|*
*************************************************************/
* Customer Form *
*************************************************************/
* Number :f000 *
* Owner Name :f001 |f002 | *
* Company :f003 *
* Address :f004 *
* f005 *
* City :f006 | State:a0 | Zipcode:f007 |
* Telephone :f008 |
*************************************************************/
* Entry Clerk :f009 | Time Entered :f010 |*
*************************************************************/
}
tables
customer
attributes
f000 = customer.customer_num, noentry;
f001 = customer.fname;
f002 = customer.lname;
f003 = customer.company;
f004 = customer.address1;
f005 = customer.address2;
f006 = customer.city;
a0 = customer.state, default="CA", upshift, autonext;
f007 = customer.zipcode, autonext;
f008 = customer.phone;
f009 = displayonly type char;
f010 = displayonly type char;

instructions
after editadd editupdate of phone
    nextfield = f001
after editadd editupdate of customer
    call stamptime()
end

The function stamptime, called by the form specification file when the entry clerk presses ESCAPE to complete the transaction, follows. In addition to the special function pf_putval defined earlier in this section, stamptime uses the system functions time, localtime, and getlogin. The login name of the order taker is obtained from the string function getlogin and is displayed in the screen field Entry Clerk.
The system time is decomposed into hours and minutes and then reconstructed into a string variable displayed in the screen field Time Entered. **PERFORM** then writes the record to the **customer** table, using the data on the screen.

```c
#include <stdio.h>
#include <time.h>
#include "ctools.h"

valueptr stamptime();

struct ufunc userfuncs[] =
{
   "stamptime", stamptime,
   0,0
};

valueptr stamptime()
{
   long seconds, time();
   char usertime[10], *getlogin();
   struct tm *timerec, *localtime();

   seconds = time((long *) 0);
   timerec = localtime(&seconds);

   pf_putval(getlogin(), CCHARTYPE, "f009");
   sprintf(usertime, "%02d:%02d",
           timerec->tm_hour, timerec->tm_min);
   pf_putval(usertime, CCHARTYPE, "f010");
}
```

To execute this example, perform the following steps:

1. Compile the form by executing the following command:
   ```
   sformbld p_ex1.per
   ```

2. Create a custom version of **sperform** by executing the following command:
   ```
   cperf stamp.c -o output_file
   ```
   *where* `output_file` *is the name of the file to contain the customer version of sperform.*
3. Run the program by executing the following command:

```plaintext
output_file p_ex1.frm
```

where `output_file` is the name of the file containing the customer version of `sperform`. 
List of Appendixes

Appendix A  The Demonstration Database and Application
Appendix B  Environment Variables
Appendix C  Native Language Support Within INFORMIX-SQL
Appendix D  Modifying termcap and terminfo
Appendix E  The ASCII Character Set
Appendix F  Reserved Words
Appendix G  System Catalogs
Appendix H  Accessing Programs from the Operating System
Appendix I  Using the INFORMIX-OnLine Dynamic Server
The Demonstration Database and Application

The stores demonstration database contains a set of tables that describe an imaginary business. You can access the data in the stores demonstration database by the demonstration programs that appear in this book, as well as by application programs that are listed in the documentation of other Informix products. The stores demonstration database is not MODE ANSI.

This appendix contains the following sections:

- Instructions for copying the demonstration database and application.
- A description of the structure of the tables in the stores demonstration database. For each table, the name and the data type of each column are listed. Any indexes on individual columns or on multiple columns are identified and classified as unique or as allowing duplicate values.
- A graphic map of the tables in the stores demonstration database, showing potential join columns.
- A discussion of the join columns that link some of the tables in the stores demonstration database, illustrating how you can use these relationships to obtain information from multiple tables.
- A listing of the data contained in each table of the stores demonstration database.
• The example forms and reports used in this manual and the INFORMIX-SQL User Guide.

Creating the Demonstration Database

As you learn about INFORMIX-SQL (I-SQL), you will want to experiment with the demonstration database. To do this, you need to make a copy of this material. To make a copy of the demonstration database, select the directory where you want to store the material (often your home directory) and make it your current working directory. Then, from the operating system command line, enter

```
isqldemo
```

The `isqldemo` program creates a subdirectory called `stores.dbs` in your current directory and places the stores demonstration database files there. It also copies all the demonstration report, form, and command files into your current directory.

If you list the contents of your current directory, you will see filenames similar to these:

```
c_index.sql  ord1.ace  ex4.sql
```
```
c_custom.sql  ord1.arc  ex5.sql
```
```
c_items.sql  ord2.ace  ex6.sql
```
```
c_orders.sql  ord2.arc  ex7.sql
```
```
c_manuf.sql  ord3.ace  ex8.sql
```
```
c_state.sql  ord3.arc  ex9.sql
```
```
c_stock.sql  mail1.ace  ex10.sql
```
```
c_stores.sql  mail1.arc  ex11.sql
```
```
customer.per  mail2.ace  ex12.sql
```
```
customer.frm  mail2.arc  ex13.sql
```
```
orderform.per  mail3.ace  ex14.sql
```
```
orderform.frm  mail3.arc  ex15.sql
```
```
sample.per  ex1.sql  ex16.sql
```
```
sample.frm  ex2.sql  ex17.sql
```
```
clist1.ace  ex3.sql  ex18.sql
```
```
clist2.arc  ex19.sql
```
```
clist2.ace  stores.dbs
```

Additional forms, reports, and command files have been included that are not part of the demonstration application. These provide further opportunities for practice once you have become familiar with the demonstration database.
Restoring the Demonstration Database

As you work with your copy of the demonstration database, you can make changes in such a way that the illustrations in this manual no longer reflect what you actually see on your screen. This can happen if you enter a great deal of new information into the stores demonstration database, delete the information that came with the database, or significantly alter the structure of the tables, forms, reports, or command files.

You can restore the demonstration database to an original condition (the one upon which the examples are based) by recreating the database with the `isqldemo` command.

You may want to make a fresh copy of the demonstration database each time you start a new chapter.

If you installed your I-SQL software according to the instructions provided in your installation letter, the files that make up the demonstration database are protected so that you cannot make any changes to the original copy.

Structure of the Tables

The stores demonstration database contains information about a fictitious sporting goods distributor that services stores in the Western United States. This database includes the following tables:

- customer
- orders
- items
- stock
- manufact
- state
The customer Table

The customer table contains information about the retail stores that place orders from the distributor. The columns of the customer table are as follows:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer_num</td>
<td>SERIAL(101)</td>
<td>system-generated customer number</td>
</tr>
<tr>
<td>fname</td>
<td>CHAR(15)</td>
<td>first name of store’s representative</td>
</tr>
<tr>
<td>lname</td>
<td>CHAR(15)</td>
<td>last name of store’s representative</td>
</tr>
<tr>
<td>company</td>
<td>CHAR(20)</td>
<td>name of store</td>
</tr>
<tr>
<td>address1</td>
<td>CHAR(20)</td>
<td>first line of store’s address</td>
</tr>
<tr>
<td>address2</td>
<td>CHAR(20)</td>
<td>second line of store’s address</td>
</tr>
<tr>
<td>city</td>
<td>CHAR(15)</td>
<td>city</td>
</tr>
<tr>
<td>state</td>
<td>CHAR(2)</td>
<td>state</td>
</tr>
<tr>
<td>zipcode</td>
<td>CHAR(5)</td>
<td>zip code</td>
</tr>
<tr>
<td>phone</td>
<td>CHAR(18)</td>
<td>phone number</td>
</tr>
</tbody>
</table>

The customer_num column is indexed and must contain unique values. The zipcode and state columns are indexed to allow duplicate values.

The orders Table

The orders table contains information about orders placed by the distributor’s customers. The columns of the orders table are as follows:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>order_num</td>
<td>SERIAL(1001)</td>
<td>system-generated order number</td>
</tr>
<tr>
<td>order_date</td>
<td>DATE</td>
<td>date order entered</td>
</tr>
<tr>
<td>customer_num</td>
<td>INTEGER</td>
<td>customer number (from customer table)</td>
</tr>
<tr>
<td>ship_instruct</td>
<td>CHAR(40)</td>
<td>special shipping instructions</td>
</tr>
<tr>
<td>backlog</td>
<td>CHAR(1)</td>
<td>indicates order cannot be filled because the item is backlogged: y = yes n = no</td>
</tr>
<tr>
<td>po_num</td>
<td>CHAR(10)</td>
<td>customer purchase order number</td>
</tr>
<tr>
<td>ship_date</td>
<td>DATE</td>
<td>shipping date</td>
</tr>
<tr>
<td>ship_weight</td>
<td>DECIMAL(8,2)</td>
<td>shipping weight</td>
</tr>
<tr>
<td>ship_charge</td>
<td>MONEY(6)</td>
<td>shipping charge</td>
</tr>
<tr>
<td>paid_date</td>
<td>DATE</td>
<td>date order paid</td>
</tr>
</tbody>
</table>

The order_num column is indexed and must contain unique values. The customer_num column is indexed to allow duplicate values.
The **items** Table

An order can include one or more items. There is one row in the **items** table for each item in an order. The columns of the **items** table are as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>item_num</td>
<td>SMALLINT</td>
<td>sequentially assigned item number for an order</td>
</tr>
<tr>
<td>order_num</td>
<td>INTEGER</td>
<td>order number (from <strong>orders</strong> table)</td>
</tr>
<tr>
<td>stock_num</td>
<td>SMALLINT</td>
<td>stock number for item (from <strong>stock</strong> table)</td>
</tr>
<tr>
<td>manu_code</td>
<td>CHAR(3)</td>
<td>manufacturer’s code for item ordered (from <strong>manufact</strong> table)</td>
</tr>
<tr>
<td>quantity</td>
<td>SMALLINT</td>
<td>quantity ordered</td>
</tr>
<tr>
<td>total_price</td>
<td>MONEY(8,2)</td>
<td>quantity ordered × unit price = total price of item</td>
</tr>
</tbody>
</table>

The **order_num** column is indexed and allows duplicate values. A multiple-column index for the **stock_num** and **manu_code** columns also permits duplicate values.

The **stock** Table

The distributor carries 41 different types of sporting goods from various manufacturers. More than one manufacturer can supply a sporting good. For example, the distributor offers racer goggles from two manufacturers and running shoes from six manufacturers.

The **stock** table is a catalog of the items sold by the distributor. The columns of the **stock** table are as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stock_num</td>
<td>SMALLINT</td>
<td>stock number that identifies type of item</td>
</tr>
<tr>
<td>manu_code</td>
<td>CHAR(3)</td>
<td>manufacturer’s code (from <strong>manufact</strong> table)</td>
</tr>
<tr>
<td>description</td>
<td>CHAR(15)</td>
<td>description of item</td>
</tr>
<tr>
<td>unit_price</td>
<td>MONEY(6,2)</td>
<td>unit price</td>
</tr>
<tr>
<td>unit</td>
<td>CHAR(4)</td>
<td>unit by which item is ordered:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>each</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pair</td>
</tr>
<tr>
<td></td>
<td></td>
<td>case</td>
</tr>
<tr>
<td></td>
<td></td>
<td>box</td>
</tr>
<tr>
<td>unit_descr</td>
<td>CHAR(15)</td>
<td>description of unit</td>
</tr>
</tbody>
</table>

The **stock_num** and **manu_code** columns are indexed and allow duplicate values. A multiple-column index for both the **stock_num** and **manu_code** columns allows only unique values.
Structure of the Tables

The **catalog Table**

The `catalog` table describes each of the items in stock. Retail stores use this catalog when placing orders with the distributor. The columns of the `catalog` table are as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>catalog_num</td>
<td>SERIAL(10001)</td>
<td>system-generated catalog number</td>
</tr>
<tr>
<td>stock_num</td>
<td>SMALLINT</td>
<td>distributor’s stock number (from <code>stock</code> table)</td>
</tr>
<tr>
<td>manu_code</td>
<td>CHAR(3)</td>
<td>manufacturer’s code (from <code>manufact</code> table)</td>
</tr>
<tr>
<td>cat_descr</td>
<td>TEXT</td>
<td>description of item</td>
</tr>
<tr>
<td>cat_picture</td>
<td>BYTE</td>
<td>picture of item (binary data)</td>
</tr>
<tr>
<td>cat_advert</td>
<td>VARCHAR(255, 65)</td>
<td>tag line underneath picture</td>
</tr>
</tbody>
</table>

The `catalog_num` column is indexed and must contain unique values. The `stock_num` and `manu_code` columns allow duplicate values. A multiple-column index for the `stock_num` and `manu_code` columns allows only unique values.

The `catalog` table appears only if you are using an INFORMIX-OnLine database engine.

The **cust_calls Table**

All customer calls for information on orders, shipments, or complaints are logged. The `cust_calls` table contains information about these types of customer calls. The columns of the `cust_calls` table are as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer_num</td>
<td>INTEGER</td>
<td>customer number (from <code>customer</code> table)</td>
</tr>
<tr>
<td>call_dtime</td>
<td>DATETIME YEAR TO MINUTE</td>
<td>date and time call received</td>
</tr>
<tr>
<td>user_id</td>
<td>CHAR(18)</td>
<td>name of person logging call</td>
</tr>
<tr>
<td>call_code</td>
<td>CHAR(1)</td>
<td>type of call:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>B</code> = billing error</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>D</code> = damaged goods</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>I</code> = incorrect merchandise sent</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>L</code> = late shipment</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>O</code> = other</td>
</tr>
<tr>
<td>call_descr</td>
<td>CHAR(240)</td>
<td>description of call</td>
</tr>
<tr>
<td>res_dtime</td>
<td>DATETIME YEAR TO MINUTE</td>
<td>date and time call resolved</td>
</tr>
<tr>
<td>res_descr</td>
<td>CHAR(240)</td>
<td>description of how call was resolved</td>
</tr>
</tbody>
</table>
A multiple-column index for both the customer_num and call_dtime columns allows only unique values. The customer_num column also has an index that allows duplicate values.

The manufact Table

Information about the nine manufacturers whose sporting goods are handled by the distributor is stored in the manufact table. The columns of the manufact table are as follows:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>manu_code</td>
<td>CHAR(3)</td>
<td>manufacturer’s code</td>
</tr>
<tr>
<td>manu_name</td>
<td>CHAR(15)</td>
<td>name of manufacturer</td>
</tr>
<tr>
<td>lead_time</td>
<td>INTERVAL DAY(3) TO DAY</td>
<td>lead time for shipment of orders</td>
</tr>
</tbody>
</table>

The manu_code column has an index that requires unique values.

The state Table

The state table contains the names and postal abbreviations for the 50 states of the United States. It includes the following two columns:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>CHAR(2)</td>
<td>state code</td>
</tr>
<tr>
<td>sname</td>
<td>CHAR(15)</td>
<td>state name</td>
</tr>
</tbody>
</table>

The code column is indexed as unique.
Map of the Demonstration Database

Figure A-1 displays the column names of the tables in the demonstration database. Shading connecting a column in one table to a column in another table indicates columns that contain the same information.

Join Columns That Link the Database

The tables of the demonstration database are linked together by the join columns shown in Figure A-1 and identified in this section. You can use these columns to retrieve and display information from several tables at once, as if the information had been stored in a single table. Figure A-1 through Figure A-8 show the join relationships among tables, and how information stored in one table supplements information stored in others.
Join Columns in the *customer* and *orders* Tables

The **customer_num** column joins the *customer* table and the *orders* table, as shown in Figure A-2.

<table>
<thead>
<tr>
<th>customer_num</th>
<th>fname</th>
<th>lname</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Ludwig</td>
<td>Pauli</td>
</tr>
<tr>
<td>102</td>
<td>Carole</td>
<td>Sadler</td>
</tr>
<tr>
<td>103</td>
<td>Philip</td>
<td>Currie</td>
</tr>
<tr>
<td>104</td>
<td>Anthony</td>
<td>Higgins</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>order_num</th>
<th>order_date</th>
<th>customer_num</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>05/20/1990</td>
<td>104</td>
</tr>
<tr>
<td>1002</td>
<td>05/21/1990</td>
<td>101</td>
</tr>
<tr>
<td>1003</td>
<td>05/22/1990</td>
<td>104</td>
</tr>
<tr>
<td>1004</td>
<td>05/22/1990</td>
<td>106</td>
</tr>
</tbody>
</table>

**Figure A-2**  
*Tables joined by the customer_num column*

The *customer* table contains a **customer_num** column that holds a number identifying a customer, along with columns for the customer’s name, company, address, and telephone number. For example, the row with information about Anthony Higgins contains the number 104 in the **customer_num** column. The *orders* table also contains a **customer_num** column that stores the number of the customer who placed a particular order.

According to Figure A-2, customer 104 (Anthony Higgins) has placed two orders since his customer number appears in two rows of the *orders* table. Since the join relationship lets you select information from both tables, you can retrieve Anthony Higgins’ name and address and information about his orders at the same time.
Join Columns in the \textit{orders} and \textit{items} Tables

The \textit{orders} and \textit{items} tables are linked by an order\_num column that contains an identification number for each order. If an order includes several items, the same order number appears in several rows of the \textit{items} table. Figure A-3 shows this relationship.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
\textbf{order\_num} & \textbf{order\_date} & \textbf{customer\_num} \\
1001 & 05/20/1990 & 104 \\
1002 & 05/21/1990 & 101 \\
1003 & 05/22/1990 & 104 \\
\hline
\end{tabular}
\caption{\textit{orders} table (detail)}
\end{table}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
\textbf{item\_num} & \textbf{order\_num} & \textbf{stock\_num} & \textbf{manu\_code} \\
1 & 1001 & 1 & HRO \\
1 & 1002 & 4 & HSK \\
2 & 1002 & 3 & HSK \\
1 & 1003 & 9 & ANZ \\
2 & 1003 & 8 & ANZ \\
3 & 1003 & 5 & ANZ \\
\hline
\end{tabular}
\caption{\textit{items} table (detail)}
\end{table}

\textbf{Figure A-3} \hspace{1em} Tables joined by the order\_num column
Join Columns in the *items* and *stock* Tables

The *items* table and the *stock* table are joined by two columns: the *stock_num* column stores a stock number for an item, and the *manu_code* column stores a code that identifies the manufacturer. You need both the stock number and the manufacturer code to uniquely identify an item.

For example, the item with the stock number 1 and the manufacturer code HRO is a Hero baseball glove, while the item with the stock number 1 and the manufacturer code HSK is a Husky baseball glove.

The same stock number and manufacturer code can appear in more than one row of the *items* table, if the same item belongs to separate orders, as illustrated in Figure A-4.

![Figure A-4 Tables joined by the stock_num and manu_code columns](image)
Join Columns in the stock and catalog Tables

The catalog table and the stock table are joined by two columns: the stock_num column stores a stock number for an item, and the manu_code column stores a code that identifies the manufacturer. You need both of these columns to uniquely identify an item. Figure A-5 shows this relationship.

<table>
<thead>
<tr>
<th>stock_num</th>
<th>manu_code</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HRO</td>
<td>baseball gloves</td>
</tr>
<tr>
<td>1</td>
<td>HSK</td>
<td>baseball gloves</td>
</tr>
<tr>
<td>1</td>
<td>SMT</td>
<td>baseball gloves</td>
</tr>
</tbody>
</table>

Figure A-5 Tables joined by the stock_num and manu_code columns

Join Columns in the stock and manufact Tables

The stock table and the manufact table are joined by the manu_code column. The same manufacturer code can appear in more than one row of the stock table if the manufacturer produces more than one piece of equipment. This relationship is illustrated in Figure A-6.

<table>
<thead>
<tr>
<th>stock_num</th>
<th>manu_code</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HRO</td>
<td>baseball gloves</td>
</tr>
<tr>
<td>1</td>
<td>HSK</td>
<td>baseball gloves</td>
</tr>
<tr>
<td>1</td>
<td>SMT</td>
<td>baseball gloves</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>manu_code</th>
<th>manu_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRG</td>
<td>Norge</td>
</tr>
<tr>
<td>HSK</td>
<td>Husky</td>
</tr>
<tr>
<td>HRO</td>
<td>Hero</td>
</tr>
</tbody>
</table>

Figure A-6 Tables joined by the manu_code column
Join Columns in the *cust_calls* and *customer* Tables

The *cust_calls* table and the *customer* table are joined by the *customer_num* column. The same customer number can appear in more than one row of the *cust_calls* table if the customer calls the distributor more than once with a problem or question. This relationship is illustrated in Figure A-7.

<table>
<thead>
<tr>
<th>customer_num</th>
<th>fname</th>
<th>lname</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Ludwig</td>
<td>Pauli</td>
</tr>
<tr>
<td>102</td>
<td>Carole</td>
<td>Sadler</td>
</tr>
<tr>
<td>103</td>
<td>Philip</td>
<td>Currie</td>
</tr>
<tr>
<td>104</td>
<td>Anthony</td>
<td>Higgins</td>
</tr>
<tr>
<td>105</td>
<td>Raymond</td>
<td>Vector</td>
</tr>
<tr>
<td>106</td>
<td>George</td>
<td>Watson</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer_num</th>
<th>call_dtime</th>
<th>user_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>106</td>
<td>1990-06-12 08:20</td>
<td>maryj</td>
</tr>
<tr>
<td>127</td>
<td>1990-07-31 14:30</td>
<td>maryj</td>
</tr>
<tr>
<td>116</td>
<td>1990-11-28 13:34</td>
<td>mannyh</td>
</tr>
<tr>
<td>116</td>
<td>1989-12-21 11:24</td>
<td>mannyh</td>
</tr>
</tbody>
</table>

*Figure A-7*  Tables joined by the *customer_num* column
Join Columns in the *state* and *customer* Tables

The *state* table and the *customer* table are joined by a column that contains the state code. This column is called `code` in the *state* table and `state` in the *customer* table. If several customers live in the same state, the same state code will appear in several rows of the table, as shown in Figure A-8.

<table>
<thead>
<tr>
<th>customer_num</th>
<th>fname</th>
<th>lname</th>
<th>...</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Ludwig</td>
<td>Pauli</td>
<td>...</td>
<td>CA</td>
</tr>
<tr>
<td>102</td>
<td>Carole</td>
<td>Sadler</td>
<td>...</td>
<td>CA</td>
</tr>
<tr>
<td>103</td>
<td>Philip</td>
<td>Currie</td>
<td>...</td>
<td>CA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>code</th>
<th>sname</th>
</tr>
</thead>
<tbody>
<tr>
<td>AK</td>
<td>Alaska</td>
</tr>
<tr>
<td>AL</td>
<td>Alabama</td>
</tr>
<tr>
<td>AR</td>
<td>Arkansas</td>
</tr>
<tr>
<td>AZ</td>
<td>Arizona</td>
</tr>
<tr>
<td>CA</td>
<td>California</td>
</tr>
</tbody>
</table>

Figure A-8 Tables joined by the state/code column

Data in the Demonstration Database

The tables that follow display the data in the stores demonstration database.
# Map of the Demonstration Database

## customer Table

<table>
<thead>
<tr>
<th>customer_num</th>
<th>fname</th>
<th>lname</th>
<th>company</th>
<th>address1</th>
<th>address2</th>
<th>city</th>
<th>state</th>
<th>zipcode</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Ludwig</td>
<td>Pauli</td>
<td>All Sports Supplies</td>
<td>213 Erstwild Court</td>
<td>Sunnyvale</td>
<td>CA</td>
<td>94086</td>
<td>408-789-8075</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>Carole</td>
<td>Sadler</td>
<td>Sports Spot</td>
<td>785 Geary St</td>
<td>San Francisco</td>
<td>CA</td>
<td>94117</td>
<td>415-822-1289</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>Philip</td>
<td>Currie</td>
<td>Phil’s Sports</td>
<td>654 Poplar</td>
<td>Palo Alto</td>
<td>CA</td>
<td>94303</td>
<td>415-328-4531</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>Anthony</td>
<td>Higgins</td>
<td>Play Ball!</td>
<td>East Shopping Cntr.</td>
<td>Redwood City</td>
<td>CA</td>
<td>94065</td>
<td>415-368-1100</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>Raymond</td>
<td>Vector</td>
<td>Los Altos Sports</td>
<td>1899 La Loma Drive</td>
<td>Los Altos</td>
<td>CA</td>
<td>94022</td>
<td>415-776-3249</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>George</td>
<td>Watson</td>
<td>Watson &amp; Son</td>
<td>1143 Carrier Place</td>
<td>Mountain View</td>
<td>CA</td>
<td>94043</td>
<td>415-369-8789</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>Charles</td>
<td>Ream</td>
<td>Athletic Supplies</td>
<td>41 Jordan Avenue</td>
<td>Palo Alto</td>
<td>CA</td>
<td>94304</td>
<td>415-536-6586</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>Donald</td>
<td>Quinn</td>
<td>Quinn’s Sports</td>
<td>587 Alvando</td>
<td>Redwood City</td>
<td>CA</td>
<td>94063</td>
<td>415-544-6729</td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>Jane</td>
<td>Miller</td>
<td>Sport Shufi</td>
<td>Mayfair Mart</td>
<td>Sunnyvale</td>
<td>CA</td>
<td>94086</td>
<td>415-723-7869</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Roy</td>
<td>Jaeger</td>
<td>AA Athletics</td>
<td>520 Topaz Way</td>
<td>Redwood City</td>
<td>CA</td>
<td>94062</td>
<td>415-743-3611</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>Frances</td>
<td>Keys</td>
<td>Sports Center</td>
<td>3199 Sterling Court</td>
<td>Sunnyvale</td>
<td>CA</td>
<td>94085</td>
<td>415-277-7245</td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>Margaret</td>
<td>Lawson</td>
<td>Runners &amp; Others</td>
<td>234 Wyandotte Way</td>
<td>Los Altos</td>
<td>CA</td>
<td>94022</td>
<td>415-867-7235</td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>Lana</td>
<td>Beatty</td>
<td>Sportstown</td>
<td>654 Oak Grove</td>
<td>Menlo Park</td>
<td>CA</td>
<td>94025</td>
<td>415-356-9982</td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>Frank</td>
<td>Albertson</td>
<td>Sporting Place</td>
<td>947 Waverly Place</td>
<td>Redwood City</td>
<td>CA</td>
<td>94062</td>
<td>415-866-6677</td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>Alfred</td>
<td>Grant</td>
<td>Gold Medal Sports</td>
<td>776 Gary Avenue</td>
<td>Menlo Park</td>
<td>CA</td>
<td>94025</td>
<td>415-356-1123</td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>Jean</td>
<td>Parmele</td>
<td>Olympic City</td>
<td>1104 Spindola Drive</td>
<td>Mountain View</td>
<td>CA</td>
<td>94040</td>
<td>415-534-8622</td>
<td></td>
</tr>
<tr>
<td>117</td>
<td>Arnold</td>
<td>Sipes</td>
<td>Kids Komer</td>
<td>850 Lypton Court</td>
<td>Redwood City</td>
<td>CA</td>
<td>94063</td>
<td>415-245-4578</td>
<td></td>
</tr>
<tr>
<td>118</td>
<td>Dick</td>
<td>Baxter</td>
<td>Blue Ribbon Sports</td>
<td>5427 College</td>
<td>Oakland</td>
<td>CA</td>
<td>94609</td>
<td>415-665-0011</td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>Bob</td>
<td>Shorter</td>
<td>The Triathletes Club</td>
<td>2405 Kings Highway</td>
<td>Cherry Hill</td>
<td>NJ</td>
<td>08025</td>
<td>609-663-6079</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>Fred</td>
<td>Jewell</td>
<td>Century Pro Shop</td>
<td>6627 N. 17th Way</td>
<td>Phoenix</td>
<td>AZ</td>
<td>85016</td>
<td>602-265-7854</td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>Jason</td>
<td>Wallack</td>
<td>City Sports</td>
<td>Lake Biltmore Mall</td>
<td>Wilmington</td>
<td>DE</td>
<td>19898</td>
<td>302-366-7511</td>
<td></td>
</tr>
<tr>
<td>122</td>
<td>Cathy</td>
<td>O’Brian</td>
<td>The Sporting Life</td>
<td>540 Nassau Street</td>
<td>Princeton</td>
<td>NJ</td>
<td>08540</td>
<td>609-342-0054</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>Marvin</td>
<td>Hanlon</td>
<td>Bay Sports</td>
<td>10100 Bay Meadows Rd</td>
<td>Jacksonville</td>
<td>FL</td>
<td>32256</td>
<td>904-823-4299</td>
<td></td>
</tr>
<tr>
<td>124</td>
<td>Chris</td>
<td>Putnum</td>
<td>Putnum’s Putters</td>
<td>4715 S.E. Adams Blvd</td>
<td>Bartksville</td>
<td>OK</td>
<td>74006</td>
<td>918-375-5704</td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>James</td>
<td>Henry</td>
<td>Total Fitness Sports</td>
<td>1800 Commonwealth Av</td>
<td>Brighton</td>
<td>MA</td>
<td>02135</td>
<td>617-232-4159</td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>Eileen</td>
<td>Neelie</td>
<td>Neelie’s Discount Sp</td>
<td>2539 South Utica Str</td>
<td>Denver</td>
<td>CO</td>
<td>80219</td>
<td>303-936-7731</td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>Kim</td>
<td>Satisfi</td>
<td>Big Blue Bike Shop</td>
<td>Blue Island Square</td>
<td>Blue Island</td>
<td>NY</td>
<td>11222</td>
<td>312-944-5691</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>Frank</td>
<td>Leisor</td>
<td>Phoenix University Athletic Department</td>
<td>1817 N. Thomas Road</td>
<td>Phoenix</td>
<td>AZ</td>
<td>85008</td>
<td>602-333-1817</td>
<td></td>
</tr>
</tbody>
</table>
Map of the Demonstration Database

items Table (1 of 2)

<table>
<thead>
<tr>
<th>item_num</th>
<th>order_num</th>
<th>stock_num</th>
<th>manu_code</th>
<th>quantity</th>
<th>total_price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1001</td>
<td>1</td>
<td>HRO</td>
<td>1</td>
<td>250.00</td>
</tr>
<tr>
<td>1</td>
<td>1002</td>
<td>4</td>
<td>HSK</td>
<td>1</td>
<td>960.00</td>
</tr>
<tr>
<td>2</td>
<td>1002</td>
<td>3</td>
<td>HSK</td>
<td>1</td>
<td>240.00</td>
</tr>
<tr>
<td>1</td>
<td>1003</td>
<td>9</td>
<td>ANZ</td>
<td>1</td>
<td>20.00</td>
</tr>
<tr>
<td>2</td>
<td>1003</td>
<td>8</td>
<td>ANZ</td>
<td>1</td>
<td>840.00</td>
</tr>
<tr>
<td>3</td>
<td>1003</td>
<td>5</td>
<td>ANZ</td>
<td>5</td>
<td>99.00</td>
</tr>
<tr>
<td>1</td>
<td>1004</td>
<td>1</td>
<td>HRO</td>
<td>1</td>
<td>250.00</td>
</tr>
<tr>
<td>2</td>
<td>1004</td>
<td>2</td>
<td>HRO</td>
<td>1</td>
<td>126.00</td>
</tr>
<tr>
<td>3</td>
<td>1004</td>
<td>3</td>
<td>HSK</td>
<td>1</td>
<td>240.00</td>
</tr>
<tr>
<td>4</td>
<td>1004</td>
<td>1</td>
<td>HSK</td>
<td>1</td>
<td>800.00</td>
</tr>
<tr>
<td>1</td>
<td>1005</td>
<td>5</td>
<td>NRG</td>
<td>10</td>
<td>280.00</td>
</tr>
<tr>
<td>2</td>
<td>1005</td>
<td>5</td>
<td>ANZ</td>
<td>10</td>
<td>198.00</td>
</tr>
<tr>
<td>3</td>
<td>1005</td>
<td>6</td>
<td>SMT</td>
<td>1</td>
<td>36.00</td>
</tr>
<tr>
<td>4</td>
<td>1005</td>
<td>6</td>
<td>ANZ</td>
<td>1</td>
<td>48.00</td>
</tr>
<tr>
<td>5</td>
<td>1006</td>
<td>6</td>
<td>SMT</td>
<td>1</td>
<td>48.00</td>
</tr>
<tr>
<td>6</td>
<td>1006</td>
<td>6</td>
<td>ANZ</td>
<td>1</td>
<td>48.00</td>
</tr>
<tr>
<td>1</td>
<td>1007</td>
<td>1</td>
<td>HRO</td>
<td>1</td>
<td>250.00</td>
</tr>
<tr>
<td>2</td>
<td>1007</td>
<td>2</td>
<td>HRO</td>
<td>1</td>
<td>126.00</td>
</tr>
<tr>
<td>3</td>
<td>1007</td>
<td>3</td>
<td>HSK</td>
<td>1</td>
<td>240.00</td>
</tr>
<tr>
<td>4</td>
<td>1007</td>
<td>4</td>
<td>HRO</td>
<td>1</td>
<td>480.00</td>
</tr>
<tr>
<td>5</td>
<td>1007</td>
<td>7</td>
<td>HRO</td>
<td>1</td>
<td>600.00</td>
</tr>
<tr>
<td>1</td>
<td>1008</td>
<td>8</td>
<td>ANZ</td>
<td>1</td>
<td>840.00</td>
</tr>
<tr>
<td>2</td>
<td>1008</td>
<td>9</td>
<td>ANZ</td>
<td>5</td>
<td>100.00</td>
</tr>
<tr>
<td>1</td>
<td>1009</td>
<td>1</td>
<td>SMT</td>
<td>1</td>
<td>450.00</td>
</tr>
<tr>
<td>1</td>
<td>1010</td>
<td>6</td>
<td>SMT</td>
<td>1</td>
<td>36.00</td>
</tr>
<tr>
<td>2</td>
<td>1010</td>
<td>6</td>
<td>ANZ</td>
<td>1</td>
<td>48.00</td>
</tr>
<tr>
<td>1</td>
<td>1011</td>
<td>5</td>
<td>ANZ</td>
<td>5</td>
<td>99.00</td>
</tr>
<tr>
<td>1</td>
<td>1012</td>
<td>8</td>
<td>ANZ</td>
<td>1</td>
<td>840.00</td>
</tr>
<tr>
<td>2</td>
<td>1012</td>
<td>9</td>
<td>ANZ</td>
<td>10</td>
<td>200.00</td>
</tr>
<tr>
<td>1</td>
<td>1013</td>
<td>5</td>
<td>ANZ</td>
<td>1</td>
<td>19.80</td>
</tr>
<tr>
<td>2</td>
<td>1013</td>
<td>6</td>
<td>SMT</td>
<td>1</td>
<td>36.00</td>
</tr>
<tr>
<td>3</td>
<td>1013</td>
<td>6</td>
<td>ANZ</td>
<td>1</td>
<td>48.00</td>
</tr>
<tr>
<td>4</td>
<td>1013</td>
<td>9</td>
<td>ANZ</td>
<td>2</td>
<td>40.00</td>
</tr>
<tr>
<td>1</td>
<td>1014</td>
<td>4</td>
<td>HSK</td>
<td>1</td>
<td>960.00</td>
</tr>
<tr>
<td>2</td>
<td>1014</td>
<td>4</td>
<td>HRO</td>
<td>1</td>
<td>480.00</td>
</tr>
<tr>
<td>1</td>
<td>1015</td>
<td>1</td>
<td>SMT</td>
<td>1</td>
<td>450.00</td>
</tr>
<tr>
<td>1</td>
<td>1016</td>
<td>101</td>
<td>SHM</td>
<td>2</td>
<td>136.00</td>
</tr>
<tr>
<td>2</td>
<td>1016</td>
<td>109</td>
<td>PRC</td>
<td>3</td>
<td>90.00</td>
</tr>
<tr>
<td>3</td>
<td>1016</td>
<td>110</td>
<td>HSK</td>
<td>1</td>
<td>308.00</td>
</tr>
<tr>
<td>4</td>
<td>1016</td>
<td>114</td>
<td>PRC</td>
<td>1</td>
<td>120.00</td>
</tr>
<tr>
<td>1</td>
<td>1017</td>
<td>201</td>
<td>NKL</td>
<td>4</td>
<td>150.00</td>
</tr>
<tr>
<td>2</td>
<td>1017</td>
<td>202</td>
<td>KAR</td>
<td>1</td>
<td>230.00</td>
</tr>
<tr>
<td>3</td>
<td>1017</td>
<td>301</td>
<td>SHM</td>
<td>2</td>
<td>204.00</td>
</tr>
<tr>
<td>1</td>
<td>1018</td>
<td>307</td>
<td>PRC</td>
<td>2</td>
<td>500.00</td>
</tr>
</tbody>
</table>
## Map of the Demonstration Database

### items Table (2 of 2)

<table>
<thead>
<tr>
<th>item_num</th>
<th>order_num</th>
<th>stock_num</th>
<th>manu_code</th>
<th>quantity</th>
<th>total_price</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1018</td>
<td>302</td>
<td>KAR</td>
<td>3</td>
<td>15.00</td>
</tr>
<tr>
<td>3</td>
<td>1018</td>
<td>110</td>
<td>PRC</td>
<td>1</td>
<td>236.00</td>
</tr>
<tr>
<td>4</td>
<td>1018</td>
<td>5</td>
<td>SMT</td>
<td>4</td>
<td>100.00</td>
</tr>
<tr>
<td>5</td>
<td>1018</td>
<td>304</td>
<td>HRO</td>
<td>1</td>
<td>280.00</td>
</tr>
<tr>
<td>1</td>
<td>1019</td>
<td>111</td>
<td>SHM</td>
<td>3</td>
<td>1499.97</td>
</tr>
<tr>
<td>2</td>
<td>1020</td>
<td>204</td>
<td>KAR</td>
<td>2</td>
<td>90.00</td>
</tr>
<tr>
<td>3</td>
<td>1020</td>
<td>301</td>
<td>KAR</td>
<td>4</td>
<td>348.00</td>
</tr>
<tr>
<td>1</td>
<td>1021</td>
<td>201</td>
<td>NKL</td>
<td>2</td>
<td>75.00</td>
</tr>
<tr>
<td>2</td>
<td>1021</td>
<td>201</td>
<td>ANZ</td>
<td>3</td>
<td>225.00</td>
</tr>
<tr>
<td>3</td>
<td>1021</td>
<td>202</td>
<td>KAR</td>
<td>3</td>
<td>690.00</td>
</tr>
<tr>
<td>4</td>
<td>1021</td>
<td>205</td>
<td>ANZ</td>
<td>2</td>
<td>624.00</td>
</tr>
<tr>
<td>1</td>
<td>1022</td>
<td>309</td>
<td>HRO</td>
<td>1</td>
<td>40.00</td>
</tr>
<tr>
<td>2</td>
<td>1022</td>
<td>303</td>
<td>PRC</td>
<td>2</td>
<td>96.00</td>
</tr>
<tr>
<td>3</td>
<td>1022</td>
<td>6</td>
<td>ANZ</td>
<td>2</td>
<td>96.00</td>
</tr>
<tr>
<td>1</td>
<td>1023</td>
<td>103</td>
<td>PRC</td>
<td>2</td>
<td>40.00</td>
</tr>
<tr>
<td>2</td>
<td>1023</td>
<td>104</td>
<td>PRC</td>
<td>2</td>
<td>116.00</td>
</tr>
<tr>
<td>3</td>
<td>1023</td>
<td>105</td>
<td>SHM</td>
<td>1</td>
<td>80.00</td>
</tr>
<tr>
<td>4</td>
<td>1023</td>
<td>110</td>
<td>SHM</td>
<td>1</td>
<td>228.00</td>
</tr>
<tr>
<td>5</td>
<td>1023</td>
<td>304</td>
<td>ANZ</td>
<td>1</td>
<td>170.00</td>
</tr>
<tr>
<td>6</td>
<td>1023</td>
<td>306</td>
<td>SHM</td>
<td>1</td>
<td>190.00</td>
</tr>
<tr>
<td>order_num</td>
<td>order_date</td>
<td>customer_num</td>
<td>ship_instruct</td>
<td>backlog</td>
<td>po_num</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------</td>
<td>--------------</td>
<td>--------------------------------</td>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>1001</td>
<td>05/20/1990</td>
<td>104</td>
<td>express</td>
<td>n</td>
<td>B77836</td>
</tr>
<tr>
<td>1002</td>
<td>05/21/1990</td>
<td>101</td>
<td>PO on box; deliver back door only</td>
<td>n</td>
<td>9270</td>
</tr>
<tr>
<td>1003</td>
<td>05/22/1990</td>
<td>104</td>
<td>express</td>
<td>n</td>
<td>B77890</td>
</tr>
<tr>
<td>1004</td>
<td>05/22/1990</td>
<td>106</td>
<td>ring bell twice</td>
<td>y</td>
<td>8006</td>
</tr>
<tr>
<td>1005</td>
<td>05/24/1990</td>
<td>116</td>
<td>call before delivery</td>
<td>n</td>
<td>2685</td>
</tr>
<tr>
<td>1006</td>
<td>05/30/1990</td>
<td>112</td>
<td>after 10 am</td>
<td>y</td>
<td>Q13557</td>
</tr>
<tr>
<td>1007</td>
<td>05/31/1990</td>
<td>117</td>
<td></td>
<td>n</td>
<td>Z78693</td>
</tr>
<tr>
<td>1008</td>
<td>06/07/1990</td>
<td>110</td>
<td>closed Monday</td>
<td>y</td>
<td>LZ230</td>
</tr>
<tr>
<td>1009</td>
<td>06/14/1990</td>
<td>111</td>
<td>next door to grocery</td>
<td>n</td>
<td>4745</td>
</tr>
<tr>
<td>1010</td>
<td>06/17/1990</td>
<td>115</td>
<td>deliver 77% King St. if no answer</td>
<td>n</td>
<td>4294Q</td>
</tr>
<tr>
<td>1011</td>
<td>06/18/1990</td>
<td>104</td>
<td>express</td>
<td>n</td>
<td>B77897</td>
</tr>
<tr>
<td>1012</td>
<td>06/18/1990</td>
<td>117</td>
<td></td>
<td>n</td>
<td>27801</td>
</tr>
<tr>
<td>1013</td>
<td>06/22/1990</td>
<td>104</td>
<td>express</td>
<td>n</td>
<td>B77930</td>
</tr>
<tr>
<td>1014</td>
<td>06/25/1990</td>
<td>106</td>
<td>ring bell, kick door loudly</td>
<td>n</td>
<td>8052</td>
</tr>
<tr>
<td>1015</td>
<td>06/27/1990</td>
<td>110</td>
<td>closed Mondays</td>
<td>n</td>
<td>MA003</td>
</tr>
<tr>
<td>1016</td>
<td>06/29/1990</td>
<td>119</td>
<td>delivery entrance off Camp St</td>
<td>n</td>
<td>PC6782</td>
</tr>
<tr>
<td>1017</td>
<td>07/09/1990</td>
<td>120</td>
<td>North side of clubhouse</td>
<td>n</td>
<td>DM2544331</td>
</tr>
<tr>
<td>1018</td>
<td>07/10/1990</td>
<td>121</td>
<td>SW corner of Baltimore Mall</td>
<td>n</td>
<td>S52042</td>
</tr>
<tr>
<td>1019</td>
<td>07/11/1990</td>
<td>122</td>
<td>closed till noon Mondays</td>
<td>n</td>
<td>Z55709</td>
</tr>
<tr>
<td>1020</td>
<td>07/11/1990</td>
<td>123</td>
<td>express</td>
<td>n</td>
<td>W2286</td>
</tr>
<tr>
<td>1021</td>
<td>07/23/1990</td>
<td>124</td>
<td>ask for Elaine</td>
<td>n</td>
<td>C3288</td>
</tr>
<tr>
<td>1022</td>
<td>07/24/1990</td>
<td>126</td>
<td>express</td>
<td>n</td>
<td>W9925</td>
</tr>
<tr>
<td>1023</td>
<td>07/24/1990</td>
<td>127</td>
<td>no deliveries after 3 p.m.</td>
<td>n</td>
<td>KF2961</td>
</tr>
</tbody>
</table>
stock Table (1 of 2)

<table>
<thead>
<tr>
<th>stock_num</th>
<th>manu_code</th>
<th>description</th>
<th>unit_price</th>
<th>unit</th>
<th>unit_descr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HRO</td>
<td>baseball gloves</td>
<td>250.00</td>
<td>case</td>
<td>10 gloves/case</td>
</tr>
<tr>
<td>1</td>
<td>HSK</td>
<td>baseball gloves</td>
<td>800.00</td>
<td>case</td>
<td>10 gloves/case</td>
</tr>
<tr>
<td>1</td>
<td>SMT</td>
<td>baseball gloves</td>
<td>450.00</td>
<td>case</td>
<td>10 gloves/case</td>
</tr>
<tr>
<td>2</td>
<td>HRO</td>
<td>baseball bat</td>
<td>126.00</td>
<td>case</td>
<td>24/case</td>
</tr>
<tr>
<td>3</td>
<td>HSK</td>
<td>baseball bat</td>
<td>240.00</td>
<td>case</td>
<td>12/case</td>
</tr>
<tr>
<td>4</td>
<td>HSK</td>
<td>football</td>
<td>960.00</td>
<td>case</td>
<td>24/case</td>
</tr>
<tr>
<td>4</td>
<td>HRO</td>
<td>football</td>
<td>480.00</td>
<td>case</td>
<td>24/case</td>
</tr>
<tr>
<td>5</td>
<td>NRG</td>
<td>tennis racquet</td>
<td>28.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>5</td>
<td>SMT</td>
<td>tennis racquet</td>
<td>25.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>5</td>
<td>ANZ</td>
<td>tennis racquet</td>
<td>19.80</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>6</td>
<td>SMT</td>
<td>tennis ball</td>
<td>36.00</td>
<td>case</td>
<td>24 cans/case</td>
</tr>
<tr>
<td>6</td>
<td>ANZ</td>
<td>tennis ball</td>
<td>48.00</td>
<td>case</td>
<td>24 cans/case</td>
</tr>
<tr>
<td>7</td>
<td>HRO</td>
<td>basketball</td>
<td>600.00</td>
<td>case</td>
<td>24/case</td>
</tr>
<tr>
<td>8</td>
<td>ANZ</td>
<td>volleyball</td>
<td>840.00</td>
<td>case</td>
<td>24/case</td>
</tr>
<tr>
<td>9</td>
<td>ANZ</td>
<td>volleyball net</td>
<td>20.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>101</td>
<td>PRC</td>
<td>bicycle tires</td>
<td>88.00</td>
<td>box</td>
<td>4/box</td>
</tr>
<tr>
<td>101</td>
<td>SHM</td>
<td>bicycle tires</td>
<td>68.00</td>
<td>box</td>
<td>4/box</td>
</tr>
<tr>
<td>102</td>
<td>SHM</td>
<td>bicycle brakes</td>
<td>220.00</td>
<td>case</td>
<td>4 sets/case</td>
</tr>
<tr>
<td>102</td>
<td>PRC</td>
<td>bicycle brakes</td>
<td>480.00</td>
<td>case</td>
<td>4 sets/case</td>
</tr>
<tr>
<td>103</td>
<td>PRC</td>
<td>front derailleur</td>
<td>20.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>104</td>
<td>PRC</td>
<td>rear derailleur</td>
<td>58.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>105</td>
<td>PRC</td>
<td>bicycle wheels</td>
<td>53.00</td>
<td>pair</td>
<td>pair</td>
</tr>
<tr>
<td>105</td>
<td>SHM</td>
<td>bicycle wheels</td>
<td>80.00</td>
<td>pair</td>
<td>pair</td>
</tr>
<tr>
<td>106</td>
<td>PRC</td>
<td>bicycle stem</td>
<td>23.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>107</td>
<td>PRC</td>
<td>bicycle saddle</td>
<td>70.00</td>
<td>pair</td>
<td>pair</td>
</tr>
<tr>
<td>108</td>
<td>SHM</td>
<td>crankset</td>
<td>45.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>109</td>
<td>PRC</td>
<td>pedal binding</td>
<td>30.00</td>
<td>case</td>
<td>6 pairs/case</td>
</tr>
<tr>
<td>109</td>
<td>SHM</td>
<td>pedal binding</td>
<td>200.00</td>
<td>case</td>
<td>4 pairs/case</td>
</tr>
<tr>
<td>110</td>
<td>PRC</td>
<td>helmet</td>
<td>236.00</td>
<td>case</td>
<td>4/case</td>
</tr>
<tr>
<td>110</td>
<td>ANZ</td>
<td>helmet</td>
<td>244.00</td>
<td>case</td>
<td>4/case</td>
</tr>
<tr>
<td>110</td>
<td>SHM</td>
<td>helmet</td>
<td>228.00</td>
<td>case</td>
<td>4/case</td>
</tr>
<tr>
<td>110</td>
<td>HRO</td>
<td>helmet</td>
<td>260.00</td>
<td>case</td>
<td>4/case</td>
</tr>
<tr>
<td>110</td>
<td>HSK</td>
<td>helmet</td>
<td>308.00</td>
<td>case</td>
<td>4/case</td>
</tr>
<tr>
<td>111</td>
<td>SHM</td>
<td>10-spd, assmbld</td>
<td>499.99</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>112</td>
<td>SHM</td>
<td>12-spd, assmbld</td>
<td>549.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>113</td>
<td>SHM</td>
<td>18-spd, assmbld</td>
<td>685.90</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>114</td>
<td>PRC</td>
<td>bicycle gloves</td>
<td>120.00</td>
<td>case</td>
<td>10 pairs/case</td>
</tr>
</tbody>
</table>
### Map of the Demonstration Database

#### stock Table (2 of 2)

<table>
<thead>
<tr>
<th>stock_num</th>
<th>manu_code</th>
<th>description</th>
<th>unit_price</th>
<th>unit</th>
<th>unit_descr</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>NKL</td>
<td>golf shoes</td>
<td>37.50</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>201</td>
<td>ANZ</td>
<td>golf shoes</td>
<td>75.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>201</td>
<td>KAR</td>
<td>golf shoes</td>
<td>90.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>202</td>
<td>NKL</td>
<td>metal woods</td>
<td>174.00</td>
<td>case</td>
<td>2 sets/case</td>
</tr>
<tr>
<td>202</td>
<td>KAR</td>
<td>std woods</td>
<td>230.00</td>
<td>case</td>
<td>2 sets/case</td>
</tr>
<tr>
<td>203</td>
<td>NKL</td>
<td>irons/wedges</td>
<td>670.00</td>
<td>case</td>
<td>2 sets/case</td>
</tr>
<tr>
<td>204</td>
<td>KAR</td>
<td>putter</td>
<td>45.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>205</td>
<td>NKL</td>
<td>3 golf balls</td>
<td>312.00</td>
<td>case</td>
<td>24/case</td>
</tr>
<tr>
<td>205</td>
<td>ANZ</td>
<td>3 golf balls</td>
<td>312.00</td>
<td>case</td>
<td>24/case</td>
</tr>
<tr>
<td>205</td>
<td>HRO</td>
<td>3 golf balls</td>
<td>312.00</td>
<td>case</td>
<td>24/case</td>
</tr>
<tr>
<td>301</td>
<td>NKL</td>
<td>running shoes</td>
<td>97.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>301</td>
<td>HRO</td>
<td>running shoes</td>
<td>42.50</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>301</td>
<td>SHM</td>
<td>running shoes</td>
<td>102.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>301</td>
<td>PRC</td>
<td>running shoes</td>
<td>75.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>301</td>
<td>KAR</td>
<td>running shoes</td>
<td>87.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>301</td>
<td>ANZ</td>
<td>running shoes</td>
<td>95.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>302</td>
<td>HRO</td>
<td>ice pack</td>
<td>4.50</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>302</td>
<td>KAR</td>
<td>ice pack</td>
<td>5.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>303</td>
<td>PRC</td>
<td>socks</td>
<td>48.00</td>
<td>box</td>
<td>24 pairs/box</td>
</tr>
<tr>
<td>303</td>
<td>KAR</td>
<td>socks</td>
<td>36.00</td>
<td>box</td>
<td>24 pair/box</td>
</tr>
<tr>
<td>304</td>
<td>ANZ</td>
<td>watch</td>
<td>170.00</td>
<td>box</td>
<td>10/box</td>
</tr>
<tr>
<td>304</td>
<td>HRO</td>
<td>watch</td>
<td>280.00</td>
<td>box</td>
<td>10/box</td>
</tr>
<tr>
<td>305</td>
<td>HRO</td>
<td>first-aid kit</td>
<td>48.00</td>
<td>case</td>
<td>4/case</td>
</tr>
<tr>
<td>306</td>
<td>PRC</td>
<td>tandem adapter</td>
<td>160.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>306</td>
<td>SHM</td>
<td>tandem adapter</td>
<td>190.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>307</td>
<td>PRC</td>
<td>infant jogger</td>
<td>250.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>308</td>
<td>PRC</td>
<td>twin jogger</td>
<td>280.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>309</td>
<td>HRO</td>
<td>ear drops</td>
<td>40.00</td>
<td>case</td>
<td>20/case</td>
</tr>
<tr>
<td>309</td>
<td>SHM</td>
<td>ear drops</td>
<td>40.00</td>
<td>case</td>
<td>20/case</td>
</tr>
<tr>
<td>310</td>
<td>SHM</td>
<td>kick board</td>
<td>80.00</td>
<td>case</td>
<td>10/case</td>
</tr>
<tr>
<td>310</td>
<td>ANZ</td>
<td>kick board</td>
<td>89.00</td>
<td>case</td>
<td>12/case</td>
</tr>
<tr>
<td>311</td>
<td>SHM</td>
<td>water gloves</td>
<td>48.00</td>
<td>box</td>
<td>4 pairs/box</td>
</tr>
<tr>
<td>312</td>
<td>SHM</td>
<td>racer goggles</td>
<td>96.00</td>
<td>box</td>
<td>12/box</td>
</tr>
<tr>
<td>312</td>
<td>HRO</td>
<td>racer goggles</td>
<td>72.00</td>
<td>box</td>
<td>12/box</td>
</tr>
<tr>
<td>313</td>
<td>SHM</td>
<td>swim cap</td>
<td>72.00</td>
<td>box</td>
<td>12/box</td>
</tr>
<tr>
<td>313</td>
<td>ANZ</td>
<td>swim cap</td>
<td>60.00</td>
<td>box</td>
<td>12/box</td>
</tr>
</tbody>
</table>
### catalog Table (1 of 7)

<table>
<thead>
<tr>
<th>catalog_num</th>
<th>stock_num</th>
<th>manu_code</th>
<th>cat_descr</th>
<th>cat_picture</th>
<th>cat_advert</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001</td>
<td>1</td>
<td>HRO8</td>
<td>Brown leather. Specify first baseman’s or infield/outfield style. Specify right- or left-handed.</td>
<td>&lt;BYTE value&gt;</td>
<td>Your First Season’s Baseball Glove</td>
</tr>
<tr>
<td>10002</td>
<td>1</td>
<td>HSK</td>
<td>Babe Ruth signature glove. Black leather. Infield/outfield style. Specify right- or left-handed.</td>
<td>&lt;BYTE value&gt;</td>
<td>All-Leather, Hand-Stitched, Deep-Pockets, Sturdy Webbing that Won’t Let Go</td>
</tr>
<tr>
<td>10003</td>
<td>1</td>
<td>SMT</td>
<td>Catcher’s mitt. Brown leather. Specify right- or left-handed.</td>
<td>&lt;BYTE value&gt;</td>
<td>A Sturdy Catcher’s Mitt With the Perfect Pocket</td>
</tr>
<tr>
<td>10004</td>
<td>2</td>
<td>HRO</td>
<td>Jackie Robinson signature glove. Highest Professional quality, used by National League.</td>
<td>&lt;BYTE value&gt;</td>
<td>High-Technology Design Expands the Sweet Spot</td>
</tr>
<tr>
<td>10005</td>
<td>3</td>
<td>HSK</td>
<td>Pro-style wood. Available in sizes: 31, 32, 33, 34, 35.</td>
<td>&lt;BYTE value&gt;</td>
<td>Durable Aluminum for High School and Collegiate Athletes</td>
</tr>
<tr>
<td>10006</td>
<td>3</td>
<td>SHM</td>
<td>Aluminum. Blue with black tape. 31”, 20 oz or 22 oz; 32”, 21 oz or 23 oz; 33”, 22 oz or 24 oz;</td>
<td>&lt;BYTE value&gt;</td>
<td>Quality Pigskin with Norm Van Brocklin Signature</td>
</tr>
<tr>
<td>10007</td>
<td>4</td>
<td>HSK</td>
<td>Norm Van Brocklin signature style.</td>
<td>&lt;BYTE value&gt;</td>
<td>Professional Football for High School and Collegiate Competitions</td>
</tr>
<tr>
<td>10008</td>
<td>4</td>
<td>HRO</td>
<td>NFL-Style pigskin.</td>
<td>&lt;BYTE value&gt;</td>
<td>最高的质量足球,用于高中和大学比赛。</td>
</tr>
<tr>
<td>10009</td>
<td>5</td>
<td>NRG</td>
<td>Graphite frame. Synthetic strings.</td>
<td>&lt;BYTE value&gt;</td>
<td>最宽的球体增强您的自然能力通过提供更多的力量。</td>
</tr>
<tr>
<td>10010</td>
<td>5</td>
<td>SMT</td>
<td>Aluminum frame. Synthetic strings.</td>
<td>&lt;BYTE value&gt;</td>
<td>肮脏网球和日光或人工光的易见性。</td>
</tr>
<tr>
<td>10011</td>
<td>5</td>
<td>ANZ</td>
<td>Wood frame, cat-gut strings.</td>
<td>&lt;BYTE value&gt;</td>
<td>木框网球与猫毛线。</td>
</tr>
<tr>
<td>10012</td>
<td>6</td>
<td>SMT</td>
<td>Soft yellow color for easy visibility in sunlight or artificial light.</td>
<td>&lt;BYTE value&gt;</td>
<td>High-Visibility Tennis, Day or Night</td>
</tr>
<tr>
<td>10013</td>
<td>6</td>
<td>ANZ</td>
<td>Pro-core. Available in neon yellow, green, and pink.</td>
<td>&lt;BYTE value&gt;</td>
<td>Durability Construction Coupled with the Brightest Colors Available</td>
</tr>
<tr>
<td>10014</td>
<td>7</td>
<td>HRO</td>
<td>Indoor. Classic NBA style. Brown leather.</td>
<td>&lt;BYTE value&gt;</td>
<td>Long-Life Basketballs for Indoor Gymnasiums</td>
</tr>
<tr>
<td>10015</td>
<td>8</td>
<td>ANZ</td>
<td>Indoor. Finest leather. Professional quality.</td>
<td>&lt;BYTE value&gt;</td>
<td>Professional Volleyballs for Indoor Competitions</td>
</tr>
<tr>
<td>10016</td>
<td>9</td>
<td>ANZ</td>
<td>Steel eyelets. Nylon cording. Double-stitched. Sanctioned by the National Athletic Congress</td>
<td>&lt;BYTE value&gt;</td>
<td>Sanctioned Volleyball Netting for Indoor Professional and Collegiate Competition</td>
</tr>
<tr>
<td>catalog_num</td>
<td>stock_num</td>
<td>manu_code</td>
<td>cat_descr</td>
<td>cat_picture</td>
<td>cat_advert</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>-----------</td>
<td>----------------------------------------------------------------------------------------------</td>
<td>-------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>10017</td>
<td>101</td>
<td>PRC</td>
<td>Reinforced, hand-finished tubular. Polyurethane belted. Effective against punctures. Mixed tread for super wear and road grip.</td>
<td>&lt;BYTE value&gt;</td>
<td>Ultimate in Puncture Protection, Tires Designed for In-City Riding</td>
</tr>
<tr>
<td>10018</td>
<td>101</td>
<td>SHM</td>
<td>Durable nylon casing with butyl tube for superior air retention. Center-ribbed tread with herringbone side. Coated sidewalls resist abrasion.</td>
<td>&lt;BYTE value&gt;</td>
<td>The Perfect Tire for Club Rides or Training</td>
</tr>
<tr>
<td>10019</td>
<td>102</td>
<td>SHM</td>
<td>Thrust bearing and coated pivot washer/spring sleeve for smooth action. Slotted levers with soft gum hoods. Two-tone paint treatment. Set includes calipers, levers, and cables.</td>
<td>&lt;BYTE value&gt;</td>
<td>Thrust-Bearing and Spring-Sleeve Brake Set Guarantees Smooth Action</td>
</tr>
<tr>
<td>10020</td>
<td>102</td>
<td>PRC</td>
<td>Computer-aided design with low-profile pads. Cold-forged alloy calipers and beefy caliper bushing; Aero levers. Set includes calipers, levers, and cables.</td>
<td>&lt;BYTE value&gt;</td>
<td>Computer Design Delivers Rigid Yet Vibration-Free Brakes</td>
</tr>
<tr>
<td>10021</td>
<td>103</td>
<td>PRC</td>
<td>Compact leading-action design enhances shifting. Deep cage for super-small granny gears. Extra strong construction to resist off-road abuse.</td>
<td>&lt;BYTE value&gt;</td>
<td>Climb Any Mountain: ProCycle's Front Derailleur Adds Finesse to Your ATB</td>
</tr>
<tr>
<td>10022</td>
<td>104</td>
<td>PRC</td>
<td>Floating trapezoid geometry with extra thick parallelogram arms. 100-tooth capacity. Optimum alignment with any freewheel.</td>
<td>&lt;BYTE value&gt;</td>
<td>Computer-Aided Design Engineers 100-Tooth Capacity Into ProCycle's Rear Derailleur</td>
</tr>
<tr>
<td>10023</td>
<td>105</td>
<td>PRC</td>
<td>Front wheels laced with 15g spokes in a 3-cross pattern. Rear wheels laced with 14g spokes in a 3-cross pattern.</td>
<td>&lt;BYTE value&gt;</td>
<td>Durable Training Wheels That Hold Tnie Under Toughest Conditions</td>
</tr>
<tr>
<td>10024</td>
<td>105</td>
<td>SHM</td>
<td>Polished alloy. Sealed-bearing, quick-release hubs. Double-butted. Front wheels are laced 15g/2-cross. Rear wheels are laced 15g/3-cross.</td>
<td>&lt;BYTE value&gt;</td>
<td>Extra Lightweight Wheels for Training or High-Performance Touring</td>
</tr>
<tr>
<td>10025</td>
<td>106</td>
<td>PRC</td>
<td>Hard anodized alloy with pearl finish. 6mm hex bolt hardware. Available in lengths of 90-140mm in 10mm increments.</td>
<td>&lt;BYTE value&gt;</td>
<td>ProCycle Stem with Pearl Finish</td>
</tr>
<tr>
<td>10026</td>
<td>107</td>
<td>PRC</td>
<td>Available in three styles: Mens racing; Mens touring; and Women's. Anatomical gel construction with lycra cover. Black or black/hot pink.</td>
<td>&lt;BYTE value&gt;</td>
<td>The Ultimate In Riding Comfort, Lightweight With Anatomical Support</td>
</tr>
<tr>
<td>catalog_num</td>
<td>stock_num</td>
<td>manu_code</td>
<td>cat_descr</td>
<td>cat_picture</td>
<td>cat_advert</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------</td>
<td>-------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>10027</td>
<td>108</td>
<td>SHM</td>
<td>Double or triple crankset with choice of chainrings. For double crankset, chainrings from 38-54 teeth. For triple crankset, chainrings from 24-48 teeth.</td>
<td>&lt;BYTE value&gt;</td>
<td>Customize Your Mountain Bike With Extra-Durable Crankset</td>
</tr>
<tr>
<td>10028</td>
<td>109</td>
<td>PRC</td>
<td>Steel toe clips with nylon strap. Extra wide at buckle to reduce pressure.</td>
<td>&lt;BYTE value&gt;</td>
<td>Classic Toeclip Improved To Prevent Soreness At Clip Buckle</td>
</tr>
<tr>
<td>10029</td>
<td>109</td>
<td>SHM</td>
<td>Ingenious new design combines button on sole of shoe with slot on a pedal plate to give riders new options in riding efficiency. Choose full or partial locking. Four plates mean both top and bottom of pedals are slotted—no fishing around when you want to engage full power. Fast unlocking ensures safety when maneuverability is paramount.</td>
<td>&lt;BYTE value&gt;</td>
<td>Ingenious Pedal/Clip Design Delivers Maximum Power And Fast Unlocking</td>
</tr>
<tr>
<td>10030</td>
<td>110</td>
<td>PRC</td>
<td>Super-lightweight. Meets both ANZI and Snell standards for impact protection. 7.5 oz. Quick-release shadow buckle.</td>
<td>&lt;BYTE value&gt;</td>
<td>Feather-Light, Quick-Release, Maximum Protection Helmet</td>
</tr>
<tr>
<td>10031</td>
<td>110</td>
<td>ANZ</td>
<td>No buckle so no plastic touches your chin. Meets both ANZI and Snell standards for impact protection. 7.5 oz. Lycra cover.</td>
<td>&lt;BYTE value&gt;</td>
<td>Minimum Chin Contact, Feather-Light, Maximum Protection Helmet</td>
</tr>
<tr>
<td>10032</td>
<td>110</td>
<td>SHM</td>
<td>Dense outer layer combines with softer inner layer to eliminate the mesh cover, no snagging on brush. Meets both ANZI and Snell standards for impact protection. 8.0 oz.</td>
<td>&lt;BYTE value&gt;</td>
<td>Mountain Bike Helmet: Smooth Cover Eliminates the Worry of Brush Snags But Delivers Maximum Protection</td>
</tr>
<tr>
<td>10033</td>
<td>110</td>
<td>HRO</td>
<td>Newest ultralight helmet uses plastic shell. Largest ventilation channels of any helmet on the market. 8.5 oz.</td>
<td>&lt;BYTE value&gt;</td>
<td>Lightweight Plastic with Vents Assures Cool Comfort Without Sacrificing Protection</td>
</tr>
<tr>
<td>10034</td>
<td>110</td>
<td>HSK</td>
<td>Aerodynamic (teardrop) helmet covered with anti-drag fabric. Credited with shaving 2 seconds/mile from winner’s time in Tour de France time-trial. 7.5 oz.</td>
<td>&lt;BYTE value&gt;</td>
<td>Teardrop Design Used By Yellow Jerseys, You Can Time the Difference</td>
</tr>
<tr>
<td>10035</td>
<td>111</td>
<td>SHM</td>
<td>Light-action shifting 10 speed. Designed for the city commuter with shock-absorbing front fork and drilled eyelets for carrying racks or bicycle trailers. Internal wiring for generator lights. 33 lbs.</td>
<td>&lt;BYTE value&gt;</td>
<td>Fully Equipped Bicycle Designed for the Serious Commuter Who Mixes Business With Pleasure</td>
</tr>
</tbody>
</table>
### Catalog Table (4 of 7)

<table>
<thead>
<tr>
<th>catalog_num</th>
<th>stock_num</th>
<th>manu_code</th>
<th>cat_descr</th>
<th>cat_picture</th>
<th>cat_advert</th>
</tr>
</thead>
<tbody>
<tr>
<td>10036</td>
<td>112</td>
<td>SHM</td>
<td>Created for the beginner enthusiast. Ideal for club rides and light touring. Sophisticated triple-butted frame construction. Precise index shifting. 28 lbs.</td>
<td>&lt;BYTE value&gt;</td>
<td>We Selected the Ideal Combination of Touring Bike Equipment, Then Turned It Into This Package Deal: High-Performance on the Roads, Maximum Pleasure Everywhere</td>
</tr>
<tr>
<td>10038</td>
<td>114</td>
<td>PRC</td>
<td>Padded leather palm and stretch mesh merged with terry back. Available in tan, black, and cream. Sizes S, M, L, XL.</td>
<td>&lt;BYTE value&gt;</td>
<td>Riding Gloves For Comfort and Protection</td>
</tr>
<tr>
<td>10039</td>
<td>201</td>
<td>NKL</td>
<td>Designed for comfort and stability. Available in white &amp; blue or white &amp; brown. Specify size.</td>
<td>&lt;BYTE value&gt;</td>
<td>Full-Comfort, Long-Wearing Golf Shoes for Men and Women</td>
</tr>
<tr>
<td>10040</td>
<td>201</td>
<td>ANZ</td>
<td>Guaranteed waterproof. Full leather upper. Available in white, bone, brown, green, and blue. Specify size.</td>
<td>&lt;BYTE value&gt;</td>
<td>Waterproof Protection Ensures Maximum Comfort and Durability In All Climates</td>
</tr>
<tr>
<td>10041</td>
<td>201</td>
<td>KAR</td>
<td>Leather and leather mesh for maximum ventilation. Waterproof lining to keep feet dry. Available in white &amp; gray or white &amp; ivory. Specify size.</td>
<td>&lt;BYTE value&gt;</td>
<td>Kansten's Top Quality Shoe Combines Leather and Leather Mesh</td>
</tr>
<tr>
<td>10042</td>
<td>202</td>
<td>NKL</td>
<td>Complete starter set utilizes gold shafts. Balanced for power.</td>
<td>&lt;BYTE value&gt;</td>
<td>Starter Set of Woods, Ideal for High School and Collegiate Classes</td>
</tr>
<tr>
<td>10043</td>
<td>202</td>
<td>KAR</td>
<td>Full set of woods designed for precision control and power performance.</td>
<td>&lt;BYTE value&gt;</td>
<td>High-Quality Woods Appropriate for High School Competitions or Serious Amateurs</td>
</tr>
<tr>
<td>10044</td>
<td>203</td>
<td>NKL</td>
<td>Set of eight irons includes 3 through 9 irons and pitching wedge. Originally priced at $489.00.</td>
<td>&lt;BYTE value&gt;</td>
<td>Set of Irons Available From Factory at Tremendous Savings: Discontinued Line.</td>
</tr>
<tr>
<td>10045</td>
<td>204</td>
<td>KAR</td>
<td>Ideally balanced for optimum control. Nylon-covered shaft.</td>
<td>&lt;BYTE value&gt;</td>
<td>High-Quality Beginning Set of Irons Appropriate for High School Competitions</td>
</tr>
<tr>
<td>10046</td>
<td>205</td>
<td>NKL</td>
<td>Fluorescent yellow.</td>
<td>&lt;BYTE value&gt;</td>
<td>Long Drive Golf Balls: Fluorescent Yellow</td>
</tr>
<tr>
<td>10047</td>
<td>205</td>
<td>ANZ</td>
<td>White only.</td>
<td>&lt;BYTE value&gt;</td>
<td>Long Drive Golf Balls: White</td>
</tr>
<tr>
<td>10048</td>
<td>205</td>
<td>HRO</td>
<td>Combination fluorescent yellow and standard white.</td>
<td>&lt;BYTE value&gt;</td>
<td>HiFlier Golf Balls: Case Includes Fluorescent Yellow and Standard White</td>
</tr>
</tbody>
</table>
### Map of the Demonstration Database

**The Demonstration Database and Application**

**A-25**

#### catalog Table (5 of 7)

<table>
<thead>
<tr>
<th>catalog_num</th>
<th>stock_num</th>
<th>manu_code</th>
<th>cat_descr</th>
<th>cat_picture</th>
<th>cat_advert</th>
</tr>
</thead>
<tbody>
<tr>
<td>10049</td>
<td>301</td>
<td>NKL</td>
<td>Super shock-absorbing gel pads disperse vertical energy into a horizontal plane for extraordinary cushioned comfort. Great motion control. Mens only. Specify size.</td>
<td>&lt;BYTE value&gt;</td>
<td>Maximum Protection For High-Mileage Runners</td>
</tr>
<tr>
<td>10050</td>
<td>301</td>
<td>HRO</td>
<td>Engineered for serious training with exceptional stability. Fabulous shock absorption. Great durability. Specify mens/womens, size.</td>
<td>&lt;BYTE value&gt;</td>
<td>Pronators and Supinators Take Heart: A Serious Training Shoe For Runners Who Need Motion Control</td>
</tr>
<tr>
<td>10051</td>
<td>301</td>
<td>SHM</td>
<td>For runners who log heavy miles and need a durable, supportive, stable platform. Mesh/synthetic upper gives excellent moisture dissipation. Stability system uses rear antipronation platform and forefoot control plate for extended protection during high-intensity training. Specify mens/womens, size.</td>
<td>&lt;BYTE value&gt;</td>
<td>The Training Shoe Engineered for Marathoners and Ultra-Distance Runners</td>
</tr>
<tr>
<td>10052</td>
<td>301</td>
<td>PRC</td>
<td>Supportive, stable racing flat. Plenty of forefoot cushioning with added motion control. Womens only. D widths available. Specify size.</td>
<td>&lt;BYTE value&gt;</td>
<td>A Woman's Racing Flat That Combines Extra Forefoot Protection With a Slender Heel</td>
</tr>
<tr>
<td>10053</td>
<td>301</td>
<td>KAR</td>
<td>Anatomical last holds your foot firmly in place. Feather-weight cushioning delivers the responsiveness of a racing flat. Specify mens/womens, size.</td>
<td>&lt;BYTE value&gt;</td>
<td>Durable Training Flat That Can Carry You Through Marathon Miles</td>
</tr>
<tr>
<td>10054</td>
<td>301</td>
<td>ANZ</td>
<td>Cantilever sole provides shock absorption and energy rebound. Positive traction shoe with ample toe box. Ideal for runners who need a wide shoe. Available in mens and womens. Specify size.</td>
<td>&lt;BYTE value&gt;</td>
<td>Motion Control, Protection, and Extra Toebox Room</td>
</tr>
<tr>
<td>10055</td>
<td>302</td>
<td>KAR</td>
<td>Re-usable ice pack with velcro strap. For general use. Velcro strap allows easy application to arms or legs.</td>
<td>&lt;BYTE value&gt;</td>
<td>Finally, An Ice Pack for Achilles Injuries and Shin Splints that You Can Take to the Office</td>
</tr>
<tr>
<td>10056</td>
<td>303</td>
<td>PRC</td>
<td>Neon nylon. Perfect for running or aerobics. Indicate color: Fluorescent pink, yellow, green, and orange.</td>
<td>&lt;BYTE value&gt;</td>
<td>Knock Their Socks Off With YOUR Socks!</td>
</tr>
<tr>
<td>10057</td>
<td>303</td>
<td>KAR</td>
<td>100% nylon blend for optimal wicking and comfort. We've taken out the cotton to eliminate the risk of blisters and reduce the opportunity for infection. Specify mens or womens.</td>
<td>&lt;BYTE value&gt;</td>
<td>100% Nylon Blend Socks - No Cotton!</td>
</tr>
<tr>
<td>catalog_num</td>
<td>stock_num</td>
<td>manu_code</td>
<td>cat_descr</td>
<td>cat_picture</td>
<td>cat_advert</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>10058</td>
<td>304</td>
<td>ANZ</td>
<td>Provides time, date, dual display of lap/cumulative splits, 4-lap memory, 10hr countdown timer, event timer, alarm, hour chime, waterproof to 50m, velcro band.</td>
<td>&lt;BYTE value&gt;</td>
<td>Athletic Watch w/4-Lap Memory</td>
</tr>
<tr>
<td>10059</td>
<td>304</td>
<td>HRO</td>
<td>Split timer, waterproof to 50m. Indicate color: Hot pink, mint green, space black.</td>
<td>&lt;BYTE value&gt;</td>
<td>Waterproof Triathlete Watch In Competition Colors</td>
</tr>
<tr>
<td>10060</td>
<td>305</td>
<td>HRO</td>
<td>Contains ace bandage, anti-bacterial cream, alcohol cleansing pads, adhesive bandages of assorted sizes, and instant cold pack.</td>
<td>&lt;BYTE value&gt;</td>
<td>Comprehensive First-Aid Kit Essential for Team Practices, Team Traveling</td>
</tr>
<tr>
<td>10061</td>
<td>306</td>
<td>PRC</td>
<td>Converts a standard tandem bike into an adult/child bike. User-tested Assembly Instructions</td>
<td>&lt;BYTE value&gt;</td>
<td>Enjoy Bicycling With Your Child On a Tandem; Make Your Family Outing Safer</td>
</tr>
<tr>
<td>10062</td>
<td>306</td>
<td>SHM</td>
<td>Converts a standard tandem bike into an adult/child bike. Lightweight model.</td>
<td>&lt;BYTE value&gt;</td>
<td>Consider a Touring Vacation For the Entire Family: A Lightweight, Touring Tandem for Parent and Child</td>
</tr>
<tr>
<td>10063</td>
<td>307</td>
<td>PRC</td>
<td>Allows mom or dad to take the baby out, too. Fits children up to 21 pounds. Navy blue with black trim.</td>
<td>&lt;BYTE value&gt;</td>
<td>Infant Jogger Keeps A Running Family Together</td>
</tr>
<tr>
<td>10064</td>
<td>308</td>
<td>PRC</td>
<td>Allows mom or dad to take both children! Rated for children up to 18 pounds.</td>
<td>&lt;BYTE value&gt;</td>
<td>As Your Family Grows, Infant Jogger Grows With You</td>
</tr>
<tr>
<td>10065</td>
<td>309</td>
<td>HRO</td>
<td>Prevents swimmer’s ear.</td>
<td>&lt;BYTE value&gt;</td>
<td>Swimmers Can Prevent Ear Infection All Season Long</td>
</tr>
<tr>
<td>10066</td>
<td>309</td>
<td>SHM</td>
<td>Extra-gentle formula. Can be used every day for prevention or treatment of swimmer’s ear.</td>
<td>&lt;BYTE value&gt;</td>
<td>Swimmer’s Ear Drops Specially Formulated for Children</td>
</tr>
<tr>
<td>10067</td>
<td>310</td>
<td>SHM</td>
<td>Blue heavy-duty foam board with Shimara or team logo.</td>
<td>&lt;BYTE value&gt;</td>
<td>Exceptionally Durable, Compact Kickboard for Team Practice</td>
</tr>
<tr>
<td>10068</td>
<td>310</td>
<td>ANZ</td>
<td>White. Standard size.</td>
<td>&lt;BYTE value&gt;</td>
<td>High-Quality Kickboard</td>
</tr>
<tr>
<td>10069</td>
<td>311</td>
<td>SHM</td>
<td>Swim gloves. Webbing between fingers promotes strengthening of arms. Cannot be used in competition.</td>
<td>&lt;BYTE value&gt;</td>
<td>Hot Training Tool - Webbed Swim Gloves Build Arm Strength and Endurance</td>
</tr>
<tr>
<td>10071</td>
<td>312</td>
<td>HRO</td>
<td>Durable competition-style goggles. Available in blue, grey, or white.</td>
<td>&lt;BYTE value&gt;</td>
<td>Swim Goggles: Traditional Rounded Lens For Greater Comfort.</td>
</tr>
<tr>
<td>catalog_num</td>
<td>stock_num</td>
<td>manu_code</td>
<td>cat_descr</td>
<td>cat_picture</td>
<td>cat_advert</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>10072</td>
<td>313</td>
<td>SHM</td>
<td>Silicone swim cap. One size. Available in white, silver, or navy. Team Logo Imprinting Available</td>
<td>&lt;BYTE value&gt; Team Logo Silicone Swim Cap</td>
<td></td>
</tr>
<tr>
<td>10073</td>
<td>313</td>
<td>ANZ</td>
<td>Silicone swim cap. Squared-off top. One size. White.</td>
<td>&lt;BYTE value&gt; Durable Squared-off Silicone Swim Cap</td>
<td></td>
</tr>
<tr>
<td>10074</td>
<td>302</td>
<td>HRO</td>
<td>Re-usable ice pack. Store in the freezer for instant first-aid. Extra capacity to accommodate water and ice.</td>
<td>&lt;BYTE value&gt; Water Compartment Combines With Ice to Provide Optimal Orthopedic Treatment</td>
<td></td>
</tr>
<tr>
<td>customer_num</td>
<td>call_dtime</td>
<td>user_id</td>
<td>call_code</td>
<td>call_descr</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>---------------</td>
<td>---------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>1990-06-12 8:20</td>
<td>maryj</td>
<td>D</td>
<td>Order was received, but two of the cans of ANZ tennis balls within the case were empty</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>1990-07-07 10:24</td>
<td>richc</td>
<td>L</td>
<td>Order placed one month ago (6/7) not received.</td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>1990-07-01 15:00</td>
<td>richc</td>
<td>B</td>
<td>Bill does not reflect credit from previous order</td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>1990-07-10 14:05</td>
<td>maryj</td>
<td>O</td>
<td>Customer likes our merchandise. Requests that we stock more types of infant joggers. Will call back to place order.</td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>1990-07-31 14:30</td>
<td>maryj</td>
<td>I</td>
<td>Received Hero watches (item # 304) instead of ANZ watches</td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>1989-11-28 13:34</td>
<td>mannyn</td>
<td>I</td>
<td>Received plain white swim caps (313 ANZ) instead of navy with team logo (313 SHM)</td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>1989-12-21 11:24</td>
<td>mannyn</td>
<td>I</td>
<td>Second complaint from this customer! Received two cases right-handed outfielder gloves (1 HRO) instead of one case lefties.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cust_calls Table</th>
<th>res_dtime</th>
<th>res_descr</th>
</tr>
</thead>
<tbody>
<tr>
<td>106</td>
<td>1990-06-12 8:25</td>
<td>Authorized credit for two cans to customer, issued apology. Called ANZ buyer to report the QA problem.</td>
</tr>
<tr>
<td>110</td>
<td>1990-07-07 10:30</td>
<td>Checked with shipping (Ed Smith). Order sent yesterday - we were waiting for goods from ANZ. Next time will call with delay if necessary.</td>
</tr>
<tr>
<td>119</td>
<td>1990-07-02 8:21</td>
<td>Spoke with Jane Akant in Finance. She found the error and is sending new bill to customer</td>
</tr>
<tr>
<td>121</td>
<td>1990-07-10 14:06</td>
<td>Sent note to marketing group of interest in infant joggers</td>
</tr>
<tr>
<td>127</td>
<td>1989-11-28 16:47</td>
<td>Shipping found correct case in warehouse and express mailed it in time for swim meet.</td>
</tr>
<tr>
<td>116</td>
<td>1989-12-27 08:19</td>
<td>Memo to shipping (Ava Brown) to send case of left-handed gloves, pick up wrong case; memo to billing requesting 5% discount to placate customer due to second offense and lateness of resolution because of holiday</td>
</tr>
</tbody>
</table>
The customer Specification

database
stores
screen
{

CUSTOMERS
Customer Number: [f000 ]
    Company : [f001 ]
        First Name: [f002 ]  Last Name: [f003 ]
    Address : [f004 ]
        [f005 ]
    City : [f006 ]  State : [a0]  Zip : [f007 ]
Telephone : [f008 ]

}
end

tables
    customer
attributes
f000 = customer_num;
f001 = company, reverse;
f002 = fname, comments = "Please enter first name if available";
f003 = lname;
f004 = address1;
f005 = address2;
f006 = city;
a0 = state, upshift, autonext, include = ("CA", "OR", "NV", "WA");
    comments = "Legal states are CA, OR, NV, or WA";
f007 = zipcode, autonext;
f008 = phone, picture = "###-###-####XXXXXX";
end
The *orderform* Specification

database stores

screen

CUSTOMER INFORMATION:
Customer Number: [c1]  | Telephone: [c10]
Company: [c4]  | Last Name: [c3]
First Name: [c2] | Address: [c5]
| [c6]
City: [c7]  | State: [c8]  | Zip: [c9]

ORDER INFORMATION:
Order Number: [o11]  | Order Date: [o12]
Stock Number: [i13]  | Manufacturer: [i16]
[manu_name]
| Quantity: [i18]
Total Price: [i19]

SHIPPING INFORMATION:
Customer P.O.: [o20]
Ship Date: [o21]  | Date Paid: [o22]

end

tables customer orders
items manufact

attributes

c1 = *customer.customer_num = orders.customer_num;
c2 = fname,
  comments = "Please enter initial if available ";
c3 = lname;
c4 = company;
c5 = address1;
c6 = address2;
c7 = city;
c8 = state, upshift, autonext,
  include = ("CA","OR","NV","WA");
c9 = zipcode;
c10 = phone, picture = "###-####-####x#####";
o11 = *orders.order_num = items.order_num;
o12 = order_date,
  default = today;
i13 = items.stock_num;
i16 = items.manu_code , lookup manu_name = manufact.manu_name,
  joining *manufact.manu_code, upshift;
Sample PERFORM Form Specifications

i18 = quantity, include = (1 to 100);
i19 = total_price;
o20 = po_num;
o21 = ship_date;
o22 = paid_date;

instructions

customer master of orders;
orders master of items;

end
The sample Specification

database stores
screen
{
========================================================================
========================================================================
CUSTOMER INFORMATION:
Customer Number: [c1 ]
   Company: [c4 ]
   First Name: [c2 ]    Last Name: [c3 ]
   Address: [c5 ]
   [c6 ]
   City: [c7 ]    State: [c8 ]    Zip: [c9 ]
   Telephone: [c10 ]
========================================================================
========================================================================
}

customers customer items stock
orders manufact
attributes
  c1 = customer.customer_num
  c2 = fname,
      comments = "Please enter initial if available";
  c3 = lname;
  c4 = company, reverse;

SHIPPING INFORMATION:
Customer P.O.: [o20 ]    Ship Charge: [d1 ]
   Backlog: [a]    Total Order Amount: [d2 ]
   Ship Date: [o21 ]
   Date Paid: [o22 ]
   Instructions: [o23 ]
}
end
Sample PERFORM Form Specifications

c5 = address1;
c6 = address2;
c7 = city;
c8 = state, upshift, autonext,
   include = ("CA","OR","NV","WA"),
   default = "CA" ;
c9 = zipcode, autonext;
c10 = phone, picture = "###-####-####x####";
o11 = *orders.order_num = items.order_num;
o12 = order_date, default = today, format = "mm/dd/yyyy";
i13 = items.stock_num;
   = *stock.stock_num, noentry, noupdate, queryclear;
i16 = items.manu_code, lookup m17 = manufact.manu_name
   joining *manufact.manu_code, upshift, autonext;
   = *stock.manu_code, noentry, noupdate,
     upshift, autonext, queryclear;
s14 = stock.description, noentry, noupdate;
s16 = stock.unit_descrip, noentry, noupdate;
s15 = stock.unit_price, noentry, noupdate;
i18 = items.quantity, include = (1 to 50),
   comments = "Acceptable values are 1 through 50" ;
i19 = items.total_price;
o20 = po_num, required,
   comments = "If no P.O. Number enter name of caller" ;
a = backlog, autonext;
o21 = ship_date, default = today, format = "mm/dd/yyyy";
o22 = paid_date, format = "mm/dd/yyyy";
o23 = ship_instruct;
d1 = displayonly type money;
d2 = displayonly type money;

instructions

customer master of orders;
orders master of items;
composites <items.stock_num, items.manu_code>
   <stock.stock_num, stock.manu_code>

before editadd editupdate of orders
nextfield = o20

before editadd editupdate of items
nextfield = i13

after editadd editupdate of quantity
let i19 = i18 * s15
nextfield = o11

after add update query of items
if (total of i19) <= 100 then
  let d1 = 7.50
else
  let d1 = (total of i19) * .04
let d2 = (total of i19) + d1

after display of orders
let d1 = 0
let d2 = 0

end
Sample ACE Report Specifications

The clist1 Specification

{ File: clist1.ace - Customer List Specification 1}
database  
end
output  
left margin 2  
end
select  
customer_num,  
fname,  
lname,  
company,  
city,  
state,  
zipcode,  
phone  
from  
customer  
order by  
city  
end
format

first page header  
print column 32, "CUSTOMER LIST"  
print column 32, "-------------"  
skip 2 lines  
print "NUMBER",  
column 9, "NAME",  
column 32, "LOCATION",  
column 54, "ZIP",  
column 62, "PHONE"  
skip 1 line
page header  
print "NUMBER",  
column 9, "NAME",  
column 32, "LOCATION",  
column 54, "ZIP",  
column 62, "PHONE"  
skip 1 line
on every row  
print customer_num using "####",  
column 9, fname clipped, 1 space, lname clipped,  
column 32, city clipped, "\", state,  
column 54, zipcode,  
column 62, phone

on last row  
skip 1 line  
print "TOTAL NUMBER OF CUSTOMERS:",  
column 30, count using "#"
end
The **clist2 Specification**

```ace
{ File: clist2.ace - Customer List Specification 2 }

database
stores
end

define variable thisstate char(2)
end

input
    prompt for thisstate using
        "Enter state (use UPPER CASE) for which you wish a customer list: "
end

output
    left margin 0
end

select
    customer_num,
    fname,
    lname,
    company,
    city,
    state,
    zipcode,
    phone
from
    customer
where
    state matches $thisstate
order by
    zipcode,
    lname
end

format

    first page header
    print column 32, "CUSTOMER LIST"
    print column 32, "-------------"
    skip 2 lines
    print "Listings for the State of ", thisstate
    skip 2 lines
    print "NUMBER",
        column 9, "NAME",
    column 32, "LOCATION",
    column 54, "ZIP",
    column 62, "PHONE"
    skip 1 line
```
Sample ACE Report Specifications

```
page header
print "NUMBER",
    column 9, "NAME",
    column 32, "LOCATION",
    column 54, "ZIP",
    column 62, "PHONE"
skip 1 line

on every row
print customer_num using "####",
    column 9, fname clipped, 1 space, lname clipped,
    column 32, city clipped, ", " , state,
    column 54, zipcode,
    column 62, phone

on last row
skip 2 lines
print "Number of customers in ",thisstate, " is ", count using "<<<<&"
end
```
The mail1 Specification

[file mail1.ace

Mailing Label Specification - 1]

{Customized report to print mailing labels. The report will be sorted by zip code and last name. The use of PRINT commands in the FORMAT section allow specific columns to be printed on a line. Blank lines will appear where data is absent, and spaces appear next to city field as not clipped as in mail2.ace specification}

database stores end

select * from customer order by zipcode, lname end

format on every row
  print fname, lname
  print company
  print address1
  print address2
  print city, ", " , state, 2 spaces, zipcode
  skip 2 lines
end
Sample ACE Report Specifications

The mail2 Specification

```
{mail2.ace file
Mailing Label Specification - 2 }

{This improved report has an OUTPUT section added, and uses nested IF statements. }

database
stores end

output
top margin 0
bottom margin 0
left margin 0
page length 9
report to "labels"
end

select 
  fname, lname, 
  company, 
  address1, 
  address2, 
  city, state, zipcode 
from customer
order by zipcode, lname 
end

format 
on every row
  if (city is not null) and
    (state is not null) then
    begin
      print fname clipped, 1 space, lname
      print company
      print address1 
      if (address2 is not null) then
        print address2
      print city clipped, ",", state, 
        2 spaces, zipcode
      skip to top of page
    end
end
```
The mail3 Specification

[file mail3.ace

Mailing Label Specification - 3 ]

[This report prints 1-3 mailing labels across a page. It stores the labels in character strings (array1, array2, and array3) as it reads each row, and prints the labels when it has read the proper number of rows. At run time, you specify the number of labels (1-3) that you want ACE to print across the page.]

database stores

end
define

variable name char(75) {holds first and last names}
variable cstzp char(75) {holds city, state, and zip}
variable array1 char(80) {Array for name line}
variable array2 char(80) {Array for street line}
variable array3 char(80) {Array for city, state, and zipcode}
variable start smallint {start of current label in array}
variable finish smallint {end of current label in array}
variable l_size smallint {label width}
variable white smallint {spaces between each label}
variable count1 smallint {number of labels across page}
variable i smallint {label counter}
end

input
prompt for count1 using "Number of labels across page? [1-3] "
end

output
top margin 0
bottom margin 0
left margin 0
report to "labels.out"
end

select
* from
customer order by zip
end

format
first page header {Nothing is displayed in this control block. It just initializes variables that are used in the ON EVERY ROW control block.}

let i = 1 {Initialize label counter.}
let l_size = 72/count1 {Determine label width (allow eight spaces total between labels).}
let white = 8/count1 \{Divide the eight spaces between the number of labels across the page.\}

on every row
let name = fname clipped, 1 space, lname
let cstzp = city clipped, *
state, 
2 spaces, zipcode
let finish = (i * l_size) + white \{This section assigns names, addresses, and zip codes to arrays 1, 2, 3 until \}
let start = finish - l_size \{i = the number of labels across a page.\}
let array1[start, finish] = name 
let array2[start, finish] = address1
let array3[start, finish] = cstzp
if i = count1 then
begin
print array1 clipped \{Print the stored addresses.\}
print array2 clipped \{Use clipped to remove trailing spaces for quicker printing.\}
skip 1 line
let array1 = " " \{Reset the arrays to spaces.\}
let array2 = " "
let array3 = " "
let i = 1
end
else
let i = i + 1
on last row
if i > 1 then
begin
print array1 clipped \{Print the last set of addresses\}
print array2 clipped \{if there were any left.\}
print array3 clipped
end
end
The *ord1* Specification

```plaintext
[file  ord1.ace

Order Specification - 1 ]

database
stores
end

output
   report to "ordlist1"
end

select
   orders.order_num ordnum, 
   order_date, customer_num, 
   po_num, ship_date, ship_charge, 
   paid_date, 
   items.order_num, stock_num, manu_code, 
   quantity, total_price
from orders, items
where orders.order_num = items.order_num
   order by ordnum
end

format

before group of ordnum
   print "Order number: ", ordnum using "#####", 
      " for customer number: ", customer_num
       using "#####"
   print "Customer P.O. : ", po_num,
      " Date ordered: ", order_date
skip 1 line
   print "Stockno", column 20, 
   "Mfcode", column 28, "Qty", column 38, "Price"

on every row
   print stock_num using "###", column 20, 
      manu_code, column 28, quantity using "###", 
      column 38, total_price using "$$$,$$$.$$$

after group of ordnum
   print 5 spaces, "Total amount for the order: ", 
      group total of total_price using "$$,$$$,$$$.$$$
   skip 3 lines

end
```

The Demonstration Database and Application  A-41
The *ord2* Specification

```ace
(file  ord2.ace
Order Specification - 2 )
database
stores
end
output
  left margin 0
  report to "ordlist2"
end
select

  customer.customer_num custnum, fname,
  lname, company,

  orders.order_num ordnum, order_date,
  orders.customer_num, po_num, ship_date,
  ship_charge, paid_date,

  items.order_num, items.stock_num snum,
  items.manu_code, quantity, total_price,

  stock.stock_num, stock.manu_code,
  description, unit_price
from customer, orders, items, stock
where customer.customer_num = orders.customer_num
  and orders.order_num = items.order_num
  and items.stock_num = stock.stock_num
  and items.manu_code = stock.manu_code
order by custnum, ordnum, snum
end
format
before group of custnum
  print "Orders for: "*, fname clipped, 1 space,
  lname print 13 spaces, company
  skip 1 line
before group of ordnum
  print "Order number: "*, ordnum using "#####"
  print "Customer P.O. : " *, po_num,
  " Date ordered: "*, order_date
  skip 1 line
  print "Stockno", column 10, "Mfcode", column 18,
  "Description", column 38, "Qty", column 43,
  "Unit price", column 55, "Total for item"```
on every row
  print snum using "###", column 10, manu_code, column 18, description clipped, column 38,
  quantity using "###", column 43,
  unit_price using "$$$$.&&", column 55,
  total_price using "$$$,$$$,$$$$.&&"

after group of ordnum
  skip 1 line
  print 4 spaces,
    "Shipping charges for the order: ",
  ship_charge using "$$$$.&&"
  skip 1 line

  print 5 spaces, "Total amount for the order: ",
  ship_charge + group total of total_price using "$$$,$$$,$$$$.&&"
  skip 3 lines

after group of custnum
  skip 2 lines

end
The ord3 Specification

{file  ord3.ace
Order Specification - 3 }
database
stores
end
define
variable begin_date date
variable end_date date
end
input
prompt for begin_date using
"Enter beginning date for report: "
prompt for end_date using
"Enter ending date for report: "
end
output
left margin 0
report to "ordList3"
end
select
customer.customer_num, fname, lname, company,
orders.order_num ordnum, orders.customer_num,
order_date, month(order_date) months,
day(order_date) days, year(order_date) years,
items.order_num, quantity, total_price
from customer, orders, items
where customer.customer_num = orders.customer_num
and orders.order_num = items.order_num and
order_date between $begin_date and $end_date
order by years, months, days, company, ordnum
end
format
first page header
print column 10, "========================================================"
print column 10, " DAILY ORDER REPORT"
print column 10, "========================================================"
skip 1 line
print column 15, "FROM: ", begin_date
using "mm/dd/yy",
column 35, "TO: ", end_date
using "mm/dd/yy"
print column 15, "Report run date: ",
today using "mmm dd, yyyy"
skip 2 lines
print column 2, "ORDER DATE", column 15,
"COMPANY", column 35, "NAME",
column 57, "NUMBER", column 65, "AMOUNT"
before group of days
skip 2 lines

after group of ordnum
print column 2, order_date, column 15,
  company clipped, column 35, fname clipped,
  1 space, lname clipped, column 55,
  ordnum using "####", column 60,
  group total of total_price using "$$,$$$,$$$,&&"

after group of days
skip 1 line
print column 21, "Total amount ordered for the day: ",
  group total of total_price using "$$,$$$,$$$,&&"
skip 1 line
print column 15, "======================================================"

on last row
skip 1 line
print column 15, "======================================================"

skip 2 lines
print "Total Amount of orders: ", total of 
  total_price using "$$,$$$,$$$,&&"

page trailer
print column 28, page using "page <<<"

end
Sample ACE Report Specifications
Environment Variables

Various environment variables affect the functionality of your Informix products. You can set environment variables that identify your terminal, specify the location of your software, and define other parameters. The environment variables discussed in this chapter are grouped and listed alphabetically.

Some environment variables are required and others are optional. For example, you must set—or accept the default setting for—certain UNIX environment variables.

This chapter describes how to use the environment variables that apply to INFORMIX-SQL (I-SQL) and shows how to set them. It is divided into three main sections:

- Informix environment variables
  This section describes some standard Informix-defined environment variables that are used with I-SQL. Many of these variables are not for frequent use, but are included in case they are necessary for correct operation of I-SQL with server products. Others are for directly specifying numeric and date formatting and the locations of message files. The settings for some of these latter variables might take precedence over those for their Native Language Support (NLS) counterparts.

- NLS environment variables
  You must set some or all these variables to benefit from NLS. These might cause your product to behave differently than when their standard Informix counterparts
Where to Set Environment Variables

You can set Informix, NLS, and UNIX environment variables in the following ways:

- **At the system prompt on the command line**
  When you set an environment variable at the system prompt, you must reassign it the next time you log into the system.

- **In a special shell file, as follows:**
  .login or .cshrc for the C shell
  .profile for the Bourne shell or the Korn shell
  When you set an environment variable in your .login, .cshrc, or .profile file, it is assigned automatically every time you log into the system.

  **Caution:** Check that you do not inadvertently set an environment variable differently in your .login and .cshrc C shell files.

- **In an environment-configuration file**
  This is a common or private file where you can define all the environment variables that are used by Informix products. Using a configuration file reduces the number of environment variables that you must set at the command line or in a shell file.
An environment-configuration file can contain comment lines (preceded by #) and variable lines and their values (separated by blanks and tabs), as in the following example:

```
# This is an example of an environment-configuration file
#
# These are Informix-defined variable settings
#
DBDATE DMY4-
DBFORMAT *:.,:DM
DBLANG german
#
# These are NLS environment variable settings
#
LANG de
```

Use the ENVIIGNORE environment variable to later override one or more entries in this file. Use the following Informix chkenv utility to check the contents of an environment-configuration file, and return an error message if there is a bad environment-variable entry in the file or if the file is too large:

```
chkenv filename
```


The common (shared) environment-configuration file resides in $INFORMIXDIR/etc/informix.rc. The permission for this shared file must be set to 644. A private environment-configuration file must be stored in the user’s home directory as .informix and must be readable by the user.

**Note:** The first time you set an environment variable in a shell or configuration file, before you work with your Informix product, you should log out and then log back in, “source” the file (C shell), or use “.” to execute an environment-configuration file (Bourne or Korn shell). This allows the process to read your entry.

### How to Set Environment Variables

You can change default settings and add new ones by setting one or more of the environment variables recognized by your Informix product. If you are already using an Informix product, some or all the appropriate environment variables might already be set.
After one or more Informix products have been installed, enter the following command at the system prompt to view your current environment settings:

**BSD UNIX:**  
`env`

**UNIX System V:**  
`printenv`

Use standard UNIX commands to set environment variables. Depending on the type of shell you use, Figure B-1 shows how you set the fictional `ABCD` environment variable to `value`.

<table>
<thead>
<tr>
<th>C shell:</th>
<th><code>setenv ABCD value</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bourne shell</strong></td>
<td><code>ABCD=value</code></td>
</tr>
<tr>
<td><strong>or Korn shell:</strong></td>
<td><code>export ABCD</code></td>
</tr>
<tr>
<td><strong>Korn shell:</strong></td>
<td><code>export ABCD=value</code></td>
</tr>
</tbody>
</table>

*Figure B-1 Setting environment variables in different shells*

When Bourne-shell example settings are shown in this chapter, the Korn shell (a superset of the Bourne shell) is implied as well. Note that Korn-shell syntax allows for a shortcut, as shown in Figure B-1.

*Note: The environment variables are case sensitive.*

The following diagram shows how the syntax for setting an environment variable is represented throughout this chapter. These diagrams indicate the setting for the C shell; for the Bourne or Korn shell, follow the syntax in Figure B-1.

```
setenv ABCD value
```

For more information on how to read syntax diagrams, see the section “Syntax Conventions” in the Introduction to this manual.

To unset most of the environment variables shown in this chapter, enter the following command:

**C shell:**  
`unsetenv ABCD`

**Bourne shell**  
`unset ABCD`

**or Korn shell:**

---

B-4  Environment Variables
Default Environment Variable Settings

The following list describes the main default assumptions that are made about your environment when you use Informix products. Environment variables used to change the specific default values are shown in parentheses. Other product-specific default values are described where appropriate throughout this chapter.

- The program, compiler, or preprocessor, and any associated files and libraries of your product have been installed in the /usr/informix directory.
- The default INFORMIX-OnLine Dynamic Server (OnLine) or INFORMIX-SE database server for explicit or implicit connections is indicated by an entry in the $INFORMIXDIR/etc/sqlhosts file. (INFORMIXSERVER)
- The default directory for message files is $INFORMIXDIR/msg. (DBLANG unset and LANG unset)
- If you are using INFORMIX-SE, the target or current database is in the current directory. (DBPATH)
- Temporary files for INFORMIX-SE are stored in the /tmp directory. (DBTEMP)
- The default terminal-dependent keyboard and screen capabilities are defined in the termcap file in the $INFORMIXDIR/etc directory. (INFORMIXTERM)
- For products that use an editor, the default editor is the predominant editor for the operating system, usually vi. (DBEDIT)
- For products that have a print capability, the program that sends files to the printer is usually:
  * lp for UNIX System V
  * lpr for BSD and other UNIX systems
  (DBPRINT)
- The default format for money values is $000.00. (DBMONEY set to $.)
- The default format for dates is MM/DD/YYYY. (DBDATE set to MDY4/)
- The field separator for unloaded data files is the vertical bar (|=ASCII 124). (DBDELEMITER set to |)
Rules of Precedence

When an Informix product accesses an environment variable, normally the following rules of precedence apply:

1. The highest precedence goes to the value as defined in the environment (shell).
2. The second-highest precedence goes to the value as defined in the private environment-configuration file in the user’s home directory (~/.informix).
3. The next-highest precedence goes to the value as defined in the common environment-configuration file ($INFORMIXDIR/etc/informix.rc).
4. The lowest precedence goes to the default value.

If NLS is activated, there is an exception to these rules. The setting for any of the X/Open categories (LC_*) takes precedence over the setting for the LANG environment variable, no matter where they are set. For more information, see Appendix C.

The lists that follow show the most common environment variables used by Informix products. These environment variables and their uses are discussed in this chapter.

List of Environment Variables

The following tables contain alphabetical lists of the Informix, NLS, and UNIX environment variables that you can set for an Informix database server and I-SQL. These environment variables are described in this chapter on the pages listed in the last column.

<table>
<thead>
<tr>
<th>INFORMIX Environment Variable</th>
<th>Restrictions</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBANSIWARN</td>
<td></td>
<td>B-8</td>
</tr>
<tr>
<td>DBDATE</td>
<td></td>
<td>B-8</td>
</tr>
<tr>
<td>DBDELIMITER</td>
<td></td>
<td>B-11</td>
</tr>
<tr>
<td>DBEDIT</td>
<td></td>
<td>B-11</td>
</tr>
<tr>
<td>DBFORM</td>
<td></td>
<td>B-12</td>
</tr>
<tr>
<td>DBFORMAT</td>
<td></td>
<td>B-14</td>
</tr>
<tr>
<td>DBLANG</td>
<td></td>
<td>B-18</td>
</tr>
<tr>
<td>DBMONEY</td>
<td></td>
<td>B-21</td>
</tr>
<tr>
<td>DBPATH</td>
<td></td>
<td>B-23</td>
</tr>
<tr>
<td>DBPRINT</td>
<td></td>
<td>B-26</td>
</tr>
</tbody>
</table>
## List of Environment Variables

### INFORMIX Environment Variable

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Restrictions</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBREMOTECMD</td>
<td><strong>OnLine</strong> only</td>
<td>B-27</td>
</tr>
<tr>
<td>DBSPACETEMP</td>
<td><strong>OnLine</strong> only</td>
<td>B-28</td>
</tr>
<tr>
<td>DBTEMP</td>
<td><strong>SE</strong> only</td>
<td>B-29</td>
</tr>
<tr>
<td>DBUSPSPACE</td>
<td></td>
<td>B-29</td>
</tr>
<tr>
<td>ENVIGNORE</td>
<td></td>
<td>B-30</td>
</tr>
<tr>
<td>INFORMIXCONRETRY</td>
<td></td>
<td>B-30</td>
</tr>
<tr>
<td>INFORMIXCONTIME</td>
<td></td>
<td>B-31</td>
</tr>
<tr>
<td>INFORMIXDIR</td>
<td></td>
<td>B-32</td>
</tr>
<tr>
<td>INFORMIXSERVER</td>
<td></td>
<td>B-33</td>
</tr>
<tr>
<td>INFORMIXSHMBASE</td>
<td><strong>OnLine</strong> only</td>
<td>B-33</td>
</tr>
<tr>
<td>INFORMIXSTACKSIZE</td>
<td><strong>OnLine</strong> only</td>
<td>B-34</td>
</tr>
<tr>
<td>INFORMIXTERM</td>
<td></td>
<td>B-35</td>
</tr>
<tr>
<td>ONCONFIG</td>
<td><strong>OnLine</strong> only</td>
<td>B-36</td>
</tr>
<tr>
<td>PSORT_DBTEMP</td>
<td><strong>OnLine</strong> only</td>
<td>B-36</td>
</tr>
<tr>
<td>PSORT_NPROCS</td>
<td><strong>OnLine</strong> only</td>
<td>B-37</td>
</tr>
<tr>
<td>SLEXEC</td>
<td></td>
<td>B-38</td>
</tr>
<tr>
<td>SRLRM</td>
<td>must be unset</td>
<td>B-38</td>
</tr>
<tr>
<td>SRLRMDIR</td>
<td>must be unset</td>
<td>B-39</td>
</tr>
</tbody>
</table>

### NLS Environment Variable

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLLCHAR</td>
<td>C-19</td>
</tr>
<tr>
<td>DBAPICODE</td>
<td>C-23</td>
</tr>
<tr>
<td>DBNLS</td>
<td>C-17</td>
</tr>
<tr>
<td>LANG</td>
<td>C-25</td>
</tr>
<tr>
<td>LC_COLLATE</td>
<td>C-27</td>
</tr>
<tr>
<td>LC_CTYPE</td>
<td>C-29</td>
</tr>
<tr>
<td>LC_MONETARY</td>
<td>C-31</td>
</tr>
<tr>
<td>LC_NUMERIC</td>
<td>C-35</td>
</tr>
</tbody>
</table>

### UNIX Environment Variable

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATH</td>
<td>B-40</td>
</tr>
<tr>
<td>TERM</td>
<td>B-41</td>
</tr>
<tr>
<td>TERMCAP</td>
<td>B-41</td>
</tr>
<tr>
<td>TERMINFO</td>
<td>B-42</td>
</tr>
</tbody>
</table>
Informix Environment Variables

This section lists alphabetically the variables that you can set when you use Informix products. It includes references to NLS environment variables that are comparable to standard Informix environment variables, where appropriate.

DBANSIWARN

The DBANSIWARN environment variable indicates that you want to check for Informix extensions to ANSI standard syntax. Unlike most environment variables, you do not need to set DBANSIWARN to a value—setting it to any value or to no value, as follows, is sufficient:

```bash
setenv DBANSIWARN
```

Setting the DBANSIWARN environment variable for I-SQL is functionally equivalent to including the `-ansi` flag when invoking the utility from the command line. If you set DBANSIWARN before you run I-SQL, warnings are displayed on the screen within the SQL menu.

At run time, the DBANSIWARN environment variable causes the SQL Communication Area (SQLCA) variable `sqlca.sqlwarn.sqlwarn5` to be set to `W` when a statement that is not ANSI-compliant is executed. (For more information on SQLCA, refer to the Informix Guide to SQL: Reference, Version 4.1.)

Once you set DBANSIWARN, Informix extension checking is automatic until you log out or unset DBANSIWARN. To turn off Informix extension checking, unset the DBANSIWARN environment variable with the following command:

```bash
C shell: unsetenv DBANSIWARN
Bourne shell: unset DBANSIWARN
```

DBDATE

The DBDATE environment variable specifies the following formats for DATE values:

- The order of the month, day, and year in a date
- Whether the year should be printed with two digits (Y2) or four digits (Y4)
- The separator between the month, day, and year
Informix Environment Variables

Environment Variables  B-9

Informix Environment Variables

- , . are characters that can be used as separators in a date format.
/ is the default separator for date formats.
0 is a character that indicates no separator for the date format.
D, M are characters representing day or month, respectively, in date formats.
Y2, Y4 are characters that represent the year, and the number of digits in the year, in date formats.

The default setting for DBDATE is MDY4/, where M represents the month, D represents the day, Y4 represents a four-digit year, and the slash (/) is a separator (for example, 12/25/1993).

Other acceptable characters for the separator are a hyphen (-), a period (.), or a zero (0). (Use the zero to indicate no separator.)

The slash (/) appears if you attempt to use a character other than a hyphen, period, or zero as a separator, or if you do not include a separator character in the DBDATE definition.

You always must specify the separator character last. The number of digits you specify for the year must always follow the Y.

Date formatting specified in a USING clause or FORMAT attribute will override the formatting specified in DBDATE.

To make the date appear as mmddyy, set the DBDATE environment variable as follows:

C shell:    setenv DBDATE MDY20
Bourne shell: DBDATE=MDY20
            export DBDATE

MDY represents the order of month, day, and year; 2 indicates two digits for the year; and 0 specifies no separator. As a result, the date is displayed as 122593.
To make the date appear in European format (dd-mm-yyyy), set the DBDATE environment variable as follows:

**C shell:**
```
setenv DBDATE DMY4-
```

**Bourne shell:**
```
DBDATE=DMY4-
export DBDATE
```

DMY represents the order of day, month, and year; 4 indicates four digits for the year; and – specifies a hyphen separator. As a result, the date is displayed as 25–12–1993.

**Usage**

The LANG setting will specify the default value for DBDATE in an active NLS environment on HP and IBM systems. For example, a French NLS locale will establish a DBDATE default of DMY2. DMY2 formats the date March 1, 1994 as 1.3.94. On SUN systems LANG has no influence on the default for DBDATE.

DBDATE can only specify the display of month and day as numeric values, and does not support character month names and day names the way USING and FORMAT do. For this reason, changes in DBLANG and LANG do not affect the results of DBDATE on the display of DATE data.

The DBDATE variable, like DBFORMAT and DBMONEY, performs its role regardless of whether or not NLS is active (DBNLS set), and is not stored in database system tables upon database creation or considered during consistency checking. This is in contrast to LANG and the LC variables, which are only active when NLS is active, are stored with a database, and are checked for consistency.
DBDELMITNER

The DBDELMITNER environment variable specifies the field delimiter used by the dbexport utility in unloaded data files or with the LOAD and UNLOAD statements in I-SQL.

```
setenv DBDELMITNER 'delimiter'
```

`delimiter` is the field delimiter for unloaded data files.

Any single character except the following is allowed:

- Hexadecimal numbers (0 through 9, a through f, A through F)
- NEWLINE or CONTROL-J
- The backslash symbol (\)

The vertical bar (|=ASCII 124) is the default. To change the field delimiter to a plus (+), set the DBDELMITNER environment variable as follows:

```
C shell:       setenv DBDELMITNER '+'
Bourne shell:  DBDELMITNER='+'
            export DBDELMITNER
```

DBEDIT

The DBEDIT environment variable lets you name the text editor you want to use. If DBEDIT is set, the specified editor is called directly. If DBEDIT is not set, you are prompted to specify an editor as the default for the rest of the session.

```
setenv DBEDIT editor
```

`editor` is the name of the text editor you want to use.

For most systems, the default editor is vi. If you use another editor, be sure that it creates ASCII files. Some word processors in document mode introduce printer control characters that can interfere with operation of your Informix product.

To specify the EMACS text editor, set the DBEDIT environment variable as follows:

```
C shell:       setenv DBEDIT emacs
Bourne shell:  DBEDIT=emacs
            export DBEDIT
```
Informix Environment Variables

DBFORM

The DBFORM variable specifies the subdirectory of $INFORMIXDIR (or full pathname) in which the menu form files for the currently active language reside. (Note that $INFORMIXDIR means "the name of the directory referenced by the environment variable INFORMIXDIR.”) Menu form files provide a set of language-translated menus to replace the standard I-SQL menus. Menu form files have the suffix .frm. Menu form files are included in language supplements, which contain instructions specifying where the files should be installed and what DBFORM settings to specify.

```
setenv DBFORM pathname
```

`pathname` specifies the subdirectory of $INFORMIXDIR or the full pathname of the directory that contains the message files.

Usage

If DBFORM is not set, the default directory for menu form files is $INFORMIXDIR/forms. The files should be installed in a subdirectory under the forms subdirectory under $INFORMIXDIR. For example, French menu files could be installed in $INFORMIXDIR/forms/french or in $INFORMIXDIR/forms/fr.88591. The English language version will normally be installed in $INFORMIXDIR/forms or $INFORMIXDIR/forms/english. Non-English menu form files should not be installed in either of the locations where English files are normally found.
The following diagram illustrates the search method employed for locating message files for a particular language (where the value set in the DBFORM variable is indicated as $DBFORM$):

![Figure B-2 Directory search order, depending on $DBFORM$]

If both DBFORM and LANG are set, LANG is ignored in establishing search order.

![Figure B-3 Directory search order, depending on $LANG$]

If the LANG variable is set, and DBFORM is not, the search order changes to the following:

If both DBFORM and LANG are set, LANG is ignored in establishing search order.
Informix Environment Variables

To specify a menu form directory, follow these steps:

1. Use the `mkdir` command to create the appropriate subdirectory in `$INFORMIXDIR/forms`.
2. Set the owner and group of the subdirectory to `informix` and the access permission for this directory to 755.
3. Set the `DBFORM` environment variable to the new subdirectory, specifying only the subdirectory name and not the full pathname.
4. Copy the `.frm` files to the new menu form directory specified by `$INFORMIXDIR/forms/$DBFORM`. All files in the menu form directory should have the owner and group `informix` and access permission 644.
5. Run your program or otherwise continue working with your product.

For example, you can store the set of menu form files for the French language in `$INFORMIXDIR/forms/french` as follows:

```
setenv DBFORM french
```

**DBFORMAT**

The Informix-defined `DBFORMAT` environment variable specifies the default format in which the user inputs, displays, or prints values of the following data types:

- DECIMAL
- FLOAT
- SMALLFLOAT
- INTEGER
- SMALLINT
- MONEY

The default format specified in `DBFORMAT` affects how numeric and monetary values are:

- Displayed and input on the screen
- Printed
- Input to and output from ASCII files using `LOAD` and `UNLOAD`

`DBFORMAT` is used to specify the leading and trailing currency symbols (but not their default positions within a monetary value) and the decimal and thousands separators. Note that the decimal and thousands separators defined by `DBFORMAT` apply to both monetary and numeric data, and override the sets of separators established by `LC_MONETARY` and `LC_NUMERIC`.
For this reason, countries which use different formatting conventions for their monetary and numeric data should use LANG, LC_MONETARY, and LC_NUMERIC, and avoid DBFORMAT.

The setting in DBFORMAT will affect the following I-SQL keywords:

- USING expression in ACE
- FORMAT attribute in PERFORM
- PRINT statement in ACE
- LET statement in ACE, where a character string is receiving a monetary or numeric value

The syntax for setting DBFORMAT is as follows:

```
setenv DBFORMAT
```

- **front** is the leading currency symbol. The `front` value is optional. The null string, represented by "*", is allowed, and means that the leading currency symbol is not applicable.

- **thousands** is a list of one or more characters that determine the possible thousands separator. The user can use any of the specified characters as the thousands separator when inputting values. The values in the list are not separated by spaces or other characters. I-SQL uses the first value specified as the thousands separator when displaying the output value.

  You can specify any characters for the thousands separator except the following:

  - Digits
  - `<`, `>`, `|`, `?`, `!`, `,`, `[` ]

  If you specify the * character, I-SQL omits the thousands separator. The `thousands` value is optional. The default value is the * . A blank space can be the thousands separator and is used for this purpose in some locales.

  Note that in versions prior to 6.0, the colon symbol (:) was not allowed as a thousands separator. In version 6.0, the colon symbol is permitted, but must be preceded by a backslash (\) symbol, as in the specification `:\:\:DM`.

- **decimal** is a list of one or more characters that determine the possible decimal separators. The user can use any of the specified characters as the decimal separator when inputting values.
I-SQL uses the first value specified as the decimal separator when displaying the output value.

You can specify any characters except the following:
- Digits
- <, >, |, ?, !, =, [, ]
- Any characters specified for the thousands value

The decimal value is optional. Specification of an asterisk symbol in the decimal position will cause displayed values not to have a decimal separator.

Note that the colon symbol is permitted as a decimal separator, but must be preceded by a backslash (\) symbol in the DBFORMAT specification.

=back is a value that determines the trailing currency symbol. The back value is optional.

You must specify all three colons in the syntax. Enclosing the DBFORMAT specification in a pair of single quotes is suggested to prevent the shell from interpreting any of the characters.

Usage

The setting in DBFORMAT directly specifies the leading and trailing currency symbol, and the numeric and decimal separators. It adds the currency symbol and changes the separators displayed on the screen in a monetary or numeric field, and in the default format of a PRINT statement. For example, if DBFORMAT is set to:

* : : , : DM

the value 1234.56 will print or display as:

1 234, 56DM

DM stands for deutch marks. In the case of a screen form, values input by the user are expected to contain commas, not periods, as decimal separators if this DBFORMAT string has been specified.

The setting in DBFORMAT also affects the way format strings in the FORMAT attribute in ACE and the USING clause in PERFORM are interpreted. In these format strings, the period symbol (.) is not a literal character but a placeholder for the decimal separator specified by DBFORMAT. Likewise, the comma symbol (,) is a placeholder for the thousands separator specified by DBFORMAT. The dollar sign is a placeholder for the leading currency symbol.
The at-sign (@) symbol is a placeholder for the trailing currency symbol. The following table illustrates the results of different combinations of DBFORMAT setting and format string on the same value:

<table>
<thead>
<tr>
<th>Value</th>
<th>Format String</th>
<th>DBFORMAT Setting</th>
<th>Displayed Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234.56</td>
<td>$$#,###.##</td>
<td>$;,::</td>
<td>$1,234.56</td>
</tr>
<tr>
<td>1234.56</td>
<td>$$#,###.##</td>
<td>::;:DM</td>
<td>1,234.56DM</td>
</tr>
<tr>
<td>1234.56</td>
<td>#,###.##@@</td>
<td>$;,::</td>
<td>1,234.56</td>
</tr>
<tr>
<td>1234.56</td>
<td>#,###.##@@</td>
<td>::;:DM</td>
<td>1,234.56DM</td>
</tr>
</tbody>
</table>

Figure B-4 Illustration of the results of different DBFORMAT settings and format strings

When the user enters values, I-SQL behaves as follows:

- Disregards any currency symbols (leading or trailing) and thousands separators that the user enters.
- If a symbol appears that is defined as the decimal separator in DBFORMAT, it is interpreted in the input value as a decimal separator.

When I-SQL displays or prints values:

- The DBFORMAT-defined leading or trailing currency symbol is displayed for MONEY values.
- If a leading or trailing currency symbol is specified by the FORMAT attribute for non-MONEY data types, the symbol is displayed.
- The thousands separator does not display, unless it is included in a FORMAT attribute or USING operator.
- The decimal separator is displayed unless the decimal separator is defined as NULL (* ) in DBFORMAT or the data type is integer (INT or SMALLINT).
When money values are converted to character strings using the LET statement in ACE, both the default conversion and the conversion with a USING clause will insert the DBFORMAT-defined separators and currency symbol into the created strings.

### DBFORMAT

The DBFORMAT setting overrides settings in DBMONEY, LC_NUMERIC, and LC_MONETARY.

The DBFORMAT variable, like DBMONEY and DBDATE, performs its role regardless of whether or not NLS is active (DBNLS set to 1 or 2). This is in contrast to the LC variables, which are only active when NLS is active.

DBFORMAT, like DBMONEY, dictates both the numeric and monetary formats for data. In some countries, including Portugal and Italy, the correct use of decimal and thousands separators differs between numeric and monetary data. For such countries, LC_NUMERIC and LC_MONETARY provide for independently defined numeric and monetary formatting. This is in contrast to DBFORMAT and DBMONEY.

LC_NUMERIC and LC_MONETARY also can activate special logic for formatting, for purposes such as discarding the decimal portion of Italian lira. There is no fractional portion in Italian currency.

The use of LC_NUMERIC and LC_MONETARY rather than DBFORMAT is encouraged.

### DBLANG

The DBLANG variable specifies the subdirectory of $INFORMIXDIR (or the full pathname) in which the message files for the currently active language reside. (Note that $INFORMIXDIR means “the name of the directory referenced by the environment variable INFORMIXDIR.”) Message files provide a set of error messages for the engine and tools that have been translated into a national language. Message files have the suffix .iem.

A language supplement contains:

- Message files
- Instructions specifying where the files should be installed and what DBLANG settings to specify.
The syntax for setting DBLANG is as follows:

```sh
setenv DBLANG pathname
```

`pathname` specifies the subdirectory of `$INFORMIXDIR` or the full pathname of the directory that contains the message files.

**Usage**

If DBLANG is not set, the default directory for message files is `$INFORMIXDIR/msg`. The files should be installed in a subdirectory under the `msg` subdirectory under `$INFORMIXDIR`. For example, French message files could be installed in `$INFORMIXDIR/msg/french` or in `$INFORMIXDIR/msg/fr.88591`. The English language version will normally be installed in `$INFORMIXDIR/msg` or `$INFORMIXDIR/msg/english`. Non-English message files should not be installed in either of the locations where English files are normally found.

The following diagram illustrates the search method employed for locating message files for a particular language (where value of the variable DBLANG is designated as `$DBLANG`):

![Diagram of search order](image-url)

*Figure B-5 Directory search order, depending on $DBLANG*
Informix Environment Variables

If the LANG variable is set, and DBLANG is not, the search order changes to the following:

\[
\text{NLS search order:}
\]

- \$INFORMIXDIR/msg/$LANG/
- \$INFORMIXDIR/$LANG/
- \$INFORMIXDIR/msg/
- \$INFORMIXDIR/msg/english/

**Figure B-6** Directory search order, depending on $LANG

If both DBLANG and LANG are set, LANG is ignored in establishing search order.

To specify a message directory, follow these steps:

1. Use the `mkdir` command to create the appropriate subdirectory in `$INFORMIXDIR/msg`.
2. Set the owner and group of the subdirectory to `informix` and the access permission for this directory to 755.
3. Set the DBLANG environment variable to the new subdirectory, specifying only the subdirectory name and not the full pathname.
4. Copy the `.iem` files to the new message directory specified by `$INFORMIXDIR/msg/$DBLANG`. All files in the message directory should have the owner and group `informix` and access permission 644.
5. Run your program or otherwise continue working with your product.
For example, you can store the set of message files for the French language in `$INFORMIXDIR/msg/french` as follows:

```
setenv DBLANG french
```

**DBMONEY**

The DBMONEY environment variable specifies the display format for MONEY values.

The syntax is as follows:

```
setenv DBMONEY $ . , back
```

- `$` is the default symbol that precedes the MONEY value.
- `.` is the default decimal separator symbol that separates the integral from the fractional part of the MONEY value.
- `,` is an alternative decimal separator symbol that separates the integral from the fractional part of the MONEY value.
- `back` represents the optional trailing currency symbol that follows the MONEY value. The `back` symbol can be up to seven characters long and can contain any character except a comma or a period.
- `front` is the alternative leading currency symbol that precedes the MONEY value instead of `$`. The `front` symbol can be up to seven characters long and can contain any character except a comma or a period.

The default setting for DBMONEY is: `$`, where a dollar sign ($) precedes the MONEY value, a period (.) separates the integral from the fractional part of the MONEY value, and no `back` symbol appears. For example, `100.50` is formatted as `$100.50`. 

**NLS**

Informix tools do not support the XPG3 category LC_MESSAGES because the use of LC_MESSAGES requires storage of messages in system directories, which is less desirable than using the standard Informix message directory method.
Suppose you want to represent MONEY values in deutsche marks, which use DM as the currency symbol and a comma as the decimal separator. Set the DBMONEY environment variable as follows:

C shell:       setenv DM,  
Bourne shell:  DBMONEY=DM,  
                export DBMONEY

Here, DM is the currency symbol preceding the MONEY value, and a comma separates the integral from the fractional part of the MONEY value. As a result, the amount 100.50 is displayed as DM100, 50.

Whenever you supply a back symbol, you must also supply the front symbol and the decimal separator (a comma or period). Similarly, if you change the value separator from a period to a comma, you must also supply the front symbol.
Selecting the period as a decimal separator dictates the use of the comma as a thousands separator for monetary and numeric values. Selecting the comma as a decimal separator dictates the use of the period as the thousands separator.

**NLS**

DBMONEY represents syntax from older versions of the product set. It is recommended that you use the LC_MONETARY and LANG, or DBFORMAT, environment variables for specifying monetary format. DBMONEY has been retained only for compatibility with older versions.

The DBMONEY variable, like DBFORMAT and DBDATE, performs its role regardless of whether or not NLS is active (DBNLS set to 1 or 2). This is in contrast to LANG and the LC variables, which are only active when NLS is active.

The contents of a declared DBMONEY environment variable take precedence over the contents of the LC_MONETARY variable that can be set for NLS. However, the LC_MONETARY setting takes precedence over the default DBMONEY format.

DBMONEY, like DBFORMAT, dictates both the numeric and monetary formats for data. In some countries, including Portugal and Italy, the correct use of decimal and thousands separators differs between numeric and monetary data. For such countries, LC_NUMERIC and LC_MONETARY provide for independently defined numeric and monetary formatting. This is in contrast to DBMONEY and DBFORMAT.

LC_NUMERIC and LC_MONETARY also can activate special logic for formatting, for purposes such as discarding the decimal portion of Italian lira. There is no fractional portion in Italian currency.

The use of LC_NUMERIC and LC_MONETARY rather than DBFORMAT or DBMONEY is encouraged. The use of DBFORMAT rather than DBMONEY is encouraged, if LC_ variables cannot be used.

See the discussion of LC_MONETARY in Appendix C, “Native Language Support Within INFORMIX-SQL.”

**DBPATH**

Use DBPATH to identify the database servers that contain databases (if you are using the OnLine server), or the directories and/or database servers that contain databases (if you are using INFORMIX-SE). The DBPATH environment variable also specifies a list of directories (in addition to the current directory) in which I-SQL looks for command scripts (.sql files).
The CONNECT, DATABASE, START DATABASE, and DROP DATABASE statements use DBPATH to locate the database under two conditions:

- If the location of a database is not explicitly stated and if the database cannot be located in the default server or
- For INFORMIX-SE, the default directory

The CREATE DATABASE statement does not use DBPATH.

You can add a new DBPATH entry to existing entries. To do so, use the $ format described for the UNIX environment variable PATH, described on page B-40.

```
setenv DBPATH

pathname //servername / full_pathname //servername

pathname is a valid relative pathname for a directory in which .sql files are stored or in which INFORMIX-SE databases are stored.

full_pathname is a valid full pathname, starting with root, for a directory in which .sql files are stored or in which INFORMIX-SE databases are stored.

servername is the name of an OnLine or INFORMIX-SE database server on which databases are stored. You cannot reference database files with a servername.

DBPATH can contain up to 16 entries. Each entry (pathname, or servername, or servername and full_pathname) must be less than 128 characters long. In addition, the maximum length of DBPATH depends on the hardware platform on which you are setting DBPATH.

When you access a database using the CONNECT, DATABASE, START DATABASE, or DROP DATABASE statement, the search for the database is done first in the directory or database server specified in the statement. If no database server is specified, the default database server as set in the INFORMIXSERVER environment variable is used. For INFORMIX-SE, if no directory is specified in the statement, the default directory is searched for the database. (The default directory is the current working directory if the database server is on the local machine, or your login directory if the database server is on a remote machine.) If a directory is specified but is not a full path, the directory is considered to be relative to the default directory.
If the database is not located during the initial search, and if DBPATH is set, the database servers or directories in DBPATH are searched for the indicated database. The entries to DBPATH are considered in order.

DBPATH with INFORMIX-SQL

If you are using I-SQL and you use the Choose option of the SQL menu without having already selected a database, you see a list of all the .sql files in the directories listed in your DBPATH. Once you select a database, the DBPATH is not used to find the .sql files. For OnLine databases, only the .sql files in the current working directory are displayed; for INFORMIX-SE databases, the .sql files in the directory containing the selected database are displayed.

Searching Local Directories

Use a pathname without a database server name to have the database server search for databases or .sql scripts on your local machine. If you are using I-SQL with INFORMIX-SE, you can search for a database and .sql scripts; with OnLine, you can look only for .sql scripts.

For example, the following DBPATH setting causes I-SQL to search for the database files in your current directory and then in Joachim’s and Sonja’s directories on the local machine:

```
setenv DBPATH /usr/joachim:/usr/sonja
```

As shown in the previous example, if the pathname specifies a directory name but not a database server name, the directory is sought on the machine running the default database server as specified by the INFORMIXSERVER environment variable. (See page B-33.) For instance, with this example, if INFORMIXSERVER is set to quality, the DBPATH value is interpreted as follows, where the double slash precedes the database server name:

```
setenv DBPATH //quality/usr/joachim://quality/usr/sonja
```

Searching Networked Machines for Databases

If you are using more than one database server, you can set DBPATH to explicitly contain the database server and/or directory names that you want to be searched for databases. For example, if INFORMIXSERVER is set to quality but you also want to search the marketing database server for /usr/joachim, set DBPATH as follows:

```
setenv DBPATH //marketing/usr/joachim:/usr/sonja
```
Specifying a Servername

You can set DBPATH to contain only database server names. This allows you to locate only databases and not locate command files.

The **OnLine** or **SE** administrator must include each database server mentioned by DBPATH in the $INFORMIXDIR/etc/sqlhosts file. For information on communication-configuration files and dbservername, see the INFORMIX-OnLine Dynamic Server Administrator’s Guide, Version 6.0, or INFORMIX-SE Administrator’s Guide, Version 6.0.

For example, if INFORMIXSERVER is set to **quality**, you can search for an **OnLine** database first on the **quality** database server and then on the **marketing** database server by setting DBPATH as follows:

```
setenv DBPATH //marketing
```

If you are using **I-SQL** in this example, the names of all the databases on the **quality** and **marketing** database servers are displayed with the Select option of the Database menu.

For **INFORMIX-SE**, you can set DBPATH to contain just the database server names (and no directory names) if you want to locate only databases and not command scripts:

- If you specify a local **SE** database server, the current working directory is searched for databases.
- If you specify a remote **SE** database server, the search for databases is done in the login directory of the user on the machine where the database server is running.

DBPRINT

The DBPRINT environment variable specifies the print program that you want to use.

```
setenv DBPRINT program
```

`program` names any command, shell script, or UNIX utility that handles standard ASCII input.

The default program is as follows:

- For most BSD UNIX systems, the default program is `lpr`.
- For UNIX System V, the default program is usually `lp`.
Informix Environment Variables

Set the DBPRINT environment variable as follows to specify the `myprint` print program:

- **C shell:**
  ```
  setenv DBPRINT myprint
  ```

- **Bourne shell:**
  ```
  DBPRINT=myprint
  export DBPRINT
  ```

### DBREMOTECMD

You can set the DBREMOTECMD environment variable to override the default remote shell used when you perform remote tape operations with OnLine. Set it using either a simple command or the full pathname. If you use the full pathname, the database server searches your `PATH` for the specified command.

- **C shell:**
  ```
  setenv DBREMOTECMD command
  ```

- **Bourne shell:**
  ```
  DBREMOTECMD=command
  export DBREMOTECMD
  ```

**command** is the command to override the default remote shell.

**pathname** is the pathname to override the default remote shell.

Informix highly recommends the full pathname syntax on the interactive UNIX platform to avoid problems with like-named programs in other directories and possible confusion with the restricted shell (`/usr/bin/rsh`).

Set the DBREMOTECMD environment variable as follows for a simple command name:

- **C shell:**
  ```
  setenv DBREMOTECMD rcmd
  ```

- **Bourne shell:**
  ```
  DBREMOTECMD=rcmd
  export DBREMOTECMD
  ```

Set the DBREMOTECMD environment variable as follows to specify the full pathname:

- **C shell:**
  ```
  setenv DBREMOTECMD /usr/bin/remsh
  ```

- **Bourne shell:**
  ```
  DBREMOTECMD=/usr/bin/remsh
  export DBREMOTECMD
  ```

For more information on DBREMOTECMD, see the discussion in the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide, Version 6.0*, about using remote tape devices with OnLine for archives, restores, and logical log backups.
Informix Environment Variables

**DBSPACETEMP**

If you are using Online, you can set your DBSPACETEMP environment variable to specify the dbspace to be used for building all temporary tables and for holding temporary files used for sorting in Online. This spreads temporary space across any number of disks.

```
setenv DBSPACETEMP temp_dbspace
```

*punct* can be either colons or commas.

*temp_dbspace* is a list of valid existing temporary dbspaces.

You can set the DBSPACETEMP environment variable to override the default dbspaces used for temporary tables and sorting space specified in the DBSPACETEMP configuration parameter in the Online configuration file. For example, you might set DBSPACETEMP as follows:

**C shell:**
```
setenv DBSPACETEMP sorttmp1:sorttmp2:sorttmp3
```

**Bourne shell:**
```
DBSPACETEMP=sorttmp1:sorttmp2:sorttmp3
export DBSPACETEMP
```

Separate the dbspace entries with either colons or commas. The number of dbspaces is limited by the maximum size of the environment variable, as defined by the UNIX shell.

The default, if left unspecified, becomes the root dbspace. Online does not create the dbspace named by the environment variable if it does not exist.

For the creation of temporary tables, if neither DBSPACETEMP nor the DBSPACETEMP parameter in the onconfig file is set, Online creates the temporary tables in the dbspace where the database was created.

For sorting space, Online uses the following disk space for writing temporary information, in the following order:

1. The operating system directory or directories specified by the environment variable PSORT_DBTEMP, if set.
2. The dbspace or dbspaces specified by the environment variable DBSPACETEMP, if set.
3. The dbspace or dbspaces specified by the onconfig parameter DBSPACETEMP.
4. The operating system file space in /tmp.
**DBTEMP**

Set the DBTEMP environment variable to specify the full pathname of the directory into which you want INFORMIX-SE to place its temporary files. You need not set DBTEMP if the default, /tmp, is satisfactory.

```
setenv DBTEMP pathname
```

*pathname* is the full pathname of the directory for temporary files.

Set the DBTEMP environment variable as follows to specify the pathname `usr/magda/mytemp`:

**C shell:**
```
setenv DBTEMP usr/magda/mytemp
```

**Bourne shell:**
```
DBTEMP=usr/magda/mytemp
export DBTEMP
```

For the creation of temporary tables, if DBTEMP is not set, the temporary tables are created in the directory of the database (that is, the .dbs directory).

**DBUPSPACE**

The DBUPSPACE environment variable lets you specify and thus constrain the amount of system disk space that the UPDATE STATISTICS statement can use when trying to simultaneously construct multiple column distributions.

```
setenv DBUPSPACE value
```

*value* represents a disk space amount in kilobytes.

For example, if DBUPSPACE is set to 2500 (kilobytes) as follows,

**C shell:**
```
setenv DBUPSPACE 2500
```

**Bourne shell:**
```
DBUPSPACE=2500
export DBUPSPACE
```

then no more than 2.5 megabytes of disk space are to be used to accomplish sorting during the execution of an UPDATE STATISTICS statement. If a table requires 5 megabytes of disk space for sorting, then UPDATE STATISTICS accomplishes the task in two passes; the distributions for one half of the columns are constructed with each pass.

If you try to set DBUPSPACE to any value less than 1024 kilobytes, it is automatically set to 1024 kilobytes, but no error message is returned. If this value is not large enough to allow more than one distribution to be constructed at a time, at least one distribution is done, even if the amount of disk space required for the one is greater than specified in DBUPSPACE.
ENVIGNORE

Use the ENVIGNORE environment variable to deactivate specified environment variable entries in the common (shared) and private environment-configuration files.

```
setenv ENVIGNORE variable
```

*variable* is the list of environment variables that you want to deactivate.

For example, to ignore the DBPATH and DBMONEY entries in the environment-configuration files, specify the following command:

**C shell:**
```
setenv ENVIGNORE DBPATH:DBMONEY
```

**Bourne shell:**
```
ENVIGNORE=DBPATH:DBMONEY
export ENVIGNORE
```

The common environment-configuration file is stored in `$INFORMIXDIR/etc/informix.rc`. The private environment-configuration file is stored in the user's home directory as `.informix`. See “Where to Set Environment Variables” on page B-2 for information on creating or modifying an environment-configuration file.

*Note:* ENVIGNORE cannot be set in an environment-configuration file.

INFORMIXCONRETRY

The INFORMIXCONRETRY environment variable lets you specify the maximum number of additional connection attempts that should be made to each server by the client during the time limit specified by the INFORMIXCONTIME environment variable.

Set the INFORMIXCONRETRY environment variable as follows:

```
setenv INFORMIXCONRETRY value
```

*value* represents the number of connection attempts to each server.

For example, set INFORMIXCONRETRY to 3 additional connection attempts (after the initial attempt) as follows:

**C shell:**
```
setenv INFORMIXCONRETRY 3
```

**Bourne shell:**
```
INFORMIXCONRETRY=3
export INFORMIXCONRETRY
```
The default value for INFORMIXCONRETRY is one retry after the initial connection attempt. The INFORMIXCONTIME setting, described below, takes precedence over the INFORMIXCONRETRY setting.

**INFORMIXCONTIME**

The INFORMIXCONTIME environment variable lets you specify the minimum time limit, in seconds, for the SQL statement CONNECT to attempt to connect to a server before it returns an error.

You might encounter connection difficulties related to system or network load problems. For instance, if the server is busy establishing new SQL client threads, some of the clients might fail because the server can not issue a network function call fast enough. The INFORMIXCONTIME and INFORMIXCONRETRY environment variables let you configure your client-side connection capability to retry to connect instead of returning an error.

Set the INFORMIXCONTIME environment variable as follows:

```bash
setenv INFORMIXCONTIME value
```

*value* represents the minimum number of seconds spent in attempts to connect to each server.

For example, set INFORMIXCONTIME to 60 seconds as follows:

**C shell:**

```
setenv INFORMIXCONTIME 60
```

**Bourne shell:**

```
INFORMIXCONTIME=60
export INFORMIXCONTIME
```

If INFORMIXCONTIME is set to 60 and INFORMIXCONRETRY is set to 3, as shown in this appendix, attempts to connect to the server (after the initial attempt at 0 seconds) will be made at 20, 40, and 60 seconds, if necessary, before aborting. This 20-second interval is the result of INFORMIXCONTIME divided by INFORMIXCONRETRY.

If execution of the CONNECT statement involves searching DBPATH, the following rules apply:

- All appropriate servers in the DBPATH setting are accessed at least once, even though the INFORMIXCONTIME value might be exceeded. Thus, the CONNECT statement might take longer than the INFORMIXCONTIME time limit to return an error indicating connection failure or that the database was not found.
- The INFORMIXCONRETRY value specifies the number of additional connections that should be attempted for each server entry in DBPATH.
• The INFORMIXCONTIME value is initially divided among the number of server entries specified in DBPATH. Thus, if DBPATH contains numerous servers, you should increase the INFORMIXCONTIME value accordingly to allow for multiple connection attempts.

The default value for INFORMIXCONTIME is 15 seconds after the initial connection attempt. The INFORMIXCONTIME setting takes precedence over the INFORMIXCONRETRY setting; retry efforts could end after the INFORMIXCONTIME value has been exceeded, but before the INFORMIXCONRETRY value has been reached.

INFORMIXDIR

The INFORMIXDIR environment variable specifies the directory that contains the subdirectories in which your product files are installed. INFORMIXDIR must be set. If you have multiple versions of the OnLine or INFORMIX-SE database server, set INFORMIXDIR to the appropriate directory name for the version that you want to access. For information about when to set the INFORMIXDIR environment variable, see the UNIX Products Installation Guide, Version 6.0.

```
setenv INFORMIXDIR pathname
```

`pathname` is the directory path where the product files are installed.

Set the INFORMIXDIR environment variable to the following recommended installation directory:

**C shell:**
```
setenv INFORMIXDIR /usr/informix
```

**Bourne shell:**
```
INFORMIXDIR=/usr/informix
export INFORMIXDIR
```
INFORMIXSERVER

The INFORMIXSERVER environment variable specifies the default database server to which an explicit or implicit connection is made by I-SQL. The database server can be either OnLine or INFORMIX-SE and can be either local or remote.

```
setenv INFORMIXSERVER dbservername
```

$dbservername$ is the name of the default database server.

The value of INFORMIXSERVER must correspond to a valid $dbservername$ entry in the $INFORMIXDIR/etc/sqlhosts file on the machine running the application. It must be specified using lowercase characters and cannot exceed 18 characters for the OnLine database server and cannot exceed 10 characters for the INFORMIX-SE database server. For example, to specify the coral database server as the default for connection, enter the following command:

C shell: `setenv INFORMIXSERVER coral`

Bourne shell: `INFORMIXSERVER=coral`

```
export INFORMIXSERVER
```

INFORMIXSERVER specifies the database server to which an application connects if the CONNECT DEFAULT statement is executed. It also defines the database server to which an initial implicit connection is established if the first statement in an application is not a CONNECT statement.

**Note:** INFORMIXSERVER must be set even if the application or I-SQL does not use implicit or explicit default connections.

INFORMIXSHMBASE

The INFORMIXSHMBASE environment variable affects only client applications connected to an OnLine server using the IPC shared-memory (ipcshm) communication protocol.

You use INFORMIXSHMBASE to specify where shared-memory communication segments are attached to the client process so that client applications can avoid collisions with other memory segments used by the
Informix Environment Variables

application. If you do not set INFORMIXSHMBASE, the memory address of the communication segments defaults to an implementation-specific value such as 0x800000.

```
setenv INFORMIXSHMBASE value
```

`value` is used to calculate the memory address.

**OnLine** calculates the memory address where segments are attached by multiplying the value of INFORMIXSHMBASE by 1024. For example, to set the memory address to the value 0x800000, set the INFORMIXSHMBASE environment variable as follows:

**C shell:**
```
setenv INFORMIXSHMBASE 8192
```

**Bourne shell:**
```
INFORMIXSHMBASE=8192
export INFORMIXSHMBASE
```

Resetting INFORMIXSHMBASE requires a thorough understanding of the application’s use of memory. Normally you do not reset INFORMIXSHMBASE. For more information, see the **INFORMIX-OnLine Dynamic Server Administrator's Guide**, Version 6.0.

**INFORMIXSTACKSIZE**

The INFORMIXSTACKSIZE environment variable affects only client applications connected to an **OnLine** server.

The **OnLine** administrator can set INFORMIXSTACKSIZE to specify the stack size (in kilobytes) that **OnLine** will use for a particular client session. Use INFORMIXSTACKSIZE to override the value of the `onconfig` configuration parameter STACKSIZE for a particular application or user.

```
setenv INFORMIXSTACKSIZE value
```

`value` is the stack size for SQL client threads in kilobytes.

For example, to decrease the INFORMIXSTACKSIZE to 20 kilobytes, enter the following command:

**C shell:**
```
setenv INFORMIXSTACKSIZE 20
```

**Bourne shell:**
```
INFORMIXSTACKSIZE=20
export INFORMIXSTACKSIZE
```

**Note:** If INFORMIXSTACKSIZE is not set, the stack size is taken from the **OnLine** configuration parameter STACKSIZE, or it defaults to an implementation-specific value. The default stack size value for the primary thread for an SQL client is 32 kilobytes for nonrecursive database activity.
Warning: See the INFORMIX-OnLine Dynamic Server Administrator’s Guide, Version 6.0, for specific instructions for setting this value. If you incorrectly set the value of INFORMIXSTACKSIZE, it can cause OnLine to crash.

INFORMIXTERM

The INFORMIXTERM environment variable specifies whether I-SQL should use the information in the termcap file or the terminfo directory. INFORMIXTERM determines terminal-dependent keyboard and screen capabilities such as the operation of function keys, color and intensity attributes in screen displays, and the definition of window border and graphics characters.

If INFORMIXTERM is not set, the default setting is termcap. When I-SQL is installed on your system, a termcap file is placed in the etc subdirectory of $INFORMIXDIR. This file is a superset of an operating system termcap file.

You can use the termcap file supplied by Informix, the system termcap file, or a termcap file that you created yourself. You must set the TERMCP environment variable if you do not use the default termcap file.

The terminfo directory contains a file for each terminal name that has been defined. It is supported only on machines that provide full support for the UNIX System V terminfo library. For details, see the Version 6.0 on-line machine notes for your machine.

Set the INFORMIXTERM environment variable to terminfo as follows:

C shell:
```bash
setenv INFORMIXTERM terminfo
```

Bourne shell:
```bash
INFORMIXTERM=terminfo
export INFORMIXTERM
```

Set the INFORMIXTERM environment variable to termcap as follows:

C shell:
```bash
setenv INFORMIXTERM termcap
```

Bourne shell:
```bash
INFORMIXTERM=termcap
export INFORMIXTERM
```

Note: If INFORMIXTERM is set to termcap, you must set the UNIX environment variable TERMCP; if INFORMIXTERM is set to terminfo, you must set the UNIX environment variable TERMINFO.
ONCONFIG

The **OnLine** administrator can set the ONCONFIG environment variable (previously known as TBCONFIG), which contains the name of the UNIX file that holds the configuration parameters for **OnLine**. This file is read as input to either the disk-space or shared-memory initialization procedure.

```bash
setenv ONCONFIG filename
```

`filename` is the name of the file in `INFORMIXDIR/etc` that contains the **OnLine** configuration parameters.

If you are not the administrator of an **OnLine** server, you need to set ONCONFIG only if more than one **OnLine** system is initialized in your `INFORMIXDIR` directory, and you want to maintain multiple configuration files with different values. If you do not set ONCONFIG, the default is `onconfig`.

Each **OnLine** server has its own `onconfig` file that must be stored in the `INFORMIXDIR/etc` directory. You might prefer to name `onconfig` so it easily can be related to a specific **OnLine** database server. For example, when the desired filename is `onconfig3`, you can set the ONCONFIG environment variable as follows:

```bash
C shell:      setenv ONCONFIG onconfig3
Bourne shell: ONCONFIG=onconfig3
              export ONCONFIG
```

For more information, see the **INFORMIX-OnLine Dynamic Server Administrator’s Guide**, Version 6.0.

PSORT_DBTEMP

The PSORT_DBTEMP environment variable affects only client applications connected to an **OnLine** server.

PSORT_DBTEMP specifies a directory or directories where the **OnLine** server writes the temporary files it uses when performing a sort. See the DBSPACETEMP environment variable on page B-28 for more information on other places **OnLine** can write information during a sort.
This variable is used even if the environment variable PSORT_NPROCS is not set.

```
setenv PSORT_DBTEMP pathname
```

`pathname` is the name of the UNIX directory used for intermediate writes during a sort.

Set the PSORT_DBTEMP environment variable as follows to specify the directory, for example, `/usr/leif/tempsort`:

**C shell:**

```
setenv PSORT_DBTEMP /usr/leif/tempsort
```

**Bourne shell:**

```
PSORT_DBTEMP=/usr/leif/tempsort
export PSORT_DBTEMP
```

For maximum performance, specify directories that reside in file systems on different disks.

You also might want to consider setting the environment variable DBSPACETEMP to place temporary files used in sorting in dbspaces rather than operating system files. See the discussion of the DBSPACETEMP environment variable on page B-28.

**PSORT_NPROCS**

The PSORT_NPROCS environment variable affects only client applications connected to an OnLine server.

PSORT_NPROCS enables the Psort parallel-process sorting package to improve performance on OnLine. The setting defines the upper limit for the number of threads used to sort a query.

```
setenv PSORT_NPROCS value
```

`value` specifies the maximum number of threads to be used to sort a query.

Set the PSORT_NPROCS environment variable as follows to specify the maximum value:

**C shell:**

```
setenv PSORT_NPROCS 4
```

**Bourne shell:**

```
PSORT_NPROCS=4
export PSORT_NPROCS
```

To maximize the effectiveness of Psort, set PSORT_NPROCS to the number of available processors in the hardware. If PSORT_NPROCS is set to zero, Psort uses three as the default number of threads.
Use the following command to disable Psort:

**C shell:** unsetenv PSORT_NPROCS

**Bourne shell:** unset PSORT_NPROCS

For additional information about the PSORT_NPROCS environment variable, see *INFORMIX-OnLine Dynamic Server Administrator’s Guide*, Version 6.0.

**SQLEXEC**

The SQLEXEC environment variable specifies the location of the Version 6.0 relay module executable that allows a Version 4.1 or earlier client to communicate indirectly with a local Version 6.0 OnLine or INFORMIX-SE database server. You must, therefore, set SQLEXEC only if you want to establish communications between a Version 4.1 or earlier client and a Version 6.0 database server.

```
setenv SQLEXEC pathname
```

`pathname` specifies the pathname for the relay module.

Set SQLEXEC as follows to specify the full pathname of the relay module, which is in the `lib` subdirectory of your `$INFORMIXDIR` directory:

**C shell:**
```
setenv SQLEXEC $INFORMIXDIR/lib/sqlrm
```

**Bourne shell:**
```
SQLEXEC=$INFORMIXDIR/lib/sqlrm
export SQLEXEC
```

If you set the SQLEXEC environment variable on the C shell command line instead of in your `.login` or `.cshrc` file, you must include curly braces around the existing INFORMIXDIR, as follows:

**C shell:**
```
setenv SQLEXEC `$(INFORMIXDIR)/lib/sqlrm`
```


**SQLRM**

In Version 6.0, if the system administrator is configuring a client/server environment in which a Version 4.1 I-SQL client accesses a local Version 6.0 database server, the SQLRM environment variable must be `unset` before SQLEXEC can be used to spawn a Version 6.0 relay module.
Unset SQLRM as follows:

C shell: `unsetenv SQLRM`
Bourne shell: `unset SQLRM`


**SQLRMDIR**

In Version 6.0, if the database administrator is configuring a client/server environment in which a Version 4.1 I-SQL client accesses a local Version 6.0 database server, the SQLRM environment variable must be *unset*.

Unset SQLRMDIR as follows:

C shell: `unsetenv SQLRMDIR`
Bourne shell: `unset SQLRMDIR`

**NLS Environment Variables**

The following variables apply to Native Language Support, and are documented in Appendix C, “Native Language Support Within INFORMIX-SQL.”

<table>
<thead>
<tr>
<th>NLS Environment Variable</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLLCHAR</td>
<td>C-19</td>
</tr>
<tr>
<td>DBAPICODE</td>
<td>C-23</td>
</tr>
<tr>
<td>DBNLS</td>
<td>C-17</td>
</tr>
<tr>
<td>LANG</td>
<td>C-25</td>
</tr>
<tr>
<td>LC_COLLATE</td>
<td>C-27</td>
</tr>
<tr>
<td>LC_CTYPE</td>
<td>C-29</td>
</tr>
<tr>
<td>LC_MONETARY</td>
<td>C-31</td>
</tr>
<tr>
<td>LC_NUMERIC</td>
<td>C-35</td>
</tr>
</tbody>
</table>
UNIX Environment Variables

Informix products also rely on the correct setting of certain standard UNIX system environment variables. The PATH and TERM environment variables must always be set. You also might have to set the TERMCAP or TERMINFO environment variable to use I-SQL effectively.

**PATH**

The PATH environment variable tells the shell the order in which to search directories for executable programs. You must include the directory that contains I-SQL in your PATH environment variable before you can use I-SQL.

```
setenv PATH $PATH:/usr/informix
```

`pathname` specifies the search path for executables.

You can specify the correct search path in various ways. Be sure to include a colon between the directory names.

The following example uses the explicit path `/usr/informix`. This path must correspond to the INFORMIXDIR setting. The C shell example is valid for `.login` or `.cshrc` files.

**C shell:**
```
setenv PATH $PATH:/usr/informix/bin
```

**Bourne shell:**
```
PATH=$PATH:/usr/informix/bin
export PATH
```

The following example specifies `$INFORMIXDIR` instead of `/usr/informix`. It tells the shell to search the directories that were specified when INFORMIXDIR was set. The C shell example is valid for `.login` or `.cshrc` files.

**C shell:**
```
setenv PATH $PATH:$INFORMIXDIR/bin
```

**Bourne shell:**
```
PATH=$PATH:$INFORMIXDIR/bin
export PATH
```

You might prefer to use this version to ensure that your PATH entry does not contradict the path that was set in INFORMIXDIR, and so that you do not have to reset PATH whenever you change INFORMIXDIR.

**Note:** If you set the PATH environment variable on the C shell command line instead of in your `.login` or `.cshrc` file, you must include curly braces with the existing INFORMIXDIR and PATH, as follows:

**C shell:**
```
setenv PATH `$(INFORMIXDIR)/bin:$PATH`
```
Environment Variables

UNIX Environment Variables

TERM

The TERM environment variable is used for terminal handling. It enables I-SQL to recognize and communicate with the terminal you are using.

```
setenv TERM type

type specifies the terminal type.
```

The terminal type specified in the TERM setting must correspond to an entry in the `termcap` file or `terminfo` directory. Before you can set the TERM environment variable, you must obtain the code for your terminal from the OnLine or INFORMIX-SE administrator.

For example, to specify the vt100 terminal, set the TERM environment variable as follows:

```
C shell:       setenv TERM vt100
Bourne shell:  TERM=vt100
               export TERM
```

TERMCAP

The TERMCAP environment variable is used for terminal handling. It tells I-SQL to communicate with the `termcap` file instead of the `terminfo` directory.

```
setenv TERMCAP pathname

pathname specifies the location of the termcap file.
```

The `termcap` file contains a list of various types of terminals and their characteristics. Set TERMCAP as follows:

```
C shell:       setenv TERMCAP /usr/informix/etc/termcap
Bourne shell:  TERMCAP=/usr/informix/etc/termcap
               export TERMCAP
```

Note: If you set the TERMCAP environment variable, be sure that the INFORMIXTERM environment variable is set to the default, `termcap`. 
UNIX Environment Variables

TERMINFO

The TERMINFO environment variable is used for terminal handling. It is supported only on machines that provide full support for the UNIX System V terminfo library.

```bash
setenv TERMINFO /usr/lib/terminfo
```

TERMINFO tells I-SQL to communicate with the terminfo directory instead of the termcap file. The terminfo directory has subdirectories that contain files pertaining to terminals and their characteristics.

Set TERMINFO as follows:

**C shell:**
```
setenv TERMINFO /usr/lib/terminfo
```

**Bourne shell:**
```
TERMINFO=/usr/lib/terminfo
export TERMINFO
```

**Note:** If you set the TERMINFO environment variable, you also must set the INFORMIXTERM environment variable to terminfo.
Native Language Support Within
INFORMIX-SQL

This appendix discusses the Native Language Support (NLS) features that are included in the 6.0 release of INFORMIX-SQL (I-SQL).

This appendix is organized as follows:

- Overview of NLS
- The Non-NLS Environment
- NLS Environments
- NLS Features Supported in I-SQL
- Classification and Precedence of Environment Variables
- Database Storage of Environment Variables
- Meta-Environment Variables
- X/Open-Defined Variables
- Informix-Defined Language and Formatting Variables
- LOAD and UNLOAD Statements
- FORMAT and USING
- Multiple Locale Support
- Language Supplements
- Glossary of NLS Terms
Overview of NLS

Native Language Support is based on the X/Open Portability Guide Version 3 (XPG3) standard, which specifies a means for localization of software to European geographical regions.

A useful NLS concept is that of the locale; a locale specifies, by way of NLS environment variables, the language and formatting environment of an I-SQL user (the user locale) or of a database at the time of its creation (the database locale). The LANG environment variable establishes an overall locale such as German or French.

Each specific feature of a locale, pertaining to one aspect of the language environment, is referred to as category. NLS categories are specified with environment variables whose names start with the letters LC followed by the underscore (_) symbol. NLS categories supported in I-SQL include:

- Sorting sequence of characters, also known as collation (LC_COLLATE)
- Character set (LC_CTYPE)
- Monetary formatting (LC_MONETARY)
- Numeric formatting (LC_NUMERIC)

LC_ variables do not have to be individually specified. The LANG variable establishes defaults for all LC_ variables. These defaults are appropriate for most users in the geographic region implied by LANG. One or more LC_ variables are used to fine-tune particular features of the locale.

Note: Since the LANG variable specifies values for LC_COLLATE, LC_CTYPE, LC_MONETARY, and LC_NUMERIC, references in the text to setting any of the LC_ variables can either mean setting the LC_ variable directly or setting it indirectly by way of LANG.

Informix-Defined Environment Variables

In addition to standard NLS categories, Informix provides its own environment variables with the 6.0 tools.

Supported Informix-created environment variables control:

- On/off status of the NLS feature set, and level of required database-user locale matching (DBNLS).
- On/off status of implicit collation-sequence mapping between user and server (COLLCHAR).
- Character set translation between the database and non-ASCII terminal equipment (DBAPICODE).
Overview of NLS

• Location of files that specify error messages, menus, and month and day names that are translated into a national language (DBLANG and DBFORM).
• Numeric, monetary, and date formatting (DBFORMAT, DBMONEY and DBDATE). The Informix-created variables predate the XPG3 NLS variables, and so are supported for reasons of backwards compatibility.

NLS Character Sets

The NLS character sets are provided to meet the needs of European countries. Most European languages contain various characters that are not found in English, such as eszet (ß), a-umlaut (ä), and enye (ñ).

The ASCII character set used for English is a standard representation system for characters on computers. Any ASCII character can be represented with seven bits of data, of which there are 128 possible combinations. Each of these combinations has a numeric value between 0 and 127. For example, the capital letter B has the ASCII value 66.

It would seem that 128 possible ASCII values would provide sufficient room for representing non-English characters. However, the ASCII set has to include all CONTROL characters, punctuation, numeric digits, uppercase and lowercase alphabetic letters, and arithmetic symbols. There are no unused ASCII values for assignment to special European characters. ASCII characters and their ASCII values are listed in Appendix E, “The ASCII Character Set.”

With the elimination of the need for computer systems to include a parity bit in 8-bit computer bytes, an eighth bit has become available for representing characters. This doubles the number of available character representation values, from 128 to 256. Particular computer manufacturers have created their own 8-bit character sets which work with their own computer equipment but not others. The first 128 values always correspond to ASCII values, but the remaining 128 values are assigned arbitrarily.

The International Standards Organization (ISO) has created two standard 8-bit character sets to support European alphabets uniformly across computer platforms. These are the ISO 8859-1 and ISO 8859-2 character sets, which support Western European and Eastern European languages, respectively.

NLS character set support provides, by way of the LC_CTYPE and DBAPICODE environment variables, the ability to:
• Specify a character set for representing the character data in a database.
• Create user-defined names such as table and column names from the set of characters supported by the database.
Overview of NLS

• Specify a different character set for the computer monitor or printer, and provide the means to translate characters between the database and the monitor or printer.

• Utilize built-in capitalization rules that assure correct conversion between uppercase and lowercase characters within a character set.

*Note:* There are several possible standards for Asian and Arabic language representation, which are not supported by way of NLS. Asian and Arabic language support will require 16-bit character (or larger) representation. In the future these languages will be supported by way of a different standard called GLS (Global Language Support), which is an extension of the NLS standard.

### NLS Collation

With the introduction of non-English characters into data, there arises a need for specifying how character data is to be alphabetized. **I-SQL 6.0** introduces the capability to sort and compare character data in a collating sequence specified by the **LC_COLLATE** category in the locale.

To accomplish this, two new data types are utilized by the server in databases created for NLS environments: **NCHAR** and **NVARCHAR**. These data types are not directly definable by the **I-SQL** user (except when using the SQL CREATE TABLE and ALTER TABLE commands in the Explicit mode, discussed below). Rather, using a process called *implicit mapping*, the server substitutes the data type **NCHAR** when the user defines or accesses a **CHAR** column, and substitutes the data type **NVARCHAR** when the user defines or accesses a **VARCHAR** column.

The effect of implicit mapping is to allow you to define the same character data types as you have in the past (pre-Version 6.0) in your applications, but have the system automatically treat this character data as sorted according to the locale.

The server’s data types **NCHAR** and **NVARCHAR** are identical to **CHAR** and **VARCHAR** respectively, except that data in an **NCHAR** or **NVARCHAR** column is sorted and compared according to the **LC_COLLATE** setting in the locale, rather than according to the default collation (US English ASCII order).
The Non-NLS Environment

Depending on the settings in the environment variables DBNLS and COLLCHAR, the user operates in either the Non-NLS environment or one of three NLS environments. The three NLS environments are Implicit NLS, Explicit NLS, and Open NLS. These are discussed in the next section.

The Non-NLS environment is specified by unsetting the DBNLS variable (or equivalently, setting it to zero). The Non-NLS environment is equivalent to using a pre-6.0 version of I-SQL. With the Non-NLS environment active, the following are true:

- The NLS environment variables LC_COLLATE, LC_CTYPE, LC_MONETARY, and LC_NUMERIC have no effect.
- All character data is sorted and compared according to US English collation.
- Only characters from the ASCII character set may be used in identifiers (such as database and table names).
- The default numeric and monetary formats (in the absence of DBFORMAT and DBMONEY settings) are ANSI compliant.

A database that is created when one of the NLS environments is active, known as an NLS database, cannot be accessed while in the Non-NLS environment.

NLS Environments

The three NLS environments (Implicit, Explicit and Open) are specified by combinations of the DBNLS and COLLCHAR environment variables in which DBNLS is set to a value of 1 or 2.

With any of the three NLS environments active, the following are true:

- The NLS environment variables LC_COLLATE, LC_CTYPE, LC_MONETARY, and LC_NUMERIC are considered in various operations.
- Character data columns are sorted and compared according to the collation sequence specified by the LC_COLLATE variable. (If the Explicit NLS environment is active and the I-SQL user wishes to create US English sorted columns in an otherwise non-English database, this is permitted, but not recommended.)
- Characters from the character set specified by LC_CTYPE are available for use in identifiers, in addition to the ASCII character set.
- Monetary and numeric formats specified by LC_MONETARY and LC_NUMERIC become active in display, input, printing, loading, and
unloading of values, provided that DBFORMAT and DBMONEY are not set.

If any of the three NLS environments is active when a database is created, the database will be permanently associated with the collation sequence (LC_COLLATE) and character set (LC_CTYPE) that are current at the time of creation, and permanently designated as an NLS database.

A database that is created when the Non-NLS environment is active, known as a non-NLS Database, can be accessed while in any of the NLS environments. However, none of the NLS variables take effect when working in a non-NLS database. To avoid confusion, it is recommended that you unset DBNLS before working with a non-NLS database.

There are two distinctions between the three NLS environments:

1. Whether or not the user is required to access an NLS database with the same collation sequence (LC_COLLATE) and character set (LC_CTYPE) variable values specified in the current environment as were specified at the time of database creation.

2. Whether or not the user can define character columns that are sorted and compared in US English in a non-English NLS database.

All version 6.0 Informix tools can utilize the Implicit NLS and Open NLS environments. All version 6.0 Informix tools except 4GL can utilize the Explicit NLS environment. Implicit NLS is the recommended NLS environment, as it provides NLS capabilities without disruption to existing applications or risk of data corruption.

Implicit NLS

Implicit NLS is the environment defined by the DBNLS environment variable set to 1 and the COLLCHAR variable also set to 1. This is the standard NLS environment in which I-SQL operates. All character columns defined by the user in the Implicit environment appear to the user as type CHAR (or VARCHAR, if variable-length), but are interpreted by the server as type NCHAR (or NVARCHAR) if the database is non-US-English. This is referred to as implicit mapping. Thus, all database columns sort according to the LC_COLLATE setting in the locale. Such columns are referred to as locale-sorted. Columns that are not locale-sorted cannot be created in the Implicit NLS environment.

The advantage of the Implicit environment for a user is that existing references to CHAR and VARCHAR data types do not have to be changed when an application is moved from a non-NLS to an NLS environment.
Locale-sorting of character data becomes available without modification. Also, the user does not need to be concerned with which columns are locale-sorted and which columns are not.

The Implicit NLS environment, like Explicit NLS, mandates consistency checking between the current LC_COLLATE and LC_CTYPE values and those that were saved with the database at the time of database creation. Access is not permitted to an NLS database with different LC_COLLATE and LC_CTYPE settings from the current environment’s settings. This helps prevent certain kinds of data corruption.

**Explicit NLS**

Explicit NLS is the environment defined by the DBNLS environment variable set to 1 and the COLLCHAR variable unset. In the Explicit environment, character columns can be defined by the user that are not implicitly mapped, that is, always sort as CHAR (or VARCHAR) data, regardless of locale. Columns that are to be locale-sorted are defined by the user explicitly as NCHAR (or NVARCHAR). Columns that are to be US English sorted are defined by the user as CHAR or VARCHAR, and these are not mapped to NCHAR and NVARCHAR at the server. Note that the NCHAR and NVARCHAR types do not appear in the listing of data types that is displayed when creating table columns from the menus. These two types can only be created by way of the SQL commands CREATE TABLE and ALTER TABLE.

The only advantage of the Explicit NLS environment is that it allows users to separately declare nationalized and non-nationalized character data types, and thereby avoid the resource-intensive NCHAR sorting when it is not needed. However, it is easy to confuse CHAR and NCHAR columns in the Explicit environment, and thereby create unintentional data corruption problems or undesired application results.

When US English sorting is acceptable, a database should be created as a non-NLS database. When locale-sorting is required, the database should be created as an NLS database in the Implicit NLS environment.

The Explicit environment and explicit creation of CHAR and VARCHAR columns in an NLS database are primarily intended for users of server tools such as ESQL/C and DB-Access, and are not recommended for I-SQL users.

The Explicit NLS environment, like Implicit NLS, mandates consistency checking between the current LC_COLLATE and LC_CTYPE values and those that were saved with the database at the time of database creation. Access is not permitted to an NLS database with different LC_COLLATE and LC_CTYPE settings from the current environment’s settings.
Open NLS

Open NLS is the environment defined by the DBNLS variable set to 2 and the COLLCHAR variable set to 1. Open NLS differs from Implicit and Explicit NLS in that the Open environment the system does not perform consistency checking when access to an NLS database is attempted. Any combination of settings of LC_COLLATE and LC_CTYPE is permitted when accessing an NLS database in the Open environment.

Consistency checking is normally a desirable feature, as it prevents data corruption such as would occur if French-sorted data were appended to a German-sorted character column or characters peculiar to Spanish were allowed in the names of tables in an Italian database. However, there are three situations in which overriding the consistency checking feature by means of specifying the Open environment are desirable:

1. When users of non-Informix tools (or Informix tools prior to version 6.0) need to access data in an NLS database stored in an Informix database engine, they would specify the Open environment.
2. When data is unloaded from an NLS database in one locale, and loaded to an NLS database in another locale.
3. When data is unloaded from a non-NLS database and loaded to an NLS database or vice versa.

Open NLS is similar to Implicit NLS in that implicit mapping is performed. All character columns created are locale-sorted, that is, NCHAR or NVARCHAR.
Summary of Environments

In the following table, behavior of NLS databases in Open NLS, Implicit NLS, Explicit NLS and Non-NLS environments is contrasted with the behavior of non-NLS databases in those environments:

<table>
<thead>
<tr>
<th>Environment</th>
<th>Settings</th>
<th>Property</th>
<th>NLS database</th>
<th>Non-NLS database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open NLS</td>
<td>DBNLS=2</td>
<td>user can access</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>COLLCHAR=1</td>
<td>user can create</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Implicit NLS</td>
<td>DBNLS=1</td>
<td>user can access</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>COLLCHAR=1</td>
<td>user can create</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Explicit NLS</td>
<td>DBNLS=1</td>
<td>user can access</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>COLLCHAR unset</td>
<td>user can create</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Non NLS</td>
<td>DBNLS=0 or</td>
<td>user can access</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>DBNLS unset</td>
<td>user can create</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

Figure C-1  Database Access In Open, Implicit, and Explicit Environments

NLS Features Supported in INFORMIX-SQL 6.0

NLS features supported in I-SQL version 6.0 include:

- Character data sorting and comparison according to the rules of a national language locale.
- Use of extended-ASCII characters permitted in user-defined names such as database, table, and column names.
- Nationalized money and numeric decimal formats in reports, screen forms, and data assignment statements.
- Character conversion between database data and national language specific keyboards and screens.
- The ability for different users to simultaneously access, on the same server, databases with different locale settings.
Figure C-2 and Figure C-3 present an overview of the affected data types and I-SQL menu options and keywords.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>Transparently maps to NCHAR if I-SQL is running in Implicit mode against an NLS database.</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>Transparently maps to NVARCHAR if I-SQL is running in Implicit mode against an NLS database.</td>
</tr>
<tr>
<td>NCHAR</td>
<td>New data type. Sorts in the order of the user locale. Available only by way of SQL CREATE TABLE, and only if I-SQL is running in Explicit mode against an NLS database.</td>
</tr>
<tr>
<td>NVARCHAR</td>
<td>New data type. Sorts in the order of the user locale. Available only by way of SQL CREATE TABLE, and only if I-SQL is running in Explicit mode against an NLS database.</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>Display depends on values in DBFORMAT, DBMONEY, and LC_NUMERIC (highest to lowest precedence).</td>
</tr>
<tr>
<td>SMALLFLOAT</td>
<td>Display depends on values in DBFORMAT, DBMONEY, and LC_NUMERIC (highest to lowest precedence).</td>
</tr>
<tr>
<td>FLOAT</td>
<td>Display depends on values in DBFORMAT, DBMONEY, and LC_NUMERIC (highest to lowest precedence).</td>
</tr>
<tr>
<td>MONEY</td>
<td>Display depends on values in DBFORMAT, DBMONEY, and LC_MONETARY (highest to lowest precedence).</td>
</tr>
<tr>
<td>DATE</td>
<td>Separator symbol and order of month, day, and year depends on the value in DBDATE. Display of language-specific month and day names depends on installation of message files, whose location is referenced by DBLANG.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>Display of language-specific month and day names depends on the installation of message files, whose location is referenced by DBLANG.</td>
</tr>
</tbody>
</table>

**Figure C-2**  
Impact of NLS Support on Data Types
### Menu Option or Keyword

<table>
<thead>
<tr>
<th><strong>IMPACT</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LOAD</strong></td>
</tr>
<tr>
<td><strong>UNLOAD</strong></td>
</tr>
<tr>
<td><strong>USING</strong></td>
</tr>
<tr>
<td><strong>CREATE TABLE, ALTER TABLE</strong></td>
</tr>
<tr>
<td><strong>FORMAT</strong></td>
</tr>
<tr>
<td><strong>ORDER BY, MATCHES, WHILE, INCLUDE, and IF</strong></td>
</tr>
<tr>
<td><strong>LET</strong></td>
</tr>
<tr>
<td><strong>UPSHIFT and DOWNSHIFT</strong></td>
</tr>
<tr>
<td><strong>ASCII (ACE)</strong></td>
</tr>
<tr>
<td><strong>DATE</strong></td>
</tr>
<tr>
<td><strong>MENU NAME</strong></td>
</tr>
<tr>
<td><strong>CALL (to C function)</strong></td>
</tr>
</tbody>
</table>

*Figure C-3 Impact of NLS Support on Menu Options and Keywords*
Classification and Precedence of Environment Variables

The environment variables used by Informix servers and tools, including I-SQL, are either X/Open-defined variables or Informix-defined variables. X/Open-defined variables include LANG and variables that start with the characters LC_. Collectively the X-Open defined variables specify the locale. Informix-defined variables are not in the X/Open standard, and include COLLCHAR and variables that start with the characters DB.

The LANG and LC_ variables, along with the Informix-defined variables DBFORMAT, DBDATE, and DBMONEY, together specify the language and formatting environment of the user. These variables are referred to as language and formatting variables. They define the following aspects of the environment:

- Sort order of character data (LC_COLLATE)
- Valid character set for identifiers (LC_CTYPE)
- Monetary data format (DBFORMAT, DBMONEY and LC_MONETARY)
- Numeric data format (DBFORMAT, DBMONEY and LC_NUMERIC)
- Date format (DBDATE)

The DBNLS, COLLCHAR, DBAPICODE, and DBLANG environment variables are referred to as meta-environment variables. They specify the following aspects of the meta-environment:

- Whether or not NLS is activated (DBNLS).
- Whether Explicit, Implicit, or Open NLS is active (DBNLS and COLLCHAR).
- The name of a character translation file for mapping characters between the database and terminal equipment (DBAPICODE).
- The location of message and menu form files (DBLANG and DBFORM).

These classifications are summarized in the following table:

<table>
<thead>
<tr>
<th>Meta-Environment</th>
<th>X/Open defined (locale)</th>
<th>Informix defined</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LANG</td>
<td>DBNLS</td>
</tr>
<tr>
<td></td>
<td>LC_COLLATE</td>
<td>COLLCHAR</td>
</tr>
<tr>
<td></td>
<td>LC_CTYPE</td>
<td>DBAPICODE</td>
</tr>
<tr>
<td>Language and</td>
<td>LC_MONETARY</td>
<td>DBLANG</td>
</tr>
<tr>
<td>Formatting</td>
<td>LC_NUMERIC</td>
<td>DBFORM</td>
</tr>
<tr>
<td></td>
<td>DBFORMAT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DBMONEY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DBDATE</td>
<td></td>
</tr>
</tbody>
</table>
It is important to distinguish between Informix-defined and XPG3-defined (locale) variables. This is because the locale variables rely on facilities provided by the computer manufacturer, and may vary in meaning from platform to platform. For example, the German locale may be specified as LANG DE on one system and LANG de on another, and the two may imply different collation, character sets, or formatting rules. Informix-defined variables, in contrast, have uniform syntax and meaning across platforms.

The following Informix-defined variables can be used without activating NLS, but interact with NLS variables because they pertain to monetary, numeric, and date formatting. They are covered in detail in Appendix B, "Environment Variables."

- DBLANG
- DBFORMAT
- DBMONEY
- DBDATE

The overall precedence hierarchy for language and formatting variables is as follows:

![Diagram](image)

**Figure C-4 Order of Precedence of Language and Formatting Environment Variables**

The Informix-defined language and formatting variables—DBFORMAT, DBMONEY, and DBDATE—take the highest precedence, the LC_ variables intermediate precedence, and the LANG variable the lowest. Any of these will override internal defaults. Therefore, a DBFORMAT setting will override an LC_NUMERIC setting, which in turn will override LANG, which will override the default.
Native Language Support Within INFORMIX-SQL

In Informix tools and engine products, the existing variables that specify language-specific or country-specific behavior need to be retained and given the highest priority, for the sake of existing user applications that use these variables.

In the XPG3 NLS standard, the LANG variable specifies a locale, and the LC_ variables are used to modify particular pieces of the locale that the user wants to be different from the standards for that locale. The LANG variable sets the default values of all of the LC_ variables. The user may override the LANG defaults for particular LC_ variables by setting those individually.

In the following example, the user is selecting the German language locale (LANG set to DE). However, the user then selects the German Swiss dictionary collation sequence to override the default German sequence (LC_COLLATE), and the ISO 8859/1 set of monetary formats for expressing money values in Swiss francs (LC_MONETARY) to override the default German monetary format:

```
setenv LANG DE
setenv LC_COLLATE DE_ch@dict
setenv LC_MONETARY CH.88591
```
The order in which environment variables is specified is not significant. However, the directory location where a locale variable is defined influences the precedence, as illustrated in the following diagram:

1. The highest precedence goes to the value as defined in the environment (shell).
2. The second-highest precedence goes to the value as defined in the private environment-configuration file in the user's home directory (~/.informix). This is a private file where you can define all the environment variables that are used by Informix products. Using a configuration file reduces the number of environment variables that you must set at the command line or in a shell file.
3. The next-highest precedence goes to the value as defined in the common environment-configuration file ($INFORMIXDIR/etc/informix.rc). This is...
the same as a private environment-configuration file in the user’s home directory, only for shared use rather than private.

4. The lowest precedence goes to the default value.

5. The setting for any of the LC_ variables takes precedence over the setting for the LANG environment variable, no matter where they are set.

**Database Storage of Environment Variables**

When the database connection is established between a user session and the database server by way of the SELECT or CREATE options in the DATABASE menu in I-SQL, the information in the local environment variables LC_COLLATE and LC_CTYPE will be transmitted with the request for database service. If a new database is being created, the database server saves these values in system tables inside the database. If the user is requesting access to an existing database, the server rejects the connection request if the current environment’s LC_COLLATE and LC_CTYPE values do not match the saved database values (and the Implicit or Explicit, rather than Open, NLS environment is being employed). This process is referred to as consistency checking.

For I-SQL, consistency checking takes place at the time an interactive database creation or selection request is issued.

These values are kept unchanged throughout the life of the database to ensure the consistent use of collating sequences and character sets. The LC_COLLATE and LC_CTYPE environment variable settings for a database cannot be changed; the data must be unloaded and reloaded into a different database to change collation or character set features of the database. Numeric and monetary format features of the locale can be changed as desired, because monetary and numeric data are saved in database tables in ANSI format. LC_MONETARY and LC_NUMERIC affect the display and input of data from the standpoint of the user, not internally at the server. However, they do affect the format of an ASCII file created by an UNLOAD operation, and the interpretation of the data in that file during a LOAD.
Meta-Environment Variables

Meta-environment variables are all Informix defined. They turn on and off features of the NLS environment and point to the locations of certain files that NLS uses.

DBNLS

The Informix-defined DBNLS environment variable enables or disables the NLS features implemented in the 6.0 tools and server products, and specifies the level of consistency checking to take place between user and database. The user activates the NLS capability in I-SQL by setting DBNLS to 1 or 2. NLS capability is deactivated by unsetting DBNLS or setting it to 0, which is equivalent to unsetting.

DBNLS is set to 0, 1, or 2 as follows:

```bash
setenv DBNLS
```

- 0
- 1
- 2

DBNLS is unset as follows:

```bash
unsetenv DBNLS
```

Usage

If DBNLS is not set or set to 0, NLS functionality is not turned on, and the settings for the other NLS environment variables do not take effect. If DBNLS is set to 2 (and COLLCHAR set to 1, which is required), NLS functionality is activated but in the Open NLS environment. The Open NLS environment allows inconsistent locales between user and database.

Unlike LC_CTYPE and LC_COLLATE, the actual value of DBNLS is not saved with a database at creation time. However, the DBNLS setting at creation time permanently determines whether the database is an NLS database or a non-NLS database.
Meta-Environment Variables

The meanings of the available settings for DBNLS are listed in the table below:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBNLS = unset or DBNLS = 0</td>
<td>This is the default setting. It specifies the non-NLS environment. Only non-NLS databases can be created or accessed if the user environment has this setting. Any attempt to access an existing NLS database results in an error. Any databases created will not be NLS compatible. I-SQL version 6.0 with this setting will behave exactly like non-NLS (pre-6.0) I-SQL.</td>
</tr>
<tr>
<td>DBNLS = 1</td>
<td>This is the recommended setting for performing NLS work. This setting establishes either the Implicit (recommended) or Explicit NLS environment, depending on the setting of COLLCHAR. Either NLS or non-NLS databases can be accessed with DBNLS set to 1, but only NLS databases can be created. When accessing an existing NLS database, the user session LC_CTYPE and LC_COLLATE values must match the corresponding database values in order for access to be permitted. A database is locked into the current user environment values of the these two variables at database creation time.</td>
</tr>
<tr>
<td>DBNLS = 2</td>
<td>This setting specifies the Open NLS environment (in combination with a COLLCHAR setting of 1). Open NLS provides access to NLS databases regardless of whether or not the user session and database server settings of LC_COLLATE and LC_CTYPE match. The server obtains its settings of these variables from the database being accessed. New databases inherit the current locale of the user when they are created. Open NLS allows tools that are not aware of internal database locale settings to access NLS databases, and provides a means to load and unload data between dissimilar locales. Otherwise, this setting is not recommended. Character comparison results can be inconsistent between the user (for example, variable1 greater than variable2) and SQL queries issued to the server (ORDER BY, WHERE, MATCHES). Database data can be corrupted using Open NLS when the user updates the database, because the active environment settings can differ from the settings in which the database is intended to operate. For example, column names could be created with one valid character set active, and then become illegal when the active character set changes.</td>
</tr>
<tr>
<td>DBNLS = any value other than 0, 1, 2 or unset</td>
<td>This is an invalid setting. A run-time error will be returned.</td>
</tr>
</tbody>
</table>

**Figure C-6 Values for the DBNLS environment variable**

Operations on non-NLS databases while in an NLS environment follow the same rules as operations in pre-6.0 I-SQL. These include:

- Sorting order based on ASCII character code
- User defined names allowed to contain only ASCII characters
**COLLCHAR**

The Informix-defined COLLCHAR environment variable turns on or off the implicit mapping feature, which is necessary for using the Implicit NLS and Open NLS environments. COLLCHAR determines whether the data types NCHAR and NVARCHAR can be accessed directly by a client application (COLLCHAR = unset), or whether they are accessed by way of implicit mapping of CHAR and VARCHAR data types (COLLCHAR = 1). COLLCHAR has no effect in the non-NLS environment (DBNLS set to 0 or unset), in which the NCHAR and NVARCHAR data types are inaccessible.

Implicit NLS is the recommended NLS environment, as it provides NLS capabilities without disruption to existing applications or risk of data corruption. Refer to the discussion of Implicit, Explicit and Open NLS starting on page C-6.

In I-SQL version 6.0, the Explicit NLS (DBNLS = 1 and COLLCHAR = unset) environment is supported. An I-SQL user in the Explicit NLS environment can define CHAR and VARCHAR columns that will remain CHAR and VARCHAR when created in the database, provided the columns are defined by way of the SQL commands CREATE TABLE and ALTER TABLE rather than at the menu interface. NCHAR and NVARCHAR columns defined by way of SQL CREATE TABLE and ALTER TABLE will remain NCHAR and NVARCHAR when created in the database. The Explicit NLS environment is not recommended for I-SQL users.

The user activates implicit mapping in I-SQL by setting COLLCHAR to 1. Explicit NLS is obtained by unsetting the COLLCHAR variable. Note that 0 is not an acceptable value for COLLCHAR, and may cause unpredictable results.

COLLCHAR is set to 1 as follows:

```
setenv COLLCHAR 1
```

COLLCHAR is unset as follows:

```
unsetenv COLLCHAR
```

**Note:** There is a significant performance penalty associated with using locale-sorted (NCHAR and NVARCHAR) data types at the server. When there is no need for sorting according to the locale-specified collating sequence, users should create non-NLS databases rather than NLS databases.
Meta-Environment Variables

### Usage

The meanings of the available settings for COLLCHAR are listed in the table below:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLLCHAR = unset</td>
<td>This is the default value. It allows the I-SQL user to specify NCHAR and</td>
</tr>
<tr>
<td></td>
<td>NVARCHAR directly (Explicit NLS) if DBNLS is set to 1. Explicit NLS is not</td>
</tr>
<tr>
<td></td>
<td>recommended for I-SQL users.</td>
</tr>
<tr>
<td>COLLCHAR = 1</td>
<td>This setting specifies Implicit NLS when DBNLS=1 and Open NLS when DBNLS=2.</td>
</tr>
<tr>
<td></td>
<td>This setting activates implicit mapping, in which the database server maps</td>
</tr>
<tr>
<td></td>
<td>all incoming CHAR requests to NCHAR, and outgoing requests to the client back</td>
</tr>
<tr>
<td></td>
<td>to CHAR again. When DBNLS=1 (Implicit), this COLLCHAR setting causes</td>
</tr>
<tr>
<td></td>
<td>consistency checking of LC_COLLATE and LC_CTYPE settings between the user</td>
</tr>
<tr>
<td></td>
<td>environment and the database. When DBNLS=2 (Open), this COLLCHAR setting</td>
</tr>
<tr>
<td></td>
<td>will enable access to NLS databases with different LC_COLLATE and LC_CTYPE</td>
</tr>
<tr>
<td></td>
<td>settings than the user session. This COLLCHAR setting requires a DBNLS</td>
</tr>
<tr>
<td></td>
<td>setting of 1 or 2 to have any effect.</td>
</tr>
<tr>
<td>COLLCHAR = any value</td>
<td>A run-time error will be returned.</td>
</tr>
<tr>
<td>other than 1 or unset</td>
<td></td>
</tr>
</tbody>
</table>

**Figure C-7 Values for the COLLCHAR environment variable**

The relationship between DBNLS and COLLCHAR is the following:

| DBNLS = unset or DBNLS=0 | Non-NLS environment. Only non-NLS databases can be accessed and created by |
| DBNLS = 1                | Explicit NLS environment. An I-SQL user can create both locale sorted and US |
|                          | English sorted columns in the same database. LC_CTYPE and LC_COLLATE of    |
|                          | user must match values of database for access.                            |
| DBNLS = 2                | Invalid combination of settings. For Open NLS, COLLCHAR must be set to 1, or|
|                          | results will be unpredictable.                                            |

**Figure C-8 DBNLS/COLLCHAR Cross Reference Table**

C-20  Native Language Support Within INFORMIX-SQL
In the Implicit NLS environment, I-SQL behaves as if all character columns were defined in the language-specific collation sequence. Although defined as CHAR or VARCHAR fields by the user, the behavior of all character columns in the Implicit environment is as if they were defined as NCHAR or NVARCHAR, that is, using locale-specified collation.

The difference between Implicit NLS (DBNLS = 1, COLLCHAR = unset) and Explicit NLS (DBNLS = 1, COLLCHAR = 1) is illustrated with the example below, in which a database table and character column are created first in Implicit NLS and then in Explicit NLS.

Implicit NLS settings and the German locale (LANG=DE) are specified at the operating system prompt:

```
setenv DBNLS 1
setenv COLLCHAR 1
setenv LANG DE
isql
```

A new database is created by selecting the Create option from the Database menu. The CREATE DATABASE screen appears and prompts for the name of the new database, which will be `test1`:

```
CREATE DATABASE >>test1
Enter the name you want to assign to the new database, then press RETURN.
--------------------------- Press CONTROL-W for Help ---------------------------
```

Figure C-9  Create Database Screen
A new table (länder—German for countries) is created by way of the CREATE TABLE screen, and two columns, one named land (German for country) and one named hauptstadt (German for capital) are created:

```
CREATE TABLE länder:
```

Since this is an NLS database (created with DBNLS set to 1) and implicit mapping is active (COLLCHAR is set to 1), the land and hauptstadt columns are created with data type NCHAR at the server. Any data saved in these columns will sort in German collating sequence.

To test this, we input the following data into the new table:

<table>
<thead>
<tr>
<th>land (country) value</th>
<th>hauptstadt (capital) value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Österreich</td>
<td>Wien</td>
</tr>
<tr>
<td>Portugal</td>
<td>Lissabon</td>
</tr>
<tr>
<td>Luxemburg</td>
<td>Luxemburg</td>
</tr>
</tbody>
</table>

At the SQL Query menu we execute the following command:

```
SELECT * FROM länder
ORDER BY land
```

The result of the query will be:

<table>
<thead>
<tr>
<th>land</th>
<th>hauptstadt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Luxemburg</td>
<td>Luxemburg</td>
</tr>
<tr>
<td>Österreich</td>
<td>Wien</td>
</tr>
<tr>
<td>Portugal</td>
<td>Lissabon</td>
</tr>
</tbody>
</table>

Note that Ö sorts between L and P, which is correct German alphabetization.
We return to the operating system prompt and enter the following setting, which specifies Explicit NLS instead of Implicit (note that DBNLS is still 1 and LANG is still DE):

\[ \text{unsetenv COLLCHAR} \]

We create a different database, this time called \texttt{test2}. We also create the \texttt{land} and \texttt{hauptstadt} columns again, but this time do so by way of an SQL query rather than the CREATE TABLE menu. NCHAR is unavailable in the CREATE TABLE menu, even when the environment settings are for Explicit mode. The following SQL statement is executed:

\[ \text{CREATE TABLE länder} \\
(\texttt{land} \texttt{CHAR(30)}, \texttt{hauptstadt} \texttt{NCHAR(28)}) \]

The same data values are input as before, and the SELECT query is executed. This time the query returns the following:

\begin{tabular}{ll}
Österreich & Wien \\
Luxemburg & Luxemburg \\
Portugal & Lissabon \\
\end{tabular}

This is correct sorting for US ASCII in an NLS database. Note that the \texttt{hauptstadt} column, because it is an NCHAR column, will sort in German order in a SELECT/ORDER BY query ordered on \texttt{hauptstadt}.

\textbf{Note:} An NLS database established with DBNLS=1 or 2 (NLS active) and LANG=EN_us (US ASCII) does not collate the same as a non-NLS database (DBNLS=unset or 0). The collation within the 128-character ASCII character set will be the same in either case, but the 128 special characters will collate ahead of ASCII in the NLS database with the EN_us locale, and will collate behind ASCII in the non-NLS database.

\section*{DBAPICODE}

The Informix-defined DBAPICODE environment variable lets a computer peripheral whose character set differs from the database character set access the database. In this context, \textit{peripheral} refers to a keyboard, monitor, or printer.
Meta-Environment Variables

DBAPICODE specifies the character-mapping file between the peripheral and the database character set. In NLS databases, the database character set is defined in the LC_CTYPE environment variable. In non-NLS systems, the database character set is the default ASCII character set.

```bash
catsetenv DBAPICODE mapfilename
```

`mapfilename` names a character mapping file to be used for translation of characters in the database character set to the keyboard and monitor character set.

Usage

DBAPICODE specifies to the system the name of a mapping file created by the Informix utility `crtcmap`. With DBAPICODE set, I-SQL communicates with the database server by using the ASCII character set, but interacts with the keyboard, monitor, or printer using the character set mapping file specified in DBAPICODE. If DBAPICODE is left unset, the system communicates with the keyboard and terminal by way of ASCII.

To use a specific DBAPICODE setting, there must be a mapping file for that character set in the message directory. For example, to use the character set FR_fr.646, there must be a mapping file called `mFR_fr.646`. This file should be located in the message directory `INFORMIXDIR/msg`. If not, DBLANG or LANG must point to the message directory. Refer to the discussion of DBLANG on pages C-24 and B-18.

The `crtcmap` utility helps you create mapping files between database and peripheral character sets for use with NLS. It prepends `m` to the name of the mapping file it creates. For example, it renames the output file `FR_fr.646` as `mFR_fr.646`. The `crtcmap` utility is described in the *Informix Guide to SQL: Reference*, Version 6.0.

An example of setting DBAPICODE to a character set file is as shown:

```
catsetenv DBAPICODE mFR_fr.646
```

**DBLANG**

The DBLANG variable specifies the subdirectory of `INFORMIXDIR` (or full pathname) in which the message files for the currently active language reside. Message files provide a set of error messages for the engine and tools that have been translated into a national language. Message files have the
suffix .iem. DBLANG also points to the location of the character mapping file, as discussed under “DBAPICODE” on page C-23. Message files are obtained as part of language supplements, which include instructions specifying where the files should be installed and what DBLANG settings to specify.

For a complete discussion of the DBLANG variable, refer to Appendix B, “Environment Variables.”

**DBFORM**

The DBFORM variable specifies the subdirectory of $INFORMIXDIR (or full pathname) in which the menu form files for the currently active language reside. Menu form files provide a set of language-translated menus to replace the standard I-SQL menus. Menu form files have the suffix .frm. Menu form files are obtained as part of language supplements, which include instructions specifying where the files should be installed and what DBFORM settings to specify.

For a complete discussion of the DBFORM variable, refer to Appendix B, “Environment Variables.”

**X-Open Defined Variables**

X-Open defined variables include LANG and variables that start with the characters LC_. These are also known as locale variables.

**LANG**

The value of the X/Open-defined LANG environment variable specifies the locale of the user’s environment. A LANG setting specifies all of the following:

- A collating sequence.
- A set of translations between uppercase and lowercase.
- A legal character set for user-defined names.
- Formatting rules for monetary and numeric data.

The LANG setting implies standard values for the language environment that you can modify or override by setting LC_COLLATE, LC_CTYPE, LC_MONETARY, or LC_NUMERIC. The LC_ environment variables are described in the sections that follow. The setting for these LC_ categories always takes precedence over the LANG setting. For example, if a LANG
setting for Swiss is specified, but LC_COLLATE is set for German, the Swiss settings apply to all categories in the locale except for character collation, which is German.

LANG should always be set to some locale value when creating or accessing NLS databases. The defaults for the LC_ variables when LANG is not set are unpredictable and vary between systems.

LANG identifies the required language, territory, and character set as illustrated in the syntax diagram below. The territory, character set and @modifier portions are optional and are not supported on some systems or with some LANG values.

The formal syntax for LANG is as follows:

```
setenv LANG language
```

- **language**  
a one to two character abbreviation such as FR for French, DE for German or EN for English.
- **_territory**  
specifies a dialect of a language, such as DE_ch (Swiss dialect of German) or EN_uk (British dialect of English).
- **.charset**  
specifies a character set other than ASCII, such as .88591 (the ISO 8859/1 character set).
- **@modifier**  
specifies a collation sequence within a language and territory, such as @dictionary (dictionary order) or @telephone (telephone book order).

In practice, the syntax for LANG is likely to be one of the following:

```
setenv LANG language
```

or

```
setenv LANG language_territory
```

There is no standardization of LANG locale values between systems. Exact values to specify to obtain particular locale settings are particular to different computer systems, and also depend on which language supplements have been installed on your system.
Locale Files

Locale files installed on your system are what actually determine the set of correct values for the LANG and LC_ variables. The implementation varies between computer platforms but the purpose on all systems is the same. Locale files do the following:

• Translate a value that you set for a locale variable into a set of rules governing system behavior.
• Define the list of acceptable values that you can set each locale variable to.

For example, on the Sun platform there are locale files found in directories called LC_COLLATE, LC_CTYPE, LC_MONETARY, and LC_NUMERIC. These are locale directories. When you set LANG to DE on a Sun workstation, you are referencing a file named DE in each of these four locale directories. When you set the environment variable LC_COLLATE to DE@telephone, a file named DE@telephone is referenced in the LC_COLLATE directory.

Other computer manufacturers such as IBM and Hewlett-Packard use other schemes for translating a locale variable setting into a pointer to a set of rules or values. The set of acceptable values for each locale variable and the effects of particular settings are dependent on which computer system you are using.

You can add to the set of valid LANG and LC_ variable settings on your system by installing language supplements.

The syntax diagrams for LANG and the LC_ variables in this manual suggest a range of possible settings for these variables, rather than an exact syntax for each variable that is applicable to all systems. Consult your language supplement documentation for specific settings.

LC_COLLATE

The X/Open-defined environment variable LC_COLLATE specifies a particular NLS collating sequence for the following:

• Sorting character data.
• Performing character string comparisons.
• Evaluating regular expressions, such as [A-Z], in MATCHES clauses.

LANG provides the default collation sequence if LC_COLLATE is not set. The value of LC_COLLATE, along with LC_CTYPE, is saved in an NLS database at database creation time, and consistency checked prior to database access.

Refer to “Database Storage of Environment Variables” on page C-16.
If `LC_COLLATE` (directly or by way of `LANG`) is set to a non-US English value, this value changes the following features of the Informix database environment:

- **Collating sequence**
  - For an ORDER BY or GROUP BY clause in SELECT statements.
  - For index generation.
  - For alphabetically listing tables, databases, report specification files, form specification files, modules, and so on in I-SQL menus.

- **Result set**
  - Of expressions containing character fields in the WHERE clause of a SELECT statement.

The formal syntax for `LC_COLLATE` is as follows:

```plaintext
setenv LC_COLLATE language

language a one to two character abbreviation such as FR for French, DE for German or EN for English.
_territory specifies a dialect of a language, such as DE_ch (Swiss dialect of German) or EN_uk (British dialect of English).
.charset specifies a character set other than ASCII, such as .88591 (the ISO 8859/1 character set).
.modifier specifies a collation sequence within a language and territory, such as @dictionary (dictionary order) or @telephone (telephone book order).
```

In practice, the syntax for `LC_COLLATE` is likely to be one of the following:

```plaintext
setenv LC_COLLATE language

or

setenv LC_COLLATE language_territory

or

setenv LC_COLLATE language_territory@modifier
```
There is no standardization of LC_ variable values between systems. Refer to the discussion of locale files on page C-27 for a better understanding of this syntax.

The following example sets the LC_COLLATE environment variable to specify the German telephone book sorting order:

```bash
setenv LC_COLLATE DE@telephone
```

An example of a result set influenced by the LC_COLLATE setting appears below. The following query will produce different result sets depending on whether the database is created with German or English collation:

```sql
SELECT hauptstadt FROM länder
WHERE land >= "Luxemburg" AND land <= "Portugal"
```

The query selects capitals of countries whose country names are alphabetically between Luxemburg and Portugal. With German collating, the capital of Austria (Vienna, or Wien in German) would be included in the result set, because the German word for Austria is Österreich. Österreich is between Luxemburg and Portugal in German; in US English in an NLS database it is not. Refer to the example in the section entitled “COLLCHAR” on page C-19.

**LC_CTYPE**

The X/Open-defined LC_CTYPE environment variable is used to specify which predefined set of characters can be legally contained in user-defined names. This includes the following:

- Names of databases, tables, and columns
- Names of views and indexes
- Names of form specification files in PERFORM
- Names of report specification files in ACE

LC_CTYPE also defines what characters result from conversion of lowercase to uppercase letters, and vice versa.

Along with LC_COLLATE, the database server saves the LC_CTYPE setting at database creation time. The LC_CTYPE setting is used throughout the lifetime of the database; therefore a database saved with a particular LC_CTYPE value will always accept the same set of naming characters.
X-Open Defined Variables

The formal syntax for LC_CTYPE is as follows:

```
setenv LC_CTYPE language
```

- **language**: a one to two character abbreviation such as FR for French, DE for German or EN for English.
- **_territory**: specifies a dialect of a language, such as DE_ch (Swiss dialect of German) or EN_uk (British dialect of English).
- **.charset**: specifies a character set other than ASCII, such as .88591 (the ISO 8859/1 character set).
- **@modifier**: specifies a collation sequence within a language and territory, such as @dictionary (dictionary order) or @telephone (telephone book order).

In practice, the syntax for setting LC_CTYPE is likely to be one of the following:

```
setenv LC_CTYPE language
```

or

```
setenv LC_CTYPE language_territory
```

or

```
setenv LC_CTYPE language.charset
```

There is no standardization of LC_ variable values between systems. Refer to the discussion of locale files on page C-27 for a better understanding of this syntax.

For example, if the language environment for your system is defined as US English (LANG=EN_us), but you wish to allow German characters in user-defined names, you would issue the following command at the operating system prompt:

```
setenv LC_CTYPE DE
```
Native Language Support Within INFORMIX-SQL

Subsequently, you could create a database whose name, gödel, contains the ö (o-umlaut) character without error:

```
CREATE DATABASE >>gödel
```

Enter the name you want to assign to the new database, then press RETURN.

------------------------------------------ Press CONTROL-W for Help ------------

Note: Use of NLS character sets other than EN_us may require you to perform the UNIX command `stty -istrip` before accessing I-SQL on some systems. This command enables you to type in characters from the keyboard that are outside the ASCII character set.

LC_MONETARY

The X/Open-defined LC_MONETARY environment variable is used to set the format of values of data type MONEY. This default format affects how monetary values are:

- Displayed and input on the screen
- Printed
- Input from ASCII files using LOAD
- Output to ASCII files using UNLOAD

LC_MONETARY specifies the locale-specific leading and trailing currency symbols, including their positions within a monetary value, and the decimal and thousands separators. Note that the decimal and thousands separators defined for monetary data by LC_MONETARY are distinct from the decimal and thousands separators defined for numeric data by LC_NUMERIC.

For example, the value 120.50 expressed as money appears as $120.50 if the locale is US English. However, by setting LC_MONETARY (or LANG) to UK English (EN_uk), the same value displays as £120.50, and by setting LC_MONETARY to German, the value displays as 120,50DM.

The setting in LC_MONETARY affects the following I-SQL keywords:

- USING expression in ACE
- FORMAT attribute in PERFORM
• PRINT statement in ACE
• LET statement in ACE, where a character string is receiving a monetary value

LC_MONETARY utilizes locale files the way LANG and the other LC_ variables do. It does not directly specify currency and separator symbols the way DBMONEY and DBFORMAT do.

When LC_MONETARY is set and not overridden by DBFORMAT or DBMONEY, logic is employed by I-SQL to determine issues such as the following:

• Whether a leading or trailing currency symbol is appropriate for this locale
• Whether the decimal portion of a monetary amount should be omitted, as for the Italian lira

The formal syntax for LC_MONETARY is as follows:

```
setenv LC_MONETARY language
```

```
language
```

A one to two character abbreviation such as FR for French, DE for German or EN for English.

```
_territory
```

Specifies a dialect of a language, such as DE_ch (Swiss dialect of German) or EN_uk (British dialect of English).

```
_charset
```

Specifies a character set other than ASCII, such as .88591 (the ISO 8859/1 character set).

```
@modifier
```

Specifies a collation sequence within a language and territory, such as @dictionary (dictionary order) or @telephone (telephone book order).

In practice, the syntax for setting LC_MONETARY is likely to be one of the following:

```
setenv LC_MONETARY language
```

```
setenv LC_MONETARY language_territory
```

C-32 Native Language Support Within INFORMIX-SQL
There is no standardization of LC_ variable values between systems. Refer to the discussion of locale files on page C-27 for a better understanding of this syntax.

**Usage**

The setting in LC_MONETARY determines the leading or trailing currency symbol, and the numeric and decimal separators. It adds the currency symbol and changes the separators displayed on the screen in a monetary field, and in the default format of a PRINT statement. For example, in a German locale the value 1234.56 prints as:

```
1234,56DM
```

DM stands for deutsche marks. In a screen form with the French or German locale values active, input by the user will be expected to contain commas, not periods, as decimal separators.

The setting in LC_MONETARY also affects the way format strings in the FORMAT attribute in `ACE` and the USING clause in `PERFORM` are interpreted. In these format strings, the period symbol (.) is not a literal character but a placeholder for the decimal separator specified by environment variables. Likewise, the comma symbol (,) is a placeholder for the thousands separator specified by environment variables. The dollar sign ($) is a placeholder for the leading currency symbol. The at symbol (@) is a placeholder for the trailing currency symbol. Thus, the format string $#,##0.## will format the value 1234.56 as follows in a US English locale:

```
$1,234.56
```

It displays as follows in a French locale:

```
£1.234,56
```

When money values are converted to character strings using the LET statement in `ACE`, both the default conversion and the conversion with a USING clause insert the locale-specific separators and currency symbol into the created strings, not the US English separators and currency symbol.
The LC_MONETARY setting is impacted by settings in DBFORMAT, DBMONEY, and LANG, according to the hierarchy of precedence. The following diagram illustrates the precedence of environment variables in specifying monetary formatting:

```
+----------------+
| DBFORMAT      |
+----------------+
| DBMONEY        |
+----------------+
| LC_MONETARY    |
+----------------+
| LANG           |
+----------------+
```

**Figure C-11  Order of Precedence of Monetary Environment Variables**

In the following statement, LC_MONETARY specifies the French currency symbol and format inherent in the ISO 8859/1 character set:

```
setenv LC_MONETARY FR_fr.88591
```

An example of printing a monetary value in ACE (without a USING clause) with German locale monetary formatting appears below. The statement:

```
PRINT 1234.56
```

will produce the result:

```
1234,56DM
```

With the USING clause added, the statement:

```
PRINT 1234.56 USING "#,###.###"
```
will print the result:

```
1.234,56DM
```

In 4GL, there is a distinction between the interpretation by the database server of monetary values enclosed in quotes and those not enclosed in quotes. In 4GL, unless values are contained in quotes, the database engine interprets all incoming data as being in ANSI SQL numeric data format (where the decimal separator is a period). However, in I-SQL, there is never any need to enclose monetary values in quotes, and doing so may generate an error.

**Note:** The use of LC_MONETARY for the specification of monetary formatting is preferable to the use of DBMONEY. DBMONEY is an obsolescent formatting construct that has been retained for backwards compatibility with older user-created programs. DBMONEY is also used by Informix engine tools that do not provide DBFORMAT. Note also that LC_MONETARY and LC_NUMERIC are the only means by which you can specify different formatting for monetary and numeric data. The formatting specified in DBFORMAT and DBMONEY apply to both monetary and numeric data. For more details, refer to the sections entitled “DBFORMAT” and “DBMONEY” in Appendix B, “Environment Variables.”

**LC_NUMERIC**

The X/Open-defined LC_NUMERIC environment variable sets the format for values of data types INTEGER, SMALLINT, DECIMAL, FLOAT, and SMALLFLOAT. This default format affects how numeric values are:

- Displayed and input on the screen
- Printed
- Input from ASCII files using LOAD
- Output to ASCII files using UNLOAD

LC_NUMERIC defines the numeric decimal and numeric thousands separators. Note that the decimal and thousands separators defined for numeric data by LC_NUMERIC are distinct from the decimal and thousands separators defined for monetary data by LC_MONETARY.

For example, the number 2345.67 in a US English locale displays as:

```
2,345.67
```
With LC_NUMERIC set for the French locale, where the thousands separator is the comma and the decimal separator the period, the value displays as:

```
2.345,67
```

The setting in LC_NUMERIC affects the following in I-SQL:
- USING expression in ACE
- FORMAT attribute in PERFORM
- PRINT statement in ACE
- LET statement in ACE, where a character string is receiving a numeric value

The formal syntax for setting the LC_NUMERIC variable is as follows:

```
setenv LC_NUMERIC language

language       a one to two character abbreviation such as FR for French, DE for German or EN for English.
_territory     specifies a dialect of a language, such as DE_ch (Swiss dialect of German) or EN_uk (British dialect of English).
_charset       specifies a character set other than ASCII, such as .88591 (the ISO 8859/1 character set).
@modifier      specifies a collation sequence within a language and territory, such as @dictionary (dictionary order) or @telephone (telephone book order).
```

In practice, the syntax for setting LC_NUMERIC is likely to be one of the following:

```
setenv LC_NUMERIC language
```

or

```
setenv LC_NUMERIC language_territory
```

There is no standardization of LC_variable values between systems. Refer to the discussion of locale files on page C-27 for a better understanding of this syntax.
Usage

The setting in LC_NUMERIC determines the numeric and decimal separators. It changes the separators displayed on the screen in a numeric field and in the default format of a PRINT statement. For example, the value 1234.56 will print or display as follows in a French or German locale:

```
1234,56
```

In the case of a screen form, in the French or German locale values input by the user will be expected to contain commas, not periods, as decimal separators.

The setting in LC_NUMERIC also affects the way format strings in the FORMAT attribute in ACE and the USING clause in PERFORM are interpreted. In these format strings, the period symbol (.) is not a literal character but a placeholder for the decimal separator specified by environment variables. Likewise, the comma symbol (,) is a placeholder for the thousands separator specified by environment variables. Thus, the format string `#,###.##` will format the value 1234.56 as follows in a US English locale:

```
1,234.56
```

but as follows in a French or German locale:

```
1.234,56
```

When numeric values are converted to character strings using the LET statement in ACE, both the default conversion and the conversion with a USING clause will insert locale-specific separators into the created strings, not US English separators.
The LC_NUMERIC setting is impacted by settings in DBFORMAT and DBMONEY, according to the hierarchy of precedence. The following diagram illustrates the precedence of environment variables in specifying numeric formatting:

![Figure C-12 Order of Precedence of Numeric Environment Variables](image)

The following command sets the LC_NUMERIC environment variable to specify a variety of German numeric formatting:

```plaintext
setenv LC_NUMERIC DE_de.88591
```

Here, DE means Deutsch (German), de represents Deutschland (Germany), and 88591 represents the Western European version of the 8-bit ISO character set (ISO 88591:1983).

An example of printing a numeric value in ACE (without a USING clause) with German locale numeric formatting appears below. The statement:

```plaintext
PRINT 1234.56
```

will produce the result:

```
1234,56
```
With the USING clause added, the statement:

```
PRINT 1234.56 USING "#,####.##"
```

will print the result:

```
1,234.56
```

In 4GL, there is a distinction between the interpretation by the database server of numeric values enclosed in quotes, and those not enclosed in quotes. In 4GL, unless values are contained in quotes, the database engine interprets all incoming data as being in ANSI SQL numeric data format (where the decimal separator is a period). However, in I-SQL, there is never any need to enclose numeric values in quotes, and doing so may generate an error.

### Informix-Defined Language and Formatting Variables

Informix-defined language and formatting variables, like the locale variables, specify the language and formatting environment of the user session. These Informix-defined variables interact with the locale variables according to the hierarchy of precedence.

#### DBFORMAT

The Informix-defined DBFORMAT environment variable specifies the default format in which the user enters and I-SQL inputs, displays, or prints values of the following data types:

- DECIMAL
- FLOAT
- SMALLFLOAT
- INTEGER
- SMALLINT
- MONEY

The default format specified in DBFORMAT affects how numeric and monetary values are:

- Displayed and input on the screen
- Printed
Informix-Defined Language and Formatting Variables

- Input from ASCII files using LOAD
- Output to ASCII files using UNLOAD

DBFORMAT is used to specify the leading and trailing currency symbols and the decimal and thousands separators. DBFORMAT specifies the currency symbols, not their default positions within a monetary value. Note that the decimal and thousands separators defined by DBFORMAT apply to both monetary and numeric data, and override the sets of separators established by LC_MONETARY and LC_NUMERIC.

The setting in DBFORMAT affects the following in I-SQL:

- USING expression in ACE
- FORMAT attribute in PERFORM
- PRINT statement in ACE
- LET statement in ACE, where a character string is receiving a monetary or numeric value

The DBFORMAT setting overrides settings in DBMONEY, LC_NUMERIC, and LC_MONETARY.

For a complete discussion of the DBFORMAT variable, refer to Appendix B, “Environment Variables.”

Note: The DBFORMAT variable, like DBMONEY and DBDATE, performs its role regardless of whether or not NLS is active (DBNLS set to 1 or 2). This is in contrast to LANG and the LC_ variables, which are only active when NLS is active.

DBMONEY

The Informix-defined DBMONEY environment variable specifies the display format for MONEY values.

For example, the DBMONEY setting

\,DM

prints the value 12345.67 as

12345, 67DM

DBMONEY formats monetary data in a rough country-specific format, but provides no facility for a thousands separator. For complete information on DBMONEY, see Appendix B, “Environment Variables.”
Informix-Defined Language and Formatting Variables

The precedence relationship between DBFORMAT, DBMONEY, and LC_MONETARY is illustrated below:

![Diagram showing precedence relationship]

DBMONEY represents syntax from older versions of the product set. It is recommended that you use the LC_MONETARY environment variable for specifying monetary format. DBMONEY is retained only for compatibility with older versions.

The DBMONEY variable, like DBFORMAT and DBDATE, performs its role regardless of whether or not NLS is active (DBNLS set to 0 or 1). This is in contrast to LANG and the LC_ variables, which are only active when NLS is active.

**DBDATE**

Refer to Appendix B, “Environment Variables,” for a complete discussion of DBDATE. The following points pertain to the relationship between DBDATE and NLS.

- DBDATE is insensitive to NLS locale, and to the effects of DBLANG. Since month and day are displayed as numeric values in all available DBDATE formats, they will not change with DBLANG the way formats containing character month names do.
- Date formatting specified in a USING clause or FORMAT attribute will override the formatting specified in DBDATE.
LOAD and UNLOAD Statements

- The LANG setting will specify the default value for DBDATE in an active NLS environment on HP and IBM systems. On SUN systems LANG has no influence on the default for DBDATE.

- The DBDATE variable, like DBFORMAT and DBMONEY, performs its role regardless of whether or not NLS is active (DBNLS set to 1 or 2). This is in contrast to LANG and the LC_ variables, which are only active when NLS is active.

Note: Although the XPG3 specification includes the ability to create locale-specific formatting of date and time data by way of the LC_TIME category, this is not supported in the 6.0 Informix tools.

LOAD and UNLOAD Statements

NLS affects the text files produced by UNLOAD. It also determines the format in which it expects LOAD files. The UNLOAD statement uses the environment variables DBFORMAT, DBMONEY, LC_NUMERIC, LC_MONETARY, and DBDATE to determine its output format. The precedence of these format specifications is consistent with that of forms and reports. The LOAD statement expects incoming data in the format specified by the environment. If there is an inconsistency between the format of the data being loaded and the value of the formatting variables, an error is reported, and the LOAD is cancelled.

The order of precedence for monetary data is as illustrated in Figure C-11 on page C-34. The precedence order for numeric data is in Figure C-12 on page C-38. Date data is currently only affected by DBDATE.

UNLOAD utilizes the user’s language and formatting environment information as follows:

- For dates, as specified in DBDATE.
- For numbers, as specified in DBFORMAT, DBMONEY, and LC_NUMERIC (although not allowing thousands separators).
- For money values, as specified in DBFORMAT, DBMONEY, and LC_MONETARY (although not allowing thousands separators).

LOAD uses the user’s environment information as follows:

- For dates, as specified in DBDATE.
- For numbers, as specified in DBFORMAT, DBMONEY, and LC_NUMERIC (including thousands separators).
- For money values, as specified in DBFORMAT, DBMONEY, and LC_MONETARY including thousands separators and currency symbols,
but not following the usual negative numbering conventions, that is, \((-\) =

This set of behaviors is adequate for unloading and loading between two non-
NLS databases, or between two databases with the same locale. For unloading
and loading between dissimilar database locales, it is necessary to perform
one of the following two sequences of operations:

• At the source database, set DBNLS to 2 and LANG to the locale of the
destination database, then perform the UNLOAD operation. LOAD the
resulting text file at the destination database using normal settings for that
database.

or

• At the source database, UNLOAD using normal settings. At the
destination database, set DBNLS to 2 and LANG to the locale of the source
database, and perform the LOAD.

Either of these approaches works equally well. An example of the first
approach would be a user in Paris unloading a database created with
LANG=FR, and loading the data in Munich to a database created with
LANG=DE. The user in Paris sets DBNLS to 2 and LANG to DE at the client side
before performing the unload. The data is sent to Munich, where the user
there performs the load normally (that is, with DBNLS set to 1 and consistent
locale variables).

Note that if cross-locale database load/unload sequences take place without
following one of the two procedures identified above, the following can
occur:

• Multiplying or dividing numeric or monetary values by orders of magni-
tude if thousands and decimal separators are different between locales.
For example, loading an English UNLOAD file into a German database
causes every value to be divided by 1000, because of the switched roles of
the comma and period symbols.

• Generating LOAD errors if characters are encountered that are outside the
character set of the database.

• Generating LOAD errors if uninterpretable currency symbols are
encountered.

FORMAT and USING

The USING operator is typically used in PRINT statements, but you can also
use it with LET to assign the formatted value to a character variable. The
FORMAT attribute is used strictly in screen forms. Format strings in USING

Native Language Support Within INFORMIX-SQL  C-43
and FORMAT have identical meanings, except that a FORMAT operator’s formatting string cannot display currency symbols, whereas USING format strings can. NLS variables influence the results obtained from USING and FORMAT strings identically, except for currency symbols not being available in FORMAT. The order of variable precedence is as illustrated in the sections on LC_MONETARY and LC_NUMERIC.

The following classifications apply to format strings:

- A format string is said to be a **monetary formatter** if either one of the following two conditions hold:
  - The format string is formatting monetary data.
  - The format string contains either of the currency placeholder symbols ( $ or @).

  Format strings in the USING operator can be monetary formatters. Format strings in the FORMAT attribute cannot.

- A format string is said to be a **date formatter** if it is not a monetary formatter and it contains one of the following tokens: `mm`, `mmm`, `dd`, `ddd`, or `yy`, `yyyy`.

- A format string is said to be a **numeric formatter** if it is neither a monetary formatter nor a date formatter.

**Monetary and Numeric Formatting**

The following format string symbols have new meanings in 6.0 1-SQL. They are influenced by settings in the LC_MONETARY, LC_NUMERIC, DBFORMAT, DBMONEY, DBDATE, and DBLANG environment variables. Any valid formatting symbol that is not mentioned below has the same meaning it had in the pre-6.0 1-SQL.

- Any appearance of a comma in the format string for non-DATE data is interpreted as a thousands separator. This symbol is not a literal. The comma stands for either the numeric thousands separator or the monetary thousands separator specified by the format environment. If the format string is a monetary-formatter, then this symbol stands for the monetary thousands separator. If the format string is a numeric-formatter, then this symbol stands for the numeric thousands separator. If the format string is a date-formatter, then this symbol stands for itself; the comma remains a comma.

- Any appearance of a period in the format string for non-DATE data is interpreted as a decimal separator. This symbol is not a literal. The period stands for the decimal separator specified by the format environment. If the format string is a monetary-formatter, then this sym-
bol stands for the monetary decimal separator. If the format string is a numeric-formatter, then this symbol stands for the numeric decimal separator. If the format string is a date-formatter, then this symbol stands for itself; the period remains a period.

$ In a USING operator format string, the dollar sign is the leading currency formatting symbol. The dollar sign is not valid syntax for the FORMAT attribute, because the FORMAT attribute cannot format the MONEY data type. Groups of dollar signs stand for formatting space provided for the currency symbols that precede a monetary value. A group of dollar signs in a row will provide formatting space for the leading currency symbol or string specified in the format environment. The USING clause right-justifies currency symbols within the sequence of $ signs.

@ In a USING operator format string, the at symbol (@) is the trailing currency symbol formatting symbol. This symbol, which is new to USING format strings, is similar in purpose to the dollar sign. Groups of @ symbols stand for formatting space provided for the currency symbol or string that follows a monetary value. The USING clause left-justifies currency symbols within the sequence of @ signs.

For example, with LANG set to DE, and columns m and c representing monetary and character data, respectively, a German ACE form specification might use the following statements to format money:

```
LET m = 1234.56
LET c = m USING "#,##0.##@@"
PRINT c
```

The result would appear as shown:

```
1.234,56DM
```

With LANG set to EN_us and DBFORMAT set to:

```
$£,:,.p
```
a British **ACE** form specification might use the following statements to format money:

```sql
LET m = 1234.56
LET c = m USING "$$.###.###"  
PRINT c
```

The result would be as follows:

`£1,234.56p`

Note the use of both a leading and a trailing currency symbol in the latter example.

### Date Formatting

If you use the **USING** operator or **FORMAT** attribute to format a **DATE** value, **USING** or **FORMAT** takes precedence over the **DBDATE** environment variable. The format-string for a date can be a combination of the characters `m`, `d`, and `y`:

- `dd`: day of the month as a 2-digit number (01 through 31 or less)
- `ddd`: day of the week as a 3-letter abbreviation (Sun through Sat)
- `mm`: month as a 2-digit number (01 through 12)
- `mmm`: month as a 3-letter abbreviation (Jan through Dec)
- `yy`: year as a 2-digit number in the 1900s (00 through 99)
- `yyyy`: year as a 4-digit number (0001 through 9999)

The **DBLANG** setting and installation of appropriate message files will determine the set of available 3-character weekday name `ddd` and month name `mmm` abbreviations.

### LET Statements

When **LET** statements are employed without **USING** clauses, certain default conversions are employed in an **NLS** environment. In particular, there is the case where a monetary or numeric value is converted to a character value (or the character variable is decoded into monetary or a numeric value). In this situation, the conversion follows the formatting rules established by the rel-
evant hierarchy of environment variables. The following example (in which columns \(c\) and \(c_2\) are of CHARACTER type, \(m\) is MONEY and \(d\) is DECIMAL) demonstrates this:

```sql
LET m = 1234.56; LET d = 9876.54
LET c = m; LET c2 = d
PRINT c, c2
```

The result in a LANG=DE environment is:

```
1.234,56DM  9876,54
```

When LET statements assign monetary and numeric constants to variables and NLS is active, the interpretation of the constants is governed by locale if the constant is in quotes, and governed by US ANSI if the constant is not in quotes. The following example (in which \(m\) and \(m_2\) are of type MONEY) illustrates this:

```sql
DEFINE m1 MONEY, m2 MONEY
LET m1 = 1234.56; LET m2 = "1234,56DM"
```

In a LANG=DE environment, this example would work correctly in I-SQL. However, in a LANG=EN_us (US English) environment, the constant being assigned to \(m_2\) would trigger an error.

**Note:** For a user performing data entry into a form, locale is active for values being entered, and quotes are not necessary.

---

### Multiple Locale Support

Under version 6.0, an NLS database is limited to one locale per database. The database locale is specified by the contents of the LC_COLLATE and LC_CTYPE environment variables stored in the database at database creation time. However, multiple databases built with different environment variable settings can reside on the same server. Furthermore, multiple users can access databases with different locales on the same server at the same time. The locale of the server machine itself is irrelevant for the purposes of determining whether or not access will be permitted; it is strictly determined by the consistency checking process between user and database as described earlier.
Language Supplements

With NLS, in order to use different languages, you need the necessary language supplements to install on the system. For example, to use German and Spanish with the I-SQL product, you need to buy an I-SQL package and two language supplements: one German and one Spanish. The installation script for the language supplements creates the appropriate directories.

Figure C-14 Multiple NLS Locales in 6.0 Architecture
In order to know the exact language supplement to purchase, you need to know the following information:

- The system (Sun, HP, and so on)
- Operating system version
- NLS version (if purchased separately)
- Information about your locale (LANG setting)

Contact your local Informix sales office for more information on language supplements.

Glossary of NLS terms

**ANSI** Acronym for the American National Standards Institute. This group sets standards for the computer industry, including standards for SQL languages. In NLS, ANSI refers to the ANSI standard for numeric formatting, which is the format in which numeric and monetary data is stored in the database, regardless of locale.

**ASCII** Acronym for the American Standards Committee for Information Interchange. Often used to describe an ordered set of printable and non-printable characters used in computers and telecommunication. In NLS, ASCII can refer to either the ASCII character set, which is a set of 128 characters and their numeric representations, or ASCII collation, which is the ordering of characters in the ASCII character set by their numeric values.

**category** In NLS, category refers to each feature of the locale, pertaining to one aspect of the language and formatting environment. Standard NLS categories include collation, character set, monetary formatting, numeric formatting, and date/time formatting. Date/time formatting is not a supported NLS category on Version 6.0 Informix tools.

**character set** A set of valid characters, each of which corresponds to an integer value from 0 to 255 (8-bit) or 0 to 127 (7-bit). In NLS, a character set is specified by way of the name of a character set file. A character set file contains all of the characters and their corresponding numeric values. The NLS character sets are provided to meet the needs of European countries. They extend the ASCII character set used for English, which consists of 128 characters (there are 128 possible combinations of 7 bits of data), to one of several possible sets of 256 characters (based on 8 bits per character). The most prevalent of the 256-character sets are the ISO 8859-1 and 8859-2 standards.
**Glossary of NLS terms**

**consistency checking**  The process of verifying that the user session’s collation and character set variable settings match settings in the database locale. In the Implicit and Explicit NLS environments, consistency checking determines whether or not access to a database is permitted. In the Open NLS environment, consistency checking is overridden.

**database locale**  Locale of the user at the time of database creation, permanently saved in database system tables and consulted when the database is accessed. Currently only the collation (LC_COLLATE) and character set (LC_CTYPE) variables are saved.

**environment variable**  A variable with an assigned value that is maintained by the operating system and made available to all programs.

**Explicit NLS environment**  The Explicit NLS environment is defined as DBNLS set to 1 and COLLCHAR unset. The Explicit NLS environment supports creation of both CHAR and NCHAR columns in a table. In Explicit NLS, CHAR (and VARCHAR) data columns sort and compare according to US English ASCII, whereas NCHAR (and NVARCHAR) data sort and collate according to collation and character set settings of the database locale. The capability of users of a tool in the Explicit environment to separately declare nationalized and non-nationalized character data types allows them to avoid the resource-intensive NCHAR sorting and comparison when it is not needed. However, the Explicit environment also can create confusion for the user, and limits the ability of third party tools to access a database that has been created in this environment. INFORMIX-SQL version 6.0 can utilize the Explicit environment (this is also true of ESQL/C, ESQL/COBOL, and DB-Access). It is, however, recommended that INFORMIX-SQL users use the Implicit environment with NLS databases.

**GLS**  Global Language Support. A proposed standard that would include Asian and Arabic language representation.

**identifier**  A sequence of letters, digits, and underscores (\_) that represents the name of a database, table, column, screen form, program variable, cursor, function, index, window, menu, synonym, alias, view, PREPAREd object, constraint, or report.

**Implicit NLS environment**  The Implicit NLS environment is defined as DBNLS set to 1 and COLLCHAR set to 1. INFORMIX-SQL operating in the Implicit NLS environment produces NLS applications in which all character data is sorted and compared according to the collation sequence established at the time of database creation. All character columns defined by the user in the Implicit environment appear to the user as type CHAR (or VARCHAR, if variable-length), but are interpreted by the server as type NCHAR (or NVARCHAR) if the
database is non-US-English. This is the recommended environment for access and creation of NLS databases, since it minimizes the possibilities for data corruption, and allows pre-6.0 applications to run unmodified against an NLS database.

**implicit mapping**  
The process which makes it seem to the user of a database that they are using CHAR or VARCHAR data types, when in reality NCHAR and NVARCHAR types are in use at the server, respectively.

**Informix-defined**  
A classification describing environment variables which did not originate in the X-Open Portability Guide version 3 (XPG3) standard but are relevant to NLS. This includes DBNLS, COLLCHAR, DBAPICODE, DBLANG, DBFORM, DBFORMAT, DBMONEY, and DBDATE. Informix-defined variables are consistent in syntax and meaning across platforms. This is in contrast to X-Open defined variables, which rely on facilities provided by the computer manufacturer, and can vary from system to system.

**language and formatting variable**  
An environment variable that controls aspects of the language and formatting environment. These include LANG, the LC_ variables, DBFORMAT, DBMONEY and DBDATE. This classification is in contrast with meta-environment variables, which control aspects of the NLS meta-environment, such as whether or not NLS is active or whether or not implicit mapping is turned on.

**language supplement**  
A product obtained from an Informix sales office that provides settings for an additional national language when installed in a 6.0 server environment.

**LC_ variable**  
Any of the four environment variables LC_COLLATE, LC_CTYPE, LC_MONETARY, or LC_NUMERIC that begin with the letters LC followed by the underbar (_) symbol.

**locale file**  
A file installed on the system, which specifies language or formatting behavior for one or more settings of one or more LC_ variables. The format, naming and use by the system of locale files varies between computer manufacturers. Locale files are installed by installing language supplements.

**locale-sorted**  
Character data which sorts in the order specified by the LC_COLLATE environment variable. Corresponds at the server to the data types NCHAR and NVARCHAR.

**meta-environment variable**  
An environment variable that controls aspects of the NLS meta-environment, such as whether or not NLS is active or whether or not implicit mapping is turned on. These include DBNLS, COLLCHAR,
Native Language Support

Based on the X/Open Portability Guide Version 3 (XPG3) standard, it specifies a means for localization of software to European geographical regions without the need for alteration to user applications. NLS is available only on UNIX systems that support X/Open NLS libraries.

NLS

Acronym for Native Language Support.

NLS database

An NLS database is a database created with NLS environment settings active (DBNLS set to 1 or 2).

NLS environment

A combination of user environment variable settings in which DBNLS is set to 1 or 2. There are three NLS environments: Implicit, Explicit and Open. Different NLS environments are selected by way of different combinations of the DBNLS and COLLCHAR environment variables. In NLS environments: 1) the LANG and LC_ variables are considered in operations, 2) collation is specified by LC_COLLATE unless the column is type NCHAR and the environment is Explicit, 3) user defined names can contain any characters contained in the character set specified by LC_CTYPE, 4) non-NLS databases can be accessed but not created, 5) LC_COLLATE and LC_CTYPE values are saved in system tables at database creation time.

Non-NLS database

A non-NLS database is a database created in a pre-6.0 server environment, or in the Non-NLS environment. It can be accessed while an NLS environment is active, but LANG and the LC_ variables have no effect.

Non-NLS environment

The Non-NLS environment is defined as DBNLS unset or set to zero. In this environment: 1) databases created are non-NLS databases, 2) NLS databases cannot be accessed, 3) LANG and the LC_ variables have no effect, 4) character collation order is US English, 4) only ASCII characters may be used in identifiers, 5) default monetary and numeric formats are ANSI compliant.

Open NLS environment

The Open NLS environment is defined as DBNLS set to 2 and COLLCHAR set to 1. Open NLS differs from Implicit and Explicit NLS in that third party tools can use the Open NLS environment to access NLS databases by way of SQL commands to the database engine. The tool sends queries to the server for processing and gets back results that are properly sorted from the standpoint of the database locale, without the tool knowing what locale it is accessing. The Open NLS environment can also be used to perform LOAD and UNLOAD operations between locales.
**user locale**  The set of X/Open-defined environment variable settings currently active in a user session.

**X-Open defined**  A classification describing environment variables which originated in the X-Open Portability Guide version 3 (XPG3) standard. This includes LANG and the LC_ variables. Collectively the values of the X-Open variables define the locale. X-Open variables rely on facilities provided by the computer manufacturer, and can vary from system to system in syntax and meaning. X-Open defined variables are distinguished from Informix-defined environment variables, which are consistent across platforms.

**XPG3**  X-Open Portability Guide version 3. The current standard for NLS on UNIX systems.
Glossary of NLS terms
Modifying termcap and terminfo

You can include color and graphics characters in your PERFORM screen forms. The meaning of these characters, however, is terminal dependent. To determine terminal-dependent characteristics, INFORMIX-SQL (I-SQL) uses the information in the termcap file or in the terminfo directory. I-SQL uses the INFORMIXTERM environment variable to determine whether to use termcap or terminfo. For more information about INFORMIXTERM, read the discussion of environment variables in the ‘Environment Variables’ appendix or in the Preface.

With I-SQL, Informix distributes termcap files that contain additional capabilities for many common terminals (such as the Wyse 50 and the Televideo 950). This appendix describes these capabilities, as well as the general format of termcap and terminfo entries.

Since terminfo does not support color, you can only use I-SQL color functionality with termcap. If you want to use color in I-SQL screen forms, you must set INFORMIXTERM to termcap.

You can use the information in this appendix, combined with the information in your terminal manual, to modify the contents of your termcap file or terminfo files. This appendix is divided into two main sections, termcap and terminfo. Depending on which you are using, you should read the appropriate section.
When I-SQL is installed on your system, a `termcap` file is placed in the `etc` subdirectory of `$INFORMIXDIR`. This file is a superset of an operating system `termcap` file. The Informix `termcap` file contains additional capabilities for many terminals. You may want to modify this file further in the following instances:

- The entry for your terminal has not been modified to include color-change and intensity-change capabilities.
- You want to specify or alter the graphics characters used for borders.

**Note:** Some terminals cannot support color or graphics characters. You should read this appendix and the user guide that comes with your terminal to determine whether or not the changes described in this appendix are applicable to your terminal.

### Format of a `termcap` Entry

This section describes the general format of `termcap` entries. For a complete description of `termcap`, refer to your operating system documentation.

A `termcap` entry contains a list of names for the terminal, followed by a list of the terminal’s capabilities. There are three types of capabilities:

- Boolean capabilities
- Numeric capabilities
- String capabilities

All `termcap` entries have the following format:

- `ESCAPE` is specified as a backslash (`\`) followed by the letter E, and `CONTROL` is specified as a caret (`^`). Do not use the `ESCAPE` or `CONTROL` keys to indicate escape sequences or control characters in a `termcap` entry.
- Each capability, including the last one in the entry, is followed by a colon (`:`).
- Entries must be defined on a single logical line; a backslash (`\`) appears at the end of each line that wraps to the next line.
Figure D-1 shows a basic **termcap** entry for the Wyse 50 terminal:

```plaintext
# Entry for Wyse 50:

w5|wy50|wyse50:\
 :if=/usr/lib/tabset/std: 
 :al=\EE:am=bs:ce=\Et:cm=\E=+ % + :cl=\E*:co#80: 
 :dc=\EW:dl=\ER:ho="":el=\kh="":im=\ic=\EQ:in:li#24:\ 
 :nd=\l:pt=se=\EG0:so=\EG4:sg#1:ug#1:\ 
 :up=\k:ku=\k:kd=\j:kl=\h:kr=\l:kb=\j: 
 :k0=\A0\M:k1=\AA\M:k2=\AB\M:k3=\AC\M:k4=\AD\M: 
 :k5=\AE\M:k6=\AF\M:k7=\AG\M: 
 :HI=\|:Po=\R:Pe=\T:
```

**Figure D-1 Wyse 50 termcap Entry**

*Note: Comment lines begin with a pound sign (#).*

**Terminal Names**

A **termcap** entry starts with one or more names for the terminal, each of which is separated by a vertical bar (|). For example, the **termcap** entry for the Wyse 50 terminal starts with the following line:

```
w5|wy50|wyse50:\
```

The **termcap** entry can be accessed using any one of these names.

**Boolean Capabilities**

A Boolean capability is a two-character code that indicates whether or not a terminal has a specific feature. If the Boolean capability is present in the **termcap** entry, the terminal has that particular feature. Figure D-2 shows some of the Boolean capabilities for the Wyse 50 terminal:

```plaintext
:bs:am:
# bs  backspace with CTRL-H
# am  automatic margins
```

**Figure D-2 Boolean Capabilities for the Wyse 50**
**Numeric Capabilities**

A numeric capability is a two-character code followed by a pound symbol (#) and a value. Figure D-3 shows the numeric capabilities for the number of columns and the number of lines on a Wyse 50 terminal:

```
:co#80:li#24:
```

- `co` number of columns in a line
- `li` number of lines on the screen

**Figure D-3  Numeric Capabilities for the Wyse 50**

Similarly, `sg` is a numeric capability that indicates the number of character positions required on the screen for reverse video. The entry `:sg#1:` indicates that a terminal requires one additional character position when reverse video is turned on or off. If you do not include a particular numeric capability, I-SQL assumes that the value is zero.

**String Capabilities**

A string capability specifies a sequence that can be used to perform a terminal operation. A string capability is a two-character code followed by an equal sign (=) and a string ending at the next delimiter (:).

Most `termcap` entries include string capabilities for clearing the screen, cursor movement, arrow keys, the underscore, function keys, and so on. Figure D-4 shows many of the string capabilities for the Wyse 50 terminal:

```
:ce=\Et:cl=\E*:\  
:nd=^L:up=^K:\  
:so=\EG4:se=\EG0:\  
:ku=^K:kd=^J:kr=^L:kl=^H:\  
:k0=^A@^M:k1=^AA^M:k2=^AB^M:k3=^AC^M:\
```

- `ce\Et` clear to end of line
- `cl=\E*` clear the screen
- `nd=^L` non-destructive cursor right
- `up=^K` up one line
- `so=\EG4` start stand-out
- `se=\EG0` end stand-out
# ku=^K up arrow key
# kd=^J down arrow key
# kr=^L right arrow key
# kl=^H left arrow key
#
# k0=^A^M function key F1
# k1=^AA^M function key F2
# k2=^AB^M function key F3
# k3=^AC^M function key F4

Figure D-4  String Capabilities for the Wyse 50

Specifying Graphics Characters in Screen Forms

I-SQL uses characters defined in the `termcap` file to draw the borders of boxes and other rectangular shapes that appear in a screen form. If no characters are defined in the `termcap` file, I-SQL uses the hyphen ( – ) for horizontal lines, the vertical bar ( | ) for vertical lines, and the plus sign ( + ) for corners.

The `termcap` file provided with I-SQL contains border character definitions for many common terminals. You can look at the `termcap` file to see if the entry for your terminal has been modified to include these definitions. If your terminal entry does not contain border character definitions, or if you want to specify alternative border characters, you or your system administrator can modify the `termcap` file.

Perform the following steps to modify the definition for your terminal type in the `termcap` file:

1. Determine the escape sequences for turning graphics mode on and off. This information is located in the manual that comes with your terminal. For example, on Wyse 50 terminals, the escape sequence for entering graphics mode is ESC H^B and the escape sequence for leaving graphics mode is ESC H^C.

   **Note:** Terminals without a graphics mode do not have this escape sequence. The procedure for specifying alternative border characters on a non-graphics terminal is discussed at the end of this section.

2. Identify the ASCII equivalents for the six graphics characters that I-SQL requires to draw the border. (The ASCII equivalent of a graphics character is the key you would press in graphics mode to obtain the indicated character.)
Figure D-5 shows the graphics characters and the ASCII equivalents for a Wyse 50 terminal.

<table>
<thead>
<tr>
<th>Position</th>
<th>Graphics Character</th>
<th>ASCII Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>upper left corner</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>lower left corner</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>upper right corner</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>lower right corner</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>horizontal</td>
<td>-</td>
<td>z</td>
</tr>
<tr>
<td>vertical</td>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>

3. Edit the `termcap` entry for your terminal. Note: You may want to make a copy of your `termcap` file before you edit it. You can use the `TERM=value` environment variable to point to whichever copy of the `termcap` file you want to access.

Use the format `termcap-capability=value` to enter values for the following `termcap` capabilities:

- `gs` The escape sequence for entering graphics mode. In the `termcap` file, `ESCAPE` is represented as a backslash (`\`) followed by the letter E; `CONTROL` is represented as a caret (`^`). For example, the Wyse 50 escape sequence `ESCAPE-H CONTROL-B` is represented as `\EH^B`.

- `ge` The escape sequence for leaving graphics mode. For example, the Wyse 50 escape sequence `ESCAPE-H CONTROL-C` is represented as `\EH^C`.

Again, this information should be located in the manual that comes with your terminal.
**gb**  The concatenated, ordered list of ASCII equivalents for the six graphics characters used to draw the border. Use the following order:

- upper left corner
- lower left corner
- upper right corner
- lower right corner
- horizontal lines
- vertical lines

Follow these guidelines when you insert information in the termcap entry:

- Delimit entries with a colon ( : ).
- End each continuing line with a backslash ( \ ).
- End the last line in the entry with a colon.

For example, if you are using a Wyse 50 terminal, you would add the following information in the termcap entry for the Wyse 50:

```
:gs=\EH^B:fg=\EH^C:gb=2135z6:
```

Terminals Without Graphics Capabilities

For terminals without graphics capabilities, you must enter a blank value for the gs and ge capabilities. For gb, enter the characters you want I-SQL to use for the window border.

The following example shows possible values for gs, ge, and gb in an entry for a terminal without graphics capabilities. In this example, window borders would be drawn using underscores ( _ ) for horizontal lines, vertical bars ( | ) for vertical lines, periods ( . ) for the top corners, and vertical bars ( | ) for the lower corners.

```
:gs=:ge=:gb=.|.|_:|
```
I-SQL uses the graphics characters in the termcap file when you specify a screen border in a PERFORM screen.

Adding Color and Intensity

Many of the terminal entries in the Informix termcap file (in the etc subdirectory of $INFORMIXDIR) have been modified to include color or intensity capabilities or both. You can view the termcap file to determine if the entry for your terminal type includes these capabilities. If your terminal entry includes the ZA capability, your terminal is set up for color or intensity or both. If it does not, you can add color and intensity capabilities by using the information in this section. The following topics are outlined in this section:

• Color and intensity
• The ZA capability
• Stack operations
• Examples

You should understand these topics before you modify your terminal entry.

Color and Intensity Attributes

You can display your PERFORM screen on either a monochrome or a color terminal. If you set up the termcap files as described here, color attributes and intensity attributes are related, as shown in Figure D-6.

<table>
<thead>
<tr>
<th>Number</th>
<th>Color Terminal</th>
<th>Monochrome Terminal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>WHITE</td>
<td>NORMAL</td>
</tr>
<tr>
<td>1</td>
<td>YELLOW</td>
<td>BOLD</td>
</tr>
<tr>
<td>2</td>
<td>MAGENTA</td>
<td>BOLD</td>
</tr>
<tr>
<td>3</td>
<td>RED</td>
<td>BOLD+</td>
</tr>
<tr>
<td>4</td>
<td>CYAN</td>
<td>DIM</td>
</tr>
<tr>
<td>5</td>
<td>GREEN</td>
<td>DIM</td>
</tr>
<tr>
<td>6</td>
<td>BLUE</td>
<td>DIM+</td>
</tr>
<tr>
<td>7</td>
<td>BLACK</td>
<td>INVISIBLE</td>
</tr>
</tbody>
</table>

Figure D-6  Color-Monochrome Correspondence

The background for colors is BLACK in all cases. In Figure D-6, the + signifies that, if the keyword BOLD is indicated as the attribute, the field will be RED on a color terminal, or if the keyword DIM is indicated as the attribute, the field will be BLUE on a color terminal.
In either color or monochrome mode, you can add the REVERSE, BLINK, or UNDERLINE attributes if your terminal supports them. You can select only one of these three attributes.

The ZA String Capability

I-SQL uses a parameterized string capability ZA in the termcap file to determine color assignments. Unlike other termcap string capabilities that you set equal to a literal sequence of ASCII characters, ZA is a function string that depends on four parameters:

- **Parameter 1 (p1)** Color number between 0 and 7 (see Figure D-6)
- **Parameter 2 (p2)** 0 = Normal; 1 = Reverse
- **Parameter 3 (p3)** 0 = No-Blink; 1 = Blink
- **Parameter 4 (p4)** 0 = No-Underscore; 1 = Underscore

ZA uses the values of these four parameters and a stack machine to determine which characters to send to the terminal. The ZA function is called and these parameters are evaluated when a color attribute specification is encountered during PERFORM. You can use the information in your terminal manual to set the ZA parameters to the correct values for your terminal.

To define the ZA string for your terminal, you use stack operators to push and pop values onto and off the stack. The next section describes several stack operators. Use these descriptions and the subsequent examples to understand how to define the string for your terminal.

Stack Operations

The ZA string uses stack operations to either push values onto the stack or pop values off the stack. Typically, the instructions in the ZA string push a parameter onto the stack, compare it to one or more constants, and then send an appropriate sequence of characters to the terminal. More complex operations are often necessary and, by storing the display attributes in static stack machine registers (named a through z), you can achieve terminal-specific optimizations.

A summary follows of the different stack operators you can use to write the descriptions. For a complete discussion of stack operators, consult your operating system documentation.

Operators that Send Characters to the Terminal

- `%d` pops a numeric value from the stack and sends a maximum of three digits to the terminal. For example, if the value 145 is at the top of the stack, `%d` pops the value off the stack and sends the ASCII repre-
sentations of 1, 4, and 5 to the terminal. If the value 2005 is at the top of the stack, \%d pops the value off the stack and sends the ASCII representation of 5 to the terminal.

\%
\%2d pops a numeric value from the stack and sends a maximum of two digits to the terminal, padding to two places. For example, if the value 145 is at the top of the stack, \%2d pops the value off the stack and sends the ASCII representations of 4 and 5 to the terminal. If the value 5 is at the top of the stack, \%2d pops the value off the stack and sends the ASCII representations of 0 and 5 to the terminal.

\%
\%3d pops a numeric value from the stack and sends a maximum of three digits to the terminal, padding to three places. For example, if the value 7 is at the top of the stack, \%3d pops the value off the stack and sends the ASCII representations of 0, 0, and 7 to the terminal.

\%
\%c pops a single character from the stack and sends it to the terminal.

**Operators that Manipulate the Stack**

\%
\%p[1-9] pushes the value of the specified parameter on the stack. The notation for parameters is p1, p2, ... p9. For example, if the value of p1 is 3, \%p1 pushes 3 on the stack.

\%
\%P[a-z] pops a value from the stack and stores it in the specified variable. The notation for variables is Pa, Pb, ... Pz. For example, if the value 45 is on the top of the stack, \%Pb pops 45 from the stack and stores it in the variable Pb.

\%
\%g[a-z] gets the value stored in the corresponding variable (P[a-z]) and pushes it on the stack. For example, if the value 45 is stored in the variable Pb, \%gb gets 45 from Pb and pushes it on the stack.

\%
\%`c` pushes a single character on the stack. For example, \%`k` pushes k on the stack.

\%
\%[n] pushes an integer constant on the stack. The integer can be any length and can be either positive or negative. For example, \%{0} pushes the value 0 on the stack.

\%
\%S[a-z] pops a value from the stack and stores it in the specified static variable. (Static storage is nonvolatile since the stored value remains from one attribute evaluation to the next.) The notation for static variables is Sa, Sb, ... Sz. For example, if the value 45 is on the top of the stack, \%Sb pops 45 from the stack and stores it in the static variable Sb. This value is accessible for the duration of the I-SQL program.
%G[a-z] gets the value stored in the corresponding static variable (S[a-z]) and pushes it on the stack. For example, if the value 45 is stored in the variable Sb, %Gb gets 45 from Sb and pushes it on the stack.

**Arithmetic Operators**

Each arithmetic operator pops the top two values from the stack, performs an operation, and pushes the result on the stack.

- `%+` Addition. For example, %{2}%{3}+% is equivalent to 2+3.
- `%-` Subtraction. For example, %{7}%{3}%- is equivalent to 7-3.
- `%*` Multiplication. For example, %{6}%{3}%* is equivalent to 6*3.
- `/` Integer division. For example, %{7}%{3}/ is equivalent to 7/3 and produces a result of 2.
- `%m` Modulus (or remainder). For example, %{7}%{3}%m is equivalent to (7 mod 3) and produces a result of 1.
**Bit Operators**

The following bit operators pop the top two values from the stack, perform an operation, and push the result on the stack:

%& Bit-and. For example, %\{12\} %\{21\} %& is equivalent to (12 and 21) and produces a result of 4.

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 0</td>
<td>12</td>
</tr>
<tr>
<td>1 0 1 0 1</td>
<td>21</td>
</tr>
</tbody>
</table>

```
------------------------- and
0 0 1 0 0 = 4
```

%| Bit-or. For example, %\{12\} %\{21\} %| is equivalent to (12 or 21) and produces a result of 29.

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 0</td>
<td>12</td>
</tr>
<tr>
<td>1 0 1 0 1</td>
<td>21</td>
</tr>
</tbody>
</table>

```
------------------------- or
1 1 1 0 1 = 29
```

%^ Exclusive-or. For example, %\{12\} %\{21\} %^ is equivalent to (12 exclusive-or 21) and produces a result of 25.

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 0</td>
<td>12</td>
</tr>
<tr>
<td>1 0 1 0 1</td>
<td>21</td>
</tr>
</tbody>
</table>

```
------------------------- exclusive or
1 1 0 0 1 = 25
```

The following unary operator pops the top value from the stack, performs an operation, and pushes the result on the stack:
%~  Bitwise complement. For example, %{25}%~ results in a value of -26, as shown in the following display.

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 1 1 0 0 1</td>
<td>= 25</td>
</tr>
<tr>
<td>1 1 1 0 0 1 1 0</td>
<td>= -26</td>
</tr>
</tbody>
</table>

Logical Operators

The following logical operators pop the top two values from the stack, perform an operation, and push the logical result (either 0 for false or 1 for true) on the stack:

% =  Equal to. For example, if the parameter p1 has the value 3, the expression %p1%{2}%= is equivalent to 3=2 and produces a result of 0 (false).

% >  Greater than. For example, if the parameter p1 has the value 3, the expression %p1%{0}%> is equivalent to 3>0 and produces a result of 1 (true).

% <  Less than. For example, if the parameter p1 has the value 3, the expression %p1%{4}%< is equivalent to 3<4 and produces a result of 1 (true).

The following unary operator pops the top value from the stack, performs an operation, and pushes the logical result (either 0 or 1) on the stack.

%!  Logical negation. This operator produces a value of zero for all non-zero numbers and a value of 1 for zero. For example, %{2}%! results in a value of 0, and %{0}%! results in a value of 1.

Conditional Statements

The condition statement IF-THEN-ELSE has the following format:

%? expr %t thenpart %e elsepart %;

The %e elsepart is optional. You can nest conditional statements in the thenpart or the elsepart.
When I-SQL evaluates a conditional statement, it pops the top value from the stack and evaluates it as either true or false. If the value is true, I-SQL performs the operations after the %t; otherwise it performs the operations after the %e (if any).

For example, the expression:

```
%?%p1{%3}%=%t;31%;
```

is equivalent to:

```
if p1 = 3 then print ";31"
```

Assuming that p1 has the value 3, I-SQL performs the following steps.

- %? does not perform an operation but is included to make the conditional statement easier to read.
- %p1 pushes the value of p1 on the stack.
- %{3} pushes the value 3 on the stack.
- %= pops the value of p1 and the value 3 from the stack, evaluates the Boolean expression p1=3, and pushes the resulting value 1 (true) on the stack.
- %t pops the value from the stack, evaluates 1 as true, and executes the operations after %t. (Since ";31" is not a stack machine operation, I-SQL prints ";31" to the terminal.)
- %; terminates the conditional statement.
Summary of Operators

Figure D-7 summarizes the allowed operations:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%d</code></td>
<td>write pop() in decimal format</td>
</tr>
<tr>
<td><code>%2d</code></td>
<td>write pop() in 2-place decimal format</td>
</tr>
<tr>
<td><code>%3d</code></td>
<td>write pop() in 3-place decimal format</td>
</tr>
<tr>
<td><code>%c</code></td>
<td>write pop() as a single character</td>
</tr>
<tr>
<td><code>%p[1-9]</code></td>
<td>push $i^{th}$ parameter</td>
</tr>
<tr>
<td><code>%[a-z]</code></td>
<td>pop and store variable</td>
</tr>
<tr>
<td><code>%g[a-z]</code></td>
<td>get variable and push on stack</td>
</tr>
<tr>
<td><code>%c</code></td>
<td>push char constant</td>
</tr>
<tr>
<td><code>%[n]</code></td>
<td>push integer constant</td>
</tr>
<tr>
<td><code>%S[a-z]</code></td>
<td>pop and store static variable</td>
</tr>
<tr>
<td><code>%G[a-z]</code></td>
<td>get static variable and push</td>
</tr>
<tr>
<td><code>%+</code></td>
<td>addition. push(pop()) op pop()</td>
</tr>
<tr>
<td><code>%</code></td>
<td>subtraction. push(pop()) op pop()</td>
</tr>
<tr>
<td><code>%*</code></td>
<td>multiplication. push(pop()) op pop()</td>
</tr>
<tr>
<td><code>%/</code></td>
<td>integer division. push(pop()) op pop()</td>
</tr>
<tr>
<td><code>%%</code></td>
<td>modulus. push(pop()) op pop()</td>
</tr>
<tr>
<td><code>&amp;&amp;</code></td>
<td>bit and. push(pop()) op pop()</td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td><code>^</code></td>
<td>bit exclusive or. push(pop()) op pop()</td>
</tr>
<tr>
<td><code>~</code></td>
<td>bitwise complement. push(op pop())</td>
</tr>
<tr>
<td><code>==</code></td>
<td>equal to. push(pop()) op pop()</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>greater than. push(pop()) op pop()</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>less than. push(pop()) op pop()</td>
</tr>
<tr>
<td><code>!</code></td>
<td>logical negation. push(op pop())</td>
</tr>
<tr>
<td><code>? expr %t thenpart %e elsepart %;</code></td>
<td>if-then-else; the %e elsepart is optional. else-if’s are possible (c’s are conditions).</td>
</tr>
</tbody>
</table>

all other characters are written to the terminal; use ‘%’ to write ‘%’.

Figure D-7 Stack Operations
Examples

To illustrate, consider the monochrome Wyse terminal. Figure D-8 shows the sequences for various display characteristics.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESCAPE G 0</td>
<td>Normal</td>
</tr>
<tr>
<td>ESCAPE G 1</td>
<td>Blank(invisible)</td>
</tr>
<tr>
<td>ESCAPE G 2</td>
<td>Blink</td>
</tr>
<tr>
<td>ESCAPE G 4</td>
<td>Reverse</td>
</tr>
<tr>
<td>ESCAPE G 5</td>
<td>Reverse and blank</td>
</tr>
<tr>
<td>ESCAPE G 6</td>
<td>Reverse and blink</td>
</tr>
<tr>
<td>ESCAPE G 8</td>
<td>Underscore</td>
</tr>
<tr>
<td>ESCAPE G 9</td>
<td>Underscore and blank</td>
</tr>
<tr>
<td>ESCAPE G :</td>
<td>Underscore and blink</td>
</tr>
<tr>
<td>ESCAPE G &lt;</td>
<td>Underscore and reverse</td>
</tr>
<tr>
<td>ESCAPE G =</td>
<td>Underscore, reverse, and blank</td>
</tr>
<tr>
<td>ESCAPE G &gt;</td>
<td>Underscore, reverse, and blink</td>
</tr>
</tbody>
</table>

Figure D-8 Wyse Escape Sequences

The characters after G form an ASCII sequence from the character 0 (zero) through ?. You can generate the character by starting with 0 and adding 1 for blank, 2 for blink, 4 for reverse, and 8 for underline.

You can construct the termcap entry in stages, as outlined in the following display. \%p1 refers to pushing the i\textsuperscript{th} parameter on the stack. The designation for is \EG. The termcap entry for the Wyse terminal must contain the following ZA entry in order for I-SQL monochrome attributes such as REVERSE and BOLD to work correctly:

```bash
ZA = "\EG
%"0" #push '0' (normal) on the stack
%?%p1%{7}>%t%{1}%| #if p1 = 7 (invisible), set the 1 bit (blank);
%e%p1%{3}%=t%{64}%| #if p1 > 3 and < 7, set the 64 flag (dim);
%?%p2%t%{4}%| #if p2 is set, set the 4 bit (reverse)
%?%p3%t%{2}% | #if p3 is set, set the 2 bit (blink)
%?%p4%t%{8}% | #if p4 is set, set the 8 bit (underline)
%c: #print whatever character
# is on top of the stack"
```
You then concatenate these lines as a single string that ends with a colon and has no embedded NEWLINEs. The actual ZA entry for the Wyse 50 terminal follows:

\[ZA = \EG%'0'%?%p1%{7}%=%t%{1}%|%e%p1%{3}%>%p1%{7}%<%&%t%{64}%|%;%;?%p3%t%{2}%|%;?%p4%t%{8}%|%;c;\]

The next example is for the ID Systems Corporation ID231, a color terminal. On this terminal, to set color and other characteristics you must enclose a character sequence between a lead-in sequence (ESCAPE [ 0) and a terminating character (m). The first in the sequence is a two-digit number that determines whether the assigned color is in the background (30) or in the foreground (40). The next is another two-digit number that is the other of 30 or 40, incremented by the color number. These characters are followed by 5 if there is blinking and by 4 for underlining. The code in Figure D-9 sets up the entire escape sequence:

```
ZA = \E[0; %?%p1%{0}%=%t%{7}%e%p1%{1}%=%t%{3}%e%p1%{2}%=%t%{5}%e%p1%{3}%=%t%{1}%e%p1%{4}%=%t%{6}%e%p1%{5}%=%t%{2}%e%p1%{6}%=%t%{4}%e%p1%{7}%=%t%{0}%;%?%p2%t30;%{40}%+%2d%e40;%{30}%+%2d%;%?%p3%t;5%;%?%p4%t;4%;m
```

Figure D-9 Sample ZA String for ID231

When you concatenate these strings, the termcap entry is as shown in Figure D-10.

```
ZA = \E[0; %?%p1%{0}%=%t%{7}%e%p1%{1}%=%t%{3}%e%p1%{2}%=%t%{5}%e%p1%{3}%=%t%{1}%e%p1%{4}%=%t%{6}%e%p1%{5}%=%t%{2}%e%p1%{6}%=%t%{4}%e%p1%{7}%=%t%{0}%;%?%p2%t30;%{40}%+%2d%e40;%{30}%+%2d%;%?%p3%t;5%;%?%p4%t;4%;m
```

Figure D-10 Concatenated ZA String for ID231
In addition to the ZA capability, you can use other termcap capabilities. ZG is the number of character positions on the screen occupied by the attributes of ZA. Like the sg numeric capability, ZG is not required if no extra character positions are needed for display attributes. The value for the ZG entry is usually the same value as for the sg entry.

terminfo

If you have set the INFORMIXTERM environment variable to terminfo, I-SQL uses the terminfo directory indicated by the TERMINO environment variable (or /usr/lib/terminfo if TERMINO is not set). I-SQL uses the information in terminfo to draw borders and display certain intensity attributes.

You may want to modify a file in the terminfo directory if you want to specify or change the graphics characters used for borders in screen forms.

Note: If you use terminfo (instead of termcap), you cannot use color or certain intensity attributes with I-SQL. To use color attributes with I-SQL, you must use termcap.

Some terminals cannot support graphics characters. You should read this appendix and the user guide that comes with your terminal to determine whether or not the changes described in this appendix are applicable to your terminal.

To modify a terminfo file, you need to be familiar with the following:

- The format of terminfo entries
- The infocomp program
- The tic program

This information is summarized in this appendix; however, you should refer to your operating system documentation for a complete discussion.

Format of a terminfo Entry

terminfo is a directory that contains a file for each terminal name that is defined. Each file contains a compiled terminfo entry for that terminal. This section describes the general format of terminfo entries. For a complete description of terminfo, refer to your operating system documentation.
A `terminfo` entry contains a list of names for the terminal, followed by a list of the terminal’s capabilities. There are three types of capabilities:

- Boolean capabilities
- Numeric capabilities
- String capabilities

All `terminfo` entries have the following format:

- ESCAPE is specified as a backslash (\) followed by the letter E, and CONTROL is specified as a caret (^). Do not use the ESCAPE or CONTROL keys to indicate escape sequences or control characters in a `terminfo` entry.
- Each capability, including the last entry, is followed by a comma (,).

Figure D-11 shows a basic `terminfo` entry for the Wyse 50 terminal:

```plaintext
w5|wy50|wyse50,
   am, cols#80, lines#24, cuul=^K, clear=^Z,
   home=^^, cufl=^L, cup=%E=%p1%’s%+%c%p2%’s%+%c,
   bw, ul, bel=^G, cr=\r, cud1=\n, cub1=\b, kpb=\b, kcud1=\n,
   kdbu1=\b, nel=\r\n, ind=\n,
   xmc#1, cbt=\Ei,
```

**Figure D-11  Wyse 50 terminfo Entry**

*Note: Comment lines begin with a period (.).*

**Terminal Names**

A `terminfo` entry starts with one or more names for the terminal (each separated by a vertical bar (|)). For example, the `terminfo` entry for the Wyse 50 terminal starts with the following line:

```plaintext
w5|wy50|wyse50,
```

The `terminfo` entry can be accessed using any one of these names.

**Boolean Capabilities**

A Boolean capability is a two- to five-character code that indicates whether or not a terminal has a specific feature. If the Boolean capability is present in the `terminfo` entry, the terminal has that particular feature.
The Figure D-12 shows some of the Boolean capabilities for the Wyse 50 terminal:

\[
\begin{array}{c}
\text{bw, am,} \\
. \text{ bw backward wrap} \\
. \text{ am automatic margins}
\end{array}
\]

**Figure D-12** Boolean Capabilities for the Wyse 50

**Numeric Capabilities**

A numeric capability is a two- to five-character code followed by a pound symbol ( # ) and a value. Figure D-13 shows the numeric capabilities for the number of columns and the number of lines on a Wyse 50 terminal:

\[
\begin{array}{c}
\text{cols#80, lines#24,} \\
. \text{ cols number of columns in a line} \\
. \text{ lines number of lines on the screen}
\end{array}
\]

**Figure D-13** Numeric Capabilities for the Wyse 50

**String Capabilities**

A string capability specifies a sequence that can be used to perform a terminal operation. A string capability is a two- to five-character code followed by an equal sign ( = ) and a string ending at the next delimiter ( , ).
Most `terminfo` entries include string capabilities for clearing the screen, cursor movement, arrow keys, underscore, function keys, and so on. Figure D-14 shows many of the string capabilities for the Wyse 50 terminal:

```
el=\ET, clear=\E*,
cuf1=^L, cuu1=^K,
smso=\EG4, rmso=\EG0,
kcuu1=^K, kcu1=^J, kcufl=^L, kcb1=^H,
kf0=^A@^M, kf1=^AA^M, kf2=^AB^M, kf3=^AC^M,
```

- `el=\ET` clear to end of line
- `clear=\E*` clear the screen
- `cuf1=^L` non-destructive cursor right
- `cuu1=^K` up one line
- `smso=\EG4` start stand-out
- `rmso=\EG0` end stand-out
- `kcuu1=^K` up arrow key
- `kcu1=^J` down arrow key
- `kcufl=^L` right arrow key
- `kcb1=^H` left arrow key
- `kf0=^A@^M` function key F1
- `kf1=^AA^M` function key F2
- `kf2=^AB^M` function key F3
- `kf3=^AC^M` function key F4

Figure D-14 String Capabilities for the Wyse 50

**Specifying Graphics Characters in Screen Forms**

I-SQL uses characters defined in a `terminfo` file to draw the borders of boxes and other rectangular shapes that appear in a screen form. If no characters are defined in the `terminfo` file, I-SQL uses the hyphen ( - ) for horizontal lines, the vertical bar ( | ) for vertical lines, and the plus sign ( + ) for corners.

You can look at the `terminfo` source file (using `infocmp`) to see if the entry for your terminal includes these definitions (look for the `acsc` capability, described later in this section). If the file does not contain border character definitions for your terminal type, or if you want to specify alternative border characters, you or your system administrator can modify the `terminfo` source file. You can refer to your operating system documentation for a complete description of how to decompile `terminfo` entries using the `infocmp` program.
Perform the following steps to specify border characters in the `terminfo` source file for your terminal:

1. Determine the escape sequences for turning graphics mode on and off. This information is located in the manual that comes with your terminal. For example, on Wyse 50 terminals, the escape sequence for entering graphics mode is `ESCAPE H^B` and the escape sequence for leaving graphics mode is `ESCAPE H^C`.

   *Note:* Terminals without a graphics mode do not have this escape sequence. The procedure for specifying alternative border characters on a non-graphics terminal is discussed at the end of this section.

2. Identify the ASCII equivalents for the six graphics characters that I-SQL requires to draw the border. (The ASCII equivalent of a graphics character is the key you would press in graphics mode to obtain the indicated character.)

   Figure D-15 shows the graphics characters and the ASCII equivalents for a Wyse 50 terminal.

<table>
<thead>
<tr>
<th>Window Border Position</th>
<th>Graphics Character</th>
<th>ASCII Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>upper left corner</td>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td>lower left corner</td>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td>upper right corner</td>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td>lower right corner</td>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td>horizontal</td>
<td><code>-</code></td>
<td><code>z</code></td>
</tr>
<tr>
<td>vertical</td>
<td>`</td>
<td>`</td>
</tr>
</tbody>
</table>

   *Figure D-15*  
   *Wyse 50 ASCII Equivalents for Border Graphics Characters*

   Again, this information should be located in the manual that comes with your terminal.

3. Edit the `terminfo` source file for your terminal (you can decompile it using `infocmp` redirected to a file).
Note: You may want to make a copy of your terminfo directory before you edit files. You can use the TERMINFO environment variable to point to whichever copy of the terminfo directory you want to access.

Use the format:

```
terminfo-capability=value
```

to enter values for the following terminfo capabilities:

- **smacs**: The escape sequence for entering graphics mode. In a terminfo file, ESCAPE is represented as a backslash (\) followed by the letter E; CONTROL is represented as a caret (^). For example, the Wyse 50 escape sequence ESCAPE-H CONTROL-B is represented as \EH^B.

- **rmacs**: The escape sequence for leaving graphics mode. For example, the Wyse 50 escape sequence ESCAPE-H CONTROL-C is represented as \EH^C.

- **acsc**: The concatenated, paired list of ASCII equivalents for the six graphics characters used to draw the border. You can specify the characters in any order, but you must pair the ASCII equivalents for your terminal with the following system default characters:

<table>
<thead>
<tr>
<th>Position</th>
<th>System Default Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>upper left corner</td>
<td>l</td>
</tr>
<tr>
<td>lower left corner</td>
<td>m</td>
</tr>
<tr>
<td>upper right corner</td>
<td>k</td>
</tr>
<tr>
<td>lower right corner</td>
<td>j</td>
</tr>
<tr>
<td>horizontal</td>
<td>q</td>
</tr>
<tr>
<td>vertical</td>
<td>x</td>
</tr>
</tbody>
</table>

*Figure D-16  System Default Characters for Border Positions*
Use the following format to specify the `acsc` value:

```
defnewdefnew...
```

where `def` is the default character for a particular border character and `new` is that terminal's equivalent for the same border character.

For example, on the Wyse 50 terminal, given the ASCII equivalents in Figure D-15 and the system default characters in Figure D-16, the `acsc` capability would be set as shown in Figure D-17.

```
acsc=l2m1k3j5qzx6
```

![Wyse 50 acsc setting](image)

4. Use `tic` to recompile the modified `terminfo` file. See your operating system documentation for a description of the `tic` program.

The following example shows the full setting for specifying alternative border characters on the Wyse 50:

```
smacs=\EH^B,  \quad \text{sets smacs to ESC H CTRL B}
rmacs=\EH^C,  \quad \text{sets rmacs to ESC H CTRL C}
acsc=l2m1k3j5qzx6, \quad \text{sets acsc to the ASCII equivalents}
   \quad \text{of graphics characters for upper}
   \quad \text{left (l), lower left (m), upper right (k),}
   \quad \text{lower right (j), horizontal (q),}
   \quad \text{and vertical (x)}
```

If you prefer, you can enter this information in a linear sequence.

```
smacs=\EH^B,rmacs=\EH^C,acsc=l2m1k3j5qzx6,
```

**Terminals Without Graphics Capabilities**

For terminals without graphics capabilities, you must enter a blank value for the `smacs` and `rmacs` capabilities. For `acsc`, enter the characters you want 1-SQL to use for the window border.

The following example shows possible values for `smacs`, `rmacs`, and `acsc` in an entry for a terminal without graphics capabilities. In this example, window borders would be drawn using underscores (_ ) for horizontal lines, vertical bars ( | | ) for vertical lines, periods ( . ) for the top corners, and vertical bars ( | | ) for the lower corners.

```
smacs=,rmacs=,acsc=l.m|k.j|q_x|,
```
I-SQL uses the graphics characters in the terminfo file when you specify a screen border in a PERFORM screen.

**Color and Intensity**

If you use terminfo, you cannot use color or the BOLD or BLINK intensity attributes with the COLOR attribute in PERFORM. If you specify these attributes they are ignored.

If the terminfo entry for your terminal contains the ul and so attributes, you can use the UNDERLINE and REVERSE intensity attributes, however. You can see if your terminfo entry includes these capabilities by using the infocmp program. Refer to your operating system documentation for information about infocmp.

If you want to use color and intensity in your I-SQL screen forms, you must use termcap (by setting the INFORMIXTERM environment variable to termcap, and by setting the TERMCPAP environment variable to $INFORMIXDIR/etc/termcap). For more information, refer to the “Environment Variables” appendix and the Preface.
The ASCII Character Set
<table>
<thead>
<tr>
<th>Num</th>
<th>Char</th>
<th>Num</th>
<th>Char</th>
<th>Num</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>^@</td>
<td>43</td>
<td>+</td>
<td>86</td>
<td>V</td>
</tr>
<tr>
<td>1</td>
<td>^A</td>
<td>44</td>
<td>,</td>
<td>87</td>
<td>W</td>
</tr>
<tr>
<td>2</td>
<td>^B</td>
<td>45</td>
<td>-</td>
<td>88</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>^C</td>
<td>46</td>
<td>.</td>
<td>89</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>^D</td>
<td>47</td>
<td>/</td>
<td>90</td>
<td>Z</td>
</tr>
<tr>
<td>5</td>
<td>^E</td>
<td>48</td>
<td>0</td>
<td>91</td>
<td>[</td>
</tr>
<tr>
<td>6</td>
<td>^F</td>
<td>49</td>
<td>1</td>
<td>92</td>
<td>\</td>
</tr>
<tr>
<td>7</td>
<td>^G</td>
<td>50</td>
<td>2</td>
<td>93</td>
<td>]</td>
</tr>
<tr>
<td>8</td>
<td>^H</td>
<td>51</td>
<td>3</td>
<td>94</td>
<td>^</td>
</tr>
<tr>
<td>9</td>
<td>^I</td>
<td>52</td>
<td>4</td>
<td>95</td>
<td>_</td>
</tr>
<tr>
<td>10</td>
<td>^J</td>
<td>53</td>
<td>5</td>
<td>96</td>
<td>'</td>
</tr>
<tr>
<td>11</td>
<td>^K</td>
<td>54</td>
<td>6</td>
<td>97</td>
<td>a</td>
</tr>
<tr>
<td>12</td>
<td>^L</td>
<td>55</td>
<td>7</td>
<td>98</td>
<td>b</td>
</tr>
<tr>
<td>13</td>
<td>^M</td>
<td>56</td>
<td>8</td>
<td>99</td>
<td>c</td>
</tr>
<tr>
<td>14</td>
<td>^N</td>
<td>57</td>
<td>9</td>
<td>100</td>
<td>d</td>
</tr>
<tr>
<td>15</td>
<td>^O</td>
<td>58</td>
<td>:</td>
<td>101</td>
<td>e</td>
</tr>
<tr>
<td>16</td>
<td>^P</td>
<td>59</td>
<td>;</td>
<td>102</td>
<td>f</td>
</tr>
<tr>
<td>17</td>
<td>^Q</td>
<td>60</td>
<td>&lt;</td>
<td>103</td>
<td>g</td>
</tr>
<tr>
<td>18</td>
<td>^R</td>
<td>61</td>
<td>=</td>
<td>104</td>
<td>h</td>
</tr>
<tr>
<td>19</td>
<td>^S</td>
<td>62</td>
<td>&gt;</td>
<td>105</td>
<td>i</td>
</tr>
<tr>
<td>20</td>
<td>^T</td>
<td>63</td>
<td>?</td>
<td>106</td>
<td>j</td>
</tr>
<tr>
<td>21</td>
<td>^U</td>
<td>64</td>
<td>@</td>
<td>107</td>
<td>k</td>
</tr>
<tr>
<td>22</td>
<td>^V</td>
<td>65</td>
<td>A</td>
<td>108</td>
<td>l</td>
</tr>
<tr>
<td>23</td>
<td>^W</td>
<td>66</td>
<td>B</td>
<td>109</td>
<td>m</td>
</tr>
<tr>
<td>24</td>
<td>^X</td>
<td>67</td>
<td>C</td>
<td>110</td>
<td>n</td>
</tr>
<tr>
<td>25</td>
<td>^Y</td>
<td>68</td>
<td>D</td>
<td>111</td>
<td>o</td>
</tr>
<tr>
<td>26</td>
<td>^Z</td>
<td>69</td>
<td>E</td>
<td>112</td>
<td>p</td>
</tr>
<tr>
<td>27</td>
<td>esc</td>
<td>70</td>
<td>F</td>
<td>113</td>
<td>q</td>
</tr>
<tr>
<td>28</td>
<td>^\</td>
<td>71</td>
<td>G</td>
<td>114</td>
<td>r</td>
</tr>
<tr>
<td>29</td>
<td>^]</td>
<td>72</td>
<td>H</td>
<td>115</td>
<td>s</td>
</tr>
<tr>
<td>30</td>
<td>^^</td>
<td>73</td>
<td>I</td>
<td>116</td>
<td>t</td>
</tr>
<tr>
<td>31</td>
<td>^_</td>
<td>74</td>
<td>J</td>
<td>117</td>
<td>u</td>
</tr>
<tr>
<td>32</td>
<td>!</td>
<td>75</td>
<td>K</td>
<td>118</td>
<td>v</td>
</tr>
<tr>
<td>33</td>
<td>#</td>
<td>76</td>
<td>L</td>
<td>119</td>
<td>w</td>
</tr>
<tr>
<td>34</td>
<td>$</td>
<td>77</td>
<td>M</td>
<td>120</td>
<td>x</td>
</tr>
<tr>
<td>35</td>
<td>%</td>
<td>78</td>
<td>N</td>
<td>121</td>
<td>y</td>
</tr>
<tr>
<td>36</td>
<td>&amp;</td>
<td>79</td>
<td>O</td>
<td>122</td>
<td>z</td>
</tr>
<tr>
<td>37</td>
<td>^</td>
<td>80</td>
<td>P</td>
<td>123</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>(</td>
<td>81</td>
<td>Q</td>
<td>124</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>)</td>
<td>82</td>
<td>R</td>
<td>125</td>
<td>)</td>
</tr>
<tr>
<td>40</td>
<td>{</td>
<td>83</td>
<td>S</td>
<td>126</td>
<td>~</td>
</tr>
<tr>
<td>41</td>
<td>}</td>
<td>84</td>
<td>T</td>
<td>127</td>
<td>del</td>
</tr>
<tr>
<td>42</td>
<td>^</td>
<td>85</td>
<td>U</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the above table, ^ represents the CONTROL key.
Reserved Words

In this release of INFORMIX-SQL (I-SQL), very few words are reserved. You can use the words that were reserved in previous releases of I-SQL as identifiers. For example, you can execute a statement such as the following:

```sql
CREATE TABLE table (column INTEGER, date DATE, char CHAR(20))
```

However, using some of the formerly reserved words can cause ambiguities in your I-SQL statements. These ambiguities can cause I-SQL to do one of the following:
- Process the statement differently than you intended.
- Produce an error.

This section discusses these potential ambiguities and syntax errors.

If your table is ANSI-compliant, some words are still reserved. For a list of the ANSI reserved words, see the section entitled “Avoiding the ANSI Reserved Words” on page F-9.
Potential Ambiguities and Syntax Errors

Although you can now use the formerly reserved words as identifiers in I-SQL statements, syntactic ambiguities can occur. Thus, a statement might not produce the desired results. This section describes the following:

- Using functions as column names
- Using keywords as column names
- Using keywords as table names
- Workarounds that use the keyword AS

Using Functions as Column Names

When using built-in function names as column names, the engine can interpret the column name as a function. For example, the following statement specifies a column named \texttt{avg}. This statement fails because the engine interprets \texttt{avg} as an aggregate function rather than as a column name:

\begin{verbatim}
SELECT avg FROM mytab
\end{verbatim}

You can avoid this ambiguity by including a table name with the column name, as shown in the following example:

\begin{verbatim}
SELECT mytab.avg FROM mytab
\end{verbatim}

This ambiguity applies to the aggregate functions (AVG, COUNT, MAX, MIN, SUM), the LENGTH function, the date functions (DATE, DAY, MDY, MONTH, WEEKDAY, YEAR), and the datetime function EXTEND. For general descriptions of these functions, refer to the Informix Guide to SQL: Tutorial, Version 4.1.

If you use the keyword TODAY, CURRENT, or USER as a column name, ambiguity can also occur, as shown in the following example:

\begin{verbatim}
CREATE TABLE mytab (user CHAR(10),
  current DATETIME HOUR TO SECOND,today DATE)
INSERT INTO mytab VALUES("josh","11:30:30","1/22/89")
SELECT user, current, today FROM mytab
\end{verbatim}
The engine interprets user, current, and today in the SELECT statement as the functions USER, CURRENT, and TODAY. Thus, instead of returning josh, 11:30:30, 1/22/89, the SELECT statement returns the current user name, the current time, and the current date.

If you want to select the actual columns of the table, you must write the SELECT statement in one of two ways:

```sql
SELECT mytab.user, mytab.current, mytab.today FROM mytab
```

or, equivalently:

```sql
SELECT * FROM mytab
```

For general descriptions of the TODAY, CURRENT, and USER functions, see the Informix Guide to SQL: Tutorial, Version 4.1.

**Using Keywords as Column Names**

I-SQL supports specific workarounds for using a formerly reserved keyword as a column name in an I-SQL statement. In some cases, I-SQL offers more than one workaround. This section describes the following:

- Using ALL as a column name
- Using UNIQUE or DISTINCT as a column name
- Using INTERVAL or DATETIME as a column name
- Using rowid as a column name

**Using ALL as a Column Name**

Using all as a column name causes the following SELECT statement to fail because the engine interprets all as a keyword rather than as a column name:

```sql
SELECT all FROM mytab
```

To include a column name all in a SELECT statement, you can include the ALL keyword prior to the all column name, as shown in the following example:

```sql
SELECT ALL all FROM mytab
```
Using Keywords as Column Names

You can also prefix the column name with the table name. For example, you could specify the following:

```
SELECT mytab.all FROM mytab
```

Using UNIQUE or DISTINCT as a Column Name

Using `unique` or `distinct` as a column name causes the CREATE TABLE statement to fail because the engine interprets `unique` as a keyword rather than as a column name:

```
CREATE TABLE mytab (unique INTEGER)
```

You can, however, name a column `unique` by using two statements. The first statement creates the column `mycol`, the second statement renames the column `mycol` to `unique`:

```
CREATE TABLE mytab (mycol INTEGER)
RENAME COLUMN mytab.mycol TO unique
```

Using INTERVAL or DATETIME as a Column Name

Using `interval` as a column name causes the following SELECT statement to fail because the engine interprets `interval` as a keyword and expects it to be followed by an INTERVAL qualifier:

```
SELECT interval FROM mytab
```

To include a column named `interval` in a SELECT statement, you should prefix the column name with the table name, as shown in the following example:

```
SELECT mytab.interval FROM mytab
```
Using Keywords as Column Names

You can also include the owner name as well as the table name:

```
SELECT josh.mytab.interval FROM josh.mytab
```

**Using rowid as a Column Name**

Every Informix database table has a virtual column named `rowid`. This column contains the record number associated with each row in a table. To avoid ambiguity, you cannot use `rowid` as a column name. The following actions cause an error:

- Creating a table or view with a column named `rowid`.
- Altering a table by adding a column named `rowid`.
- Renaming a column to `rowid`.

You can, however, use the term `rowid` as a table name:

```
CREATE TABLE rowid (column INTEGER, date DATE, char CHAR(20))
```

**Using Keywords as Table Names**

If you use a previously reserved word as the name for a table, some SQL statements may be ambiguous. You can avoid ambiguous statements by prefixing the table name with the name of the table's owner.

For example, using `statistics` as a table name causes the following UPDATE statement to fail because the engine interprets it as part of the UPDATE STATISTICS syntax rather than as a table name in an UPDATE statement:

```
UPDATE statistics SET mycol = 10
```

You can, however, include a table named `statistics` in an UPDATE statement by specifying an owner name with the table name, as shown in the following example:

```
UPDATE josh.statistics SET mycol = 10
```
Using Keywords as Column Names

Using outer as a table name causes the following SELECT statement to fail because the engine interprets outer as a keyword for performing an outer join:

```
SELECT mycol FROM outer
```

Again, by specifying an owner name with the table name, you can avoid any ambiguity and create a SELECT statement that executes properly, as shown in the following example:

```
SELECT mycol FROM josh.outer
```

Workarounds that Use the Keyword AS

Some formerly reserved words cannot be used as column labels or table aliases. As a workaround, I-SQL provides the AS keyword that lets you use these words as column labels and table aliases.

Because the AS keyword syntax is a part of the proposed ANSI SQL2 standard, but is not included in the ANSI SQL89 standard, the engine generates ANSI warnings if you use the AS keyword and one of the following is true:

- The DBANSIWARN environment variable is set.
- You specify the -ansi flag when invoking I-SQL.

The syntax for using AS with a column label is as follows:

```
column-name AS display-label FROM table-name
```

The syntax for using AS with a table alias is as follows:

```
SELECT select-list FROM table-name AS table-alias
```

Both of these options are described below.

Using AS With Column Labels

To use one of the following keywords as a column label, you must use the AS keyword:

- AS
- FROM
- UNITS
- YEAR
Using Keywords as Column Names

- MONTH
- DAY
- HOUR
- MINUTE
- SECOND
- FRACTION

For example, the following statement fails because the engine interprets **units** as a DATETIME qualifier for the column named **mycol**:

```
SELECT mycol units FROM mytab
```

By using the keyword **AS**, however, you can avoid any ambiguity, as shown in the following example:

```
SELECT mycol AS units FROM mytab
```

You must also use the **AS** keyword to select a column labeled **as** or **from**. For example, the following statement fails because **I-SQL** does not find the required FROM clause. **I-SQL** interprets the column label **as** as the keyword **AS**. **I-SQL** then interprets the keyword FROM as the column label to assign to **mycol**:

```
SELECT mycol as FROM mytab
```

By using the keyword **AS**, however, you can avoid any ambiguity, as shown in the following example:

```
SELECT mycol AS as FROM mytab
```

The following statement fails because the engine expects a table name to follow the first **from**:

```
SELECT mycol from FROM mytab
```
Using Keywords as Column Names

By using the keyword AS, however, you can identify the first `from` as a column label, as shown in the following example:

```
SELECT mycol AS from FROM mytab
```

**Using AS with Table Aliases**

To use one of the following keywords as a table alias, you must use the AS keyword:
- ORDER
- FOR
- GROUP
- HAVING
- INTO
- UNION
- WHERE

For example, the following statement fails because the engine interprets `order` as part of an ORDER BY clause:

```
SELECT * FROM mytab order
```

By using the keyword AS, however, you can identify `order` as a table alias:

```
SELECT * FROM mytab AS order
```

You must also use the keyword AS to give a table the alias of WITH, CREATE, or GRANT. For example, the following statement fails because the engine interprets `with` as part of the WITH CHECK OPTION syntax:

```
SELECT * FROM mytab with
```

By using the keyword AS, however, you can identify `with` as a table alias, as shown in the following example:

```
SELECT * FROM mytab AS with
```
Avoiding the ANSI Reserved Words

The following statement fails because the engine interprets the keyword `create` as part of the syntax to create an entity such as a table, synonym, or view:

```
SELECT * FROM mytab create
```

By using the keyword `AS`, however, you can identify `create` as a table alias, as shown in the following example:

```
SELECT * FROM mytab AS create
```

Avoiding the ANSI Reserved Words

If you have an ANSI-compliant table, several words are reserved. I-SQL generates a warning if you use an ANSI reserved word as an identifier in a statement and one or both of the following is true:

- The `DBANSIWARN` environment variable is set.
- You specify the `-ansi` flag when invoking I-SQL.

The ANSI reserved words are as follows:

- `all`
- `and`
- `any`
- `as`
- `asc`
- `avg`
- `begin`
- `between`
- `by`
- `char`
- `character`
- `check`
- `close`
- `cobol`
- `commit`
- `continue`
- `count`
- `create`
- `current`
- `cursor`
- `dec`
- `decimal`
- `declare`
- `delete`
- `desc`
- `distinct`
- `double`
- `end`
- `escape`
- `exec`
- `exists`
- `fetch`
- `float`
- `for`
- `fortran`
- `found`
- `from`
- `go`
- `goto`
- `grant`
- `group`
- `having`
- `in`
- `indicator`
- `insert`
- `int`
- `integer`
- `into`
- `is`
- `language`
- `like`
- `max`
- `min`
- `module`
- `not`
- `null`
- `numeric`
- `of`
- `on`
- `open`
- `option`
- `or`
- `order`
- `pascal`
- `pli`
- `plsql`
- `precision`
- `privileges`
- `procedure`
- `public`
- `real`
- `rollback`
- `schema`
- `section`
- `select`
- `set`
- `some`
- `sql`
- `sqlcode`
- `sqlerror`
- `sum`
- `to`
- `union`
- `unique`
- `update`
- `user`
- `values`
- `view`
- `where`
- `work`
Avoiding the ANSI Reserved Words
System Catalogs

Information about the database is maintained in the system catalogs. The system catalogs are as follows:

- **systables**: describes database tables.
- **syscolumns**: describes columns in tables.
- **sysindexes**: describes indexes on columns.
- **systabauth**: identifies table-level privileges.
- **syscolauth**: identifies column-level privileges.
- **sysdepend**: describes how views depend on tables.
- **syssynonyms**: lists synonyms for tables.
- **sysusers**: identifies database-level privileges.
- **sysviews**: defines views.
- **sysconstraints**: records constraints placed on database tables.
- **syssyntable**: used for mapping of synonyms.
The following list gives a brief description of the system catalogs.

The SYSTABLES Catalog

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabname</td>
<td>char(18)</td>
<td>name of table</td>
</tr>
<tr>
<td>owner</td>
<td>char(8)</td>
<td>owner of table</td>
</tr>
<tr>
<td>dirpath</td>
<td>char(64)</td>
<td>directory path for the table file</td>
</tr>
<tr>
<td>tabid</td>
<td>serial</td>
<td>internal table identifier</td>
</tr>
<tr>
<td>rowsize</td>
<td>smallint</td>
<td>row size</td>
</tr>
<tr>
<td>ncols</td>
<td>smallint</td>
<td>number of columns</td>
</tr>
<tr>
<td>nindexes</td>
<td>smallint</td>
<td>number of indexes</td>
</tr>
<tr>
<td>nrows</td>
<td>integer</td>
<td>number of rows</td>
</tr>
<tr>
<td>created</td>
<td>date</td>
<td>date table created</td>
</tr>
<tr>
<td>version</td>
<td>integer</td>
<td>table version number</td>
</tr>
<tr>
<td>tabtype</td>
<td>char(1)</td>
<td>table type (T = table, V = view, S = synonym)</td>
</tr>
<tr>
<td>audpath</td>
<td>char(64)</td>
<td>audit filename (full pathname)</td>
</tr>
</tbody>
</table>

Index

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabname</td>
<td>unique</td>
<td>tabname, owner</td>
</tr>
<tr>
<td>tabid</td>
<td>unique</td>
<td>tabid</td>
</tr>
</tbody>
</table>

The SYSCOLUMNS Catalog

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>colname</td>
<td>char(18)</td>
<td>column name</td>
</tr>
<tr>
<td>tabid</td>
<td>integer</td>
<td>table identifier</td>
</tr>
<tr>
<td>colno</td>
<td>smallint</td>
<td>column number</td>
</tr>
<tr>
<td>coltype</td>
<td>smallint</td>
<td>column type</td>
</tr>
<tr>
<td>collength</td>
<td>smallint</td>
<td>column length (physical)</td>
</tr>
</tbody>
</table>

Index

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>column</td>
<td>unique</td>
<td>tabid, colno</td>
</tr>
</tbody>
</table>
### The SYSINDEXES Catalog

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>idxname</td>
<td>char(18)</td>
<td>index name</td>
</tr>
<tr>
<td>owner</td>
<td>char(8)</td>
<td>owner of index</td>
</tr>
<tr>
<td>tabid</td>
<td>integer</td>
<td>table identifier</td>
</tr>
<tr>
<td>idxtype</td>
<td>char(1)</td>
<td>index type (U = unique, D = dups)</td>
</tr>
<tr>
<td>clustered</td>
<td>char(1)</td>
<td>clustering</td>
</tr>
<tr>
<td>part1</td>
<td>smallint</td>
<td>column number</td>
</tr>
<tr>
<td>part2</td>
<td>smallint</td>
<td>column number</td>
</tr>
<tr>
<td>part3</td>
<td>smallint</td>
<td>column number</td>
</tr>
<tr>
<td>part4</td>
<td>smallint</td>
<td>column number</td>
</tr>
<tr>
<td>part5</td>
<td>smallint</td>
<td>column number</td>
</tr>
<tr>
<td>part6</td>
<td>smallint</td>
<td>column number</td>
</tr>
<tr>
<td>part7</td>
<td>smallint</td>
<td>column number</td>
</tr>
<tr>
<td>part8</td>
<td>smallint</td>
<td>column number</td>
</tr>
</tbody>
</table>

**Index Name**

<table>
<thead>
<tr>
<th>Type</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>dups</td>
<td>tabid</td>
</tr>
<tr>
<td>unique</td>
<td>idxname, tabid</td>
</tr>
</tbody>
</table>

### The SYSTABAUTH Catalog

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantor</td>
<td>char(8)</td>
<td>grantor of permission</td>
</tr>
<tr>
<td>grantee</td>
<td>char(8)</td>
<td>grantee (receiver) of permission</td>
</tr>
<tr>
<td>tabid</td>
<td>integer</td>
<td>table identifier</td>
</tr>
<tr>
<td>tabauth</td>
<td>char(7)</td>
<td>authorization type</td>
</tr>
</tbody>
</table>

**Index Name**

<table>
<thead>
<tr>
<th>Type</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique</td>
<td>tabid, grantor, grantee</td>
</tr>
<tr>
<td>dups</td>
<td>tabid, grantee</td>
</tr>
</tbody>
</table>
### The SYSCOLAUTH Catalog

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantor</td>
<td>char(8)</td>
<td>grantor of permission</td>
</tr>
<tr>
<td>grantee</td>
<td>char(8)</td>
<td>grantee (receiver) of permission</td>
</tr>
<tr>
<td>tabid</td>
<td>integer</td>
<td>table identifier</td>
</tr>
<tr>
<td>colno</td>
<td>smallint</td>
<td>column number</td>
</tr>
<tr>
<td>colauth</td>
<td>char(2)</td>
<td>authorization type</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index Name</th>
<th>Type</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>colgtor</td>
<td>unique</td>
<td>tabid, grantor, grantee, colno</td>
</tr>
<tr>
<td>colgtee</td>
<td>dups</td>
<td>tabid, grantee</td>
</tr>
</tbody>
</table>

### The SYSDEPEND Catalog

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>btabid</td>
<td>integer</td>
<td>tabid of base table or view</td>
</tr>
<tr>
<td>btype</td>
<td>char(1)</td>
<td>base object type (table or view)</td>
</tr>
<tr>
<td>dtabid</td>
<td>integer</td>
<td>tabid of dependent table</td>
</tr>
<tr>
<td>dtype</td>
<td>char(1)</td>
<td>dependent object type (only view now)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index Name</th>
<th>Type</th>
<th>Columns</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>btabid</td>
<td>dups</td>
<td>btabid</td>
<td></td>
</tr>
<tr>
<td>dtabid</td>
<td>dups</td>
<td>dtabid</td>
<td></td>
</tr>
</tbody>
</table>

### The SYSSYNONYMS Catalog

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>char(8)</td>
<td>user name of owner</td>
</tr>
<tr>
<td>synonym</td>
<td>char(18)</td>
<td>synonym identifier</td>
</tr>
<tr>
<td>created</td>
<td>date</td>
<td>date synonym created</td>
</tr>
<tr>
<td>tabid</td>
<td>integer</td>
<td>table identifier</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index Name</th>
<th>Type</th>
<th>Columns</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>synonym</td>
<td>unique</td>
<td>owner, synonym</td>
<td></td>
</tr>
<tr>
<td>syntabid</td>
<td>dups</td>
<td>tabid</td>
<td></td>
</tr>
</tbody>
</table>
### The SYSUSERS Catalog

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>char(8)</td>
<td>user login identifier</td>
</tr>
<tr>
<td>usertype</td>
<td>char(1)</td>
<td>D = DBA, R = RESOURCE, C = CONNECT</td>
</tr>
<tr>
<td>priority</td>
<td>smallint</td>
<td>reserved for future use</td>
</tr>
<tr>
<td>password</td>
<td>char(8)</td>
<td>reserved for future use</td>
</tr>
</tbody>
</table>

**Index**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>users</td>
<td>unique</td>
<td>username</td>
</tr>
</tbody>
</table>

### The SYSVIEWS Catalog

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabid</td>
<td>integer</td>
<td>table identifier</td>
</tr>
<tr>
<td>seqno</td>
<td>smallint</td>
<td>sequence number</td>
</tr>
<tr>
<td>viewtext</td>
<td>char(64)</td>
<td>portion of SELECT statement</td>
</tr>
</tbody>
</table>

**Index**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>view</td>
<td>unique</td>
<td>tabid, seqno</td>
</tr>
</tbody>
</table>

### The SYSCONSTRAINTS Catalog

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>constrname</td>
<td>char(18)</td>
<td>constraint identifier</td>
</tr>
<tr>
<td>owner</td>
<td>char(8)</td>
<td>user name of owner</td>
</tr>
<tr>
<td>tabid</td>
<td>integer</td>
<td>table identifier</td>
</tr>
<tr>
<td>idxname</td>
<td>char(18)</td>
<td>index name</td>
</tr>
</tbody>
</table>

**Index**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>constrname</td>
<td>unique</td>
<td>constrname, owner</td>
</tr>
<tr>
<td>constrtabid</td>
<td>dups</td>
<td>tabid</td>
</tr>
</tbody>
</table>
The SYSSYNTABLE Catalog

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabid</td>
<td>integer</td>
<td>table identifier</td>
</tr>
<tr>
<td>servername</td>
<td></td>
<td>used on INFORMIX-OnLine only</td>
</tr>
<tr>
<td>dbname</td>
<td></td>
<td>used on INFORMIX-OnLine only</td>
</tr>
<tr>
<td>owner</td>
<td></td>
<td>used on INFORMIX-OnLine only</td>
</tr>
<tr>
<td>tabname</td>
<td></td>
<td>used on INFORMIX-OnLine only</td>
</tr>
<tr>
<td>btabid</td>
<td>integer</td>
<td>tabid of base table or view</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index Name</th>
<th>Type</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>synntabid</td>
<td>unique</td>
<td>tabid</td>
</tr>
<tr>
<td>synntabid</td>
<td>dups</td>
<td>btabid</td>
</tr>
</tbody>
</table>
Accessing Programs from the Operating System

You can access the modules that comprise INFORMIX-SQL (I-SQL) in three ways:

- From the I-SQL Main menu
- Directly from the operating system command line using the module names
- Directly from the operating system command line using a shortened version of the I-SQL Main menu options

Chapter 1 of this manual describes the I-SQL Main menu.

If you are accessing a program through either the Main menu or the User-menu option, and you receive an error message indicating that you have run out of space in memory, exit to the command line. This action clears memory. Then enter `isql` from the command line and you can resume your operations.

The command line syntax for accessing each I-SQL module is described later in this appendix. The procedure for using the shortened version of the I-SQL Main menu from the command line is described in the next section.
For easy reference, the I-SQL Main menu options follow:

<table>
<thead>
<tr>
<th>Menu Options</th>
<th>Menu Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>Run, Modify, Generate, New, Compile, Drop</td>
</tr>
<tr>
<td>Report</td>
<td>Run, Modify, Generate, New, Compile, Drop</td>
</tr>
<tr>
<td>Query-language</td>
<td>New, Run, Modify, User-editor, Output, Choose,</td>
</tr>
<tr>
<td></td>
<td>Save, Info, Drop</td>
</tr>
<tr>
<td>Database</td>
<td>Create, Drop</td>
</tr>
<tr>
<td>User-menu</td>
<td>Run, Modify</td>
</tr>
<tr>
<td>Table</td>
<td>Create, Alter, Info, Drop</td>
</tr>
</tbody>
</table>

Figure H-1  INFORMIX-SQL Main menu Options

Accessing FORM Menu Options

The operating system command-line syntax for accessing I-SQL FORM menu options is:

```
isql -s database-name -f form-name - options
```

- `isql` is the program call for I-SQL.
- `-s` calls the `silent` option and suppresses all non-essential screen messages.
- `database-name` is the name of a database in your current directory or a directory cited in your DBPATH environment variable.
- `-f` calls the `Form` option from the I-SQL Main menu.
- `options` are the first letters of the FORM menu options you select. Do not include a blank space between `-f` and any option letters.
- `form-name` is the name of the form you want to access. Do not include an extension.

I-SQL returns you to the operating system after you complete the specified operation.

The following command runs the `customer` form:

```
isql -fr customer
```

The following generates a form based on the `stores2` database:

```
isql stores2 -fg
```
Accessing REPORT Menu Options

The operating system command-line syntax for accessing I-SQL REPORT menu options is:

```
isql [-s] database-name -ansi [-r] options report-name
```

- `isql` is the program call for I-SQL.
- `-s` calls the `silent` option and suppresses all non-essential screen messages.
- `database-name` is the name of a database in your current directory or a directory cited in your DBPATH environment variable.
- `-ansi` causes I-SQL to generate a warning if it encounters an Informix extension to the SELECT statement when compiling your report.
- `-r` calls the `Report` option from the I-SQL Main menu.
- `options` are the first letters of the REPORT menu options you select. Do not include a blank space between `-r` and any option letters.
- `report-name` is the name of the report you want to access. Do not include an extension in the report name.

I-SQL returns you to the operating system after you complete the specified operation.

The following command compiles the `clist1` report:

```
isql -rc clist1
```
Accessing QUERY-LANGUAGE Menu Options

The operating system command-line syntax for accessing I-SQL QUERY-LANGUAGE menu options is:

```
isql -s database-name -ansi -q options command-file-name
```

- **isql** is the program call for I-SQL.
- **-s** calls the *silent* option and suppresses all non-essential screen messages.
- **-** indicates that the database name is created or established in the command file.
- **database-name** is the name of a database in your current directory or a directory cited in your DBPATH environment variable.
- **-ansi** causes I-SQL to generate a warning whenever it encounters an Informix extension to ANSI syntax.
- **-q** calls the *Query-language* option from the I-SQL Main menu.
- **options** are the first letters of the QUERY-LANGUAGE menu options you select. Do not include a blank space between -q and any option letters.
- **command-file** is the name of the .sql file that you want to access. Do not include an extension in the command filename.

I-SQL returns you to the operating system after you complete the specified operation.

The following command chooses the ex1 file and makes the SQL statements it contains the current statements:

```
isql -s stores2 -qc ex1
```
Accessing DATABASE Menu Options

The operating system command-line syntax for accessing I-SQL DATABASE menu options is:

```
isql -s database-name -d options database-name
```

- `isql` is the program call for I-SQL.
- `-s` calls the `silent` option and suppresses all non-essential screen messages.
- `database-name` is the name of a database in your current directory or a directory cited in your DBPATH environment variable.
- `-d` calls the `Database` option from the I-SQL Main menu.
- `options` are the first letters of the DATABASE menu options you select. Do not include a blank space between `-d` and any option letters.
- `database-name` is the name of the database you want to access.

I-SQL returns you to the operating system after you complete the specified operation.

The following statement calls the `Select` option on the DATABASE menu:

```
isql -s -ds
```

Accessing USER-MENU Menu Options

The operating system command-line syntax for accessing I-SQL USER-MENU menu options is:

```
isql -s database-name -u options user-menu-name
```

- `isql` is the program call for I-SQL.
- `-s` calls the `silent` option and suppresses all non-essential screen messages.
- `database-name` is the name of a database in your current directory or a directory cited in your DBPATH environment variable.
- `-u` calls the `User-menu` option from the I-SQL Main menu.
Accessing TABLE Menu Options

options are the first letters of the USER-MENU menu options you select. Do not include a blank space between -u and any option letters.

user-menu-name is the name of the User-menu you want to run.

I-SQL returns you to the operating system after you complete the specified operation.

The following command runs the User-menu for the stores2 database:

isql stores2 -ur

Accessing TABLE Menu Options

The operating system command-line syntax for accessing I-SQL TABLE menu options is:

```
isql -s database-name -t options -table-name
```

- **isql** is the program call for I-SQL.
- **-s** calls the *silent* option and suppresses all non-essential screen messages.
- **database-name** is the name of a database in your current directory or a directory cited in your DBPATH environment variable.
- **-t** calls the *Table* option from the I-SQL Main menu.
- **options** are the first letters of the TABLE menu options you select. Do not include a blank space between -t and any option letters.
- **table-name** is the name of the table you want to access.

I-SQL returns you to the operating system after you complete the specified operation.

The following command creates a table in the stores2 database:

```
isql -s stores2 -tc
```
The command syntax for compiling a customized screen form directly from the operating system is:

```
sformbld -d filename
```

- `sformbld` is the program call for FORMBUILD.
- `-d` replaces `filename` and instructs FORMBUILD to prompt you for the information required to create and compile a default form specification.
- `-s` calls the `silent` option and suppresses all non-essential screen messages.
- `-l lines` are optional symbols and an integer to specify the total number of lines of characters (measured vertically) that the terminal can display. (The default is 24.)
- `-c cols` are optional symbols and an integer to specify the width of the screen, in characters. (The default is the number of characters in the longest line of the screen layout, as specified in the SCREEN section.)
- `-v` tells FORMBUILD to verify that the fields contained in the screen section of the form specification are consistent with the field widths of the corresponding columns.

**Usage**

If the compilation is successful, FORMBUILD creates a compiled form specification named `filename.frm`. You can use this compiled form specification with PERFORM as a screen form. If the compilation is unsuccessful, FORMBUILD creates an error file named `filename.err`. You must edit the error file, remove the error messages, and recompile with FORMBUILD before you can use the screen form.
You can also create a customized screen form directly from the operating system using the shortened version of the **I-SQL** Main menu options. This method is described earlier in this appendix.
PERFORM

The command syntax for running a compiled screen form directly from the operating system is:

```
sperform filename
```

- `sperform` is the program call for PERFORM.
- `filename` is the name of the compiled form specification file. Do not include the `.frm` extension (`filename.frm`) on the command line.

**Usage**

The maximum number of `filenames` you can include on the command line is operating system dependent.

PERFORM displays each form in the order that it appears on the command line.

If PERFORM cannot display a form, it aborts. When multiple filenames are included on the command line, subsequent filenames are not displayed.

You can also run a compiled screen form directly from the operating system using the shortened version of the I-SQL Main menu options. This method is described earlier in this appendix.
ACEPREP

The command syntax for compiling a customized report form directly from the operating system is:

```
saceprep -s filename -ansi -o directory-name
```

- `saceprep` is the program call for ACEPREP.
- `-s` calls the *silent* option and suppresses all non-essential screen messages.
- `-ansi` tells ACEPREP to generate a warning when it encounters an Informix extension to the SELECT statement in your report.
- `-o directory-name` tells ACEPREP to place the output file (either the compiled report specification or the error file) in the indicated directory.
- `filename` is the name of the report specification file. Do not include the `.ace` extension (*filename.ace*) on the command line.

**Usage**

If the compilation is successful, ACEPREP creates a compiled report specification file named *filename.arc*. You can use this compiled report specification with ACEGO to produce a report. If the compilation is unsuccessful, ACEPREP creates an error file named *filename.err*. You must edit the error file, remove the error messages, and recompile with ACEPREP before you can run the report.

You can also compile a customized report form directly from the operating system using the shortened version of the I-SQL Main menu options. This method is described in the section "Accessing REPORT Menu Options" on page H-3.
The command syntax for running a compiled report directly from the operating system is:

```
sacego -s -d database-name filename
```

**sacego** is the program call for ACEGO.

* -s calls the silent option and suppresses all non-essential screen messages.

* -d database-name overrides the database that is named in the report specification and substitutes database-name.

* filename is the name of the compiled report specification. Do not include the .ace extension (filename.ace) on the command line.

**Usage**

The maximum number of filenames you can include on the command line is operating system dependent.

ACEGO executes each report in the order in which it appears on the command line.

If ACEGO cannot execute a report, it aborts. When multiple filenames are included on the command line, subsequent filenames are not executed.

You can also run a compiled report directly from the operating system using the shortened version of the I-SQL Main menu options. This method is described in the section “Accessing REPORT Menu Options” on page H-3.
Accessing Programs from the Operating System
Using the INFORMIX-OnLine Dynamic Server

This appendix summarizes how to use INFORMIX-SQL (I-SQL) with an INFORMIX-OnLine Dynamic Server database server. This appendix is organized as follows:

- Using the Table option of the Main menu
- Using the INFO menu
- Querying VARCHAR, TEXT, and BYTE data
- Viewing TEXT and BYTE data in PERFORM
- Creating and altering tables
- Loading and unloading VARCHAR, TEXT, and BYTE data
- Creating reports with VARCHAR, TEXT, and BYTE data
- Creating forms with VARCHAR, TEXT, and BYTE data
- Specifying remote databases and tables in forms
Using the Table Option of the Main Menu

If you use the Table option to create a table, you get only the default size for initial and next extents. If you want other sizes, you must use the I-SQL Interactive Editor to execute the CREATE TABLE statement that contains the explicit extent sizes.

When you are connecting to an OnLine database server, the ADD or MODIFY TYPE menu includes an additional choice of Variable-length as shown in the following screen:

If you select Variable-length, the menu shows types unique to OnLine, as shown here:

You can select any of these types to set up a variable-length column in your table.
If you select the VARCHAR data type, you are prompted for the column length. A VARCHAR column has two lengths: a maximum size and a minimum space. You can specify these two numbers at the subsequent prompts, as shown in the following screens:

ADD MAXIMUM LENGTH >>
Enter maximum length of data from 1 to 255. RETURN adds it.

ADD MINIMUM SPACE >>
Enter amount of space to reserve for each item from 0 to max length.

If you select either TEXT or BYTE, you must indicate where the data is stored. You see the BLOBSPACE menu, as shown here:

ADD BLOBSPACE tab1: Table BLOBSpace-name
Column data is stored in the same table-space as other columns.
If you choose Table, the column data is stored in the same dbspace as the other columns. If you choose BLOBSpace-name, you see the following prompt:

```
ADD BLOBSPACE NAME >>
Enter name of BLOBSpace
```

You can enter the name of any existing blobspace at the prompt.

### Using the INFO Menu

You can use the INFO menu to display information about a table. To display the INFO menu, follow these steps:

1. Select the **Query-Language** option from the Main menu.
2. Select the **Info** option from the SQL menu.

You are then prompted for the name of a table as follows:

```
INFO FOR TABLE>>
```

In addition to the tables listed, you can request information about external tables. To specify an external table, you must enter the expanded table name at the prompt. For example, the following entry requests information from the `richard.customer` table in the `stores2` database that accesses the INFORMIX-OnLine system called `central`:

```
INFO FOR TABLE >> stores2@central:richard.customer
```

You can also use synonyms in place of the extended table name.

If you select the Status option of the INFO menu, I-SQL displays information on the dbspace that contains the table. The Status option does not display audit trail information because the logging facility replaces audit trails.
Querying VARCHAR, TEXT, and BYTE Data

The INFORMIX-SQL Interactive Editor displays the results of a query in a format that depends on the data type of the selected column. If you execute a query on a VARCHAR column, I-SQL displays the entire VARCHAR value, just as it displays CHAR values. If you select a TEXT column, I-SQL displays the contents of the TEXT column and you can scroll through the contents one screen at a time by using the Next option. If you select a BYTE column, I-SQL displays the words <BYTE value>.

Viewing Blobs in PERFORM

You can use the View option of the PERFORM menu to display the contents of TEXT fields, and of BYTE fields that are referenced in your form with the PROGRAM attribute. Blobs (Binary Large Objects) include the TEXT and BYTE data types. You can only display the contents of the blob; you cannot change the blob from within the PERFORM form.

The following screen shows the PERFORM menu with the View option highlighted:

When you select the View option, I-SQL positions the cursor on the first TEXT field, or the first BYTE field that uses the PROGRAM attribute. To display the blob, type an exclamation point (!). Press RETURN, TAB, or the down arrow key to skip to the next blob field that can be displayed; type an up arrow key to move to the previous blob field. Press ESCAPE to exit the View option and to redisplay the Main menu.

If you select the View option and the form contains no blob fields, I-SQL displays the following error message:

There are no BLOB fields to view.
Creating and Altering Tables

When you use the CREATE TABLE and ALTER TABLE statements, you can place quotes around blobspace names as shown in the following example:

```
CREATE TABLE mytab (column1 TEXT IN "blob1")
```

In this case, the quotes are optional. However, if the name of your blobspace is `table`, I-SQL requires the quotes to distinguish the blobspace name with the keyword `TABLE`. This is demonstrated in the following ALTER TABLE statement:

```
ALTER TABLE mytab ADD (column1 TEXT IN "table")
```

In this case, the quotes are required to avoid any ambiguity with the keyword `TABLE`.

Loading and Unloading VARCHARs and Blobs

You can use the LOAD and UNLOAD statements to transfer data between a table and an operating system file of ASCII data. This file contains only printable ASCII and newline characters.

You can use these statements on tables and files that contain the VARCHAR, TEXT, and BYTE data types. You should read these sections if you are loading or unloading files that contain VARCHAR or blob data.

For more information on using the LOAD and UNLOAD statements, see Informix Guide to SQL: Syntax, Version 6.0.

UNLOAD Statement

If you are unloading files containing VARCHAR, TEXT, or BYTE data types, note the following information:

- BYTE items are written in hexadecimal dump format with no spaces or newline characters. Thus the logical length of an unloaded file that contains BYTE items can be very long, and it might be impossible to print or edit such a file.
- Trailing blanks are retained in VARCHARs.
• Do not use the following characters as delimiting characters in an unload file:
  o 0-9
  o a-f
  o A-F
  o space
  o tab
  o \n
LOAD Statement
If you are loading files containing VARCHAR, TEXT, and BYTE data types, note the following information:
• You can give the LOAD statement data in which the character (including VARCHAR) fields are longer than the column size; the excess characters are disregarded.
• You can have leading and trailing blanks in noncharacter fields, except BYTE fields.
• In all character fields (including VARCHAR and TEXT), embedded delimiter and backslash characters are escaped with the backslash.
• In VARCHAR columns, you must escape newline characters.
• Data being loaded into a BYTE column must be in ASCII-hexadecimal form. BYTE columns cannot contain preceding blanks.
• Do not use the following characters as delimiting characters in a load file:
  o 0-9
  o a-f
  o A-F
  o space
  o tab
  o \
Creating Reports with VARCHARs and Blobs

You can use VARCHAR and TEXT columns in ACE report routines. You cannot use columns of type BYTE in a report.

Printing VARCHARs

VARCHAR columns and variables in expressions act the same way as CHAR columns and variables. When you define a VARCHAR in a report, do not give the min-space parameter with which the VARCHAR was defined for the database. Rather, indicate how many characters you want printed. For example, if in the employee table history is defined as VARCHAR(255,10), in the report you should define it as VARCHAR(255). If you only want to output a portion of the column, you can define VARCHAR with a shorter length, such as VARCHAR(120).

For example, you could specify the following in a report:

```plaintext
define
  variable history varchar(255)
end```

Printing TEXT

You cannot name a TEXT column in any arithmetic, aggregate, or Boolean expression, or in a BEFORE GROUP OF or AFTER GROUP OF clause.

You can name a TEXT column in a PRINT statement. The PRINT statement acts like a PRINT FILE statement with the TEXT item as a file.

Creating Forms with VARCHAR, TEXT, and BYTE Data

When you create forms using FORMBUILD in I-SQL you can use the data types VARCHAR, TEXT, and BYTE. The following sections summarize how to specify and work with VARCHARs and blobs in forms.

Specifying VARCHARs in Forms

You can designate a field in a form as type VARCHAR. VARCHARs are similar to character fields; both are supported by the multiline editor. You must assign the WORDWRAP attribute to VARCHAR fields to enable the multiline
Creating Forms with VARCHAR, TEXT, and BYTE Data

editor. For example, the following excerpt from a form specification shows the VARCHAR field history in the employee table and the attributes assigned to the field:

```
history [f002 ]
[f002 ]
[f002 ]
```

attributes

```
f002 = employee.history, WORDWRAP COMPRESS;
```

If you generate a default form for a table that has a VARCHAR column, the VARCHAR field is broken into subscripted fields. To enable WORDWRAP, revise the form and use the same field tag for all the components of the VARCHAR field; then add the WORDWRAP and COMPRESS attributes.

You can use VARCHAR as the data type for a display-only (FORMBUILD) field.

You can use the DEFAULT attribute to give a VARCHAR field a default value.

### Specifying TEXT and BYTE Data in Forms

You can use columns of types TEXT or BYTE as fields. Assign the WORDWRAP attribute to a TEXT field to display the field so that it fits into the display without having any lines start with a blank. For a TEXT field, the WORDWRAP attribute only affects how the value is displayed; WORDWRAP does not enable the multiline editor. If you want to edit a TEXT field, you must use the PROGRAM attribute to indicate the name of an external editor.

A BYTE field is never displayed; the phrase `<BYTE value>` is shown in its display field to indicate that the user cannot see the BYTE data. The following excerpt from a form specification shows a TEXT field resume and a BYTE field photo. You do not need to have more than one line on a form for a TEXT field. The BYTE field is short because only the words `<BYTE value>` are displayed.

```
resume [f003 ]
photo [f004 ]
```

attributes

```
f003 = employee.resume
f004 = employee.photo
```
You can also use the keywords TEXT and BYTE when you define a display-only (FORMBUILD) field. There is marginal value in designating one field as type BYTE because only the words <BYTE value> are displayed on the screen.

You cannot use the DEFAULT attribute with fields of type TEXT or BYTE.

**Specifying External Programs for TEXT and BYTE Data in Forms**

You can use the field attribute, PROGRAM, with a BYTE or TEXT column to call an external program to work with the TEXT or BYTE data. You invoke an external program by pressing the exclamation key while your cursor is in a blob field. The external program then takes over control of the screen. When you exit the external program, the form is restored on your screen.

If you call the program on an empty field, when you finish working in the external program and save your work, the data is stored in the blob field. If you call the program from a field that already contains data, the specified program works on the data in that field. In either case, INFORMIX-OnLine writes the blob to a temporary file, which is then passed to the external program. The external program must write its changes back to the temporary file. You do not need to know the name of the temporary file; the application development tool keeps track of it. For example, you might use PROGRAM to call a CAD or graphics program to display a drawing that you have stored. You can also use PROGRAM to invoke an editor for a TEXT field.

For example, a TEXT field might be tagged with the following line:

```
f003 = personnel.resume, WORDWRAP, PROGRAM = "edit";
```

The syntax of the PROGRAM attribute follows.
PROGRAM

Use the PROGRAM attribute to specify an external application program for use with screen fields of type TEXT or BYTE.

field-tag = [ table. ] column, PROGRAM = "name"

field-tag is the field tag used in the SCREEN section.

table.column is the name of a field, either related to a column, form-only field, or display-only (I-SQL) field.

PROGRAM is a required keyword.

name is a command string to or the name of a batch file that invokes an editing program.

Usage

When you display a field with data type TEXT I-SQL displays as many of the leading characters as will fit in the defined field. When you display a field with data type BYTE I-SQL displays <BYTE value>.

When you place the cursor in a TEXT field, and you press the exclamation-mark key in the first character position of a TEXT or BYTE field, the external program is invoked. This program receives the contents of the field and takes control of the screen to permit editing or alteration of the field. When the program is finished, your application regains control of the screen and continues execution.

One of the following programs is invoked for a TEXT field:

- The program specified by the PROGRAM ="name" attribute defined for that field, if any.
- The program named in the DBEDIT environment variable, if one is defined.
- The default editor, which depends on the host operating system.

You must explicitly define the external program for a BYTE field; the default editor is not called, and the DBEDIT variable is not examined.

Before invoking the program, I-SQL copies the BYTE or TEXT field to a temporary disk file. It then issues a system command composed of the name that you specify after the PROGRAM keyword followed by the name of the temporary file.
The name string need not be a single word. You can add additional command parameters. The program can also be a shell script, so that you can initiate a whole series of actions.

If you are invoking an external program from PERFORM, the data is stored in the blob column when you complete the Add or Update. For example:

```
f010 = contract, PROGRAM = "edit";
```

### Specifying Remote Databases and Tables in Forms

You can specify remote databases, external tables, and external, distributed tables in forms. You can make a remote database the current database for use with the form, or you can specify a table external to the current database in your form.

#### Remote Databases

You can specify the full name of a remote database in the DATABASE section of the form. List the simple names of the tables in the TABLES section. You can also use table-name synonyms in the TABLE section as long as they have been defined for the current database. These synonyms can stand for tables in the current database or in other databases.

In the ATTRIBUTES section of the form, refer to tables in the current database with their simple table names. You can also use synonyms in the ATTRIBUTES section as follows:

```
ATTRIBUTES
  f0 = table-name.colname, ......;
  f1 = synonym.colname, ......;
  f2 = view-name.colname, ......;
  .
  .
  .
```
External Tables and Synonyms

If you want to use a table or synonym that has a multipart name, you must first give it an alias. Examples of objects that require table aliases follow:

- External tables
- Tables within the current database that are qualified by their owner names
- Multipart synonyms

Use the following syntax to give an object a table alias.

```
TABLES
  
  table-alias=database@servername:[owner.]table
  table-alias=[owner.]table
```

The table-alias is a single-word identifier.

In the ATTRIBUTES and INSTRUCTIONS sections of the form, refer to external tables in the following way:

```
ATTRIBUTES
  
  f3 = table-alias.colname, .....;
  
  for example, if you have the following declaration in your tables section:

  timecard_a = otherdb:acctg.timecard

  part of your attributes section might look like this:

  f3 = timecard_a.sickleave
```

You see the alias rather than the actual table name in error messages. If you are using PERFORM, you see the alias instead of the table name on the second line of the screen.
Index

This index covers both the User and the Reference Manual. Page numbers that end in U can be found in the User, and those that end in R can be found in the Reference Manual. Special symbols are listed in ASCII order at the end of the index.

A

ACE report
  calling C functions 6-4R
  DEFINE section in specification file 6-4R
  example 6-33R, 6-35R
  FORMAT section in specification file 6-4R
  how to compile 6-8R
  steps in creating 8-4U
  using sacego 6-32R
  using saceprep 6-8R
ACE report writer
  addition 8-36U
  AFTER GROUP OF control block 8-34U, 4-43R
  aggregates 8-36U
  arithmetic 8-36U
  ASCII expression 4-71R
  ASCII statement 4-16R
  AVERAGE aggregate 8-40U
  BEFORE GROUP OF control block 8-34U, 4-45R
  BOTTOM MARGIN statement 8-26U, 4-27R
  calculations 8-36U
  calling C functions from 8-49U
  clauses in 8-20U, 4-12R
  CLIPPED expression 4-73R
  COLUMN expression 4-74R
  column headings 8-29U
  command line options 4-8R
  comments 8-21U
  compiling a report specification 8-13U, 8-18U, 4-5R
  compound statement 4-55R
  control block 8-20U, 8-27U, 4-41R
  correcting a report specification 8-18U
COUNT aggregate 8-40U, 8-43U
creating a report specification 8-11U
CURRENT expression 4-75R
custom report 8-27U
DATABASE section 8-8U, 4-13R, 4-14R
DATE expression 4-76R
DATE( ) function 4-77R
DAY( ) function 4-78R
default report 8-26U
DEFINE section 4-13R
definition of 1-20U, 8-4U
displaying a report 8-47U
division 8-36U
END keyword 8-20U
error messages 8-19U, 4-11R
every ROW statement 8-26U, 4-39R
expressions 4-10R
expressions, formatting 4-86R
filename conventions 4-9R
fill characters 8-38U, 8-39U
FIRST PAGE HEADER control block 4-47R
FOR statement 4-56R
FORMAT section 8-10U, 8-22U, 8-26U, 4-13R
formatting a report 8-25U
formatting number expressions 8-37U, 4-86R
fill characters 8-38U, 8-39U
FIRST PAGE HEADER control block 8-28U, 4-52R
PAGE HEADER control block 8-28U, 4-52R
PAGE LENGTH statement 8-26U, 4-28R
page numbers 8-30U
PAGE TRAILER control block 8-11U, 4-54R
PAGENO expression 8-30U, 4-82R
PARAM statement 4-18R
parentheses 8-37U
PAUSE statement 4-62R
PRINT FILE statement 4-65R
PRINT statement 4-63R
PRINT USING statement 8-37U
PROMPT FOR statement 4-21R
READ section 4-13R
READ statement 4-33R
report specifications 4-12R, A-34R
REPORT TO statement 4-23R
RIGHT MARGIN statement 4-25R
running a report 8-14U, 4-5R
sample report specification 8-7U
SELECT section 8-8U, 8-22U, 4-13R
SELECT statement 4-30R
selecting all rows and columns 8-23U
selecting data for a report 8-22U
selecting data from several tables 8-23U
SKIP statement 4-66R
SKIP TO TOP OF PAGE statement 4-67R
SPACES expression 4-83R
subtraction 8-36U
summary of sections 8-20U, 4-13R
TIME expression 4-84R
TODAY expression 4-85R
TOP MARGIN statement 8-26U, 4-24R
TOP OF PAGE statement 4-29R
TOTAL aggregate 8-40U
trailers 8-27U
USING expression 8-37U, 4-86R
using with menus 4-5R
WEEKDAY( ) function 4-93R
WHILE statement 4-68R
WORDWRAP expression 4-94R
YEAR( ) function 4-95R

ON EVERY ROW control block 8-30U, 4-49R
ON LAST ROW control block 4-51R
OUTPUT section 8-25U, 8-47U, 4-13R
PAGE HEADER control block 8-28U, 4-52R
PAGE LENGTH statement 8-26U, 4-28R
page numbers 8-30U
PAGE TRAILER control block 8-11U, 4-54R
PAGENO expression 8-30U, 4-82R
PARAM statement 4-18R
parentheses 8-37U
PAUSE statement 4-62R
PRINT FILE statement 4-65R
PRINT statement 4-63R
PRINT USING statement 8-37U
PROMPT FOR statement 4-21R
READ section 4-13R
READ statement 4-33R
report specifications 4-12R, A-34R
REPORT TO statement 4-23R
RIGHT MARGIN statement 4-25R
running a report 8-14U, 4-5R
sample report specification 8-7U
SELECT section 8-8U, 8-22U, 4-13R
SELECT statement 4-30R
selecting all rows and columns 8-23U
selecting data for a report 8-22U
selecting data from several tables 8-23U
SKIP statement 4-66R
SKIP TO TOP OF PAGE statement 4-67R
SPACES expression 4-83R
subtraction 8-36U
summary of sections 8-20U, 4-13R
TIME expression 4-84R
TODAY expression 4-85R
TOP MARGIN statement 8-26U, 4-24R
TOP OF PAGE statement 4-29R
TOTAL aggregate 8-40U
trailers 8-27U
USING expression 8-37U, 4-86R
using with menus 4-5R
WEEKDAY( ) function 4-93R
WHILE statement 4-68R
WORDWRAP expression 4-94R
YEAR( ) function 4-95R
Index

ACEGO, running a compiled report with H-11R
ACEPREP, compiling report specifications with 6-8R, H-10R
Active table definition of 3-5U, 5-4U selecting a different 5-4U
ADD NUMBER menu, schema editor 2-15U
Add option ALTER TABLE menu 2-24U CREATE TABLE menu 2-11U, 2-13U PERFORM 3-12U, 3-19R
ADD STARTING NUMBER screen, schema editor 2-18U
ADD TYPE menu, schema editor 2-14U, 2-18U
Adding data, in PERFORM 3-12U
AFTER control block, in PERFORM 2-64R
AFTER GROUP OF control block 8-10U
AFTER GROUP OF control block, in ACE reports 8-34U, 4-43R
Aggregate functions, listed 2-74R
Alias, table I-13R
Alter option, TABLE menu 2-10U, 2-24U, 1-17R
ALTER TABLE menu Build-new-table Exit option 2-24U Drop option 9-7U options 2-24U
ALTER TABLE statement C-7R, C-11R, C-19R
ANSI C-5R, C-16R, C-35R, C-39R, C-47R ANSI-compliance and DBANSIWARN B-8R and reserved words F-1R, F-9R -ansi flag B-8R
Arrows in syntax diagrams Intro-7R AS keyword F-6R
ASCII character chart E-2R
ASCII character set C-2R, C-3R, C-5R, C-9R, C-11R, C-24R, C-31R
ASCII collation C-4R, C-18R, C-23R
ASCII expression, in ACE reports 4-71R
ASCII file and reports 4-17U and UNLOAD statement 4-17U as input to an application 4-17U retrieving data for a report 8-24U
ASCII statement, in ACE reports 8-25U, 4-16R
At (@) symbol. See Trailing currency symbol.
Audit trail definition of 9-9U versus a transaction log 9-9U
AUTONEXT attribute 2-29R
B

BEFORE control block, in PERFORM 2-63R
BEFORE GROUP OF control block, in ACE reports 8-34U, 4-45R
Blank characters
  as list separators Intro-7R
  default character value 2-9R
Blob
  querying with INFORMIX-OnLine I-5R
  specifying external programs in forms I-10R
  specifying in forms I-9R
  supported in OnLine Intro-14U
  transferring with LOAD and UNLOAD I-6R
  viewing in PERFORM I-5R
BOTTOM MARGIN statement, in ACE reports 8-26U, 4-27R
Bourne shell
  how to set environment variables B-4R
  .profile file B-2R
BYTE data type
  definition of 2-15U
  determining length 2-15U
  in forms 3-10R
  in NLS C-4R, C-6R, C-10R, C-19R
Character set
  defined C-3R
  mentioned C-24R, C-31R
CHOOSE DATABASE screen, selecting options from 2-9U
Choose option, SQL menu 7-14U, 1-13R
CLIPPED expression, in ACE reports 4-73R
CLOSE DATABASE statement 9-5U
Clustered indexes 2-21U
Collation C-2R, C-4R, C-5R, C-9R, C-11R, C-14R, C-21R, C-27R
COLLCHAR environment variable
  interaction with DBNLS C-17R, C-20R
  mentioned C-2R, C-5R, C-12R
  settings C-20R
  syntax C-19R
COLOR attribute
  in PERFORM 6-20U, 2-30R
  intensity list 2-30R

C shell
  how to set environment variables B-4R
  .cshrc file B-2R
  .login file B-2R
cace program, used to customize sacego 6-32R
Calculations in reports 8-36U
CALL keyword
  in ACE report specification file 6-7R
  in PERFORM form specification file 6-10R
Calling C functions
  in ACE 6-6R
  in PERFORM 6-32U
catalog table in stores demonstration database A-6R
Category, NLS C-2R
CHANGE ANYWAY menu, schema editor 9-7U
CHAR data type
  definition of 2-15U
  determining length 2-15U
  in forms 3-10R
  in NLS C-4R, C-6R, C-10R, C-19R
Character set
  defined C-3R
  mentioned C-24R, C-31R
CHOOSE DATABASE screen, selecting options from 2-9U
Choose option, SQL menu 7-14U, 1-13R
CLIPPED expression, in ACE reports 4-73R
CLOSE DATABASE statement 9-5U
Clustered indexes 2-21U
Collation C-2R, C-4R, C-5R, C-9R, C-11R, C-14R, C-21R, C-27R
COLLCHAR environment variable
  interaction with DBNLS C-17R, C-20R
  mentioned C-2R, C-5R, C-12R
  settings C-20R
  syntax C-19R
COLOR attribute
  in PERFORM 6-20U, 2-30R
  intensity list 2-30R
Color, setting INFORMIXTERM for B-35R
COLUMN expression, in ACE reports 4-74R

Columns
adding, using the TABLE menu 9-6U
assigning a data type 2-14U, 2-18U
CHAR data type 2-15U
column names 1-6U
correspondence between data type and PERFORM field entry 3-13U
DATE data type 2-17U
DATETIME data type 2-17U
DECIMAL data type 2-15U
defining a column 2-12U
defining for a table 2-12U
definition of 1-6U
displaying with the Info option 7-17U
dropping, using the ALTER TABLE menu 9-7U
FLOAT data type 2-16U
in stores demonstration database A-4R to A-7R
in stores demonstration database tables A-4R
indexing 2-19U
INTEGER data type 2-16U
INTERVAL data type 2-18U
joining in multiple-table forms 5-7U
locale-sorted C-6R, C-8R
MONEY data type 2-17U
naming conventions 2-13U
naming with the Add option 2-13U
NULL value in 2-21U
number data type 2-15U
searching for 1-8U
searching for within rows 1-8U
SERIAL data type 2-16U
SMALLFLOAT data type 2-16U
SMALLINT data type 2-16U

Comma symbol. See Thousands separator.

Command files
guidelines for naming 7-14U
how to choose 7-15U
saving statements 7-14U

Command line
accessing ACE 4-8R
accessing DATABASE menu H-5R
accessing FORM menu H-2R
accessing I-SQL modules H-1R
accessing PERFORM H-9R
accessing QUERY-LANGUAGE menu H-4R
accessing REPORT menu H-3R
accessing TABLE menu H-6R
accessing USER-MENU menu H-5R
compiling a customized report form H-10R
compiling a customized screen form H-7R
using ACEGO H-11R
using ACEPREP H-10R
using sperform 2-6R

Comments
displaying on screen in PERFORM 6-21U
in SQL statements 7-16U
including in a report specification 8-21U
COMMENTS attribute, in PERFORM 6-21U, 2-33R

Compile option
FORM menu 1-11R
REPORT menu 1-15R
COMPILE REPORT menu 8-19U

Compiling
definition of 6-7U
form specifications 1-11R, 6-14R
report specifications 8-18U, 6-8R, H-10R
reports 1-15R
with saceprep 6-8R
with sformbld 6-14R

Compiling form specifications H-7R
Composite joins, in PERFORM 2-57R

Compound statement, in ACE 4-55R

CONNECT statement, and
INFORMIXSERVER environment variable B-33R

Connection
setting the INFORMIXCONRETRY environment variable B-30R
setting the INFORMIXCONTIME environment variable B-31R
Consistency checking
defined C-7R, C-16R
mentioned C-7R, C-27R, C-47R
overriding with Open NLS C-8R

Control blocks
AFTER GROUP OF 4-43R
BEFORE GROUP OF 4-45R
FIRST PAGE HEADER 4-47R
FORMBUILD 2-62R
in ACE specification file 8-20U, 8-27U, 6-6R
in FORMAT section of ACE report 4-41R
ON BEGINNING 6-12R
ON ENDING 6-12R
ON EVERY ROW 4-49R
ON LAST ROW 4-51R
PAGE HEADER 4-52R
PAGE TRAILER 4-54R

Conventions
typographical Intro-5U, Intro-5R
cperf program, used to customize sperform 6-32R
Create Database screen 2-6U
Create option
DATABASE menu 2-6U, 1-8R
TABLE menu 1-17R
CREATE TABLE menu
building a schema with 1-17U, 2-12U
correcting mistakes 2-12U
definition of 2-11U
CREATE TABLE statement 7-8U, C-4R, C-7R, C-11R, C-19R
CREATE VIEW statement, WITH CHECK OPTION 9-10U
Creating
a menu with the User-menu 5-8R
a script menu 5-24R
Criteria, for database query 4-10U
crtcmap utility C-24R
Current database
changing 2-8U
definition of 2-7U
CURRENT expression, in ACE reports 4-75R
Current list
definition of 4-8U
in PERFORM 2-62R
scanning rows with Next and Previous options 4-9U
with multiple-table forms 5-14U
Current option
changing 1-13U
definition of 1-12U
in multiple-table forms 5-14U
in PERFORM 3-20R
Current statement, definition of 7-4U
cursor movement, in PERFORM 6-17U
customer table
columns in 1-7U
in stores demonstration database A-4R
Customized screen form, FORMBUILD 2-5R
cust_calls table in stores demonstration database A-6R

D

Data
adding to a database with PERFORM 3-12U
checking in PERFORM fields 3-15U, 3-15R
confirming your entry in PERFORM fields 3-15U
definition of 1-4U
displaying on the screen with PERFORM 5-10U, 2-12R
entering for a menu 5-10R
entering in a table using a screen form 3-3U
entering in fields with PERFORM 3-13U
entering into fields with PERFORM 3-9R
entering with the User-menu 5-13R
how it is stored 1-4U
how to query on 4-3U
including from multiple tables in one form 3-5U
retrieving from a database 1-7U
updating in PERFORM 3-35R
verifying 5-10U
Data type
assigning to a column 2-14U, 2-18U
BYTE Intro-14U, I-3R, I-11R
CHAR 2-15U
CHAR, in forms 3-10R
CHAR, in NLS C-4R, C-6R, C-10R, C-19R
choosing with INFORMIX-OnLine I-3R
DATE 2-17U
DATETIME 2-17U
DATETIME, in forms 3-11R
DATE, in forms 3-11R
DATE, in NLS C-10R
DECIMAL 2-15U
DECIMAL, in forms 3-10R
DECIMAL, in NLS C-10R
definition of 3-9R
FLOAT 2-16U
FLOAT, in forms 3-11R
FLOAT, in NLS C-10R
for number columns 2-15U
INTEGER 2-16U
INTEGER, in forms 3-10R
INTERVAL 2-18U
INTERVAL, in forms 3-11R
MONEY 2-17U
MONEY, in forms 3-10R
MONEY, in NLS C-10R
NCHAR C-4R, C-6R, C-10R, C-19R
NVARCHAR C-4R, C-6R, C-10R, C-19R
SERIAL 2-16U
SERIAL, in forms 3-10R
SMALLFLOAT 2-16U
SMALLFLOAT, in forms 3-10R
SMALLFLOAT, in NLS C-10R
SMALLINT 2-16U
SMALLINT, in forms 3-10R
synonyms 3-9R
table of 2-17U
TEXT Intro-14U, I-3R, I-11R
VARCHAR Intro-14U, I-3R
VARCHAR, in NLS C-4R, C-6R, C-10R, C-19R
what it does 2-14U

Database
adding data using PERFORM 3-12U
changing the current 2-8U
changing the structure 9-3U
creating tables in 2-9U
creating through the Main menu 2-5U
current 2-7U
definition of 1-4U
how to retrieve data 1-7U
map of stores demonstration A-8R
naming conventions 2-6U
querying on 1-7U, 4-3U
relational 1-9U
removing 9-5U
removing a table 9-4U
removing an index 9-4U
renaming a table 9-4U
retrieving information from 1-7U
selecting a different current 2-8U
stores demonstration described
Intro-8U, A-1R
stores2 Intro-11R
system files in 2-8U
tables in demonstration 5-4U
user-menu for 1-25U
Database engine features Intro-13U
Database locale C-2R
Database management system
definition of 1-3U
relational 1-9U

DATABASE menu
Create option 1-8R
Drop option 1-8R
Exit option 1-8R
how to use 1-8R
options 2-6U
Select option 1-8R

DATABASE section
in form specifications 6-12U, 2-6R, 2-9R
in forms, WITHOUT NULL INPUT 2-34R
in report specifications 8-8U, 4-14R

Database server, specifying default for connection B-33R

DATE data type
default in form specifications 6-24U
definition of 2-17U
explanation of 2-17U
formatting in form 2-36R
in forms 3-11R
in NLS C-10R
DATE expression, in ACE reports 4-76R
DATE value formatting B-8R, B-18R

DATETIME data type
acceptable values 3-11R
definition of 2-17U
in forms 3-11R

Index 21
DATE( ) function, in ACE reports 4-77R
DAY( ) function, in ACE reports 4-78R
DBANSWARN environment variable B-8R
DBAPICODE environment variable C-2R, C-12R, C-23R, C-24R
DBDATE environment variable B-8R,
    C-3R, C-12R, C-41R, C-42R
DBDELMITER environment variable B-11R
DBEDIT environment variable B-11R
DBEDIT utility
    specifying field delimiter with DBDELIMITER
DBFORM environment variable B-12R,
    C-3R, C-12R, C-25R
DBFORMAT environment variable B-14R, C-3R, C-5R, C-6R, C-12R,
    C-35R, C-38R, C-39R, C-41R, C-42R
DBLANG environment variable B-18R,
    C-3R, C-12R, C-24R, C-41R
dbexport utility
    specifying field delimiter with DBDELIMITER
DBMONEY environment variable B-21R,
    C-3R, C-5R, C-6R, C-12R, C-35R, C-38R, C-40R, C-42R
DBNLS environment variable interaction with COLLCHAR C-17R,
    C-20R
    mentioned C-2R, C-5R, C-12R
    settings C-18R
    syntax C-17R
DBPATH environment variable B-23R
DBPRINT environment variable B-26R
DBREMODECMD environment variable B-27R
dbschema utility
    definition of 1-17U
DBSPACETEMP environment variable B-28R
DBTEMP environment variable B-29R
DBUPTSPACE environment variable B-29R
DECIMAL data type
    definition of 2-15U
    formatting in form 2-36R
    in forms 3-10R
    in NLS C-10R
    scale and precision 2-15U
Decimal separator 4-87R, B-14R, B-21R,
    C-31R, C-33R, C-35R, C-40R, C-44R
Declaring C functions, in ACE 6-4R, 6-5R
Default assumptions for your environment B-5R
DEFAULT attribute
    not using with TEXT or BYTE I-10R
    PERFORM 6-23U
    with WITHOUT NULL INPUT 2-34R
DEFAULT attribute, in PERFORM 2-34R
DEFINE section
    ASCII statement 4-16R
    declaring a C function 6-4R
    PARAM statement 4-18R
    PROMPT FOR statement 4-21R
    VARIABLE statement 4-19R
Delete option, in PERFORM 3-19U
Deleting data, in PERFORM 3-19U
Delimiters
    definition of 3-4U
    FORMBUILD 2-12R
Demonstration database Intro-8U,
    Intro-11R
    copying A-2R
    map of A-8R
    restoring the original database A-3R
    sample forms 2-3R
    stores, tables in A-1R
    tables in 1-5U, 5-4U, A-4R to A-7R
Designing a menu with the User-menu 5-5R
Detail option, in PERFORM 5-11U, 3-21R
Detail table, selecting 5-11U
Directory, database 2-8U
Display field
    FORMBUILD 2-12R
    order 2-20R
Display-only field, FORMBUILD 2-12R
Distributed query across databases
    Intro-14U
    Dollar sign. See Leading currency symbol.
Dominant column
    FORMBUILD 2-27R
    joins 2-27R

22 Index
<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOWNSHIFT attribute in NLS C-11R in PERFORM 6-22U, 2-35R DROP INDEX statement 9-4U Drop option ALTER TABLE menu 2-24U, 9-7U CREATE TABLE menu 2-11U DATABASE menu 2-6U, 1-8R FORM menu 1-11R REPORT menu 1-15R SQL menu 1-13R TABLE menu 1-17R</td>
</tr>
<tr>
<td>Editor</td>
</tr>
<tr>
<td>multiline 3-18U, 3-14R specifying with DBEDIT B-11R specifying with WORDWRAP attribute 3-18U SQL 7-9U system 7-10U ENVIGNORE environment variable B-30R</td>
</tr>
<tr>
<td>Environment configuration file example B-2R where stored B-3R</td>
</tr>
</tbody>
</table>
setting at the command line B-2R
setting in a shell file B-2R
SQLEXEC B-38R
SQLRM B-38R
SQLRMDIR B-39R
TERM B-41R
TERMCAP B-41R
TERMINFO B-42R
UNIX environment variables, listing B-7R
where to set B-2R
X/Open-defined C-12R, C-13R
Errors, correcting in SQL statements 7-12U
EVERY ROW statement, in ACE reports 8-30U, 4-39R
Exit option
ALTER TABLE menu 2-24U
CREATE TABLE menu 2-11U
DATABASE menu 2-6U, 1-8R
FORM menu 1-11R
PERFORM 3-22R
REPORT menu 1-15R
SQL menu 1-13R
TABLE menu 1-17R
USER-MENU menu 1-18R
Explicit NLS environment
defined C-7R, C-20R
disadvantages of C-7R
element C-21R
mentioned C-4R, C-5R, C-19R
Expressions
definition of 6-10R, 6-14R
in ACE 4-10R
in PERFORM control block 6-10R, 6-14R
Extension checking, specifying with
DBANSIWARN B-8R
Field tag
assignments 6-17U
definition of 6-12U
FORMBUILD 2-12R
Field width
explanation of 6-17U
FORMBUILD 2-12R
Fields
character, in PERFORM 3-13U
correcting your data entry in PERFORM 3-15U
confirming your data entry in PERFORM 3-15U
correspondence to columns 1-18U, 3-4U
DATETIME, in PERFORM 3-14U
DATE, in PERFORM 3-14U
definition of 1-18U, 3-3U
delimters, in PERFORM 2-59R
display 2-12R
display order 2-24R
displaying text in color 6-20U
display-only 2-24R
ting in PERFORM 3-17U
ting, in PERFORM 3-17U
entering data, in PERFORM 3-13U
FORMBUILD 2-12R
INTERVAL, in PERFORM 3-15U
join 5-3U
linked to database columns 2-22R
look up 2-22R
making text invisible 6-21U
MONEY, in PERFORM 3-14U
number, in PERFORM 3-13U
querying on in PERFORM 4-11U
too short for search 4-15U
File
command 7-13U
environment configuration B-2R
output, formatting in PERFORM 4-20U
perform.out 4-18U
aving current statements in 7-13U
sening query results with the Output
option 7-16U
shell B-2R
temporary for OnLine B-28R
temporary for SE B-29R
writing reports to 8-48U
F

Facilities
for database management 1-16U
in INFORMIX-SQL 1-16U
report writing 1-20U
schema editor 1-17U
screen form 1-18U
structured query language 1-23U
User-menu 1-25U
File extension
for compiling ACE reports 6-8R
for compiling PERFORM forms 6-14R
.ACE 4-6R, 4-7R, 4-9R, 6-8R
.ARC 4-7R, 4-8R, 4-9R
.C 6-32R
.DAY 2-23U
.DBS 2-8U
.EC 6-32R
.ERR 2-5R, 4-7R, 4-9R
.FRM 6-8U, 2-5R, B-12R
.IDX 2-23U
.IEM B-18R
.OUT 4-18U
.PER 6-8U, 2-5R, 6-14R
.SQL 7-14U
Fill characters, definition of 8-38U
FIRST PAGE HEADER control block, in
ACE reports 4-47R
FLOAT data type
definition of 2-16U
formatting in form 2-36R
in forms 3-11R
in NLS C-10R
FOR statement, in ACE reports 4-56R
FORM menu
Compile option 1-11R
Drop option 1-11R
Exit option 1-11R
Generate option 1-11R
Modify option 1-11R
New option 1-11R
Run option 1-11R, 3-4R
FORM OUTPUT FILE menu, in
PERFORM 3-25R
Form specifications
ATTRIBUTES section 6-13U, 2-7R, 2-20R
changing default values 6-23U
choosing tables for 6-6U
compiling 6-7U, 1-11R, H-7R
controlling case of input 6-22U
correcting errors in 6-8U
customizing 2-4R
DATABASE section 6-12U, 2-6R, 2-9R
default 6-3U, 6-7U, 2-4R, 2-5R
definition of 6-3U
editing 1-11R
field tag assignments 6-17U
field tags in 6-12U
files that make up 6-8U
INSTRUCTIONS section 6-32U, 2-7R, 2-56R
joining fields in 6-28U
JOINING keyword in 6-31U
lookup joins on 6-30U
maximum number of tables in 6-6U
modifying 6-4U, 6-10U
naming 6-5U
sample file 2-82R
SCREEN section 6-12U, 2-7R, 2-10R
sections in 6-10U, 2-6R
specifying acceptable values in 6-25U
TABLES section 6-12U, 2-7R, 2-18R
using today as default date 6-24U
verify joins with 6-30U
verifying input in 6-27U
Format
date data 2-36R, 4-88R, B-8R, B-18R, C-46R
monetary data 4-86R, B-14R, C-31R, C-44R
numeric data 2-36R, 4-86R, B-14R, C-35R, C-44R
FORMAT attribute
in NLS C-11R, C-31R, C-33R, C-37R, C-40R, C-41R, C-43R
in PERFORM 2-36R
FORMAT section of report specification
AFTER GROUP OF control block
8-10U, 4-43R
BEFORE GROUP OF control block
4-45R
custom blocks 4-41R
EVERY ROW statement 4-39R
FIRST PAGE HEADER control block
4-47R
FOR statement 4-56R
IF THEN ELSE statement 4-57R
LET statement 4-59R
NEED statement 4-61R
ON EVERY ROW control block
8-10U, 4-49R
ON LAST ROW control block
8-10U, 4-51R
PERFORM HEADER control block 8-11U, 4-52R
PERFORM TRAILER control block 8-11U, 4-54R
PAUSE statement 4-62R
PRINT FILE statement 4-65R
PRINT statement 4-63R  
sample section 8-10U  
SKIP statement 4-66R  
SKIP TO TOP OF PAGE statement 4-67R  
WHILE statement 4-68R  

Formatting  
ACE 4-86R  
data types in form 2-36R  
report 8-25U  

FORMBUILD transaction form generator  
ATTRIBUTES section 2-7R  
BEGIN keyword 2-80R  
BELL keyword 2-79R  
compiling a customized form 2-4R, 6-14R, H-7R  
control block 2-62R  
creating a customized form 2-4R  
CURRENT keyword 2-75R  
current list 2-62R  
DATABASE section 2-6R  
DATE format 2-36R  
default form-specification file 2-4R, 2-5R  
delimiters 2-12R  
display field 2-12R  
display-only field 2-12R, 2-47R  
dominant column 2-27R  
END keyword 2-80R  
EXISTNOW keyword 2-77R  
field 2-12R  
field tag 2-12R  
field width 2-12R  
FLOAT format 2-36R  
forms, sample 2-3R  
INSTRUCTIONS section 2-7R  
operating system, use in form creation 2-5R, H-7R  
ORDER INFORMATION screen 2-84R  
REVERSE keyword 2-79R  
SCREEN section 2-7R  
sormbld 2-6R  
SMALLFLOAT format 2-36R  
subscripting a CHAR column 2-13R  
TABLES section 2-7R  
TODAY 2-75R  
verify join 2-27R  

Functions  
aggregate, listed 2-74R  
C, calling from ACE 8-49R  
Execute, in PERFORM 3-16U  
Help, in PERFORM 3-16U  
Interrupt, in PERFORM 3-16U  
math, for ACE reports 8-40U  
names used as column names F-2R  
to control PERFORM screens 6-22R  

Forms  
ATTRIBUTES section 6-13U  
choosing which tables to include 6-6U  
compiling 1-11R  
compiling the default specification 6-7U  
compiling, linking, and running 6-32R  
creating and compiling custom 2-4R  
creating with INFORMIX-OnLine I-8R  
creating with the operating system 2-5R  
DATABASE section 6-12U  
default 1-11R  
entering data on a screen form 3-3U  
graphics characters in 2-15R  
how to create 6-4U  
how to name 6-5U  
modifying an existing one 6-10U  
multiple-page 4-17U  
multiple-table 5-3U  
orderform 5-5U, 5-8U  
querying on 4-10U  
runtime 1-11R  
screen form 1-18U  
SCREEN section 6-12U  
specifying acceptable values 6-25U  
specifying remote databases I-12R  
specifying remote tables I-12R  
TABLES section 6-12U  
using default field values 6-23U  
using synonyms for external tables I-13R  
verifying input 6-27U  
with blobs I-9R  
with external tables I-13R  
with VARCHARs I-8R  

Function library  
pf_gettype 6-23R  
pf_getval 6-25R  
pf_msg 6-31R  
pf_nxfield 6-29R  
pf_putval 6-27R  

INDEX
COMPOSITES 2-57R
DELIMITERS 2-59R
DISPLAY 2-71R
EDITADD 2-65R
EDITUPDATE 2-65R
IF-THEN-ELSE 2-80R
in form specification 6-32U
in form specifications 2-7R
LET 2-74R
MASTER OF 2-60R
NEXTFIELD 2-77R
QUERY 2-69R
REMOVE 2-70R
UPDATE 2-68R

INTEGER data type
definition of 2-16U
in forms 3-10R
with display fields 2-9R
Intensity attributes, setting
INFORMIXTERM for B-35R
Interrupt key Intro-12U
INTERVAL data type 2-18U
acceptable values 3-11R
in forms 3-11R
INVISIBLE attribute, in PERFORM 6-21U, 2-40R
items table in stores demonstration database A-5R

Join
composite 2-57R
definition of 1-9U
dominant column 2-27R
fields 6-28U
fields on screen forms 5-7U
field, definition of 5-7U
FORMBUILD 2-26R, 2-27R
in multiple-table form 5-3U
in PERFORM 5-7U
in the stores demonstration database 1-9U
JOINING keyword 6-31U
lookup 5-10U, 6-30U
verify 5-10U, 6-30U, 2-27R

Join columns
definition of 5-7U
in screen forms 2-26R
stores demonstration database columns A-8R
JOINING keyword in a form specification 6-31U

K

Keys
arrow, with SQL editor 7-9U
backspace, in PERFORM 3-16U, 3-17U
basic keys for SQL Intro-12U
change mode, in PERFORM 3-17U
clear screen, in PERFORM 3-17U
cursor positioning, in PERFORM 3-17U, 3-12R
delete a character, in PERFORM 3-17U
delete a character, with SQL editor 7-9U
delete forward, in PERFORM 3-17U
delete rest of line, with SQL editor 7-10U
documenting keys, in PERFORM 3-17U
documenting keys, in SQL 7-9U
documenting escape, with SQL editor 7-9U
documenting execute, in PERFORM 3-17U
documenting fast backspace, in PERFORM 3-17U
documenting fast forward, in PERFORM 3-16U
documenting field editing, in PERFORM 3-17U, 3-13R
documenting for displaying HELP menu 1-16U
documenting for leaving a menu 2-9U
documenting for moving the highlight 1-13U
documenting forward, in PERFORM 3-16U, 3-17U
documenting help, in PERFORM 3-16U
documenting insert mode, with SQL editor 7-9U
documenting interrupt, in PERFORM 3-16U
documenting next field, in PERFORM 3-16U
documenting repeat data, in PERFORM 3-17U
documenting special function, in PERFORM 3-12R

Keywords
columns used as column names F-3R
typographic convention Intro-5U, Intro-5R

Keywords in ACE
COLUMN 8-28U
DELIMITER 8-24U
END 8-18U, 8-20U
SPACES 8-28U
USING 8-37U

Keywords in PERFORM
ABORT 2-73R
AFTER 2-64R
AFTER ADD OF 2-67R
AFTER DISPLAY OF 2-71R
AFTER QUERY OF 2-69R
AFTER UPDATE OF 2-68R
ALLOWING INPUT 2-24R
ATTRIBUTES 6-13U
AUTONEXT 2-29R
BEFORE 2-63R
BELL 2-79R
BY 2-10R
COLOR 2-30R
COMMENTS 2-33R, 2-79R
COMPOSITES 2-57R
COMPRESS 2-53R
DATABASE 6-12U, 2-9R
DEFAULT 2-34R
DELIMITERS 2-59R
DISPLAYONLY 2-24R
DOWNSHIFT 2-35R
EDITADD 2-65R
EDITUPDATE 2-65R
END 6-11U, 2-18R
EXITNOW 2-77R
FORMAT 2-36R
IF-THEN-ELSE 2-80R
INCLUDE 2-38R
INVISIBLE 2-40R
JOINING 6-31U, 2-41R
LET 2-74R
LOOKUP 6-31U, 2-41R
MASTER OF 2-60R
NEXTFIELD 2-77R
NOT NULL 2-24R
NOUPDATE 2-44R
OF 2-63R, 2-64R
PICTURE 2-45R
QUERYCLEAR 2-47R
REMOVE OF 2-70R
REQUIRED 2-48R
REVERSE 2-49R, 2-79R
RIGHT 2-50R
SCREEN 6-12U, 2-10R
SIZE 2-10R
TABLES 2-18R
TO 6-25U
TYPE 2-24R
UPSHIFT 2-51R
VERIFY 2-52R
WHERE 6-21U, 2-30R
WITHOUT NULL INPUT 6-23U, 2-9R
WORDWRAP 2-53R
ZEROFILL 2-55R

Korn shell
how to set environment variables
B-4R
.profile file B-2R

Language and formatting environment variable
LANG environment variable
effect of where set on precedence
C-15R
interaction with DBLANG B-20R
lack of standardization C-26R
mentioned C-2R, C-12R, C-13R,
C-16R, C-21R, C-24R, C-31R,
C-34R, C-40R, C-42R, C-43R,
C-49R
setting default for LC_ variables C-2R,
C-14R, C-25R
syntax C-26R

Language supplement C-25R, C-27R,
C-48R

LC_ variable
defined C-12R
effect of where set on precedence
C-15R
lack of standardization C-2R, C-13R,
C-16R, C-25R, C-40R

LC_COLLATE environment variable
database storage of value C-7R, C-16R
defined C-27R
mentioned C-2R, C-4R, C-5R, C-6R,
C-12R, C-14R
syntax C-28R

LC_CTYPE environment variable
database storage of value C-7R,
C-16R, C-29R
defined C-29R
interaction with DBAPICODE C-24R
mentioned C-2R, C-5R, C-12R
syntax C-30R

Index  29
LC_MONETARY environment variable defined C-31R
  mentioned C-2R, C-5R, C-12R, C-14R, C-35R, C-41R, C-42R
  syntax C-5R, C-32R
LC_NUMERIC environment variable defined C-35R
  mentioned C-2R, C-5R, C-12R, C-42R
  syntax C-36R
Leading currency symbol 8-40U, 4-87R, B-14R, B-21R, C-33R, C-40R, C-45R
LEFT MARGIN statement, in ACE reports 8-26U, 4-24R
LET statement
  conversion of MONEY to CHAR C-33R, C-37R
  in ACE reports 4-59R
  in NLS C-11R, C-46R
Levels of menus in a User-menu structure 5-5R
Library functions
  in PERFORM 6-22R
    pf_gettype() 6-23R
    pf_getval() 6-25R
    pf_msg() 6-31R
    pf_nxfield() 6-29R
    pf_putval() 6-27R
LINEFEED characters between statements Intro-7R
LINENO expression, in ACE reports 4-79R
LOAD statement
  in NLS C-8R, C-11R, C-16R, C-35R, C-40R, C-42R
  specifying field delimiter with DBDELIMITER B-11R
  with INFORMIX-OnLine I-7R
with VARCHARs and blobs I-7R
Locale database locale C-2R
  defined C-2R
  mentioned C-16R, C-25R
  user locale C-2R
Locale consistency checking. See Consistency checking
Locale file C-27R
Locale variable C-12R, C-13R
Locale-sorted columns C-6R, C-8R
LOCK TABLE statement, in PERFORM 3-19R
LOOKUP attribute, in PERFORM 2-41R
Lookup fields definition of 5-11U, 6-30U
  in form specifications 6-30U
  specifying with the LOOKUP attribute 2-22R
Lookup joins definition of 6-30U
  in multiple-table forms 5-10U
  in PERFORM 5-11U
  on a screen form 5-10U
Loops in syntax diagrams Intro-7R
Lowest value operator, in PERFORM 4-14U, 3-31R

M

Main menu
  accessing 1-12U
  creating a database 2-5U
  creating a table 2-9U
Database option 1-14U, 2-6U
Exit option 1-14U
Form option 1-12U, 1-13U, 3-6U, 6-5U, 1-11R, 2-4R
how to exit 1-10R
map of INFORMIX-SQL menu hierarchy 1-5R
Query-language option 1-14U, 7-5U, 1-13R
Report option 1-13U, 1-15R
  shown 1-3R
Table option 1-14U, 1-17R
User-menu option 1-14U, 1-18R
manufact table in stores demonstration database A-7R
Mapping file C-24R, C-25R
Master option, in PERFORM 5-13U, 3-23R
Master-detail relationship
  Detail menu option, in PERFORM 3-21R
  introduction to 5-11U
  Master menu option, in PERFORM 3-23R
  specifying 2-60R
Match, exact in PERFORM query 4-4U
Math functions, in ACE reports 8-40U
MDY( ) function, in ACE reports 4-80R
Menu form file C-25R
menuform screen form
accessing the sysemnuiems table 5-9R
accessing the sysemmenus table 5-9R
definition of 5-9R
entering data in fields 5-15R
Menu Name field 5-16R
Menu Title field 5-17R
Selection Action field 5-22R
Selection Number field 5-18R
Selection Text field 5-21R
Selection Type field 5-19R

Menus
ADD TYPE 2-14U
ALTER TABLE 2-24U
calling with the User-menu facility 1-25U
CHANGE ANYWAY 9-7U
CREATE TABLE 2-11U
creating a script 5-24R
creating custom with User-menu facility 1-25U, 5-3R
creating your own 5-8R
current option 1-12U
custom (user-menus) 1-25U
DATABASE 2-6U
designing 5-5R
entering data 5-10R
FORM 1-11R
HELP 1-16U
how to access 5-4R
in a national language B-12R
INFO 7-17U
INFORMIX-SQL Main menu 1-12U, 2-5U, 1-3R
levels in the user-menu structure 5-5R
map of hierarchy 1-5R
maximum number of options per menu 5-5R
modifying ones you created 5-14R
naming ones you create 5-16R
PERFORM 3-10U
REPORT 8-5U, 1-15R
selecting options 1-13U
SQL 7-5U, 1-13R
SYNTAX 7-6U
TABLE 2-10U, 1-17R

TABLE, with INFORMIX-OnLine I-2R
USER-MENU 1-18R
Message file B-18R, C-24R, C-46R
Message line of a menu 1-12U
Meta-environment variable C-12R
Mode
insert, in PERFORM 3-17U, 3-13R
real, invoking INFORMIX-SQL 1-11U
typeover, in PERFORM 3-17U, 3-13R

Modify option
ALTER TABLE menu 2-24U, 9-7U
CREATE TABLE menu 2-11U
FORM menu 1-11R
REPORT menu 1-15R
SQL menu 1-13R
USER-MENU menu 1-18R
MODIFY REPORT menu
Compile option 8-17U
Discard-and-exit option 8-17U
Save-and-exit option 8-17U

MODIFY REPORT screen 8-16U
Modifying data, with PERFORM 3-18U
MONEY data type
definition of 2-17U
determining column length 2-17U
in forms 3-10R
in NLS C-10R
Monospace typeface Intro-5U, Intro-5R
MONTH( ) function, in ACE reports 4-81R

Moving the highlight 1-13U
Multiline editor
how to invoke 3-14R
in PERFORM 3-18U

Multiple-table form
active table for 5-4U
current option 5-14U
definition of 5-3U
join fields 5-7U
lookup join 5-10U
master-detail relationship 5-11U
orderform 5-8U
selecting the detail table 5-12U
selecting the master table 5-13U
verify join 5-10U
Index

N
Naming conventions
   for a database 2-6U
   for a table 2-11U
   for columns 2-13U
   for report specifications 8-12U
User-menu names 5-16R

Native Language Support
   classification of variables C-12R
   database access restrictions C-7R
   defined C-2R
   environment variables listed B-7R, B-39R
   features supported in Version 6.0 C-9R
   multiple locales C-47R
NCHAR data type C-4R, C-6R, C-10R, C-19R
NEED statement, in ACE reports 4-61R
Network environment variable
   SQLRM B-38R
   SQLRMDIR B-39R
New option
   FORM menu 1-11R
   REPORT menu 1-15R
   SQL menu 1-13R
Next option, in PERFORM 4-9U, 3-24R
NLS database
   defined C-5R
   mentioned C-17R
   performance penalty C-19R
NLS environments
   defined C-5R
   distinctions between C-6R
   summary of C-9R
NLS. See Native Language Support.
Non-NLS database
   defined C-6R
   mentioned C-17R
   US English NLS sorting different from C-23R
   when preferable to NLS database C-7R
Non-NLS environment
   defined C-5R, C-20R
NOUPDATE attribute, in PERFORM 2-44R
NULL values
   in a column 2-21U
   in display fields 2-34R
Numbers, appearance in reports 8-37U
NVARCHAR data type C-4R, C-6R, C-10R, C-19R

O
ON BEGINNING control block, in PERFORM 6-12R
ON ENDING control block, in PERFORM 6-12R
ON EVERY ROW control block, in ACE reports 8-10U, 8-30U, 4-49R
ON LAST ROW control block, in ACE reports 8-11U, 4-51R
ONCONFIG environment variable B-36R
onconfig file, specifying with ONCONFIG B-36R
OnLine database engine Intro-14U
   programming issues using Intro-14U
On-line documentation notes Intro-10R
OnLine environment I-1R
On-line error messages Intro-6U, Intro-10R
Open NLS environment
   defined C-8R, C-20R
   mentioned C-3R
Operating system
   using to create a form 2-5R
Operators
   arithmetic, combining with math functions 8-45U
   arithmetic, in reports 8-36U
   range, in query 4-10U
   relational, in query 4-10U
   value, in query 4-10U
Option, how to select 1-13U
Order
   of display fields 2-20R
   of tables 2-20R
   orders table in stores demonstration database A-4R
OUTPUT FORMAT screen, in PERFORM 4-20U, 3-26R
Index

Output option
PERFORM 4-17U, 3-25R
SQL menu 7-16U, 1-13R

OUTPUT section
BOTTOM MARGIN statement 4-27R
in ACE reports 8-25U, 8-47U
LEFT MARGIN statement 4-24R
PAGE LENGTH statement 4-28R
REPORT TO statement 4-23R
RIGHT MARGIN statement 4-25R
TOP MARGIN statement 4-26R
TOP OF PAGE statement 4-29R

Output, displaying for reports 8-47U
Output, for a report 8-25U

PAGE HEADER control block, in ACE reports 8-10U, 8-28U, 4-52R
Page layout 2-12R
PAGE LENGTH statement, in ACE reports 8-26U, 4-28R
PAGE TRAILER control block, in ACE reports 8-11U, 4-54R
PAGENO expression, in ACE reports 4-82R
PARAM statement, in ACE reports 4-18R

Passing values to a C function 6-18R
PATH environment variable B-40R
Pathname
including in SQLEXEC B-38R
specifying with DBPATH B-23R
specifying with PATH B-40R
PAUSE statement, in ACE reports 4-62R
PERFORM
accessing from the command line H-9R
accessing from the Main menu 3-4R
accessing with the menuform form 5-9R
active table in 5-4U
Add option 3-19R
altering a menu structure 5-3R
CALL keyword 6-10R
calling C functions 6-9R
checking data 3-15R
checking your data entry 3-15U
COLOR attribute 6-20U
COMMENTS attribute 6-21U

comparison with SQL 7-7U
confirming your entry in a field 3-15U
creating a menu 5-8R
creating a menu structure 5-3R
current list 4-8U
Current option 5-14U, 3-20R
data types 3-9R
DEFAULT attribute 6-23U
Detail option 5-11U, 3-21R
displaying blobs L-5R
DOWNSHIFT attribute 6-22U
entering menu data 5-10R
entering search criteria 4-10U
entering search criteria for query 4-4U
equivalent form 6-37R
Execute function 3-16U
Exit option 3-22R
expression 6-10R, 6-14R
field editing feature 3-17U
field editing keys 3-13R
FORM OUTPUT FILE menu 3-25R
Help function 3-16U
how to access 3-6U
how to compile 6-14R
how to exit 3-19U
how to run 3-3R
INCLUDE attribute 6-25U
information lines section 3-8U
INSTRUCTIONS section 6-9R
Interrupt function 3-16U
INVISIBLE attribute 6-21U
invoking the multiline editor 3-14R
library functions 6-22R
locating all rows in a table 4-9U
Master option 5-11U, 3-23R
menu options 3-10U
Next option 3-24R
ON BEGINNING control block 6-12R
ON ENDING control block 6-12R
OUTPUT FORMAT menu 3-26R
Output option 4-17U, 3-25R
positioning the cursor 3-16U, 3-12R
Previous option 3-28R
query operators and short fields 4-15U
Query option 4-4U, 3-29R
querying on a range of values 4-13U
querying on alternative values 4-15U
querying on CHAR fields 4-12U
querying on DATE fields 4-11U
querying on DATETIME fields 4-11U
querying on detail table 5-11U
querying on FLOAT and
SMALLFLOAT fields 4-16U
querying on highest value 4-14U
querying on INTERVAL fields 4-11U
querying on lowest value 4-14U
querying on MONEY fields 4-12U
querying on number fields 4-11U
Remove option 3-32R
removing a row 3-19U
running operating system commands 3-9R
screen for orderform form 5-5U
screen form section 3-9U
Screen option 3-33R
Screen-format option 4-20U
searching for a group of rows 4-7U
special functions 3-12R
storing a row 3-18U
Table option 3-34R
detail of sections of screen 3-8U
Update option 3-35R
updating a row 3-18U
UPSHIFT attribute 6-22U
using sperform 6-32R
using the menuform screen form 5-8R
VIEW attribute 6-27U
View option 3-36R
PERFORM functions
pf__gettype( ) 6-23R
pf__getval( ) 6-25R
pf__msg( ) 6-31R
pf__nxfield( ) 6-29R
pf__putval( ) 6-27R
PERFORM screen
divided into three sections 3-5R
entering data 3-9R
information lines 3-6R
menu options listed 3-7R
screen form 3-8R
status lines 3-9R
PERFORM screen transaction processor
accessing from Main menu 3-4R
Add option 3-12U, 3-7R
adding data with 3-12U, 3-9R
CHAR fields in 3-13U, 4-12U
comments line 2-33R
current list 4-8U, 2-62R
Current option 3-7R
cursor positioning keys 3-16U, 3-12R
data types 3-9R
DATE fields in 3-14U, 4-11U
DATETIME fields in 3-14U, 4-11U,
4-14U
Delete option 3-19U
deleting data with 3-19U
Detail option 5-11U, 3-7R
displaying comments on the screen 6-21U
editing keys in 3-17U
equal to operator 4-11U
Exit option 3-7R
exiting from 3-19U, 3-7R
explanation of queries in 4-10U
field editing 3-13R
highest value operator 4-14U, 3-31R
how to access 3-6U
how to call up 3-3R
Information lines in 3-8U, 3-5R, 3-7R
insert mode 3-17U, 3-13R
INTERVAL fields in 3-15U, 4-11U,
4-14U
introduction to 3-5U
LOCK statement with 3-19R
lowest value operator 4-14U, 3-31R
Master option 5-13U, 3-7R
menu options 3-10U, 3-7R
modifying data with 3-18U
MONEY fields in 3-14U, 4-12U
multiple-page forms 4-17U
Next option 4-9U, 3-7R
number fields in 3-13U, 4-11U
operating system commands within 3-9R
Output option 3-7R
pictures in 3-15U
Previous option 4-9U, 3-7R
Query option 3-7R
range operator 4-13U
relational operators in 4-11U
Remove option 3-7R
sample queries 4-16U
screen forms with 3-8R
Screen option 4-17U, 3-7R
search criteria in 4-4U
SERIAL fields in 3-13U
Status lines in 3-10U, 3-5R, 3-9R
Table option 5-4U, 3-7R
typeover mode 3-17U, 3-13R
Update option 3-18U, 3-7R
View option 3-7R
what it is 3-5U
wildcard characters in 4-12U, 3-30R
Period symbol. See Decimal separator.
Peripheral device C-23R
pf_gettype() function, in PERFORM 6-23R
pf_getval() function, in PERFORM 6-25R
pf_msg() function, in PERFORM 6-31R
pf_nxfield() function, in PERFORM 6-29R
pf_putval() function, in PERFORM 6-27R
PICTURE attribute 3-15U
Precedence, rules for environment variables B-6R
Pre-Version 6.0 tool C-5R, C-18R
Previous option, in PERFORM 4-9U, 3-28R
PRINT FILE statement, in ACE reports 4-65R
PRINT statement, in ACE reports 4-63R
PRINT USING statement 8-37U
Printing query results with SQL 7-16U
Printing, specifying print program with DBPRINT B-26R
Privilege, displaying with the Info option 7-17U
Program calling with the User-menu facility 1-25U
demonstration database samples 6-33R
example, p_ext1.per 6-37R
example, stamp.c 6-39R
PROGRAM attribute I-11R
PROMPT FOR statement, in ACE reports 4-21R
PSORT_DBTEMP environment variable B-36R
Punctuation symbols Intro-6R

Query
by example 4-4U
definition of 1-7U, 4-3U
eamples with PERFORM 4-16U
finding all rows in a table 4-9U
interactive 7-4U
locating a single row 4-4U
moving between screens 4-17U
on detail table 5-11U
range operators 4-10U
relational operators 4-10U
saving results in PERFORM 4-17U
saving results with SQL 7-16U
searching for a group of rows 4-7U
searching for a range of values 4-13U
searching for alternative values 4-15U
searching for an exact match 4-4U
searching for columns 1-8U
searching for columns within rows 1-8U
searching for highest value 4-14U
searching for lowest value 4-14U
searching for rows 1-8U
searching in FLOAT and SMALLFLOAT fields 4-16U
sending results to a file 7-16U
sending results to a printer 7-16U
specifying search criteria for 4-10U
structured 1-23U
syntax for PERFORM 3-30R
using SQL 1-23U
using wildcard characters 4-10U, 4-12U
value operators 4-10U
Query language, definition of 1-23U, 7-4U
Query option, in PERFORM 4-4U, 3-29R
QUERY screen, in PERFORM 4-5U
Querying the database distributed across databases Intro-14U
VARCHAR, TEXT and BYTE data I-5R
Query-language option, Main menu 1-13R
Quotation (" ) marks single and double Intro-6R
Quotes, enclosure of monetary values by C-35R, C-47R
Quotes, enclosure of numeric values by C-47R

Index 35
R

Range operators
in PERFORM 4-13U, 3-30R
with DATETIME fields, in PERFORM 4-14U
with INTERVAL fields, in PERFORM 4-14U

READ section
of report specification 8-24U
READ statement 4-33R

READ statement, in a report
specification 8-24U, 4-33R

Real mode
invoking INFORMIX-SQL 1-11U

Relational DBMS
advantages of 1-10U
definition of 1-9U

Relational operators
CHAR fields, in PERFORM 4-12U
DATE fields, in PERFORM 4-11U
DATETIME fields, in PERFORM 4-11U
in PERFORM 4-11U
INTERVAL fields, in PERFORM 4-11U
MONEY fields, in PERFORM 4-12U
numeric fields, in PERFORM 4-11U

Relay Module
SQLRM environment variable B-38R
SQLRMDIR environment variable B-39R

Remote databases and tables in forms
I-12R

Remove option, in PERFORM 3-19U,
3-32R

RENAME TABLE statement 9-4U

REPORT menu
Compile option 8-5U
Drop option 8-5U, 1-15R
Exit option 1-15R
Generate option 8-5U, 8-12U, 1-15R
how to use 1-15R
Modify option 8-16U, 1-15R
New option 8-5U, 1-15R
Run option 8-14U, 1-15R
summary of options 8-5U

Report specification
compiling 8-18U
correcting 8-18U

DATABASE section 8-8U
default, compiling 8-13U
default, creating 8-12U
definition of 8-4U
effects in 8-18U
FORMAT section 8-10U
group functions 8-45U
including comments in 8-21U
modifying 8-16U
OUTPUT section 8-25U
READ section 8-24U
saving 8-17U
SELECT section 8-8U
sequence of sections 8-20U
summary of sections 8-20U
what it contains 8-20U

REPORT TO statement, in ACE reports
8-48U, 4-23R

Report writer, ACE 8-3U

Reports
arithmetic operators in 8-36U
basic formatting 8-25U
calculations in 8-36U
compiling 1-15R, 4-5R
compiling, linking, and running 6-32R
counting rows in 8-43U
creating on INFORMIX-OnLine I-8R
definition of 8-3U
displaying output 8-47U
dollar sign in 8-40U
editing 1-15R
formatting 8-22U
group control blocks 8-34U
grouping data 8-33U
headers 8-27U
how numbers appear in 8-37U
modifying 8-16U
naming conventions 8-12U
numbering pages automatically 8-30U
organizing the data 8-32U
printing a heading 8-28U
printing column headings 8-29U
printing rows of data 8-30U
retrieving data from an ASCII file 8-24U
running 8-14U, 1-15R, 4-5R
sample report 8-6U
sample SELECT statements 8-22U
selecting data for 8-22U
steps in creating with ACE 8-4U trailers 8-27U
using control blocks to customize 4-41R
using fill characters 8-38U
writing 1-20U
writing to a file 8-48U
writing with ACE 8-3U
Reserved words F-1R
Return value macros, in C program structure 6-21R
RIGHT MARGIN statement, in ACE reports 4-25R
Row
adding rows to a table 1-6U
changing the contents with
PERFORM 3-18U
counting in ACE report 8-43U
definition of 1-6U
in the stores demonstration database 1-6U
locating all in a table 4-9U
locating with a query, in PERFORM 4-4U
removing with PERFORM 3-19U
searching for 1-8U
searching for a group, in PERFORM 4-7U
storing with PERFORM 3-18U
Run option
FORM menu 1-11R
REPORT menu 1-15R
SQL menu 7-12U, 1-13R
USER-MENU menu 1-18R
RUN REPORT screen 8-14U
Running SQL statements 7-12U
Run-time program
setting DBANSIWARN B-8R

Sample form specifications
customer A-29R
orderform A-30R
sample A-32R
Sample report specifications, ACE A-34R
Save option, SQL menu 1-13R
Saving query results with SQL 7-16U
Schema editor
ADD NUMBER menu 2-15U
ADD STARTING NUMBER screen 2-18U
ADD TYPE menu 2-18U
adding a column to a table 9-6U
ALTER TABLE menu 9-6U
assigning data types 2-14U
CHANGE ANYWAY menu 9-7U
changing a table 2-24U
correcting mistakes 2-12U
creating tables 1-17U
defining a column 2-12U
definition of 1-12U
dropping a column from a table 9-7U
exiting 2-22U
modifying a column in a table 9-7U
naming a table 2-11U
Schema, for a table 1-17U, 2-12U
Screen
CHOOSE 7-15U
INFO FOR TABLE 7-17U
menu 1-3R
menuform fields 5-15R
NEW 7-9U
PERFORM 3-8U, 5-5U, 3-5R
RUN FORM 3-7U, 1-4R
sample PERFORM, customer information 2-84R
sample PERFORM, order information 2-84R
SAVE 7-14U
text entry 1-4R
USE-EDITOR 7-11U
Screen forms
active table on 5-4U
choosing tables when creating 6-6U
choosing the one you want 3-7U
creating multiple 3-6U
default 6-9U
definition of 1-18U, 3-3U
definition of multiple-table 5-3U
entering data in a table 3-3U
features 1-18U

sacego
and C function calls in ACE 8-49U
customizing for ACE 6-32R
syntax H-11R
using the cace program 6-32R
saceprep, compiling report specifications 6-8R, H-10R

Index 37
how to use 3-6U
in the demonstration database 3-4U
including data from multiple tables 3-5U
join fields on 5-7U
master-detail relationships on 5-3U
maximum number of tables on 3-5U, 6-6U
modifying 6-4U
naming 6-5U
verify joins on 5-10U

Screen option
ALTER TABLE menu 2-24U
CREATE TABLE menu 2-11U
in PERFORM 4-17U, 3-33R

SCREEN section
FORMBUILD 2-7R
graphics characters in 2-7R

Script menu, how to create 5-24R
Search criteria
definition of 4-4U
in PERFORM 4-4U

Select option, DATABASE menu 2-6U, 1-8R
SELECT section, of report specification 8-8U
SELECT statement
distributed across databases Intro-14U
in a report specification 8-8U
in ACE reports 4-30R
SERIAL data type
assigning starting number 2-18U
definition of 2-16U
in forms 3-10R
Setting environment variables B-4R
sformbld
and the .PER extension 6-14R
definition of 2-6R
syntax H-7R
Shaded syntax diagram elements Intro-7R
Shared memory parameters, specifying file with ONCONFIG B-36R
Shell
setting environment variables in a file B-2R
specifying with DBREMOTE CMD B-27R
SKIP statement, in ACE reports 4-66R

SKIP TO TOP OF PAGE statement, in ACE reports 4-67R
SMALLFLOAT data type
definition of 2-16U
formatting in form 2-36R
in forms 3-10R
in NLS C-10R
SMALLINT data type
definition of 2-16U
in forms 3-10R

Sorting, PSORT_DBTEMP environment variable B-36R

SPACES expression, in ACE reports 4-83R
sperform
creating a new form 2-6R
customizing for PERFORM 6-32R
syntax H-9R
using the cperf program 6-32R

SQL
command file 7-14U
commenting 7-16U
correlation to PERFORM 7-7U
correcting errors in statements 7-12U
creating a table 7-8U
current statements 7-4U
definition of 7-4U
dropping an index 9-4U
editing keys 7-9U
editor 7-9U, 7-10U, 7-12U
text editor 7-9U
finding statements 7-9U
errors, correcting 7-12U
formatting statements 7-15U
how to access from the Main menu 7-5U
how to use 7-3U
introduction to 2-4U
learning how to use 7-6U
long statements 7-10U
naming current statements 7-14U
options on SQL menu 7-5U
renaming a table 9-4U
running statements 7-12U
saving statements 7-14U
selecting statements with semicolons 7-16U
specifying a system editor 7-10U
statements, how to use 1-23U, 7-4U
SYNTAX menu 7-6U
system editors 7-10U
Use-editor option 7-10U
Index 39

SQL Communications Area (SQLCA)
- effect of setting DBANSIWARN B-8R

SQL menu
- Choose option 7-5U, 7-14U, 1-13R
- Drop option 7-5U, 1-13R
- Exit option 7-5U, 1-13R
- how to use 1-13R, 1-14R
- Info option 7-5U, 7-17U, 1-13R
- introduction to 7-5U
- Modify option 7-5U, 7-13U, 1-13R
- New option 7-5U, 7-9U, 1-13R
- options 7-5U
- Output option 7-5U, 7-16U, 1-13R
- Run option 7-5U, 7-12U, 1-13R
- Save option 7-5U, 7-14U, 1-13R
- Use-editor option 7-5U, 7-10U, 1-13R

SQL statements
- correcting errors in 7-12U
- current 7-4U
- examples 7-16U
- formatting 7-15U
- how to enter 7-9U
- how to run 7-12U
- how to use 7-4U
- indicating comments 7-16U
- modifying 1-13R
- running 1-13R
- saving 1-13R
- selecting 1-13R
- separating with semicolons 7-16U

SQLEXEC environment variable B-38R
SQLRM environment variable B-38R
SQLRMDIR environment variable B-39R

state table in stores demonstration
database A-7R

Statement segments
- notational conventions Intro-6R

Statements
- ACE 4-12R
- ASCII 4-16R
- BOTTOM MARGIN 4-27R
- correcting errors 7-12U
- CREATE TABLE 7-8U
- current 7-4U
- definition of 7-4U
- editing in SQL 7-9U
- EVERY ROW 4-39R
- FOR 4-56R
- how to format 7-15U
- how to run 7-12U

IF THEN ELSE 4-57R
in ACE report specification 8-20U
LEFT MARGIN 4-24R
LET 4-59R
link 2-20R
NEED 4-61R
PAGE LENGTH 4-28R
PARAM 4-18R
PAUSE 4-62R
PRINT 4-63R
PRINT FILE 4-65R
PROMPT FOR 4-21R
READ 4-33R
REPORT TO 4-23R
RIGHT MARGIN 4-25R
saving in a command file 7-14U
SKIP 4-66R
SKIP TO TOP OF PAGE 4-67R
SQL definition of 1-23U
TOP MARGIN 4-26R
TOP OF PAGE 4-29R
VARIABLE 4-19R
WHILE 4-68R

Status
- displaying with INFORMIX-OnLine I-4R
- displaying with the Info option 7-17U lines, in PERFORM 3-10U

Status lines
- about 2-33R
- in PERFORM 3-10U

stores demonstration database
catalog table A-6R
copying A-2R
creating A-3R
customer table 9-6U, A-4R
customer table columns A-4R
cust_calls table A-6R
data values A-14R
described Intro-8U, Intro-11R, A-1R
elements table A-5R
items table columns A-5R
join columns 1-9U, A-8R, A-8R to A-14R
manufact table A-7R
manufact table columns A-7R
map of A-8R
orders table A-4R
orders table columns A-4R
report specifications 8-6U, 8-7U
restoring the original A-3R
state table A-7R
state table columns A-7R
stock table A-5R
structure of tables A-3R, A-8R
summary of tables 1-5U
user-menu design outline 5-7R
stty -istrip command C-31R
Subdiagram
  box symbol Intro-7R
  graphic notation Intro-7R
Syntax
  for ATTRIBUTES section in
    PERFORM 2-28R
  for DISPLAY instructions in
    PERFORM 2-72R
  for Query option in PERFORM 3-30R
sysmenuitems table
  accessing with PERFORM 5-9R
  definition of 5-9R
sysmenus table
  accessing with PERFORM 5-9R
  definition of 5-9R
  sample data in 5-11R
System catalogs
  syscolauth G-4R
  syscolumns G-2R
  sysconstraints G-5R
  sysdepend G-4R
  sysindexes G-3R
  syssynonyms G-4R
  syssyntable G-6R
  systabauth G-3R
  systables G-2R
  sysusers G-5R
  syssviews G-5R
System catalog, definition of 2-8U
System editor 7-11U
System files
  extensions for 2-23U
  where they are stored 2-23U

Table
  active, in PERFORM 3-5U, 5-4U
  alias I-13R
  changing the structure 2-24U, 9-5U
  connecting through column joins 1-9U
  creating a master-detail relationship 2-60R
  creating and altering with
    INFORMIX-OnLine I-6R
  creating through menus 1-17U, 2-11U
  defining columns for 2-12U
  definition of 1-5U
  detail, selecting 5-11U
  displaying information in 7-17U
  dropping a column 9-7U
  finding all rows with Query option 4-9U
  how to build with the schema editor 2-12U
  how to create with SQL 7-8U
  in system catalog 2-8U
  in the stores demonstration database 1-5U
  joining columns 1-9U
  joining tables A-8R
  listing those in a database 7-17U
  master, returning to 5-13U
  modifying a column 9-7U
  multiple, in a form 5-3U
  naming 2-11U
  naming conventions 2-11U
  order in forms 2-21R
  removing a row 3-19U
  removing, using the TABLE menu 9-4U
  renaming 9-4U
  schema 1-17U, 2-12U
  selecting data for a report 8-23U
  storing a row 3-18U
  structure in stores demonstration database A-3R
  sysmenuitems 5-9R
  sysmenus 5-9R
  system files that make up a table 2-23U
  using the schema editor 9-6U

TABLE menu
  Alter option 1-17R
  Create option 1-17R
  Drop option 1-17R
  Exit option 1-17R
  how to use 1-17R
  Info option 1-17R
  schema editor 9-6U
  summary of options 2-10U
  with INFORMIX-OnLine I-2R
Table option, in PERFORM 5-4U, 3-34R
Table schema, how to reproduce 1-17U
TABLES section, in form specifications
6-12U, 2-7R
Temporary
files, specifying directory with
DBTEMP B-29R
tables, specifying dbspace with
DBSPACETEMP B-28R
TERM environment variable B-41R
TERMCAP environment variable B-41R
termcap file
and TERM environment variable
B-41R
color and intensity D-8R
description of D-2R
graphics characters in screen form
D-5R
graphics characters in screen forms
2-16R
selecting with INFORMIXTERM
B-35R
Terminal characteristics
termcap file D-2R
terminfo directory D-18R
Terminal handling
and TERM environment variable
B-41R
and TERM environment variable
B-41R
and TERM environment variable
variable B-42R
Terminator, in syntax diagrams Intro-7R
terminfo directory
and TERM environment variable
variable B-42R
description of D-18R
graphics characters in screen form
D-21R
graphics characters in screen forms
2-16R
selecting with INFORMIXTERM
B-35R
TERMINFO environment variable B-42R
TERMINFO environment variable B-42R
TEXT data type Intro-14U
defining columns I-3R
printing in reports I-8R
specifying in forms I-9R
Text editor, specifying with DBEDIT
B-11R
Text entry screen 1-4R
Text, how to enter 1-4R
Thousands separator 4-87R, B-14R,
B-23R, C-31R, C-33R, C-35R,
C-40R, C-44R
TIME expression, in ACE reports 4-84R
TODAY expression, in ACE reports
4-85R
TOP MARGIN statement, in ACE reports
8-26U, 4-26R
TOP OF PAGE statement, in ACE reports
4-29R
Trailing currency symbol 4-87R, B-14R,
B-21R, C-33R, C-40R, C-45R
Transaction log versus an audit trail 9-9U
TRANSACTION menu, in an
ANSI-compliant database 9-9U
Transactions, definition of 9-8U
Typeover mode, in PERFORM 3-17U,
3-13R
Typographical conventions Intro-5U,
Intro-5R
UNIX
default print capability in BSD B-5R,
B-26R
default print capability in System V
B-5R, B-26R
environment variable setting in BSD
and System V B-4R
environment variables listed B-7R
terminfo library support in System V
B-35R
viewing environment settings in BSD
B-4R
viewing environment settings in
System V B-4R
UNLOAD statement
in NLS C-8R, C-11R, C-16R, C-35R,
C-40R, C-42R
specifying field delimiter with
DBDELIMITER B-11R
with INFORMIX-OnLine I-6R
with VARCHARs and blobs I-6R
Update option, in PERFORM 3-18U, 3-35R

UPDATE STATISTICS statement and DBUPSPACE environment variable B-29R

Uppercase characters typographic convention Intro-5U, Intro-5R

UPSHIFT attribute in NLS C-11R in PERFORM 6-22U

Use-editor option, SQL menu 7-10U, 1-13R

User locale C-2R

User locale. See Locale, user locale.

User-menu
definition of 1-25U
designing a menu 5-5R
entering menu data 5-10R
guidelines for using 1-18R
how to access 5-4R
layout specifications 5-5R
levels of menus 5-5R
maximum options per menu 5-5R
menu data, entering 5-10R
modifying a menu you created 5-14R
sample design outline 5-7R
steps for entering your own data 5-13R
sysmenuitems information 5-9R
sysmenus information 5-9R

USER-MENU menu
Exit option 1-18R
how to access from the Main menu 5-4R
how to use 1-18R
Modify option 1-18R
Run option 1-18R

USING expression
in ACE reports 8-37U, 4-86R
in NLS C-11R, C-31R, C-33R, C-39R, C-40R, C-41R, C-43R
NLS example C-34R, C-45R
Utilities, calling with the User-menu facility 1-25U

V

Values
NULL, in a column 2-21U
querying on a range in PERFORM 4-13U
querying on highest in PERFORM 4-14U
querying on lowest in PERFORM 4-14U
returned to ACE 6-21R
returned to PERFORM 6-21R

VARCHAR data type
defining columns I-3R
in NLS C-4R, C-6R, C-10R, C-19R
printing in reports I-8R
querying with INFORMIX-OnLine I-5R
specifying in forms I-8R
supported in OnLine Intro-14U
transferring with LOAD and UNLOAD I-6R

VERIFY attribute, in PERFORM 6-27U
Verify joins
definition of 6-30U
FORMBUILD 2-27R
in form specifications 6-30U
in multiple-table forms 5-10U
in PERFORM 2-27R
on a screen form 5-10U

View
advantages of 9-10U
definition of 9-10U
View option, in PERFORM 3-36R

W

WEEKDAY( ) function, in ACE reports 4-93R

WHILE statement, in ACE reports 4-68R

Wildcard characters
in PERFORM 4-12U, 3-30R
WITHOUT NULL INPUT option 2-34R

WORDWRAP expression, in ACE reports 4-94R

WORDWRAP keyword I-9R
Writing query results to a file with SQL 7-16U
\textbf{X}

XPG3 C-2R, C-3R
X/Open C-2R
X/Open-defined environment variable
C-12R, C-13R

\textbf{Y}

YEAR() function, in ACE reports 4-95R

\textbf{Z}

Zero, default INTERVAL value 2-9R
Dear Reader,

At Informix Software, we think documentation is very important. After all, manuals are an integral part of our product line. Because documentation is important, we want to know what you think about our manuals. You can tell us by filling out and returning the Reader-Response Card.

Thanks for your help!

1. **Overall, how do you rate this manual?**
   - [ ] Outstanding
   - [ ] Very good
   - [ ] Good
   - [ ] Average
   - [ ] Poor

2. **Is the manual effective?**
   - Is it organized so you can find things? [ ] Yes [ ] No
   - Is the index adequate? [ ] Yes [ ] No
   - Is the table of contents easy to use? [ ] Yes [ ] No
   - If applicable, do you find the statement syntax readable? [ ] Yes [ ] No
   - Are topics presented in a useful sequence? [ ] Yes [ ] No
   - Are the examples useful? [ ] Yes [ ] No

3. **What did you think of the level of description?**
   - Writing quality: [ ] Very good [ ] Good [ ] Average [ ] Poor
   - Clarity: [ ] Very clear [ ] Average [ ] Hard to understand
   - Level: [ ] Too technical [ ] Just right [ ] Oversimplified

4. **When you need to find information quickly, which part of the documentation do you use?**
   - Index: [ ] Often [ ] Sometimes [ ] Rarely [ ] Never
   - Table of Contents: [ ] Often [ ] Sometimes [ ] Rarely [ ] Never
   - Quick Reference Card: [ ] Often [ ] Sometimes [ ] Rarely [ ] Never
   - Chapter summaries: [ ] Often [ ] Sometimes [ ] Rarely [ ] Never
   - On-line help: [ ] Often [ ] Sometimes [ ] Rarely [ ] Never
   - Browse through manuals: [ ] Often [ ] Sometimes [ ] Rarely [ ] Never

5. **How long have you been developing database applications?**
   - How many years have you been programming? [ ] 0 [ ] 1-3 [ ] 4-5 [ ] >5
   - How many years have you been using databases? [ ] 0 [ ] 1-3 [ ] 4-5 [ ] >5
   - How many years have you been using Informix products? [ ] 0 [ ] 1-3 [ ] 4-5 [ ] >5

6. **Have you used other programming languages? If so, which ones?**

7. **Have you used other database products? If so, which ones?**
8. What other Informix products do you use?

________________________________________________________________________

9. What is your job title and area of responsibility?

________________________________________________________________________

10. Is the format of this manual easy to use? Consider page size, typeface, binding, diagrams, and code examples.

________________________________________________________________________

11. Additional comments:

________________________________________________________________________

________________________________________________________________________

Your Name: __________________ Company Name: _________________________________
Address: ________________________________________________________________
City: __________________________ State/Province: _____________________________
Country: ______________________ Zip/Postal Code: _____________________________
Telephone Number: ________________________________________________________

INFORMIX is a registered trademark of Informix Software, Inc.

To mail this card, please fold on the dotted line and staple or tape the open end.