AT&T 585-350-916

Oracle SQL*FORMS Designer's Guide Version 5.0

ORACLE®

The Relational Database Management System

Reader's Comment Form

SQL*Forms Designer's Reference
Version 3.0
Part Number 3304-V3.0 0691

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any suggestions for improvement, please indicate the topic, chapter, and page number below:

Please send your comments to:

SQL*Forms Product Manager
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065

If you would like a reply, please give your name, address, and telephone number below:

Thank you for helping us improve our documentation.

READ THIS FIRST

The *SQL*Forms Designer's Reference* is one of five documents in the SQL*Forms Version 3.0 documentation set. Before you attempt to use it, you should be familiar with the following topics:

- SQL*Forms documentation set
- structure of this book
- typographic conventions

This section tells you if you are ready to use this book and guides you to correct and efficient use of its information.

SQL*Forms Documentation Set

The documentation set for SQL*Forms Version 3.0 consists of the following five documents

- *SQL*Forms Designer's Quick Reference* Version 3.0, Part No. 3708-V3.0
- *SQL*Forms Designer's Reference* Version 3.0, Part No. 3304V3.0
- *SQL*Forms Designer's Tutorial* Version 3.0, Part No. 3302-V3.0
- *SQL*Forms Operator's Guide* Version 3.0, Part No. 3301-V3.0
- *SQL*Forms Operator's Quick Reference* Version 3.0, Part No. 3704-V3.0

The documentation set also includes an online help system.

Each document has a specific purpose and audience. The following table briefly presents each piece of the set, its intended audience, and any suggested prerequisite reading.

Document	***Primary Audience***	***Prerequisite Reading***
Operator's Guide	novice operator	none
Designer's Tutorial	novice designer	Operator's Guide
Online Help System	novice designer	Designer's Tutorial
Designer's Reference	advanced designer	Designer's Tutorial
Operator's Quick Reference	advanced operator	Operator's Guide
Designer's Quick Reference	advanced designer	Designer's Tutorial

Note: As an application designer using Version 3.0 of SQL*Forms, you should also be familiar with the following documents:

- *SQL Language Reference* Version 6.0, Part No. 778-V6.0
- *PL/SQL User's Guide and Reference* Version 1.0, Part No. 800-V1.0
- *0racle*Terminal User's Guide* Version 1.0, Part No. 5206V1.0
- the ORACLE Installation and User's Guide for your particular operating system

Structure of this Book

This book provides comprehensive information about SQL*Forms, presented in concise modules. Reference information (i.e., descriptions and explanations) appears separately from procedural information (i.e., "how to do it" sections) so you do not have to read through information in which you are not currently interested.

If you have not used previous versions of SQL*Forms, you should read Chapter 1, "SQL*Forms Basics." This chapter will introduce you to what kind of tool SQL*Forms is and the conceptual information you need to begin building your own applications.

Major Sections of this Book

The *SQL*Forms Designer's Reference* consists of the following major sections, or parts:

- Part I: Introduction
- Part II: Concepts
- Part III: Objects
- Part IV: Processing
- Part V: Components and Products
- Part VI: Testing
- Part VII: Appendixes

See the table of contents for a chapter-level overview of the major sections of the book.

Use the Index

You can read this book from cover to cover. However, it is primarily designed for you to locate individual topics by looking them up in the index. The index covers the entire book and is located at the back of the book.

The index contains an entry for each section of the book. Each section, where applicable, also includes a list of "Related Topics." This list presents topics that you might also look at to fully understand the current section. Items in the list appear alphabetically. Each item appears with a page number that marks where the primary discussion of this related topic is located. Each item also corresponds directly with an entry in the index in case you want to look for additional page entries or secondary topics.

Topics that appear in the same chapter usually do not reference each other as related topics-the structure of the book implies that such topics are related Instead, “Related Topics” lists point out connections throughout the book that might not be immediately obvious. For example, the “Related Topics” list that appears on the first page of Chapter 12, “Fields,” points you to two sections, “Creating a Field” and “Modifying a Field,” that are related to "Fields” but are covered elsewhere in the book.

Note: If a “Related Topics” entry contains a comma, it implies that the corresponding index entry it refers to is a secondary entry. For example, an entry specified as "NEXT_FIELD, packaged procedure” means that the related index entry is the secondary entry “packaged procedure” under the primary entry “NEXT_FIELD.”

Typographic Conventions

This manual uses typographic conventions to distinguish important elements from the body text of the manual. You should be familiar with these conventions.

Function Keys

SQL*Forms function keys are represented by the name of the SQL*Forms function enclosed in square brackets (e.g., [Next Field]). You can refer to a keypad diagram or the SQL*Forms Show Keys screen (online) for a mapping of these function keys to physical keys on your keyboard.

Screen Messages

Hint messages and error messages appear in regular monotype. For example:

```
Field must be entered.
```

Command Syntax

Commands and parts of commands appear in monotype. For example

```
HOST ( { 'command_ string' |'variable' } [, NOSCREEN] ) ;
```

Parts of commands are set off from each other by the following typographic conventions:

Contention	*Explanation*
`regular monotype`	Indicates a part of the command that you must enter exactly as shown.
`underlined monotype`	Indicates a default parameter. If you indicate no parameter in a parameter set, SQL*Forms applies the default parameter.
`italic monotype`	Indicates a variable. You must substitute an appropriate value when you enter the command.
`|(vertical bar)`	Separates alternative items.
`{ } (braces)`	Encloses a set of alternative items from which you are required to choose.
`[ ] (brackets)`	Encloses an optional item or a set of alternative items from which you have the option to choose.

Your Comments Are Welcome

Oracle knows the value of good documentation. One of the ways we continue to improve our products is by asking for your feedback on product documentation.

Please contact us if you have comments on this manual or on any piece of the SQL*Forms documentation set. Use the Reader's Comment Form at the back of this book, or call 415-598-8000 and ask for the SQL*Forms Product Manager.

CONTENTS

	Read This First	**i**
	SQL*Forms Documentation Set	ii
	Structure of this Book	iii
	Typographic Conventions	iv
	Your Comments Are Welcome	v
PART I	**INTRODUCTION**	
Chapter 1	**SQL*Forms Basics**	**1-1**
	Application Development	1-3
	SQL*Forms Objects	1-4
	A Closer Look at Objects	1-5
	SQL*Forms Processing Model	1-10
	Components of SQL*Forms	1-15
	SQL*Forms and Database terms	1-16
Chapter 2	**Procedural Information**	**2-1**
	Basic Tasks	2-2
	Screen Painter Tasks	2-9

PART II **CONCEPTS**

Chapter 3
Fundamental Processing 3-1
Navigation 3-2
Validation 3-6
Locking 3-11
Commit Processing 3-16
Query Processing 3-24
Function Key Processing 3-31
Field Adjustment 3-32

Chapter 4
Trigger Concepts 4-1
V3 Triggers versus V2 Triggers 4-2
Trigger Definition Level 4-2
Trigger Scope 4-3
Trigger Processing 4-5

Chapter 5
Interface Elements 5-1
Alert 5-3
Button 5-4
Check Box 5-5
Dialog Box 5-6
Field 5-6
Form 5-8
List of values 5-9
Menu 5-11
Message Line 5-12
Scroll Region 5-13
Spread Table 5-14
Status Line 5-16

Chapter 6	**Context**	**6-1**
	Context and Main Menu Choices	6-2
	Context and Zoom Connections	6-2
	Context for Forms, Blocks, and Fields	6-3
	Context for Triggers	6-5
	Context in the Screen Painter	6-6
	Changing Context with Function Keys	6-6
Chapter 7	**Function Keys**	**7-1**
	Keyboard Map	7-2
	Function Key Descriptions	7-3
	Function Key Groupings	7-17
Chapter 8	**Help System**	**8-1**
	Organization of the Help Material	8-2
	Features of the Help System	8-3
PART III	**OBJECTS**	
Chapter 9	**Forms**	**9-1**
	Defining Forms	9-2
	Form Characteristics	9-3
	Non-enterable Form Type	9-6
Chapter 10	**Pages**	**10-1**
	Defining Pages	10-2
	Page Characteristics	10-4
Chapter 11	**Blocks**	**11-1**
	Defining Blocks	11-2
	Block Characteristics	11-6
	Block Types	11-18
	Master-Detail Relationship	11-20

Chapter 12	**Fields**	**12-1**
	Defining Fields	12-2
	Field Characteristics	12-4
	Field Types	12-26
Chapter 13	**Pop-up Windows**	**13-1**
	Pop-up Field Editor	13-2
	Pop-up List of Values	13-4
	Pop-up Page	13-5
Chapter 14	**Triggers**	**14-1**
	Defining Triggers	14-2
	Trigger Characteristics	14-4
	Trigger Types	14-8
Chapter 15	**Pl/sql**	**15-1**
	How SQL*Forms Uses PL/SQL	15-2
	PL/SQL Blocks in SQL*Forms	15-2
	SQL/Forms Objects in PL/SQL	15-7
	Packaged Procedures in PL/SQL	15-8
	PL/SQL Restrictions within SQL*Forms	15-8
	Executable Statement Failure	15-8
	PL/SQL Compilation	15-11
Chapter 16	**Packaged Procedures**	**16-1**
	Functions without Packaged Procedures	16-2
	Packaged Procedure Definitions	16-2
	Unrestricted Packaged Procedures	16-34
	Restricted Packaged Procedures	16-35
Chapter 17	**Packaged Functions**	**17-1**
	Packaged Function Definitions	17-2

Chapter 18	**Variables**	**18-1**
	PL/SQL Local Variables	18-2
	Global Variables	18-2
	System Variables	18-3
Chapter 19	**User Exits**	**19-1**
	Types of User Exits	19-3
	Creating a User Exit	19-4
	Pop-up Window Routines	19-10
Chapter 20	**Working with Objects**	**20-1**
	Object Naming Conventions	20-2
	Object Referencing	20-4
PART IV	**PROCESSING**	
Chapter 21	**Event Processing**	**21-1**
	Event Flow Charts	21-2
	Events and Sub-events	21-131
Chapter 22	**Constraints**	**22-1**
	Object constraints	22-2
	Cursor Constraints	22-3
	16-bit Environment Constraints	22-4
PART V	**COMPONENTS AND PRODUCTS**	
Chapter 23	**Components**	**23-1**
	Automatic Login	23-2
	Remote Login	23-2
	SQL*Forms (Convert)	23-3
	SQL*Forms (Design)	23-5
	SQL*Forms (Generate)	23-6
	SQL*Forms (Run Form)	23-7
	Command Line Switches	23-8

Chapter 24 **Form Storage** **24-1**
Form Storage Formats 24-2
Portability among Systems 24-3
SQL*Forms (Design) Base Tables 24-4
INP File Format 24-13

Chapter 25 **Connecting to other Products** **25-1**
Connecting to the ORACLE RDBMS 25-2
Connecting to Application Development Tools 25-2
Using ORACLE Precompilers with SQL*Forms 25-9
Calling Forms from Programs 25-10
Using Oracle*Terminal 25-11

PART VI **TESTING**

Chapter 26 **Options** **26-1**
SQL*Forms (Run Form) Options 26-2
SQL*Forms (Design) Options 26-5
User Preference File 26-7

Chapter 27 **Application Documentation** **27-1**
Detail Documentation 27-2
Fields Documentation 27-3
Object Reference Documentation 27-4
Pages Documentation 27-4
Procedures Documentation 27-5
Summary Documentation 27-5
Triggers Documentation 27-6

Chapter 28	**Debug Facility**	**28-1**
	Debug Messages	28-2
	Break Processing	28-2
Chapter 29	**Edit Keystroke Facility**	**29-1**
	Using the Edit Keystroke Facility	29-2
	Key Script File	29-3
PART VII	**APPENDIXES**	
Appendix A	**Summary of Features**	**A-1**
	New Features	A-2
	Changed Features	A-3
Appendix B	**Upward Compatibility**	**B-1**
	Converting Existing Forms to Version 3.0 Status	B-2
	Translating Triggers into PL/SQL	B-3
	Changes to Data Types	B-9
Appendix C	**Advisory**	**C-1**
Appendix D	**System-specific Considerations**	**D-1**
	Block Mode Environment	D-2
	Bit-Mapped Environment	D-4
Appendix E	**Messages**	**E-1**
	SQL*Forms (Convert) Messages	E-2
	SQL*Forms (Design) Messages	E-11
	SQL*Forms (Generate) Messages	E-30
	SQL*Forms (Run Form) Messages	E-42

Appendix F **V2 Triggers** **F-1**
Defining V2 Triggers F-2
Trigger Step Characteristics F-4
Commands in Trigger Steps F-8
V2 Function Codes F-12
CASE Statement F-27
Variables in Trigger Steps F-28
V2 Trigger Processing F-29

Index

Reader's Comment Form

PART

I

INTRODUCTION

CHAPTER

1

SQL*FORMS BASICS

SQL*Forms is a general-purpose tool for developing and executing forms-based interactive applications. The design component of this tool is speially designed for application developers and programmers, and it allows you to perform the following tasks:

- define transactions that combine data from multiple tables into a single form
- quickly create default forms using a rich set of defaults, including automatic master-detail relationships
- customize all aspects of an application definition

SQL*Forms uses Oracle Corporation's standard fill-in-the-form user interface to enhance development productivity and reduce leaning time.

The execution component of SQL*Forms allows operators to execute, or run, the forms-based applications that you design.

This chapter covers the terms and concepts related to SQL*Forms and to relational databases with which you must be familiar to successfully use this reference manual. It is intended for new users, and those users who wish to review basic concepts of SQL*Forms.

Specifically, this chapter covers the following topics:

- application development
- SQL*Form objects
- a closer look at objects
- SQL*Forms processing model
- components of SQL*Forms
- SQL*Forms and database terms

This chapter serves as an overview of many topics relating to SQL*Forms. Much of the material in this chapter is repeated and explained in more depth in other places throughout this reference. The "Related Topics" lists tell you where to look for more information on a particular topic.

Application Development

Over the past two decades, the technology used to build applications has slowly evolved. The evolution started with completely procedural assemblers, and moved through third generation languages (3GLs) such as FORTRAN and COBOL to fourth generation languages (4GLs). The trend in this evolution has been to provide more and more functionality in a non-procedural way. Rather than specifying how something is to be done, the application developer simply specifies what should be done.

The SQL*Forms Solution

SQL*Forms furthers the trend in application development by providing large amounts of default functionality-in fact, complete working applications. SQL*Forms provides an overall application structure and a framework of functionality for all objects within an application. As a result, even the simplest SQL*Forms application has certain "default" functionality, such as the ability to validate data types, perform navigation throughout a form, and access the database.

Once you have created a default application, you can modify many of its functional characteristics and its appearance. The fill-in-the-form user interface allows you to define the specifics of an application's definition by typing the information into text areas on your screen or by "tuning on or off" items in a definition list. For example, assume that your default application defines an area, or field, in the application's display that shows data from a database column and allows operators to modify that data. If you want to modify the definition of this field so that an operator cannot modify the displayed data, you simply turn off the Input Allowed characteristic for that element. You perform modifications like this example without any programming.

In addition to the fill-in-the-form interface, SQL*Forms provides a powerful screen painter to edit screen images and add constant text, prompts, and graphic elements.

If an application requires special features or application logic, you can add commands by using PL/SQL, the integrated procedural language, from within the interface or by providing calls to third generation language programs.

This approach-of using broad default functionality and a fill-in-the-form user interface-leads to significant gains in productivity.

Portable Development Environment

In addition to its ability to quickly create powerful applications, SQL*Forms provides a portable application development environment. In the past, building applications that were to run on many different machines or display devices often meant writing and maintaining many different applications. SQL*Forms applications are portable across hardware, operating systems, and windowing systems.

SQL*Forms Objects

A SQL*Forms application is made up of objects. These objects contain all of the information that is needed to manipulate and produce a SQL*Forms application.

The primary object, or component, of a SQL*Forms application is the form. This object can comprise an entire application, or it can link with other forms, groups of forms, menus, reports, and other components to create a more complex application. Within SQL*Forms, you create forms and, within forms, any links to other components.

A form is made up of additional objects. These objects link the form to database elements, such as columns and tables, and provide control over the flow of execution.

Blocks	Describe each section or subsection of the form, and serve as the basis of default database interaction.
Fields	Represent columns or data entry areas and describe how the data should be displayed and validated and how an operator should interact with the data while it is entered.
Pages	Are collections of display information, such as constant text and graphics. All fields are displayed on some page.
Triggers	Are sets of processing commands associated with event points, such as when a particular function key is pressed by the operator.
Form-level Procedures	Are sets of processing commands that can be invoked from any trigger or form-level procedure, and can take arguments.

Every form usually contains at least one block, one page, and one or more fields. Each object in a form has a set of attributes, or characteristics, which provide information about the object. Fields are named, for instance, so they can be referenced in computations and when manipulating data in the database.

You create most objects through a default facility, and most or all of the default characteristics are usually sufficient When you want to modify object characteristics or attach custom processing commands, you use elements in the SQL*Forms design interface.

Reusing Objects

As you create an application, you usually build the objects with which the application is made. However, since objects are separate elements, you can build an application by copying existing objects into the application definition or by referencing existing objects in the definition. (Referencing is the same as copying, except SQL*Forms limits modifications to referenced objects.) Because objects are "reusable," you can create models for commonly used objects. Using models for commonly used triggers-for example-reduces typing, reduces the possibility of errors, and provides a good way of guaranteeing that designers use proven and efficient triggers.

Related Topics: Referencing objects 20-4

A Closer Look at Objects

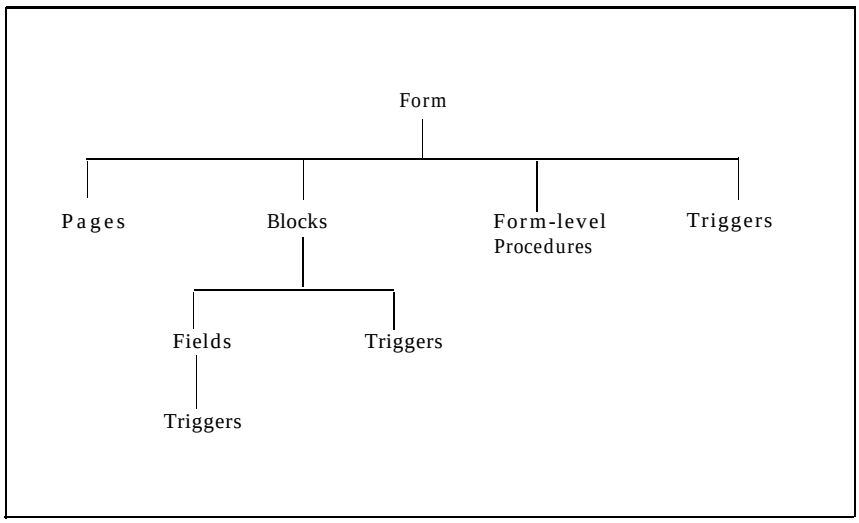
The previous pages described how objects are the basic building blocks of applications and briefly introduced those objects: forms, blocks, fields, pages, triggers, and form-level procedures. This section describes SQL*Forms objects in more detail and explains how they relate to each other. Later chapters in this book describe in detail the characteristics of each of these objects and how you define them through the design interface.

Note: You might have to refer to the "SQL*Forms and Database Terms" section in this chapter to understand the following discussion.

Object Hierarchy

SQL*Forms objects exist in a hierarchy. Figure 1-1 illustrates this hierarchy and shows how higher level objects own lower-level objects.

FIGURE 1-1
The SQL*Forms Object Hierarchy



The form is the primary object, or component, of a SQL*Forms application, and it therefore resides at the top of the hierarchy. Within an application, a form object can own all other objects. The exceptions are objects that are referenced by a form but belong to another form. Note that an object can be shared by many different forms (through object referencing) but has only one owner.

Individual blocks can own fields and triggers, and individual fields can own triggers. Trigger objects are unique in that they can exist at more than one place in the hierarchy. A trigger's definition level can affect the scope of its actions.

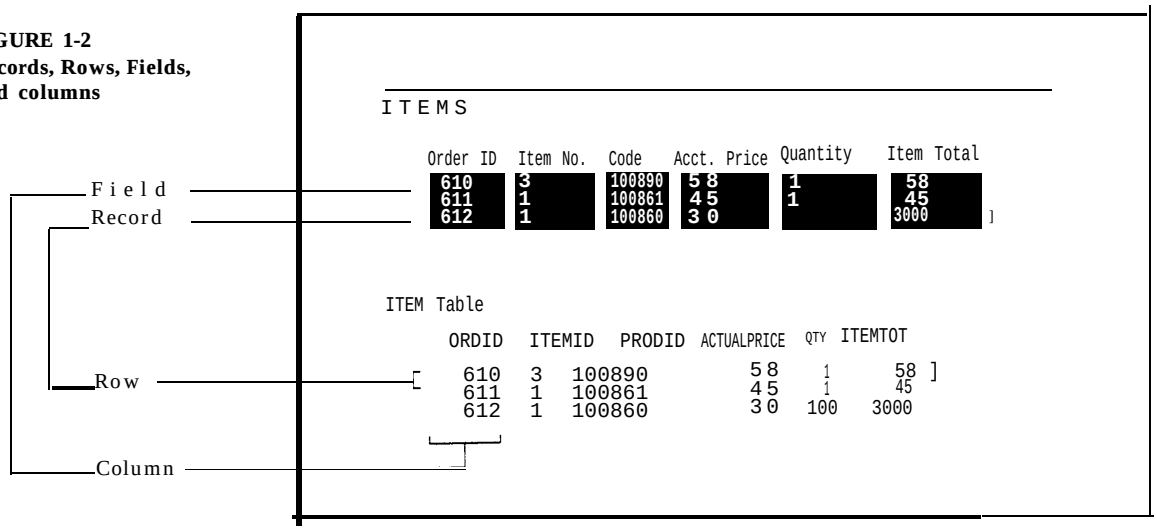
Related Topics: Context 6-1
Trigger scope 1-14

Blocks

A form contains one or more blocks. Each block can relate directly to a single database table or view. This means, as shown in Figure 1-2, that each field in the block is associated with a column in the table (known as the base table) or view. By default, this direct relationship allows operators to query, update, insert, and delete data in the base table or view. SQL*Forms performs these operations automatically when a base table is specified. Alternatively, a block can relate indirectly to, or contain information from, any number of database tables or views.

A block displays data in records. As shown in Figure 1-2, these records are groupings of fields with related information that correspond roughly to rows in a table.

FIGURE 1-2
Records, Rows, Fields, and columns



Blocks have characteristics that you can define. These characteristics determine attributes such as how to order the information that SQL*Forms retrieves into a block during a query, the number of records of data that should be displayed in a block, and whether to perform certain types of data validation.

Related Topics: Blocks 11-1

Relationships between Blocks

You can relate blocks to each other through master-detail relationships. A master-d-relationship corresponds to a primary-foreign key relationship between the base tables of two blocks. SQL*Forms allows you to automatically establish master-detail relationships when you create blocks. This feature allows you to create a form that shows all rows in the "detail" block whenever the associated row in the "master" block displays-without having to manually add any special processing commands.

Related Topics: Master-detail relationship 11-20

Fields

At the most basic level, fields serve as containers for data within a form. The values in a field can be manipulated by the operator-as when he enters a value into the field-or by a trigger or procedure-as when the total for an order is calculated and displayed. A field is always owned by, or associated with, a block. Each block normally owns one or more fields.

The fields in a block usually correspond to columns of data in the base table for the block that owns them. When this is the case, entering data into a field can affect the values in a row in the database, and the field can be populated by a query on the base table. However, fields do not always correspond to columns in the base table for the block; they sometimes contain calculated values, display related information from other tables, or accept operator input for later processing.

A form can display any number of "instances" of the same field. That is, you can define a block to display more than one record of information at a time. Each displayed record contains an instance of each field in the block. Each instance potentially corresponds to an individual row in the database, but all field instances share the same characteristics.

Fields have a number of characteristics that describe them. These characteristics specify, among other things, what kind of assistance an operator can get when in the field, how data should be displayed in the field, and what types of data are valid entries in the field.

Related Topics: Fields 12-1

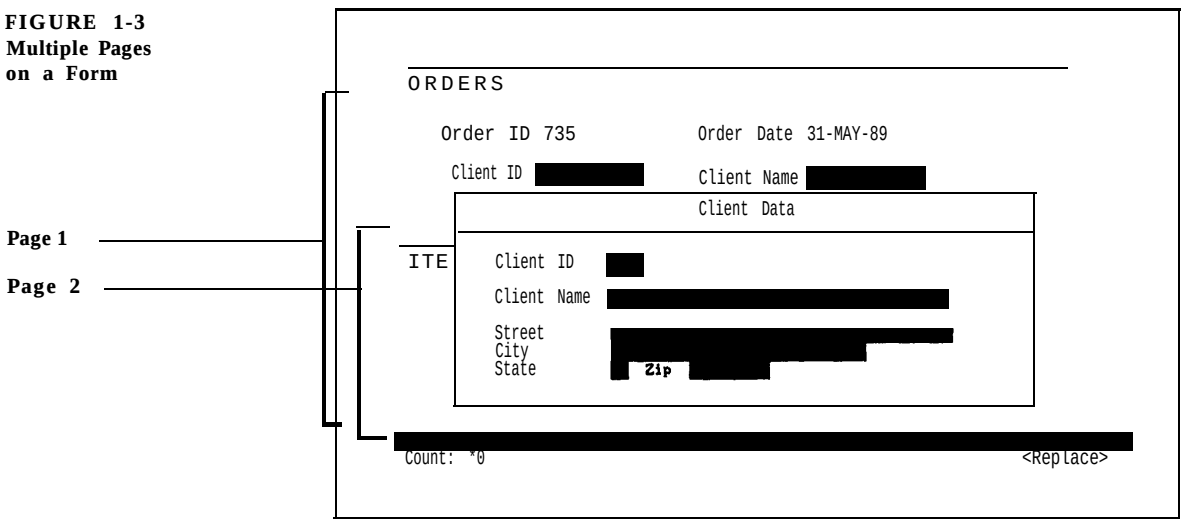
Pages

Pages are collections of display information. Similar in concept to a slide projection, a page displays fields and constant text to operators when they execute a form. Note that pages only serve to display information. They own the constant text (e.g., field and block labels) that they display, but they do not own the fields that they display nor the blocks that are sometimes associated with them. While all of the fields in a single block often display together on a single page, they can appear on any number of different pages and in proximity to another block's fields. Logic ties fields to blocks; display necessities tie fields to pages.

Pages have a number of characteristics, such as size and whether a page should display as a pop-up window.

Do not confuse pages with screens. Screens are terminal display areas, and are not directly related to pages. For example, multiple pages can appear on the screen at one time as shown in Figure 1-3. In addition, pages might have sizes greater than the screen can display.

FIGURE 1-3
Multiple Pages on a Form



Related Topics: Pages 10-1

Triggers

Triggers are sets of processing commands that you write. Different types of triggers are associated with different event points in SQL*Forms processing. When an event occurs with which a trigger is associated, the trigger "fires," executing the coremands that it contains. For example, when an operator presses [Next Field] to navigate from one field to another, the Key-NXTFLD trigger fires.

Most of the triggers you write, regardless of type, will be V3 (Version 3) triggers. This style of trigger consists of commands written in PL/SQL, a complete procedural language integrated with the standard database language SQL. You can put the following types of commands in a V3 trigger:

- SQL statements
- procedural logic statements
- form processing commands

Note: V2 style triggers are written in steps, and Oracle Corporation maintains them for compatibility with older versions of SQL*Forms.

Related Topics: Triggers, concepts 4-1

SQL Statements in Triggers

SQL, or Structured Query Language, statements can be included in any trigger. While certain restrictions exist on what statements should be executed in what type of trigger, in general any valid SQL statement can be used directly in a trigger. The ORACLE RDBMS executes all such statements.

Procedural Logic Statements in Triggers

Procedural logic statements-such as conditional logic, looping, and error handling-are also available within a trigger. Such statements can use field values, system variables, and other control information to continue processing. In addition, these statements can change field values via direct assignment.

Form Processing Commands in Triggers

Additional commands, known as packaged procedures, can affect form objects and are extensions to PL/SQL that are available only in SQL*Forms. These commands allow you to perform actions such as moving the cursor, invoking another form, or changing field characteristics. Packaged procedures perform these commands under programmatic control based on events in the application.

Form-Level Procedures

Form-level procedures are callable sets of commands. These procedures can use any commands that a trigger can use. They can also take arguments and return values, just as subroutines do in a third generation language such as COBOL.

Form-level procedures are form-local in scope. That is, they can be called by another form-level procedure or a trigger in the form in which they are defined or referenced. This feature allows you to create general routines that you can use many times. This reduces the amount of logic that you have to write for any task, and leads to more consistent and efficient applications.

Related Topics: Form-level procedures 15-4

SQL*Forms Processing Model

When a form is executed, SQL*Forms follows a predefine set of rules for how actions should occur. These actions include navigation within the application and the validation of data. Within the processing rules, you can customize the default behavior of SQL*Forms to meet the needs of your applications.

Related Topics: Fundamental processing 3-1

Events and Functions

All processing centers around events. Put simply, events are things that occur when a form is executed. SQL*Forms knows about events and handles them by executing functions. An example of an event is the operator pressing the [Next Field] key. When this event occurs, SQL*Forms executes a predefined behavior, which can be the default behavior (executing the NEXT_FIELD function which moves the cursor to the next field in sequence) or a custom behavior that you have defined (such as executing the MESSAGE function and the NEXT_FIELD function to display a message for the operator before moving the cursor).

Note that during processing, events are usually nested. That is, the occurrence of one event usually invokes functions that invoke other events.

Inherent in the functions that events invoke-and that are invoked by your triggers-are the processes of navigation and validation.

Related Topics: Event processing 21-1

Trigger points

Every function that an event calls might have one or more trigger points associated with it. A trigger point is a temporal "place" in an event with which a specific trigger type is associated. When SQL*Forms is processing an event and reaches a trigger point, it executes the associated trigger if you have defined a trigger of that type. For example, in the execution of the NEXT_FIELD function there is a trigger point for the Post-Field trigger. If you have defined a Post-Field trigger, it fires when SQL*Forms reaches that point in processing.

Trigger points and triggers are your primary tools for modifying the way that SQL*Forms handles, or processes, a particular event. Note that not all events have trigger points and triggers. For example, when a character is typed on a character mode terminal an event occurs; that is, SQL*Forms processes the keystroke and echoes the character to the field. However, there is no trigger point and you cannot attach a trigger to this event

Events in Different Environments

The display technology you use can have an impact on how SQL*Forms handles events. For example, when an operator presses the TAB key on a character mode device, SQL*Forms invokes the NEXT_FIELD function and processes any triggers attached to the associated trigger points. However, on a block mode device TAB invokes a local editing function and SQL*Forms does not process any functions or triggers-thereby allowing the operator to move between fields without interrupting the host and producing an event.

If you design applications that must run on a number of different devices, you should keep these differences in mind.

Related Topics: System-specific considerations D-1

Navigation

Navigation is an internal function that is invoked by specific events. SQL*Forms performs navigation primarily to move the cursor from one location to another. However, SQL*Forms also performs navigation during events when the cursor does not move, but SQL*Forms must navigate outside of an object to perform processing on it.

The main concepts of navigation are the navigation unit and the cursor. The navigation unit is an internal object that keeps track of the object that SQL*Forms is currently processing. This navigation unit is always defined as a specific form, block, record, or field in the form, or as outside the form. SQL*Forms navigates to a specific unit, by moving the navigation unit from one object to another.

The cursor is an externally visible object that indicates the active field in a form while the form is running. Most navigation involves moving the cursor from its current, or initial, position to a target position. To move the cursor, SQL*Forms first moves the navigation unit until it is defined as the target object.

For example, when an operator presses [Next Record], SQL*Forms invokes the navigation function. To navigate the cursor from the current field (record1.field1) to the target field (record2.field1), SQL*Forms moves the navigation unit from record1.field1 to record1 to block1 to record2 to record2.field1. As SQL*Forms “enters” and “leaves” each object, a sub-event occurs, which can fire associated triggers. Once all of the sub-processing is completed and the navigation unit is defined successfully as record2.field1 (the target field), SQL*Forms moves the cursor to that location.

Related Topics: Navigation 3-2

Validation

Validation is an internal function that is invoked by specific events. Validation is the process by which SQL*Forms determines whether the data in an object is valid or correct. SQL*Forms can validate form, block, record, and field objects.

When SQL*Forms performs default validation on an object, it checks to make sure that the object conforms to all of the rules for that object. For example, SQL*Forms checks fields for a valid data type, a specified range, and several other criteria. For records, blocks, and forms, SQL*Forms checks that each of the objects that the record, block, or form owns are valid. Although validation often occurs when navigation occurs-e.g., when you leave a field or enter a record-the two processes are independent.

You can override default validation by defining triggers at validation trigger points. You can also control the extent and frequency of validation in a form by defining the Validation Unit form characteristic.

Related Topics: Validation 3-6

Trigger Processing

Events invoke functions, which have trigger points. When SQL*Forms processes a trigger point, it executes, or fires, the associated trigger. Every trigger point has a specific type of trigger associated with it. When you define a trigger, that trigger is automatically associated with the appropriate trigger point This means that if you define a Post-Block trigger, it fires when the operator navigates out of the block, not when the operator navigates out of the field or deletes a record.

When a trigger fires, SQL*Forms executes all of its statements in the order defined by the flow of the PL/SQL block. Each statement can succeed or fail-a SQL*Forms status, not a PL/SQL status it can raise a pre-defined PL/SQL exception. Based on a statement's " outcome, you can also explicitly raise a user-defined PL/SQL exception. SQL*Forms handles this processing according to the individual statement and any logic that you add to determine later behavior.

Triggers-the collection of individual statements-can also succeed or fail. If they fail, SQL*Forms will not continue the event that it was processing. For example, if the Post-Field trigger fails, SQL*Forms reverses the navigation and re-enters the field it was attempting to leave. This behavior allows you to influence how SQL*Forms continues its processing after the tigger finishes execution.

If a trigger executes statements that would cause SQL*Forms to handle other events, SQL*Forms will suspend processing the current trigger to process the new event. For example, if a trigger invokes the NEXT_FIELD function, SQL*Forms completely processes the NEXT_FIELD event (and any events it causes to occur) before continuing to process the original trigger. This behavior is similar to how a subroutine is called in a 3GL program.

Related Topics: Trigger processing 45

Scoping and Inheritance

In addition to being associated with events, triggers are also associated with objects. When you define a trigger, you always associate it with a form, block, or field. The object that owns a trigger determines the scope of that trigger.

When SQL*Forms processes a trigger point, it expects to execute a particular type of trigger. Since you can define many triggers of the same type, SQL*Forms decides which trigger to fire by choosing the one assigned to the smallest applicable object. For example, if processing calls for an On-Validate-Field trigger to fire, SQL*Forms first checks for an On-Validate-Field trigger assigned to the current field. If SQL*Forms finds one, it executes the trigger, overriding any On-Validate-Field triggers assigned to the current block or form. However, if SQL*Forms does not find a trigger assigned to the current field, it checks for one assigned to the current block and-if still none-the current form. Because SQL*Forms uses inheritance for all triggers, any On-Validate-Field trigger that is assigned to a higher object level is available to all objects that are owned by the higher level object

Components of SQL*Forms

SQL*Forms consists of the following programs, or components, that you can execute independently from the command line. As a designer, you most often use the SQL*Forms (Design) component, from which you can access the other components as you require them.

SQL*Forms (Design)	Allows you to define forms using menus, fill-in-the-form screens, and a comprehensive online help system. SQL*Forms (Design) stores object definitions in a set of tables in an ORACLE database.
SQL*Forms (Run Form)	Allows operators to execute pre-defined forms interactively. You can also use this component to test and debug forms.
SQL*Forms (Generate)	Allows you to create a form definition file that SQL*Forms (Run Form) can execute from a text file, or flat file, representation.
SQL*Forms (Convert)	Allows you to convert between various representations of a form: • Create text files from definitions stored in database tables. • Insert form definitions (from SQL*Forms (Design)) and text files into database tables. • Create text files in the Version 3.0 format from text files created by previous versions of SQL*Forms.

Related Topics: Components 23-1

SQL*Forms and Database Terms

To use this reference effectively, you should have a working knowledge of the following SQL*Forms terms and database terms.

SQL*Forms Terms

Figures 1-4 and 1-5 illustrate some basic SQL*Forms terms.

FIGURE 1-4
Page 1 of a Sample Form

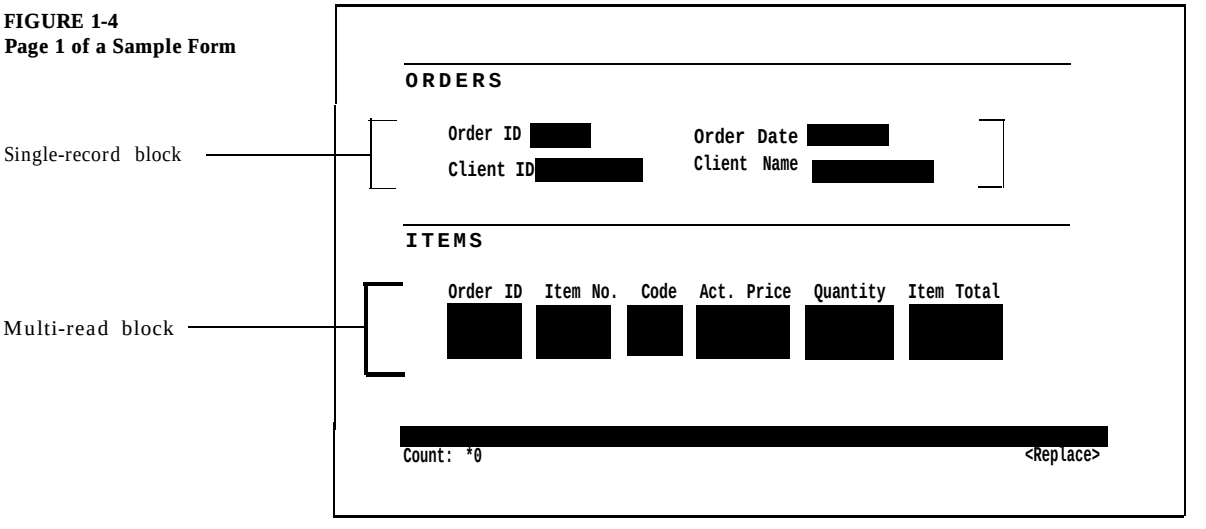
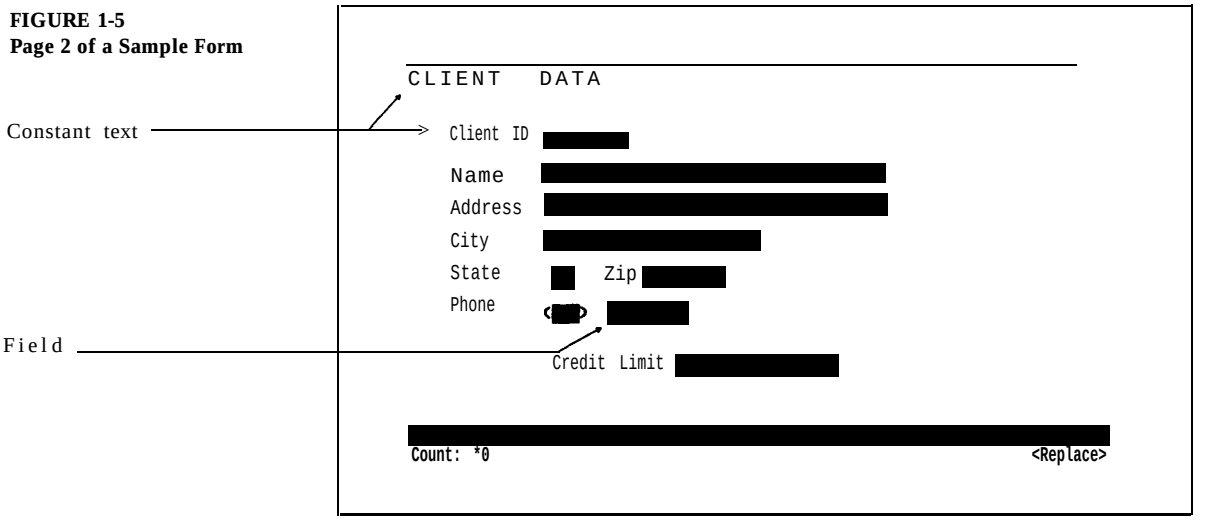


FIGURE 1-5
Page 2 of a Sample Form



Anonymous block	APL/SQL block that has no name and does not require the explicit presence of the BEGIN and END keywords to enclose the executable statements. You can use anonymous blocks in the trigger text of triggers that are written in PL/SQL (i.e., V3 triggers).
Application	A form or a series of forms that satisfies a business function. For example, you might build an application to serve as an order entry system. An application can also incorporate menus, reports, and other components.
Base table	The database table on which a block is based.
Base table field	A field that corresponds to a column in the base table of the block that owns the field.
Block	(1) A logical collection of fields in a form. A block can correspond to one table in the database or to no table. (2) A collection of PL/SQL commands. See the *PL/SQL User's Guide and Reference.*
Constant text	Text that appears in a form every time it is run. You enter this text-including field titles, directions, and graphics-from the screen painter. See Figure 1-5. Constant text is also known as boilerplate text.
Context	A concept that you use to determine what parts of a form definition you can currently access from within the SQL*Forms (Design) interface.
Designer	An application developer or programmer who uses SQL*Forms to create and modify forms.
Field	An area on a page that can display data and accept operator input. Fields usually appear highlighted or underlined. The data that they display can correspond to data-from a column in a database table. See Figure 1-5.

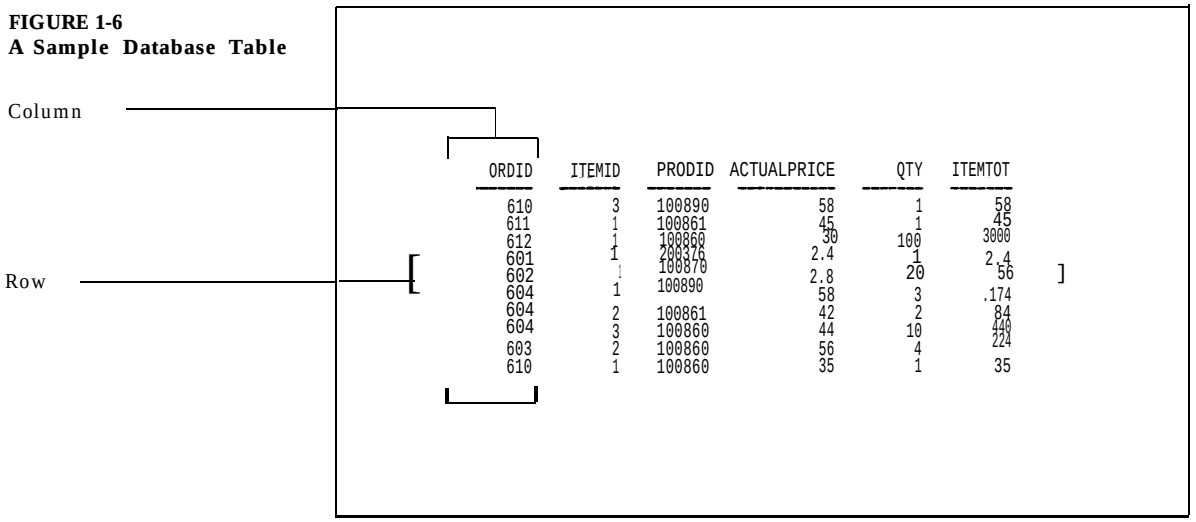
Form	A logical collection of blocks, fields, triggers, and form-level procedures. Its definition determines how an application will work and how it will appear to operators. When an operator executes a form, the form can display its elements on the computer or terminal screen.
Form-level procedure	A PL/SQL block that has a name and requires the full PL/SQL syntax. You can reference form-level procedures in the trigger text of V3 triggers in a form.
Global variable	A data store that exists across forms. When a form creates a global variable SQL*Forms maintains the variable until the application is exited, or until a later form explicitly removes it.
Login account	A username and password to use the ORACLE RDBMS. This account is usually separate from your operating system account.
Multi-record block	A block that can display more than one record at a time. See Figure 1-4.
Object	(1) A group of data, such as a form, block, field, or trigger, that you can copy, move, or delete in a single operation. (2) A named group of data in the ORACLE database, such as a table or index.
Operator	An end-user of a SQL*Forms application.
Packaged procedure	A built-in PL/SQL procedure that is available in SQL*Forms. Each packaged procedure executes a SQL*Forms function, such as clearing a field or executing a query.

Page	A collection of display information. Similar in concept to a slide projection, a page displays fields and constant text to operators on their computer or terminal screens when they execute a form. A form usually has one or more pages and can have as many pages as you wish to create. See Figure 1-4. **Note:** Since pages can vary in size and can overlay each other, it is possible for the screen to display more than one page of a form at a time, or a fraction of a page.
Pop-up window	A SQL*Forms object that overlays an area of the current display. SQL*Forms displays a pop-up window in response to some event or user action. A pop-up window can be a pop-up page, a pop-up list of values, or the pop-up field editor.
Record	Data from one row in a database table or view, as represented in a form.
S cope	The domain, or range, in which a trigger operates. This domain is determined by the level (form, block, or field) at which you define the trigger.
Screen painter	A SQL*Forms "work area" where you can modify the layout of forms. The screen painter displays one page area at a time.
Single-record block	A block that can display only one record at a time. See Figure 1-4.
System variable	A status or information indicator that provides information about the form or the objects within the form. For example, a system variable might contain the name of the current field or the value of the current field.
Trigger	A piece of logic that is executed at, or "triggered" by, a SQL*Forms (Run Form) event.

Database Terms

Figure 1-6 illustrates two basic relational database terms. The database and its functions are completely described in the *ORACLE RDBMS Database Adminstrator's Guide.*

FIGURE 1-6
A Sample Database Table



Column	In a database table, a "vertical" group of cells that represents one kind of data. See Figure 1-6.
Constraint	A rule or restriction concerning a piece of data (such as a NOT NULL restriction on a column) that is enforced at the data level, rather than the object or application level.
Data dictionary	A set of tables and views owned by the Database Administrator. It is a central source of information for the ORACLE RDBMS.
Database	A collection of tables controlled by one data dictionary.
Database Administrator (DBA)	An ORACLE user who oversees the administration of a database. The DBA is authorized to grant and revoke other users' access to the database, modify ORACLE options that affect all users, and perform other administrative functions.
Foreign key	A value or column in one table that refers to a primary key in another table.

Index	An optional structure associated with a table that is used by the RDBMS to locate rows of the table quickly, and (optionally) to guarantee that every row is unique.
Lock	A restriction that assigns temporary ownership, or control, of a database resource (e.g., a table or a row) to a user. A lock can prevent other users from changing data that a user is updating. Locks allow operators to perform updates without compromising the integrity of the database.
Primary key	Information used to identify a row in a table. Also known as a key.
Row	In a database table, a "horizontal" group of column values. See Figure 1-6.
SQL	The English-like data language that is the basic user interface for storing and retrieving information in an ORACLE database. For further information on SQL, consult the *SQL Language Reference Manual.*
Table	The basic unit of information in a relational database management system. A table is a two dimensional grid that is made up of rows and columns. See Figure 1-6.
Transaction	A logical unit of work. Specifically, a transaction is the group of events that occurs between any of the following events: the user connects to ORACLE, disconnects from ORACLE, commits changes to the database, or rolls changes back.

CHAPTER

2

PROCEDURAL INFORMATION

This chapter provides brief step-by-step descriptions of selected tasks that you perform with SQL*Forms (Design). Specifically, this chapter divides this procedural information into tasks that you perform in the screen painter and other basic tasks.

The procedural information in this chapter describes the most common tasks involved in using SQL*Forms. For information on additional tasks, refer to the *SQL*Forms Designer's Tutorial.*

Basic Tasks

This section presents brief descriptions of some basic tasks that you perform in the SQL*Forms (Design) interface. Specifically, this section covers the following tasks in alphabetical order:

- copying or referencing an object
- creating a control block
- creating a default block
- creating a field
- creating a form
- creating application documentation
- executing a form
- generating a form
- invoking online help
- loading a form
- modifying a field
- opening a form
- quitting SQL*Forms (Design)
- saving a form
- starting SQL*Forms (Design)

Copying or Referencing an Object

1. From the SQL*Forms (Design) interface, press [Copy Object].

   The Source Object dialog box and Target Object dialog box appear.

2. Specify COPY or REFERENCE in the Copy or Reference Object field.

3. Complete the information on the source object, pressing [Next Field] to move from field to field. You must provide all information that is associated with the object you are copying or referencing. For example, to reference a field you must specify the owner (ORACLE username), form, and block that the field is associated with.

   Note:

   - You can only reference objects that exist in the current database.
   - You cannot reference an object that is owned by the current form. You can copy an object that is owned by the current form by leaving the Owner field and Form field blank.
   - You cannot reference constant text. You can copy constant text by filling in the Text Page field with the page number of the page that owns the text. Copying blocks and fields does not copy any constant text.

4. Complete the information on the target object. You must provide all information that is associated with the object to which you are copying or referencing. For example, to copy a field you must specify the block that will own the field and the location at which the field should display (X coordinate, Y coordinate, and page number).

5. Press [Accept].

   SQL*Forms copies the source object to the target objector references the source object from the target object.

Related Topics: Referencing objects 20-4

Creating a Control Block

1. From the Block menu, select the Modify option.

   The Block Definition form or spread table appears.

2. Complete the form or spread table. Be sure to specify the Block Name characteristic and not to specify the Base Table characteristic.

3. Press [Accept] to accept the block definition.

4. Select the Field option from the main menu.

   The Field menu appears.

5. Select the Modify option.

   The Field Definition form or spread table appears.

6. Create the fields that you wish to assign to the control block. Be sure to turn off the Base Table characteristic.

Related Topics: Blocks, characteristics 11-6
Control block 11-18

Creating a Default Block

1. From the Block menu, select the Default option.

   The Default Block form appears.

2. Complete the Default Block form.

   You are required to specify the following block characteristics:

   - Block Name
   - Base Table
   - Sequence Number
   - Records Displayed
   - Base Line
   - Page Number

3. Press [Accept] to accept the block definition.

Related Topics: Blocks, characteristics 11-6

Creating a Field

1. Display the Field Definition form or Field Definition spread table.
2. If the cursor is not in a new record, press [Insert Record].

   SQL*Forms creates a new record and moves the cursor to it.

3. Complete the Field Definition form or spread table.

   You are required to specify the following field characteristics:

   - Field Name
   - Sequence Number
   - Data Type
   - Select Attributes
   - Screen Position

4. Press [Accept] to accept the field definition.

Related Topics: Creating a field, in block mode D-3
Fields, characteristics 12-4

Creating a Field from the Screen Painter

When you create a field, you might want to position the new field while looking at the rest of the form. If so, you can create a field by starting from the screen painter.

1. In the screen painter, move the cursor to the position you want to make the left endpoint of the field. Press [Select].
2. Move the cursor to the position you want to make the right endpoint of the field. Press [Select].
3. Press [Define Field].

   SQL*Forms creates the field in the screen painter and displays the Field Definition form.
4. Complete the Field Definition form.
5. Press [Accept] to accept the field definition.

Creating a Form

1. From the Action menu, select the New option.

   The New Form dialog box appears.
2. In the Name field, type the name that you want to assign the new form. Press [Accept].
3. Continue building the form by defining the form and creating blocks.

Creating Application Documentation

1. Make current the form for which you want to create documentation. This step requires that you have loaded or opened the form.
2. From the Action menu, select the Print Doc option.

   The Print Doc menu appears.
3. Select the type of documentation that you want to create.

   The Print to File dialog box appears.
4. If you are creating object reference documentation, specify the exact object for which you want information in the first six fields. Use [Next Field] to move between the fields.
5. Move to the File field and type in the name of the file to which you want to write the documentation.

6. If you want to print the file, move to the Send to Printer check box and press [Select] to activate that option.
7. Press [Accept].

   SQL*Forms writes the documentation to the indicated file. If you have selected the Send to Printer option, SQL*Forms also sends the documentation to the system default printer.

Related Topics: Application documentation 27-1
Context 6-1

Executing a Form

Before you can execute, or run, a form, you must have the form in FRM format. (You can create an FRM file by generating the form.)

1. From the Action menu, select the Execute option.

   The Execute Form dialog box appears. The Name field defaults to the name of the current form (if a form is current).
2. In the Name field, indicate the name of the form you want to run.
3. Press [Accept].

   SQL*Forms (Design) invokes SQL*Forms (Run Form) and runs the indicated form. When you exit the form, you return to the SQL*Forms (Design) main menu.

Generating a Form

1. From the Action menu, select the Generate option.

   The Generate Form dialog box appears. The Name and File fields default to the name of the current form (if a form is current).
2. Indicate the name of the form you want to generate and press [Next Field].
3. Indicate the name of the file in which you want to save the generated form. Press [Accept].

   SQL*Forms (Design) generates the indicated form.

Note: If you want SQL*Forms to generate the current context form automatically before executing it from SQL*Forms (Design), turn on the Generate Before Executing Form option.

Related Topics: Generate Before Executing Form design option 26-6

Invoking Online Help

1. To receive context-sensitive help, press [Help] from any location within SQL*Forms (Design).

   To receive an online introduction to the help system, select the Help System option from the Help menu or press [Help] from any point in the help system.

2. Leave the help system by selecting Exit from the help system menu.

Related Topics: Help system 8-1

Loading a Form

1. From the Action menu, select the Load option.

   The Load Form dialog box appears.

   Note: If there is a current form, SQL*Forms will display an alert that you must acknowledge.

2. In the File field, type the name of the INP file that corresponds to the form you want to load. Press [Accept].

   The loaded form becomes the current form.

Related Topics: INP file format 24-3

Modifying a Field

1. In the Field Definition form or Field Definition spread table, make current the record that corresponds to the field you want to modify.
2. Make the changes you want and press [Accept] to accept those changes.

Modifying a Field from the screen Painter

When you modify a field, you might want to select the field while you are working in the screen painter. If so, you can modify a field by starting in the screen painter.

1. From the screen painter, move the cursor to the field that you want to modify. Press [Define Field].

   SQL*Forms displays the Field Definition form.

2. Make the changes you want and press [Accept] to accept those changes.

Opening a Form

1. From the Action menu, select the Open option.

   The Open Form dialog box appears.

   Note: If there is a current form, SQL*Forms might alert you that the form will be discarded.

2. In the Name field, type the name of the form you want to open. Press [Accept].

   SQL*Forms retrieves the specified form from the database and makes the form the current form. From this point, you can modify the form definition.

Quitting SQL*Forms (Design)

1. From the Action menu, select the Quit option.

   If you have made changes and have not saved them, an alert appears.

2. If an alert appears, select the appropriate option.

Saving a Form

1. From the Action menu, select the Save option.

   The Save Form dialog box appears. The Name field defaults to the name of the current form.

2. In the Name field, specify the name by which you want to save the form. Press [Accept].

   SQL*Forms saves the current form in the database.

Starting SQL*Forms (Design)

1. At the system prompt, type the following command

   `SQLFORMS` *`[options] [username/password]`*

   where:

`options`	Specifies any command line switches you want to activate for this SQL*Forms session.
`username/password`	Specifies your ORACLE username and password.

   The main menu appears.

   Note: If you do not specify your username and password at the system prompt, SQL*Forms displays the Login screen which prompts you for them.

Related Topics: Command line switches 23-8

Screen Painter Tasks

This section presents descriptions of some basic tasks that you perform in the screen painter. Specifically, this section covers the following tasks:

- selecting objects in the screen painter
- cutting and pasting objects
- undoing screen painter actions
- using constant text

Selecting Objects in the Screen Painter

By using [Select], you can indicate an object to be cut, pasted, or defined. Objects that can be selected include fields, screen lines, and areas. An area can be anywhere on the form as long as its boundaries do not cross a field and any fields it contains are in the current block.

Related Topics: [Select] 7-15

Selecting a Field

Position the cursor on the field and press [Select].

Selecting a Screen Line

1. Position the cursor at one end of the line and press [Select].
2. Position the cursor at the other end of the line and press [Select].

Selecting an Area

1. Position the cursor at a corner of the area and press [Select].
2. Position the cursor at the diagonally opposite corner of the area and press [Select].

Cutting and Pasting Objects

By using [Cut] and [Paste], you can rearrange a form by copying, moving, and deleting objects. Objects that can be cut and pasted include fields, lines, and areas. An area can be anywhere on the form as long as its boundaries do not cut across a field and any fields it contains are in the current block.

Cutting an Object

When you cut an object, you effectively delete it. However, SQL*Forms stores the most recently cut object in a buffer so that you can paste it somewhere else, as many times as you want.

1. Select the field, line, or area.
2. Press [cut].

Pasting an Object

You can always paste the object that is in the paste buffer, which is the object that you last cut during the current SQL*Forms session. You can paste objects anywhere on the form, including another page, as long as a pasted field fits on the page and does not overlap another field. Also, you can paste objects from one form to another.

1. Place the cursor where you want the upper left corner of the object to appear.
2. Press [Paste].

When you paste a field, SQL*Forms does not retain that field's sequence number. Rather, SQL*Forms assigns the pasted field the number after the highest sequence number already assigned in the current block.

Undoing Screen Painter Actions

Press [Undo] to negate your most recent screen painter action. Continue to press [Undo] to reverse successively earlier actions (unless you logged on with the-1 switch, which limits the UNDO buffer to hold only one operation).

[Undo] has no effect on work that you have done in the object definition forms or spread tables. [Undo] also does not affect constant text that you have typed on a form page.

Note that you can indirectly affect constant text by undoing a function that modified it, such as [Cut] or [Paste].

Related Topics: -1 switch, SQL*Forms (Design) 23-12

Using Constant Text

Constant text appears in a form every time it is run. This text can include field titles, directions, and graphics.

Entering Constant Text

To enter constant text, position the cursor where you want to place the text, and type. You can write constant text anywhere on a screen page except in a field. You can be in either insert or replace mode and any block can be the current block.

Editing Constant Text

To edit constant text, use [Delete Character] or [Delete Backward] to delete text. (You can also select and cut the text.) If you are working in replace mode, you can type over the existing text. If you are working in insert mode, you can insert text at the cursor position, moving existing text to the right.

Text Ownership

Constant text belongs to the page and not to a specific block or field. Therefore, you can write constant text in any part of the screen-no matter which block is current. When you delete a block, you do not delete the constant text that appears to be associated with it.

Drawing Lines and Boxes

Use lines and boxes in your forms to visually separate fields, blocks, or areas of constant text.

Drawing a Line

1. Select the endpoints of the line.

   Note: Since you can only draw vertical or horizontal lines, the endpoints that you select must be in the same line or column. If they are not on the same plane, you will create a box instead of a line.

2. Press [Draw Box/Line].

Drawing a Box

1. Select two diagonally opposite corners of the box.
2. Press [Draw Box/Line].

Related Topics: Constant text, block mode environment D-4

PART

II CONCEPTS

CHAPTER

3 FUNDAMENTAL PROCESSING

This chapter discusses the fundamental internal processing of SQL*Forms (Run Form). Specifically, it covers the following topics:

- navigation
- validation
- locking
- commit processing
- query processing
- function key processing
- field adjustment

Read this chapter for an overview of SQL*Forms (Run Form) processing. To fully understand the specifics of processing, you should review the flow charts in Chapter 21, "Event Processing." Each major section in the current chapter names the specific flow charts that you should look at for details.

This chapter does not cover trigger processing.

Related Topics: Event processing 21-1
Trigger processing 4-5

Navigation

In SQL*Forms (Run Form), navigation is performed primarily to move the cursor from one location to another. However, SQL*Forms also performs navigation during events when the cursor does not move (such as during commit promising).

Two major concepts of navigation are the navigation unit and the cursor.

Related Topics: Navigation, bit-mapped environment D-5

Navigation Unit

During a SQL*Forms (Run Form) session, SQL*Forms keeps track of an internal object called the navigation unit. The navigation unit is always defined as a field instance, a record, a block, the form, or as "outside the form." Think of the navigation unit as being defined at some point in the following hierarchy of object levels:

- outside the form
- form level
- block level
- record level
- field level

Navigation consists of changing the navigation unit step by step along the object hierarchy until the unit becomes the current navigation unit. SQL*Forms allows the navigation unit to change only in steps from one unit to the immediately higher or lower unit.

To "enter" a unit means to change the navigation unit from the next higher unit to that unit. To "leave" a unit means to change the navigation unit from that unit to the next higher unit.

Cursor

During a SQL*Forms (Run Form) session, SQL*Forms keeps track of an externally visible object called the cursor. SQL*Forms defines the cursor as an instance of a field, outside the form, or undefined.

outside the form	The cursor is outside the form when the form first starts execution and when the form is exited.
field instance	The cursor is a field instance most of the time that the form is running.
undefined	The cursor is undefined when it is defined as a field instance that gets removed from the block. When this occurs, SQL*Forms automatically redefines the cursor to the proper field instance.

Unlike the navigation unit, the cursor cannot be associated with any object other than a field. It is important to distinguish between the navigation unit and the cursor because navigation can occur without moving the cursor. Navigation independent of the cursor is necessary because there are processing events that operate above the field level which require SQL*Forms to navigate to a particular navigation unit. For example, commit processing acts on blocks and therefore requires SQL*Forms to navigate to a block unit before processing the block. When SQL*Forms navigates for this purpose, it is navigating to a navigation unit, but is not moving the cursor.

Navigation that does involve the cursor requires moving the cursor from its current, or initial position, and putting it at a target position.

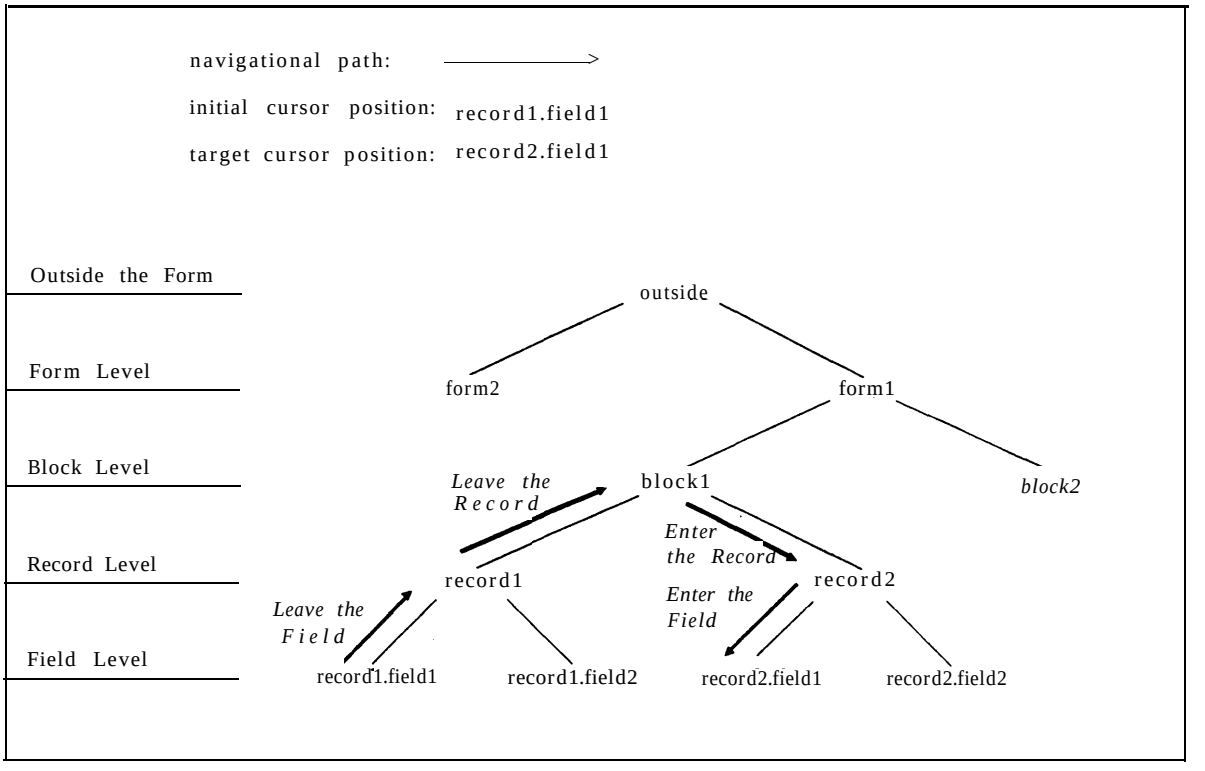
initial cursor position	The position of the cursor at the start of an event or a series of events. It is also called the current cursor position. The initial cursor position is the beginning point of a navigational event.
target cursor position	The position where the current event is directing the cursor. It is the ending point of a navigational event.

While the cursor must always be defined within a form as a field instance, the target unit might not be a field, but rather a record or a block. When the target unit is not specified as a field instance, SQL*Forms uses rules to determine which field instance within the target unit to move the cursor to. For example, if the target position of a navigational event is a particular record, SQL*Forms "puts the cursor at" the first enterable field in that record. Note that if the target unit is defined as outside the form, SQL*Forms exits the form.

Before SQL*Forms can put the cursor into a field instance, that field instance must be the current navigation unit. Before SQL*Forms can put the cursor outside the form, "outside the form" must be the current navigation unit.

Figure 3-1 illustrates an example navigational event. When an operator presses [Next Record] from record1.field1, SQL*Forms changes the navigation unit from record1.field1 to record1 to block1 to record2 to record2.field1. Once the navigation unit is record2.field1, SQL*Forms puts the cursor at record2.field1.

FIGURE 3-1
A Sample Navigational Path



The navigational path that the navigation unit takes is predefined for every initial unit and target unit. However, a processing failure (such as a trigger failure or a validation failure) can make SQL*Forms alter the intended navigational path. If some step of a navigational path cannot be completed, SQL*Forms changes the navigation unit and attempts to put the cursor within the new navigation unit.

The results of processing failures on navigation are covered in the flow charts in Chapter 21, "Event Processing."

Note: You cannot explicitly alter the navigational path by placing navigational commands in the triggers that fire during navigational events. These commands are restricted packaged procedures.

Related Topics: Navigate to the <Navigation Unit> Level 21-79
Put Cursor At 21-107

Events That Involve Navigation

Many SQL*Forms (Run Form) events involve navigation (such as the NEXT_FIELD and PREVIOUS_BLOCK events). Each navigational event has a predefine navigational path associated with it. You can refer to the event flow charts to identify each event and its implied path.

Note: The side effects of processing during an event can alter the navigational path.

Related Topics: Event flow charts 21-2

Navigation Failure

If navigation processing fails while attempting to complete a navigational path, SQL*Forms ties additional navigation to reach a consistent state. The specifics of navigation failure are covered in the flow charts in Chapter 21, "Event Processing."

Consider an example of navigation failure. The cursor is in record1.field1, and the operator presses [Next Field] to make the cursor move to record1.field2. The predefined navigational path involves moving the navigation unit to record1 and then to record1.field2. Suppose, however, that the Pre-Field trigger on record1.field2 fails during navigation processing. This trigger failure causes a failure in the Enter the Field event, leading to navigation failure processing. This processing moves the navigation unit back up to record1 and then to record1.field1.

SQL*Forms does not move the cursor until the new target cursor location (record1.field1) becomes the current navigation unit. In this example, the cursor does not move since the initial cursor location is the same as the target cursor location.

Related Topics: Event flow charts 21-2

Current Objects

Whenever a form is running, the cursor has a current location. That location is defined as the current object. Specifically, that location consists of the current block, record, and field that the cursor is in.

While there can be only one current block in a form at one time (and thus one current cursor record and cursor field), SQL*Forms does keep track-for each block-of what the current record would be if a non—current block in the form became current.

The current record for a non-current block is defined as the record that the cursor would attempt to move into if the cursor navigated to that block. This is the record that the cursor was in when the block was last current. If the block has not been current during the present SQL*Forms session, its current record is defined as the first enterable record and field in the block. The current field is defined as the first enterable field in that record.

What Navigation Is Not

Navigation is *not* what occurs when SQL*Forms "moves" internally to an object during a validation event. Instead, during validation events (Validate the Form, Validate the Block, and Validate the Record), SQL*Forms points internally to, or "looks at," each object within the object being validated and performs processing on it.

For example, during Validate the Record, SQL*Forms looks at each field in the record, one at a time, and initiates the Validate the Field event.

Validation

Validation is the process that determines if data entered in an application is valid. That is, it determines if the data meets the requirements defined by the application in order to be entered into the application or the database. These requirements range from checking data type requirements to ensuring that entered data adheres to any specified format mask requirements. The result of validation-whether or not a SQL*Forms object is valid or not-determines further SQL*Forms processing.

SQL*Forms has four validation procedures-one for each of the following objects:

- field
- record
- block
- form

Field validation consists of standard validation checks (such as range checking and fixed field length checking) and the firing of any Post-Change trigger and On-Validate-Field trigger. Field validation begins with a status check to determine whether the field is already valid. If the field is already valid, the validation process stops. If the field is not already valid, the validation process continues.

Record validation consists of validating all fields in the record followed by the firing of any On-Validate-Record trigger. Record validation begins, like field validation, with a status check to determine whether the record is already valid. If the record is already valid, the validation process stops. If the record is not already valid, the validation process continues.

Block validation consists of validating all records in the block. Form validation consists of validating all blocks in the form. Neither of these validation processes begins with a status check because blocks and forms do not have validation statuses.

Related Topics: Validate the Block 21-126
Validate the Field 21-127
Validate the Form 21-129
Validate the Record 21-130

Validation Unit

The validation unit is a form characteristic that you set (via the Validation Unit form characteristic). This characteristic determines the maximum amount of data that can be entered before SQL*Forms initiates validation. It also determines the largest object through which SQL*Forms can navigate before SQL*Forms initiates validation.

The validation unit corresponds to a unit of data in SQL*Forms. This unit can be the field, record, block, or form. The validation unit is most often the field, which means that SQL*Forms validates data in a field when SQL*Forms navigates out of that field.

When you set the validation unit above the field unit, the timing of validation changes. For example, if you define the validation unit as the block, SQL*Forms validates all new or changed data within the current block when the Leave the Block event occurs. In this example, SQL*Forms will not validate new or changed data in any field or record when the operator presses a function key that attempts to move the cursor from the current field to another field in the current record or in another record within the current block (i.e., a move *within* the validation unit).

Note: In a block mode environment, most applications are run with a validation unit of the block.

Related Topics: Setting the validation unit in block mode D-2
Validation Unit form characteristic 9-5

When Validation Occurs

SQL*Forms performs validation for the validation unit under the following conditions:

- when SQL*Forms navigates out of the validation unit, as a result of an operator pressing certain function keys or a trigger executing certain packaged procedures
- when the ENTER packaged procedure is invoked or the operator presses [Enter].

SQL*Forms performs validation for the form-no matter what the validation unit is-when the COMMIT function is invoked.

Validation does not occur in Enter Query mode. This behavior allows operators to enter query criteria that SQL*Forms would otherwise consider invalid (e.g., >100).

Related Topics: Enter Query mode 3-25

Status Checking

The first step in field and record validation is status checking. Status checking determines if SQL*Forms needs to perform any additional validation (such as standard validation checks and the firing of any Post-Change triggers in fields).

- If a field's status is New or Changed, validation continues. If a field's status is Valid, field validation stops.
- If a record's status is Changed, validation continues. If a record's status is New or Valid, record validation stops.

Field Validation States

SQL*Forms marks each field in a form as either New, Changed, or Valid.

When a Field Is New

When a record is created, SQL*Forms marks every field in that record as New. This is true even if SQL*Forms populates a field with a default or copied value (via the Default Value or Enforce Foreign Key field Characteristics).

When a Field Is Changed

SQL*Forms marks a field as Changed under the following conditions:

- When a field is altered, SQL*Forms immediately marks it as Changed. This is true whether an operator types a value into a field or a trigger selects, copies, or defaults a value into a field.
- When any field in a New record is altered, SQL*Forms marks all of the fields in the record as Changed.

Note: If the current value in a field is re-entered into that field, a character mode terminal registers a change and SQL*Forms marks the field as Changed. However, a block mode terminal cannot register such a change, and SQL*Forms would not change the field's status. A block mode terminal only registers a change if the entered value is different than the current value.

When a Field Is Valid

SQL*Forms marks a field as Valid under the following conditions:

- After SQL*Forms successfully validates a field it marks the field as Valid.
- If validation changes the value of the field being validated (through an On-Validate-Field, On-Validate-Record, or Post-Change trigger), and validation succeeds, SQL*Forms marks the field as Valid and does not re-validate the field.

  Note: This behavior is necessary to avoid validation loops. Be aware that it does allow you to commit an invalid value to the database.

- SQL*Forms marks as Valid all fields in records that are fetched from the database into the workspace. This is true even if Post-Change triggers (fired during Fetch Records) change field values.
- After SQL*Forms successfully commits data to the database, it sets all fields in the form to Valid. SQL*Forms does not validate any changes caused by triggers during the commit transaction.

  (Exceptions to this rule are explained in the section "Changing Data During Commit Processing.")

- Each field in a duplicated record (created by executing the DUPLICATE_RECORD packaged procedure or pressing [Duplicate Record]) inherits the field state of its source. Therefore, any field that is a duplicate of a Valid field will also be Valid.

  Note: A source field can only be New or Changed if the validation unit is the block or form, which would allow a field to be duplicated without being validated first.

Related Topics: Changing data, during commit processing 3-21
DUPLICATE_RECORD, packaged procedure 16-16

Record Validation States

SQL*Forms marks each record in a form as either New, Changed, or Valid. Note that a record's validation status is not equivalent to the record's status as recorded by the SYSTEM.RECORD_STATUS system variable.

Related Topics: SYSTEM.RECORD_STATUS 18-11

When a Record Is New

SQL*Forms marks a record as New when the record is first seated, before anything is changed. This is true even if SQL*Forms populates the record via the Default Value or Enforce Foreign Key field characteristic.

When a Record Is changed

SQL*Forms marks a record as Changed anytime that a field in the record is marked Changed.

When a Record Is Valid

SQL*Forms marks a record as Valid under the following conditions:

- After SQL*Forms successfully validates all of the Changed fields in a Changed record (changing their statuses to Valid), it marks the record as Valid. If all fields in a Changed record do not pass validation, the record remains as changed.
- After SQL*Forms successfully validates all of the New fields in a New record (changing their statuses to Valid), it marks the record as Valid. If not all of the fields in a New record pass validation, SQL*Forms marks the record as Changed.
- SQL*Forms marks all records fetched from the database into the workspace as Valid.
- After SQL*Forms successfully commits data to the database, it marks all records in the form as Valid. (Exceptions to this rule are explained in the section "Changing Data During Commit Processing.")
- A duplicated record (seated by executing the DUPLICATE_RECORD packaged procedure or by pressing [Duplicate Record]) inherits the record state of its source. Therefore, any record that is a duplicate of a Valid record will also be Valid.

  Note: A source record can only be New or Changed if the validation unit is the block or form, which would allow a record to be duplicated without being validated first.

Related Topics: Changing data, during commit processing 3-21
DUPLICATE_RECORD, packaged procedure 16-16

Standard Validation Checks

Validation of a New or Changed field includes standard validation checks. These checks assure that field data meets the requirements that you set via the following field characteristics:

- Format Mask
- Required
- Fixed Length
- Data Type
- Valid Range

Related Topics: Fields, characteristics 12-4
Validate the Field 21-127

Format Mask Status

If you specify a format mask for a field (via the Format Mask field characteristic), the standard validation checks involve checking the format mask status. SQL*Forms sets the format mask status whenever a value is entered into a field. The status can have one of two values:

Valid	This status indicates that the entered field value satisfies the specified format mask. Field validation continues.
Invalid	This status indicates that the entered field value does not satisfy the specified format mask. Field validation fails.

Related Topics: Enter the Value Into a Field 21-46
Format Mask field characteristic 12-15
Validate the Field 21-127

Locking

Locking tables and rows is an essential part of maintaining the consistency and integrity of your database. Locks prevent the inconsistencies that can arise when more than one user attempts to look at or change the same data at the same time.

SQL*Forms uses locks to ensure that no changes are lost when multiple users are updating the same data. When one user updates a row, SQL*Forms obtains locks to ensure that no other user tries to update it before the first user commits or rolls back his or her changes.

ORACLE and SQL*Forms automatically provide the basic provisions for data locking. However, designers and operators can exercise control over when and how resources are locked.

For a complete explanation of ORACLE locking methods and lock types, refer to the *ORACLE RDBMS Database Administrator's Guide. Also,* for a detailed account of the internal events of row locking, refer to the Lock the Row flow chart.

Note This section assumes that you are running the ORACLE RDBMS Version 6.0 using row-level locking. If you are running ORACLE Version 6.0 with table locking, consult the *ORACLE RDBMS Database Administrator's Guide* for information on table locks.

Related Topics: Lock the Row 21-73

Types of Locks

The *ORACLE RDBMS Database Administrator's Guide* extensively discusses the types of locks that ORACLE uses. The SQL*Forms *Designer's Reference* only covers the most common types of locks encountered in SQL*Forms.

Note that the types of row and table locks that SQL*Forms uses depend on whether events are invoked by operators or by triggers.

Exclusive Row Lock

An exclusive row lock prevents other operators from updating or deleting a row and from acquiring any locks on that row. However, other users can still query a row that is exclusively locked.

Exclusive Table Lock

An exclusive table lock prevents other operators from updating or deleting any row in that table and from acquiring any locks on that table. However, other users can still query a table that is exclusively locked.

Locks Caused by Operators

SQL*Forms attempts to place an exclusive lock on a row as a result of an operator changing a database field in a record that corresponds to a row in the database or as a result of the operator explicitly requesting a lock on such a record.

Locks Caused by Triggers

When a trigger makes a change to a table through a Data Manipulation Language (DML) command, the database attempts to place an exclusive lock on the affected row.

SQL*Forms also attempts to place an exclusive lock on each row corresponding to a fetched record when a trigger uses the following packaged procedures:

- DELETE_RECORD
- ENTER_QUERY (with the FOR_UPDATE parameter)
- EXECUTE_QUERY (with the FOR_UPDATE parameter)
- LOCK_RECORD

Methods of Locking Rows

Implicit locking and explicit locking are the two methods that SQL*Forms uses to guarantee that rows are locked during an update or delete. These methods of locking perform identical functions, but they differ as to when they perform these functions.

Implicit Locking

All SQL*Forms applications use implicit locking, which means that operators never have to manually lock rows.

Through implicit locking, SQL*Forms tries to place an exclusive lock on a row as soon as the operator attempts to type a change into a field in the associated record. On block mode terminals, SQL*Forms attempts to place the lock as soon as the operator presses a function key.

When the validation unit is the field, SQL*Forms does not allow the change if it cannot acquire the lock.

When the validation unit is the record, block, or form, SQL*Forms allows the edit and attempts to place an exclusive lock on the row during record validation.

Related Topics: Check the Field for Edit 21-15
Lock the Row 21-73
Validate the Record 21-130

Explicit Locking

Explicit locking means an operator or a trigger requests that SQL*Forms lock a particular row immediately. An operator can use [Lock Record] to explicitly lock the rows associated with records in a block.

You can cause SQL*Forms to explicitly lock rows associated with records in a block by designing a form that uses any of the following packaged procedures:

- ENTER_QUERY (with the FOR_UPDATE parameter)
- EXECUTE_QUERY (with the FOR_UPDATE parameter)
- LOCK_RECORD

You can use these packaged procedures anytime you want to ensure locking of all rows involved in a query, including those rows that the operator has not changed. You can use them in addition to other methods of locking.

Of course, you can also use SQL to explicitly lock rows and tables.

Related Topics: ENTER_QUERY, packaged procedure 16-17
EXECUTE_QUERY, packaged procedure 16-18
LOCK_RECORD, packaged procedure 16-23
Lock the Row 21-73

Failing to Obtain a Lock

SQL*Forms can fail to obtain a lock on a row or a table if there is resource contention among users.

Failing to Obtain a Row Lock

When there is resource contention on a multi-user system, SQL*Forms might not be able to obtain a lock on a row immediately. If this is the Case, SQL*Forms displays the message, `Attempting to reserve record for update or delete (CTRL-C to cancel) . . . .` The operator can choose to wait for SQL*Forms to acquire the lock or terminate the locking procedure.

The specific results of failing to obtain a row lock depend on when SQL*Forms attempts the locking (i.e., during what processing event) and are covered in Chapter 21, "Event Processing."

Related Topics: Lock the Row 21-73

Failing to Obtain a Table Lock

When there is resource contention on a multi-user system, SQL*Forms might not be able to obtain a requested lock on a table immediately. If this is the case, SQL*Forms waits for the table to be available without displaying any message. The operator can wait for SQL*Forms to acquire the lock or terminate the locking procedure by pressing CTRL-C, or its equivalent

Failing to obtain a table lock has serious implications. If one user has a table exclusively locked, other users can only *query* records associated with that table. This condition persits, without notice, until the resource becomes available. Therefore, Oracle Corporation strongly recommends that you do not use exclusive table locks in multi-user environments unless it is absolutely necessary.

When Locks Are Released

SQL*Forms releases all locks under the following circumstances:

- The form commits successfully.
- The form aborts.
- SQL*Forms performs a full SQL*Forms rollback-not a rollback to a savepoint. (This might occur after an attempted commit fails.)

SQL*Forms maintains locks when it rolls back to a savepoint, as might happen after an unsuccessful commit. This behavior allows the operator to make any necessary changes and re-commit.

If a trigger explicitly locked a batch of rows via the FOR_UPDATE parameter, those rows need to be re-queried after a commit to re-lock all of the rows.

Note:

- Clearing a record or clearing a block does not release the locks on any corresponding rows. This behavior differs from that of SQL*Forms versions that run on Version 5.1 (or earlier) of the ORACLE RDBMS.
- If you use a ROLLBACK or COMMIT statement in a PL/SQL anonymous block or a form-level procedure, SQL*Forms releases all locks. SQL*Forms interprets a ROLLBACK statement as a CLEAR_FORM packaged procedure with no parameters. It interprets a COMMIT statement as a COMMIT_FORM packaged procedure.
- SQL*Forms does not support a "raw" commit (one which does not include the normal SQL*Forms commit processing). For example, SQL*Forms does not support a commit from an ORACLE Precompiler user exit.

Related Topics: COMMIT_FORM, flow chart 21-23

Altering the Default Locking

You can alter the processing that SQL*Forms performs when locking resources. The On-Lock trigger replaces standard SQL*Forms processing.

Related Topics: On-Lock trigger 1415
On-Insert trigger 1415
On-Update trigger 1418
Post and Commit Transactions 21-92

Locking Recommendations

Deadlocks occur when one user needs a resource that a second user has locked and the second user needs a resource that the first user has locked. In this case, neither user can proceed and ORACLE automatically rolls back the work of one of the users. You can help prevent deadlocks by following these guidelines:

- Do not use an exclusive table lock unless it is absolutely necessary.
- Monitor those applications that do exclusively lock tables to ensure that they lock tables in the same sequence. The risk of a deadlock increases if one form locks the ORD table first and the ITEMS table second, and another form locks them in reverse order.
- Instruct operators to commit their work frequently, thereby releasing any held locks. Alternatively, design your forms to automatically commit changes at specific points.

Commit Processing

SQL*Forms commit processing is the event that attempts to make the data in the database identical to the data in a form. This process involves writing, or "posting," data to the database and then committing the data in the database. Specifically, this process:

- deletes rows in database tables that correspond to records that an operator or trigger deleted in the form
- updates rows in database tables that correspond to records in the form for which an operator or trigger changed database field values
- inserts rows in database tables that correspond to records (with database fields) that an operator or trigger inserted in the form
- commits the data to the database

All changes are permanent (unless changed in a future transaction) and are reflected in future queries of all ORACLE users.

A detailed account of the internal events of commit processing appears in the COMMIT_FORM flow chart and the Post and Commit Transactions flow chart in Chapter 21, "Event Processing."

Related Topics: COMMIT_FORM, flow chart 21-23
Post and Commit Transactions 21-92

Posting Data

Posting consists of writing deletes, inserts, and updates in the form to the database, but not committing these transactions to the database.

Posting always occurs during commit processing. However, posting can occur separately from a commit via the POST packaged procedure. (There is no SQL*Forms (Run Form) function key that corresponds to this packaged procedure.)

Any data that you post to the database is committed in the database by the next COMMIT_FORM that executes during the current SQL*Forms (Run Form) session. This data can also be rolled back via the CLEAR_FORM packaged procedure.

Related Topics: Post and Commit Transactions 21-92
POST, packaged procedure 16-26

Savepoints and Rollbacks

Within SQL*Forms, a rollback is an internal function that undoes operations in the current transaction. To regulate rollbacks, SQL*Forms can divide a transaction into segments with savepoints.

SQL*Forms can issue savepoints immediately before posting data and when calling a form.

Savepoints Issued when Posting

Whenever the Post and Commit Transactions event occurs, SQL*Forms issues a savepoint for that event. If there is an error during the event, SQL*Forms rolls back the transaction to that event's savepoint. In effect, this rollback undoes any changes that were made during the Post and Commit Transactions event.

Note: The Post and Commit Transactions event occurs during the following events:

- CLEAR_BLOCK
- CLEAR_FORM
- COMMIT_FORM
- COUNT_QUERY
- ENTER_QUERY
- Execute the Query
- POST

Related Topics: Post and Commit Transactions 21-92

Savepoints Issued when Calling a Form

Whenever processing calls a form (via the CALL or CALL_QUERY functions), SQL*Forms issues a savepoint for the called form. If a CLEAR_FORM TO_SAVEPOINT packaged procedure is processed when the called form is the current form, SQL*Forms rolls back all uncommitted changes to that form's savepoint. In effect, this rollback undoes any uncommitted changes in the current form and in any forms that the current form has called during the SQL*Forms (Run Form) session.

For example, assume that you post changes in form A, call form B, and then post changes in form B. At this point, a CLEAR_FORM TO_SAVEPOINT rolls back all of the changes posted in form B. The savepoint that SQL*Forms issued when it called form B stops the rollback. However, if you post changes in form A and in form B, and then return to form A, a CLEAR_FORM TO_SAVEPOINT rolls back all posted changes in both forms (and any unposted changes in form A). Note that the effective savepoint for the first form in a call stack of forms is the beginning of the SQL*Forms (Run Form) session.

Note: Whenever a CLEAR_FORM FULL_ROLLBACK is processed, SQL*Forms rolls back all uncommited changes throughout the retire application.

Related Topics: CALL, flow chart 21-10
CALL_QUERY, flow chart 21-12
CLEAR_FORM, packaged procedure l6-9

Processing Inserts, Updates, and Deletes

When data is posted to the database, SQL*Forms processes inserts, updates, and deletes for all blocks in a form. Inserts, updates, and deletes are pending transactions that involve any records which have been marked for insertion, update, or deletion since the last post during the current SQL*Forms (Run Form) session.

Inserts, updates, and deletes might not be processed if you alter the default SQL*Forms commit processing.

Related Topics: Commit processing, altering the default behavior 3-22

Inserts

In SQL*Forms, an insert is the pending transaction of a record that has been marked for insertion as a new row in the database. Every record that is marked as an insert has a SQL INSERT statement associated with it that is executed when the insert is posted. The INSERT statement takes the following form

```
INSERT INTO table [(column, column, . . .)1
    VALUES ( value, value, . . .) ;
```

where:

table	Specifies the name of the base table for the current block that SQL*Forms is processing.
column (s)	Specifies the columns in the specified base table that correspond to the database fields in the record that SQL*Forms is processing.
value (s)	Specifies the values that SQL*Forms will insert into the specified columns in the base table.

Updates

In SQL*Forms, an update is the pending transaction of a record that has been marked as an update to an existing row in the database. Every record that is marked as an update has a SQL UPDATE statement associated with it that is executed when the update is posted. The UPDATE statement takes the following form:

```
UPDATE table
    SET (column = value, column = value, . . .)
        WHERE ROWID = rowid_value;
```

where:

table	Specifies the name of the base table for the current block that SQL*Forms is processing.
column (s)	Specifies all columns in the specified base table that correspond to the database fields in the record that SQL*Forms is processing.
value (s)	Specifies the values that SQL*Forms will insert into the specified columns in the base table.
rowid_value	Specifies the ROWID value for the row in the database that corresponds with the record that SQL*Forms is processing.

Notice that the condition specified in the WHERE clause only specifies a ROWID value. This condition identifies the one correct row in the database that the database should update.

Note: If an operator does not have update privileges on a column and the Column Sec block characteristic is turned on, SQL*Forms does not include the column in the block's UPDATE statement. Therefore, the block's UPDATE statement can change from operator to operator. However, the statement remains the same for the duration of the operator's SQL*Forms (Run Form) session.

Related Topics: Column Sec block characteristic 11-9

Deletes

In SQL*Forms, a delete is the pending transaction of a record that has been marked for the deletion of its corresponding row in the database. Every record that is marked as a delete has a SQL DELETE statement associated with it that is executed when the delete is posted. The DELETE statement takes the following form:

```
DELETE FROM table
    WHERE ROWID = rowid_value;
```

where:

table	Specifies the name of the base table for the current block that SQL*Forms is processing.
rowid_value	Specifies the ROWID value for the row in the database that corresponds with the record that SQL*Forms is processing.

Notice that the condition specified in the WHERE clause only specifies a ROWID value. This condition identifies the one correct row in the database that the database should delete.

When Commit Processing Occurs

Commit processing occurs under the following conditions:

- when a trigger invokes the COMMIT_FORM packaged procedure
- when an operator presses [Commit]
- when there are changes to database fields in the current block since the last commit *and* a trigger invokes the CLEAR_BLOCK packaged procedure (with the DO_COMMIT parameter)
- when there are changes to database fields in any block in the current form since the last commit *and* a trigger invokes the CLEAR_FORM packaged procedure (with the DO_COMMIT parameter)
- when an operator answers `Yes` to the `Do you want to commit the changes you have made?` alert

The Do you want to commit the changes you have made? alert appears under the following conditions:

- Changes have been made to database fields in the current block since the last commit and any of the following events occur:
  - CLEAR_BLOCK (with the ASK_COMMIT parameter or with no parameter)
  - COUNT_QUERY
  - ENTER_QUERY
  - Execute the Query
- Changes have been made to database fields in any block in the current form since the last commit *and* any of the following events occur:
  - CLEAR_FORM (with the ASK_COMMIT parameter or with no parameter)
  - Leave the Form

Note: When a PL/SQL block issues a database commit from within SQL*Forms (via the SQL COMMIT statement), SQL*Forms commit processing occurs--just as if the COMMIT_FORM packaged procedure had been invoked.

Related Topics: CLEAR_BLOCK, flow chart 21-18
CLEAR_FORM, flow chart 21-21
COMMIT_FORM, flow chart 21-23
COUNT_QUERY, flow chart 21-26
ENTER_QUERY, flow chart 21-47
Execute the Query 21-49
Leave the Form 21-67
Post and Commit Transactions 21-92

Changing Data during Commit Processing

Commit processing performs validation and fires a number of triggers. As a result, a commit event can generate changes to database fields. Depending on the type of trigger you use to change data during a commit, SQL*Forms might attempt to commit those changes to the database during the current commit event.

Related Topics: Post and Commit Transactions 21-92

Changing Data before Posting

Be cautious if you use Pre- Delete, Insert, and Update triggers to make changes to database fields. Database changes made in these triggers might be committed without being validated.

For example, assume that a Pre-Insert trigger selects a value into a database field in the form:

- If the affected record has not already been processed for updates, the changes will be committed during the current commit event but they will not be validated before being committed.
- If the affected record has already been processed for updates, the changes will not be committed during the current commit event but they will cause SQL*Forms to mark the affected fields and records as Changed. If there is an attempt to update such a record before it is committed, an error occurs.

Changing Data after Posting

Be cautious if you use Post-Commit, Delete, Insert, and Update triggers to make changes to database fields. Database changes made in these triggers might not be committed during the current commit event.

For example, assume that a Post-Insert trigger selects a value into a database field in the form:

- If the affected record is in the block that the current commit is processing currently or a block that the current commit already processed, SQL*Forms does not commit the changes during the current commit event.

  However, SQL*Forms does mark the affected fields and records as Changed. If the current commit succeeds, SQL*Forms marks all fields and records as Valid. This results in an error during subsequent processing. Therefore you should avoid making such changes.

- If the affected record is in a block that has not been processed yet by the current commit, SQL*Forms validates the changes and they are committed during the current commit event.

Altering the Default Commit Processing

You can alter the processing that SQL*Forms performs while posting data during a commit. The On-Insert, On-Update, and On-Delete triggers all replace standard SQL*Forms processing.

Related Topics: On-Delete trigger 14-14
On-Insert trigger 14-15
On-Update trigger 14-18
Post and Commit Transactions 21-92

ROWID

Whenever you create a table, ORACLE automatically creates a column called ROWID in that table. The ROWID column contains a value for each row that uniquely identifies that row. Similarly, when you create a block in a form, SQL*Forms adds a non-displayed, non-updatable field in the block named ROWID. (This field also does not display in the design interface.)

SQL*Forms uses ROWID values to establish a one-to-one correspondence between rows in a database table and records in a base table block. In this way, when SQL*Forms wants to work on a row in the database based on some action in a form it uses a ROWID value to identify the specific row that it wants to affect.

Specifically, SQL*Forms uses ROWID to determine:

- what rows to fetch from the database or reserve in the database
- what rows to update or delete during posting

When SQL*Forms retrieves a row into a block, it retrieves the value of the ROWID column into the ROWID field. This value is valid from the time a row is locked until a successful commit (when locks are released). A ROWID value might be invalid for a row that has not been locked due to the results of transactions by other users. Note that the value of ROWID in a record is NULL under the following circumstances:

- the record does not correspond to a row in the database (i.e., a New record or a record with no database fields)
- the record corresponds to a row in a non-updatable table or view

Note: In the case where the record corresponds to a non-updatable table or view, a record can still be fetched into a block, but the fetched records cannot be reserved or changed in the database.

Once a row is deleted from the database, ORACLE can reassign that row's ROWID value to a newly inserted row.

Using ROWID in Triggers

In most cases you do not need to be concerned with ROWID values-they exist to help SQL*Forms manage transactions. However, you might want to use ROWID values to update a table that is not associated with a block in the current form. In such a situation, you can use ROWID values to keep track of which table rows correspond to which form records.

For example, assume that you have a form that uses EMP as a base table and that you want to record all changes made to the EMP table through this form in a table called EMP_HISTORY. You could create a Pre-Update trigger and a Pre-Delete trigger (for the block based on the EMP table) that contained the following statement:

```
INSERT INTO EMP_HISTORY
    SELECT * FROM EMP
        WHERE ROWID = :EMP.ROWID;
```

Whenever record updates or deletes are posted for the block, SQL*Forms adds a row to EMP_HISTORY. The specific row that it adds is chosen by the SELECT clause in the Pre-Update or Pre-Delete trigger. It is the row of EMP whose ROWID value equals the value in the current instance of the ROWID field in the current block (i.e., the row of EMP that is being updated or deleted).

You also might want to display the value of ROWID while debugging a form. To do so, use a Post-Query trigger to select the column or field value into a displayed field. This displayed field should have a field length of at least 26 characters to hold the entire ROWID value.

Note: Do not allow operators or a trigger to change the ROWID value for any row. Invalid ROWID values will interfere with the posting of data to the database.

Query Processing

"Query processing" refers to the SQL*Forms events that determine how rows are retrieved from the database and displayed as records in a block. Query processing covers the following major events:

- entering a query
- executing a query
- counting query hits

While performing query processing, SQL*Forms operates with special restrictions.

Entering a Query

"Entering a query" refers to how an operator specifies conditions for a query's retrieval within a particular block. To enter a query, a trigger or the operator must invoke the ENTER_QUERY function. This action puts SQL*Forms into Enter Query mode.

Note: For a detailed account of the internal events of entering a query, refer to the ENTER_QUERY flow chart in Chapter 21, "Event Processing."

Related Topics: ENTER_QUERY, flow chart 21-47
ENTER_QUERY, packaged procedure 16-17

Enter Query Mode

In Enter Query mode, SQL*Forms accepts query conditions and establishes the query SELECT statement. SQL*Forms is in Enter Query mode from the time the ENTER_QUERY function is invoked until an operator presses [Exit/Cancel] *or* presses [Enter] or [Execute Query] and the query successfully returns at least one row.

SQL*Forms processing while in Enter Query mode enforces special restrictions.

Note: When SQL*Forms (Run Form) is not in Enter Query mode or some other special mode, it is in "normal" mode.

Related Topics: Enter Query mode, restrictions 3-29

Executing a Query

"Executing a query" refers to how an operator or a trigger opens a query and attempts to fetch selected records into the block. To execute a query, a trigger or the operator must invoke the EXECUTE_QUERY function.

Related Topics: EXECUTE_QUERY, packaged procedure 16-18
Execute the Query 21-49

Opening a Query

SQL*Forms opens a query when the EXECUTE_QUERY function is invoked. "Opening a query" performs the following actions:

- identifies the rows in the database that match the selection criteria
- takes a "snapshot" of the selected rows so that a later fetch retrieves all rows from the database as of the same time as the current fetch
- attempts to lock all of the matching rows if an ENTER_QUERY(FOR_UPDATE) or EXECUTE_QUERY (FOR_UPDATE) command has been specified
- establishes the number of rows that SQL*Forms can move into the record buffer at one time
- invokes the fetching of the appropriate records

A query remains open until SQL*Forms explicitly closes it. SQL*Forms closes a query under the following conditions:

- there are no more rows to fetch
- the ABORT_QUERY function is invoked
- the block is cleared

Note: For a detailed account of the internal events of executing a query, refer to the Execute the Query flowchart in Chapter 21, "Event Processing."

Related Topics: Execute the Query 2149
Fetch Records 21-53
Open the Query 21-88

Fetching Records

Fetching records is the part of executing a query that performs the following actions:

- brings selected rows into the record buffer
- creates records in the block and populates these records with database values from the corresponding rows
- populates any non-database fields through Post-Change or Post-Query triggers

Note: For a detailed account of the internal events of fetching a record, refer to the Fetch Records flow chart in Chapter 21, "Event Processing."

Related Topics: Fetch Records 21-53

Record Selection Criteria

Executing a query issues a SELECT statement to the database that specifies the selection criteria for fetching records into a form. That SELECT statement is the query SELECT statement.

You can affect the record selection criteria with block definitions and with triggers. Operators can affect the record selection criteria by specifying conditions while in Enter Query mode.

Query SELECT Statement

The query SELECT statement, which specifies the selection criteria for fetching records into a block, has the following form:

```
SELECT select_list FROM table
    [ WHERE
        [ default_where_clause [AND]]
        [column_condition [AND]] . . .
        [(query_where_condition)]]
    [ORDER BY default_ordering I ORDER BY query_where_ordering]
```

where:

select_list	Specifies the column names that correspond to all of the database fields, and ROWID.
default_where_clause	Specifies the WHERE condition set by the block's default WHERE clause.
column_condition	Specifies the value of an individual field as specified in the example record.
query_where_condition	Specifies the WHERE condition set by the operator in the Query Where dialog box.
default_ordering	Specifies the ORDER BY condition set by the block's default ORDER BY clause.
query_where_ordering	Specifies the ordering condition set by the operator in the Query Where dialog box.

By default, the query SELECT statement retrieves all appropriate records for the block, in an unspecified order. That is, the default query SELECT statement is `SELECT select_list FROM table.` However, both designers and operators can modify this SELECT statement by indirectly specifying its parameters.

The following table indicates the parameters that can be modified, the elements that correspond to the parameters (i.e., what the designer or operator must change in order to affect the SELECT statement), and who can modify the parameters:

Parameter	*Element*	*Modifier*
column_ condition	example record	designer, operator
default_ ordering	default ORDER BY clause	designer
default_where_clause	default WHERE clause	designer
query_where_condition	Query Where statement	operator
query_where_ordering	Query Where statement	operator

Whenever SQL*Forms executes a query SELECT statement, it stores the text of the statement in the SYSTEM.LAST_QUERY system variable.

Related Topics: SYSTEM.LAST_QUERY 18-10

Default ORDER BY Clause

The query SELECT statement uses as its ORDER BY clause any default ORDER BY clause that you specified in the Default Where / Order By characteristic for the block. If the operator specifies an ORDER BY clause in the Query Where statement, it overrides the default ORDER BY clause.

Default WHERE Clause

The query SELECT statement uses as its WHERE clause any default WHERE clause that you specified in the Default Where / Order By characteristic for the block. SQL*Forms applies the default WHERE clause to every query in a block. If the example record or Query Where statement also specifies WHERE conditions, SQL*Forms applies those conditions in addition to the default WHERE clause.

Example Record

The example record is a SQL*Forms record that is not stored in the database and is only used to specify query conditions. SQL*Forms creates and initializes an example record whenever the ENTER_QUERY, EXECUTE_QUERY, or COUNT_QUERY function is invoked.

When an example record is created, it contains the same fields as any record in the current block. Each field of the example record has a NULL value unless the field is populated through the Enforce Foreign Key field characteristic.

The example record can affect the query SELECT statement in the following ways:

- The example record can supply (non-NULL) values for database and non-database fields that are referenced in the WHERE clause.
- The example record can specify conditions for database fields. For example, it can specify that the query only retrieves records where the value in the QUANTITY field is less than 100.

Operators can enter field values directly into the example record while in Enter Query mode. Designers can enter field values through the Enforce Foreign Key field characteristic and the Pre-Query trigger. Note that changes made to the example record by the Pre-Query trigger override those values specified by an operator. This feature allows you to screen illegal queries.

Query Where Statement

The query SELECT statement:

- uses as its WHERE clause any default WHERE clause that you specified in the Default Where/Order By characteristic for the block
- uses as its ORDER BY clause any default ORDER BY clause that you specified in the Default Where/Order By characteristic for the block

However, the WHERE conditions specified by the Query Where statement supplement the conditions in the default WHERE clause, and the ORDER BY conditions specified in the Query Where statement replace the conditions in the default ORDER BY clause.

Only operators can specify the Query Where statement. If an operator is entering a query and specifies a variable in one or more database fields, when the operator presses [Execute Query] the Query Where dialog box appears. In the dialog box, the operator enters the WHERE and ORDER BY conditions.

Note: For more information on complex queries, refer to the *SQL*Forms Operator's Guide*.

Query Processing Restrictions

SQL*Forms operates with a number of restrictions when it is processing query-related events. Specifically, SQL*Forms performs differently while in Enter Query mode and while fetching records.

Enter Query Mode Restrictions

When in Enter Query mode, SQL*Forms follows a modified set of processing rules. Specifically, SQL*Forms:

- does not perform any validation
- does not fire any triggers
- uses standard function key definitions (not those defined in key triggers)
- disables any function keys related to inter-record navigation

While in Enter Query mode, an operator cannot use the following function keys:

[Block Menu]	[Next Block]
[Clear Form/Rollback]	[Next Record]
[Commit]	[Next Set of Records]
[Down]	[Previous Block]
[Duplicate Field]	[Previous Record]
[Duplicate Record]	[scroll Down]
[Insert Record]	[scroll Up]
[Lock Record]	[Up]

Fetch Processing Restrictions

When SQL*Forms is populating a record with fetched values, it processes any Post-Change triggers defined as V2 triggers with the following restrictions:

- SQL*Forms ignores any attempt to change a database field in the record.
- SQL*Forms does not execute any SELECT statement that only affects database fields in the record.
- SQL*Forms does not execute any SELECT statement that does not have an INTO clause.

If SQL*Forms does not execute a SELECT statement in a V2 trigger step, it treats the trigger step as if the step succeeded (even if the Reverse Return Code trigger step characteristic is turned on).

During fetch processing, Post-Change triggers defined as V3 triggers do not operate with these restrictions. Despite the trigger style, during a record fetch SQL*Forms does not perform any validation checks, but it does mark the record and all of its fields as Valid after firing the Post-Change trigger for each field.

Note: The Post-Query trigger does not have the restrictions of the Post-Change trigger. Therefore, you can use the Post-Query trigger to make changes to the fetched database values. Such changes mark the corresponding fields and records as Changed.

Related Topics: Fetch Records 21-53

Counting Query Hits

"Counting query hits" refers to how SQL*Forms counts the rows in the database that match the current selection conditions. In order to count query hits, the application or the operator must invoke the COUNT_QUERY function.

When COUNT_QUERY is invoked, SQL*Forms uses the selection criteria to select matching rows from the block's base table. Specifically, SQL*Forms issues a query SELECT statement similar to the one issued during the Execute the Query event. However, for counting query hits, the SELECT statement substitutes the COUNT(*) function for the `select_list` parameter and does not issue any ORDER BY clause.

The results of counting a query display on the SQL*Forms (Run Form) status line.

Related Topics: COUNT_QUERY, flow chart 21-26
COUNT_QUERY, packaged procedure 16-11

Function Key Processing

Every SQL*Forms (Run Form) function key has a corresponding SQL*Forms function. When a form is in normal mode and the function key has not been redefined, pressing a function key invokes the corresponding function. For example, when an operator presses [Next Field], SQL*Forms processes the NEXT_FIELD function.

However, the actual processing might be different under the following conditions:

- the form is paused
- a list of values is active
- the cursor is in a prompt field
- the form is in Enter Query mode
- the function key is illegal in the current context
- there is an applicable key trigger defined
- there is an applicable Key-Others trigger defined

Related Topics: Process the Function Key 21-105

Legal SQL*Forms (Run Form) Function Keys

All function keys are legal in normal mode. However, certain function keys are not legal in Enter Query mode, in Edit mode, in an alert field, during a pause, or in a list of values. The *SQL*Forms Operator's Guide* details the contexts in which each function key is legal. Also, SQL*Forms (Run Form) displays a context-sensitive list of legal function keys when an operator presses [Show Keys].

Note: While some function keys are legal in more than one context, the actual processing that results from pressing a function key can differ among contexts.

Field Adjustment

During processing, SQL*Forms often needs to get, or reference, the value of a field. For example, when validating a field, SQL*Forms needs to "get" the field's value before it can validate that value. Similarly, when you reference a field's value in a trigger statement (e.g., :ORD.ORDID:=:ITEM.ORDID), SQL*Forms needs to get the field's value.

Whenever SQL*Forms must get a field value, it first adjusts the field to meet internal requirements. Specifically, SQL*Forms might perform the following actions on the field and its value

- remove trailing blank spaces from the field value
- change the field's value to NULL if it consists of all blank spaces
- apply the field's format mask
- remove leading zeros if the field's data type is INT, MONEY, NUMBER, RINT, RMONEY, or RNUMBER-unless the field's format mask permits leading zeros
- assign the first character space of the field as the field's current cursor position

CHAPTER

4 TRIGGER CONCEPTS

A trigger is a piece of code that is executed at, or "triggered" by, a SQL*Forms (Run Form) event. There are many types of triggers and many uses for them. Generally, you can use triggers to perform the following tasks:

- perform data validation, data entry, and data deletion
- control the logical flow of an application

This chapter introduces the basic concepts and terminology of SQL*Forms triggers. Specifically, it covers the following topics:

- V3 triggers versus V2 triggers
- trigger definition level
- trigger scope
- trigger processing

Related Topics: Triggers, types 14-8

V3 Triggers versus V2 Triggers

In Version 3.0 of SQL*Forms, there are two styles of triggers:

V3 trigger style	This is the "new" trigger style. This style, which uses the PL/SQL programmingg language, is available in Version 3.0 of SQL*Forms and in all subsequent versions. The V3 style allows you to design triggers with more flexibility and power by taking advantage of the capabilities of PL/SQL.
V2 trigger style	This is the "old" trigger style. This style, which uses SQL and SQL*Forms commands in trigger steps, is the only style of trigger that was available prior to Version 3.0 of SQL*Forms.

Oracle Corporation strongly recommends that you design your applications with V3 triggers. V2 triggers are only available for compatibility with previous versions of SQL*Forms.

If you have applications that you built with previous versions of SQL*Forms and that therefore use V2 triggers-Oracle Corporation recommends that you convert your triggers into the V3 style. When you convert V2 triggers to the V3 style, you experience the following benefits:

- The code is clearer than the corresponding V2 code.
- The performance of your applications might improve.
- The future maintenance and enhancement of your applications is easier.

Related Topics: Translating triggers into PL/SQL B-3
V2 triggers F-1

Trigger Definition Level

To employ a trigger in a form, you must attach it to an object in the form. Specifically, you can attach a trigger to a specific field, block, or form.

The trigger definition level refers to the object levels to which you can attach a particular type of trigger. For example, you can define a Next-Field trigger at the form, block, or field level. If you define a Next-Field trigger at the block level, its definition level is the block.

The field is the lowest definition level, the block is higher, and the form is highest. Where you define a trigger in this definition level hierarchy directly affects the trigger's scope.

Related Topics: Triggers, block mode considerations D-3

Trigger Scope

The scope of a trigger is the domain or range in which it operates. Any trigger that is part of a form has a scope. A trigger's scope is always defined as the definition level and below.

For example, assume that you want to define a Post-Field trigger. Provided that the validation unit is the field, this trigger fires whenever the Leave the Field event occurs.

- If you define the Post-Field trigger at the field level, that specific trigger fires whenever Leave the Field occurs for the field to which the trigger is attached.
- If you define the Post-Field trigger at the block level, that specific trigger fires every time Leave the Field occurs for any field within the block to which the trigger is attached.

When you define a particular trigger type at more than one level, the trigger with the lowest applicable definition level takes precedence. Figure 4-1 illustrates this concept. The arrows indicate the objects to which triggers are attached. The trigger descriptions explain how the scoping rules apply.

Pre-Block (form level)
Works on the ITEMS block but not on the ORDERS block

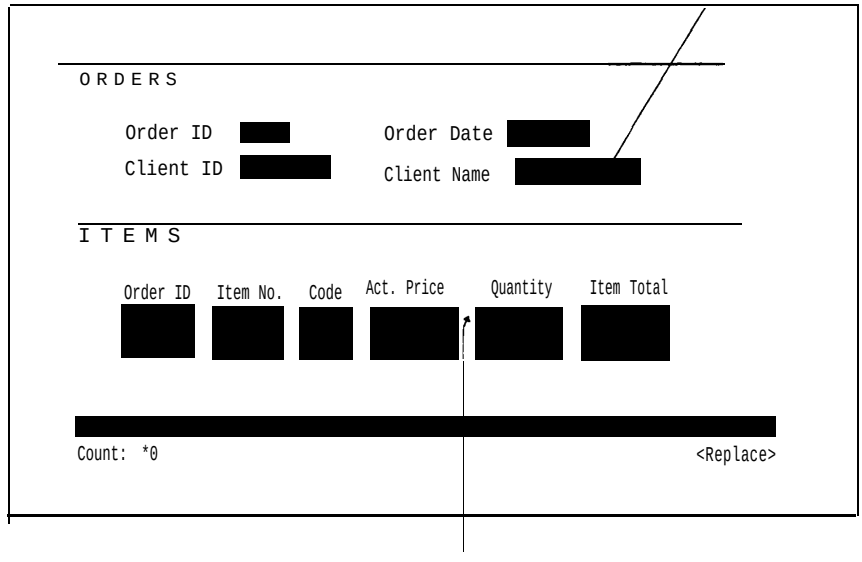
On-Validate-Field (form level)
Works on every field in the form.

Key-NXTBLK (field level)
Works on the Client Name field.

FIGURE 4-1
Trigger Scope

Pre-Block (block level)
Works on the ORDERS block, overriding the Pre-Block trigger defined at the form level.

Key-NXTFLD (block level)
Works on every field in the ITEMS block except the Quantity field.



Key-NXTFLD (field level)
Works on the Quantity field, overriding the Key-NXTFLD trigger defined at the block level.

Trigger Processing

This section discusses how SQL*Forms processes V3 triggers. Specifically, it covers the following topics:

- processing triggers within events
- processing executable statements
- trigger outcomes

Related Topics: Locks caused by triggers 3-12
PL/SQL 15-1
Triggers 14-1

Processing Triggers within Events

Triggers fire as a result of event processing. For example, when an operator executes a query the Pre- and Post-Query triggers fire during the event at designated trigger points.

The order in which triggers fire during an event is determined by event processing. In the case of executing a query, for example, SQL*Forms processing dictates that the Pre-Query trigger always fires before the query is executed and that the Post-Query trigger fires after each selected row is fetched.

Related Topics: Event flow charts 21-2

Processing Executable Statements

When a trigger fires, SQL*Forms executes the executable statements within the trigger in order, one at a time. Any changes that an executable statement makes (such as changing a field value) are visible to any statements that follow the current statement.

Note that you can change the order that SQL*Forms executes statements by using the branching, looping, and exception handling capabilities of PL/SQL.

Processing SQL Statements

All SQL Data Manipulation Language (DML) statements are legal in some type of SQL*Forms trigger. These statements follow:

- COMMIT
- DELETE
- INSERT
- LOCK
- ROLLBACK
- SELECT
- UPDATE

The "Trigger Types" section identifies which types of statements are legal, or recommended, in which trigger types. While you can write a V3 trigger that uses any DML statement, Oracle Corporation advises that you follow the recommendations that are stated for each trigger type. Using DML statements in certain triggers can desynchronize the state of records in SQL*Forms and rows in the database, and can cause unexpected behavior.

SQL Data Definition Language (DDL) and Data Command Language (DCL) statements are not legal in any SQL*Forms triggers.

Note: The outcomes of SQL statements are covered in the *PL/SQL User's Guide and Reference.*

Related Topics: Triggers, types 14-8

Processing Packaged Procedures

A packaged procedure can have one of three outcomes: success, failure, or fatal error.

success	A packaged procedure succeeds if it executes properly.
failure	A packaged procedure fails if it cannot be executed for any reason. (These reasons are covered on an individual basis in the event flow charts in Chapter 21, "Event Processing." An event is defined for each packaged procedure.)
fatal error	A packaged procedure causes a fatal error if the statement causes the operating system to run out of memory or results in a similar error.

Related Topics: Event flow charts 21-2

Processing User Exit Calls

A call to a user exit can have one of three outcomes: success, failure, or fatal error.

success	A call to a user exit succeeds if the user exit executes properly.
failure	A call to a user exit fails if the specified user exit does not exist.
fatal error	A call to a user exit causes a fatal error if the user exit returns a FAIL or FATAL code.

Related Topics: User exits 19-1

Trigger Outcomes

A trigger can either succeed or fail. A V3 trigger always succeeds unless it ends with an unhandled exception. You can explicitly direct a trigger to fail by raising the built-in exception, FORM_TRIGGER_FAILURE.

Trigger Failure

To direct a trigger to fail, you must raise the FORM_TRIGGER_FAILURE exception somewhere in the trigger. This exception is a predefine PL/SQL exception that you can only use from within SQL*Forms. (Refer to the *PL/SQL User's Guide and Reference* for information on PL/SQL exceptions.) When a trigger raises this exception, the trigger fails. SQL*Forms stops executing the trigger and performs any appropriate post-failure processing.

Note: If the Post-Change trigger fails during the Enter the Record or Fetch Records event, SQL*Forms disregards the failure and continues processing as if the trigger succeeded.

You might want to direct a trigger to fail if a particular trigger statement results in a failure or a fatal error. For example, assume that you have a Pre-Insert trigger that makes a change to the COUNT_REC table before posting a new record to the database. If the COUNT_REC table is not present, the Pre-Insert trigger fails. In this case, the change is not made, and you want the trigger to fail so that commit processing stops. The following trigger text satisfies this example:

```
UPDATE  COUNT_REC
    SET  COUNT_REC.NEW_RECORDS  =  COUNT_REC.NEW_RECORDS  +  1;
IF FORM_ FATAL
    THEN RAISE FORM_TRIGGER_FAILURE ;
END IF;
```

Related Topics: Event flow charts 21-2

Results of Trigger Failures

When a trigger fails, SQL*Forms stops executing the trigger and branches the current event processing to the appropriate post-failure processing. The following table shows the most likely post-failure precessing for each type of trigger.

Note: The conditions for (and exceptions to) this processing are covered in detail in the flow charts in Chapter 21, "Event Processing."

Trigger Type	*Result of Trigger Failure*
Key (all)	Trigger failure is ignored.
On-Clear-Block	Trigger failure is ignored.
On-Database-Record	Trigger failure is ignored.
On-Delete	SQL*Forms rolls back all database changes made by the current commit process (if necessary) and attempts to navigate the cursor to the first enterable field in the current record of the block in error.
On-Error	Trigger failure is ignored.
On-Insert	SQL*Forms rolls back all database changes made by the current commit process (if necessary) and attempts to navigate the cursor to the first enterable field in the current record of the block in error.
On-Lock	Locking fails.
On-Message	Trigger failure is ignored.
On-New-Field-Instance	Trigger failure is ignored.
On-New-Record	Record creation fails. If the current event is Enter the Block, SQL*Forms attempts to navigate the cursor to the initial tumor location.
On-Remove-Record	Trigger failure is ignored.
On-Update	SQL*Forms rolls back all database changes made by the current commit process (if necessary) and attempts to navigate the cursor to the first enterable field in the current record of the block in error.
On-Validate-Field	Field validation fails. SQL*Forms sets the error location.
On-Validate-Record	Record validation fails. SQL*Forms sets the error location.

Table continued on next page.

Trigger Type	*Result of Trigger Failure*
Post-Block	SQL*Forms attempts to navigate the cursor to the initial cursor location.
Post-commit	The current post or commit fails. SQL*Forms does not post or commit any transactions.
Post-Change	During the Enter the Record event, SQL*Forms ignores the failure and continues as if the trigger succeeded. During the Fetch Records event, SQL*Forms ignores the failure and continues as if the trigger succeeded. During the Validate the Field event, SQL*Forms sets the error location.
Post-Delete	SQL*Forms rolls back all database changes made by the current commit process (if necessary) and attempts to navigate the cursor to the first enterable field in the current record of the block in error.
Post-Field	SQL*Forms attempts to navigate the cursor to the initial cursor location.
Post-Form	SQL*Forms attempts to navigate the cursor to the initial cursor location.
Post-Insert	SQL*Forms rolls back all database changes made by the current commit process (if necessary) and attempts to navigate the cursor to the first enterable field in the current record of the block in error.
Post-Query	SQL*Form flushes the fetched record from the block and attempts to process another row.
Post-Record	SQL*Forms attempts to navigate the cursor to the initial cursor location.
Post-Update	SQL*Forms rolls back all database changes made by the current commit process (if necessary) and attempts to navigate the cursor to the first enterable field in the current record of the block in error.

Table contiued on next page.

Trigger Type	*Result of Trigger Failure*
Pre-Block	SQL*Forms attempts to navigate the cursor to the initial cursor location.
Pre-Commit	The current post or commit fails. SQL*Forms does not post or commit any transactions.
Pre-Delete	SQL*Forms rolls back all database changes made by the current commit process (if necessary) and attempts to navigate the cursor to the first enterable field in the current record of the block in error.
Pre-Field	SQL*Forms attempts to navigate the cursor to the initial cursor location.
Pre-Form	SQL*Forms attempts to navigate the cursor to the initial cursor location.
Pre-Insert	SQL*Forms rolls back all database changes made by the current commit process (if necessary) and attempts to navigate the cursor to the first enterable field in the current record of the block in error.
Pre-Query	SQL*Forms flushes any example record from the current block.
Pre-Record	SQL*Forms attempts to navigate the cursor to the initial cursor location.
Pre-Update	SQL*Forms rolls back all database changes made by the current commit process (if necessary) and attempts to navigate the cursor to the first enterable field in the current record of the block in error.
User-named (all)	Post-failure processing depends on what processing the user-named trigger is performing when it fails.

Related Topics: Event flow charts 21-2

CHAPTER

5 INTERFACE ELEMENTS

This chapter describes the generic types of elements in the SQL*Forms (Design) interface. Specifically, it discusses the following elements

- alert
- button
- check box
- dialog box
- field
- form
- list of values
- menu
- message line
- scroll region
- spread table
- status line

For a description of an individual interface element (such as the Field menu or the Edit Object dialog box), refer to the SQL*Forms (Design) online help system. For descriptions of how function keys interact with interface elements, refer to Chapter 7, “Function Keys.”

Note: This chapter describes interface elements as they appear in a character mode environment. These elements might appear differently in other environments. In addition, how you interact with these elements might be different. If you are using SQL*Forms (Design) in a bit-mapped environment, be sure to read "Bit-Mapped environment."

Related Topics: Bit-mapped environment D-4
Function keys 7-1

Alert

An alert is a window that appears in the middle of the screen, overlaying a portion of the current display. Alerts notify you of a condition that has arisen due to your last action. They force you to respond to that condition.

An alert contains a label, a text message, and one or more action items. The label indicates the severity of the alert and can be one of the following:

CAUTION	Indicates that the alert contains a cautionary statement or question.
NOTE	Indicates that the alert contains an information statement.
STOP	Indicates that the alert identifies a severe problem.

The text message explains the current situation that activated the alert. The action items give you choices to determine what action SQL*Forms should take or to acknowledge the current condition. When you select an action (via [Select]), the alert disappears and SQL*Forms executes the appropriate action.

Figure 5-1 shows the alert that appears when you choose the Quit option on the Action menu.

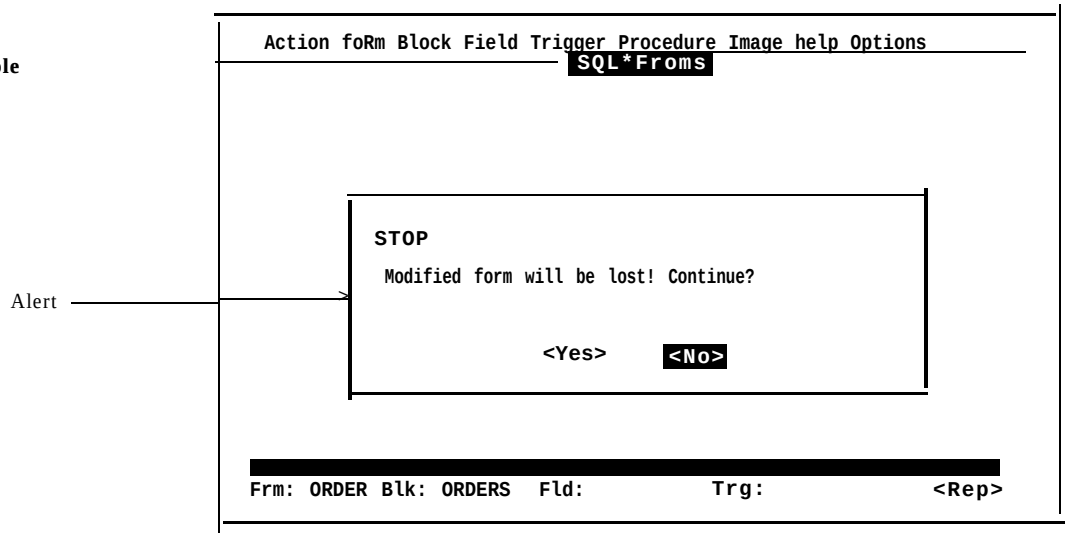


FIGURE 5-1
Alert Example

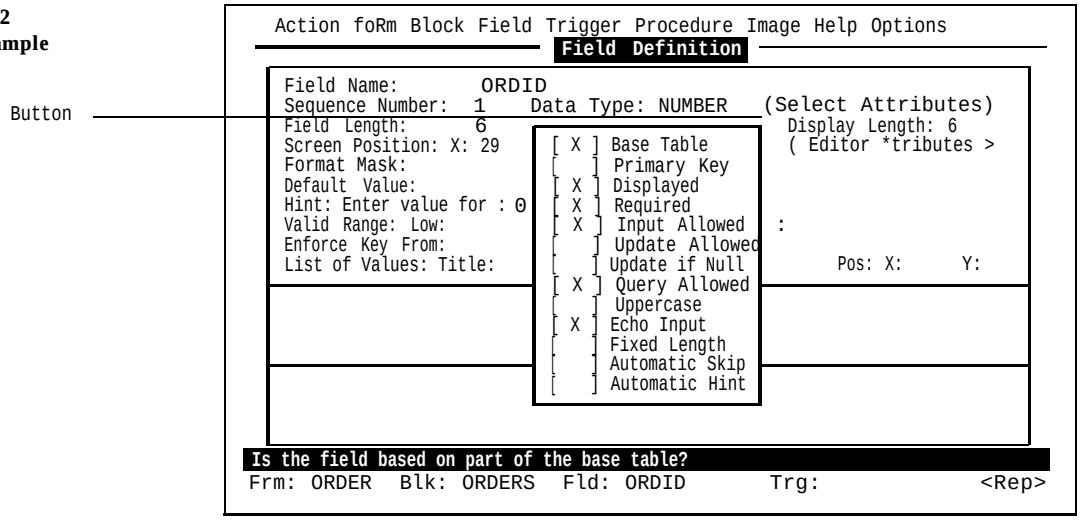
Button

A button appears as a text label in parentheses. When you select a button (via [Select]), SQL*Forms displays a list of available options or a dialog box.

- If a list appears, you can select, or make active, one or more items on the list. When you press [Accept], SQL*Forms temporarily saves your choices and the list disappears.
- If a dialog box appears, you can assign the fields and check boxes. When you press [Accept], SQL*Forms temporarily saves your choices and the dialog box disappears.

Figure 5-2 shows the Select Attributes button on the Field Definition form and the associated list of attributes.

FIGURE 5-2
Button Example



Check Box

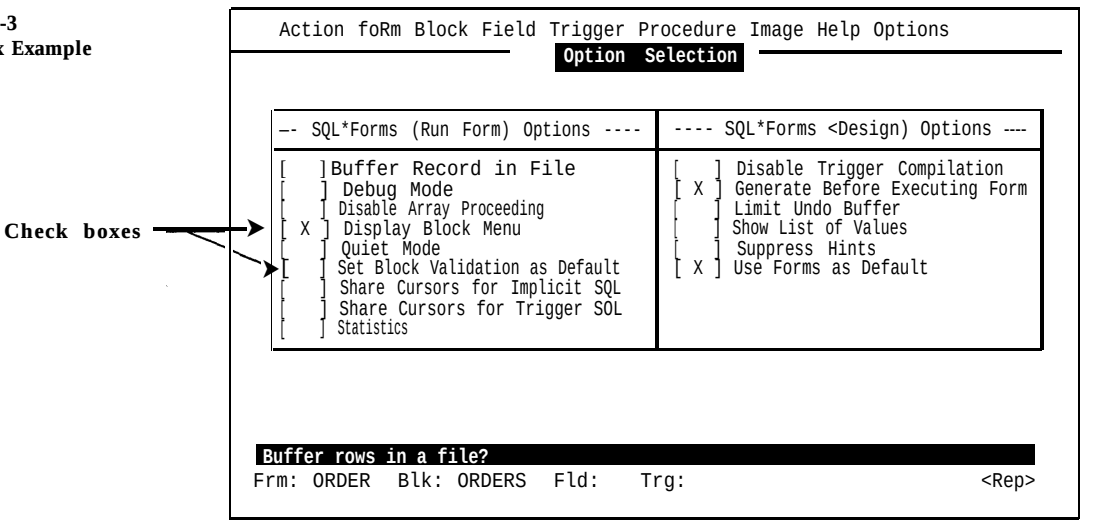
A check box is a two choice list from which you select one choice: "on" or "off."

- When a check box contains a blank space, the characteristic associated with the check box is turned off.
- When a check box contains an "x," the characteristic associated with the check box is turned on.

Press [Select] to turn on or off the characteristic associated with a check box. Alternatively, type any character to turn the characteristic on or type a blank space to turn it off.

Figure 5-3 shows how check boxes appear.

FIGURE 5-3
Check Box Example



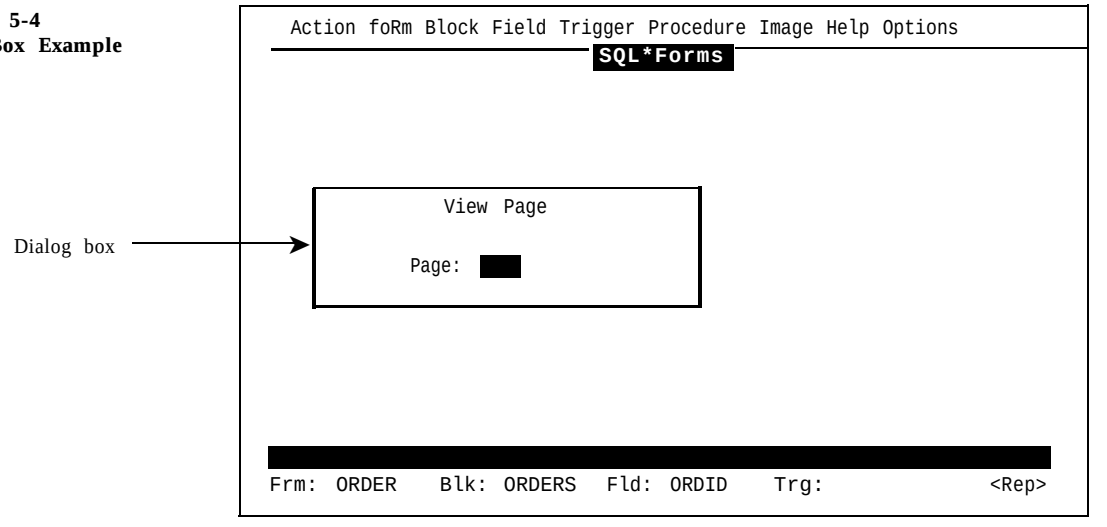
Dialog Box

A dialog box is a small form that appears in the middle of the screen, overlaying a portion of the current display. It contains one or more fields which you can fill in or edit. The information you enter in a dialog box can determine what object SQL*Forms should act on (e.g., a form name in the Open Form dialog box).

When you press [Accept], the dialog box disappears and SQL*Forms takes the appropriate action.

Figure 5-4 shows the View Page dialog box.

FIGURE 5-4
Dialog Box Example



Field

A field in the design interface is an area where you enter, edit, or delete data. That data either defines an object, as with the Field Name field in the Field Definition form, or it makes a processing choice, as with the Page field in the View Page dialog box.

Fields appear in forms, spread tables, and dialog boxes. Figures 5-5 and 5-6 show the Field Name field as it appears in both the Field Definition form and the Field Definition spread table.

FIGURE 5-5
Field in a Form

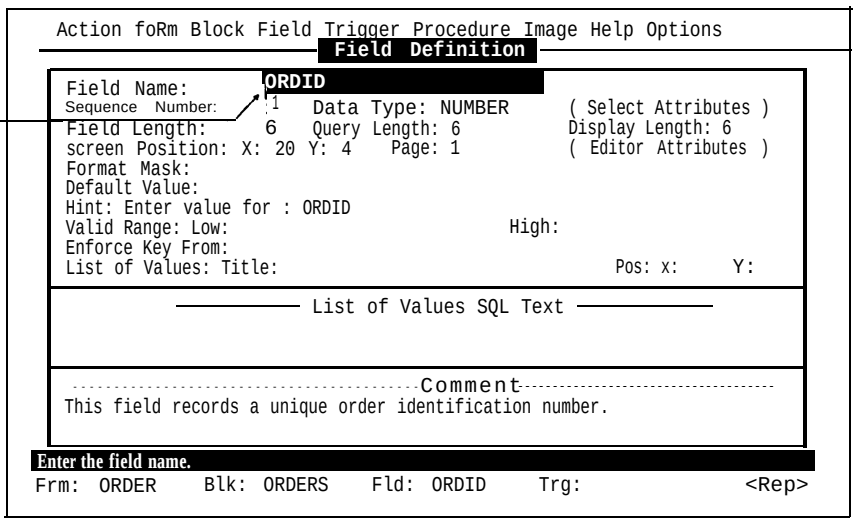
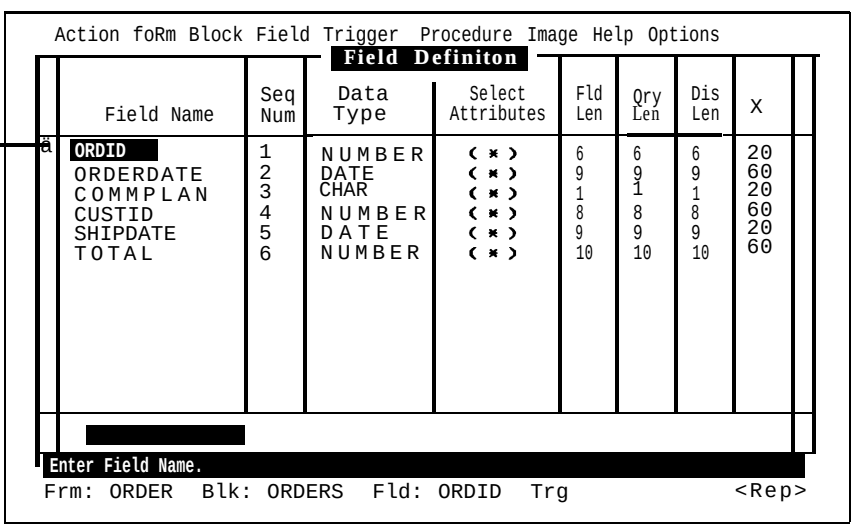


FIGURE 5-6
Field in a Spread Table



Field Validation

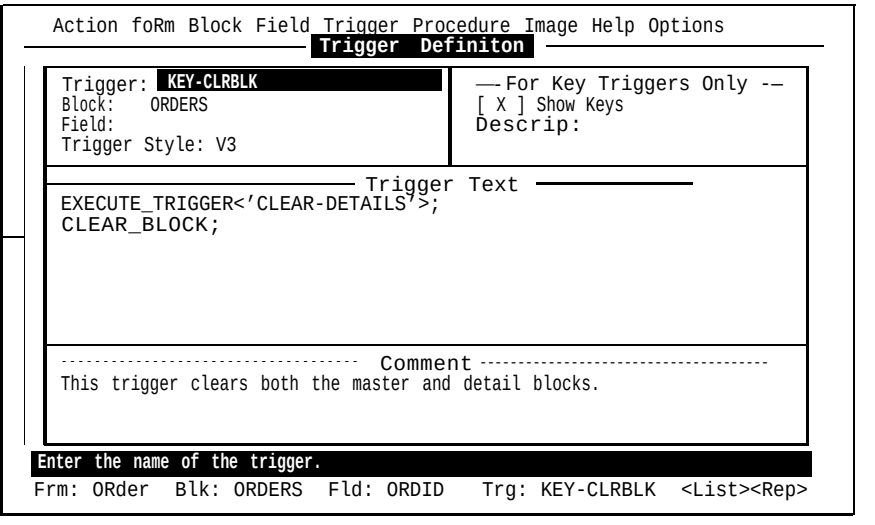
SQL*Forms validates field entries based on other information that you have already specified for the application. If a field fails validation, SQL*Forms prompts you for corrective action.

Form

A form in the design interface is a one page area that can contain buttons, check boxes, fields, and scroll regions. You navigate to the elements in a form using various function keys. When you press [Accept], SQL*Forms temporarily saves your choices and the form disappears.

Figure 5-7 shows the Trigger Definition form.

FIGURE 5-7
Form Example



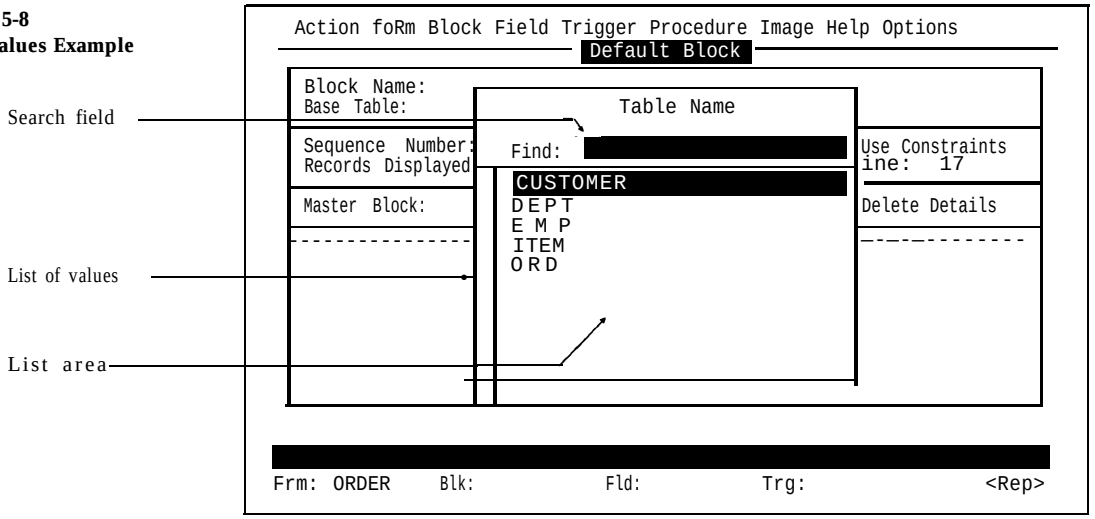
Most forms are associated with spread tables in the design interface. Where applicable, use [Change Display Type] to switch from a form to a spread table and vice versa.

List of Values

A list of values is a window that appears on the screen, overlaying a portion of the current display. Each list of values corresponds to one and only one field in the design interface. A list of values can consist of a title, a list area, and a search field. (Not all lists contain a search field.)

Figure 5-8 shows the Table Name list of values.

FIGURE 5-8
List of Values Example



You can use a list of values to view currently valid values and to enter a value into the field to which the list of values corresponds. To enter a value into the field, move the cursor to the item you want in the list of values list area and press [Select].

Note that you need not use the list of values to enter a value into a field that has a list of values.

List of Values List Area

The list area is a modified scroll region that contains one or more values. You can enter any of the listed values into the field to which the list of values corresponds. You can also use function keys to move up and down through the listed values.

Figure 5-8 shows the list area for the Table Name list of values.

Related Topics: Scroll region 5-13

List of values Search Field

The search field, or Find field, allows you to search for a value in the list of values list area.

Figure 5-8 shows the search field for the Table Name list of values.

List of values Search Function

The list of values search function allows you to look for a specific value in the list of values list area. To do so, specify a value in the list of values search field and press [List]. SQL*Forms re-displays the list, showing only those values that match the value you entered in the search field.

For example, assume that you want to narrow the number of values in the Table Name list of values to only those table names that end with "EMP." Enter `%EMP` in the search field and press [List]. From this point, you can narrow the list further by repeating the procedure with a more specific search value.

The list of values search function is case-insensitive. Also, it accepts standard SQL wildcard characters:

%	Represents any number of characters.
_ (underscore)	Represents any one character.

SQL*Forms implicitly appends the % wildcard on the end of every entry.

Menu

A menu is a list of available choices that allows you to determine what action to take next or what element to go to next.

The main menu for SQL*Forms (Design), shown in Figure 5-9, displays as a row of choices across the top of the screen when you first logon.

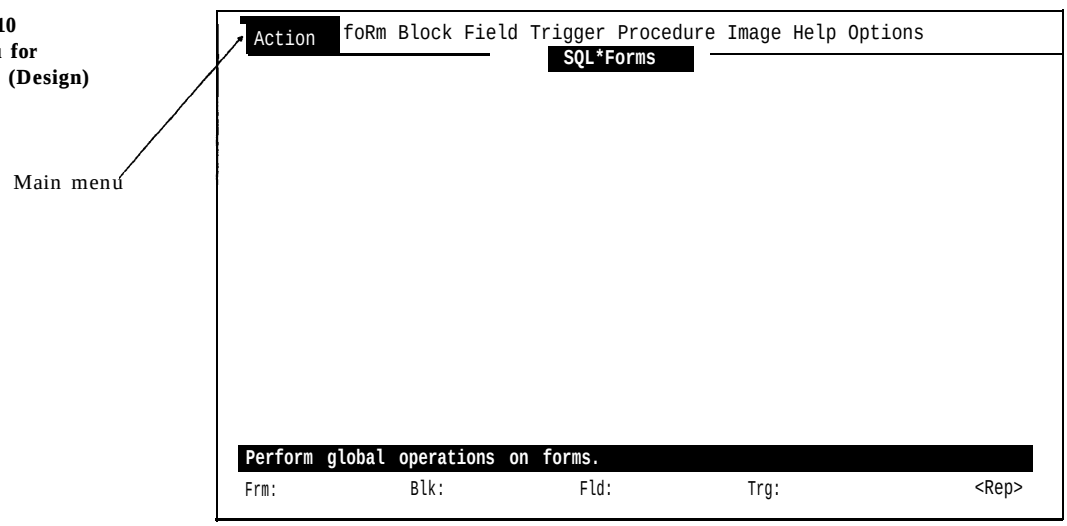


FIGURE 5-10
Main Menu for
SQL*Forms (Design)

All other menus are sub-menus of the main menu and appear as vertical lists of choices, like the Action menu shown in Figure 5-10.

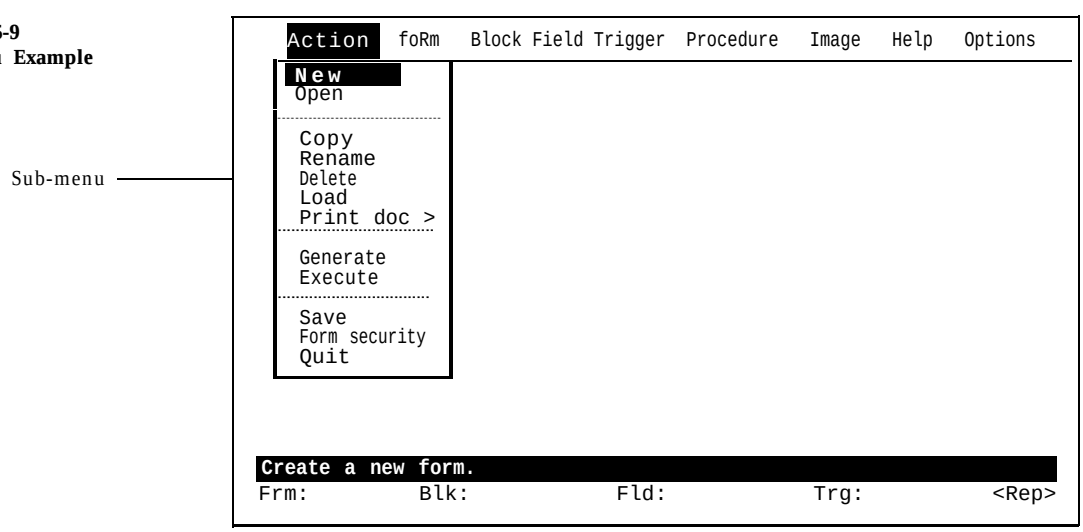


FIGURE 5-9
Sub-menu Example

You can select a menu choice in two standard ways

- Position the cursor on the desired item and press [Select].

or

- Type the first letter (or character) of the desired item. The exception to this practice is the Form option on the SQL*Forms main menu. Pressing the letter r (or R) selects this option.

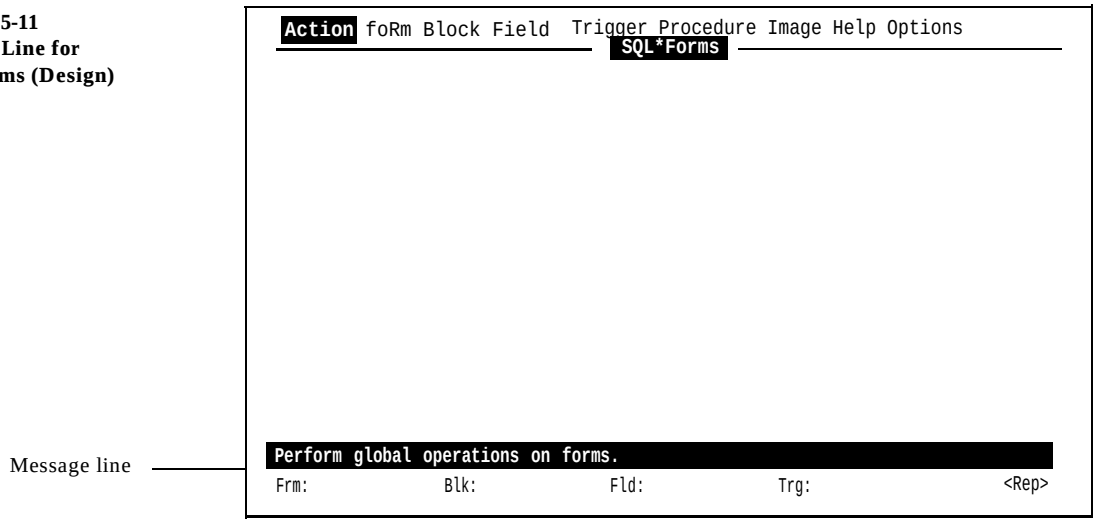
You can only select a menu choice if it is highlighted. SQL*Forms only highlights those choices which are logical in the current context.

Note: First letter menu selection does not work on a block mode terminal.

Message Line

The message line, shown in Figure 5-11, is the next to last line of every SQL*Forms (Design) screen. It displays hints and messages.

FIGURE 5-11
Message Line for SQL*Forms (Design)

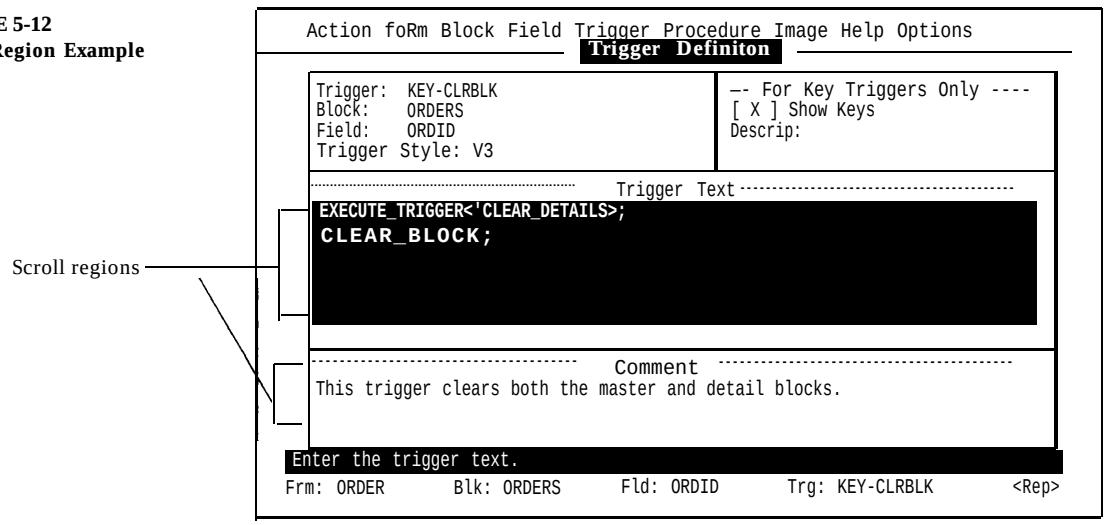


Scroll Region

A scroll region is an area that allows you to scroll vertically through information.

Figure 5-12 shows the Trigger Definition form, which contains the Trigger Text scroll region and the Comment scroll region.

FIGURE 5-12
Scroll Region Example



In a scroll region, as in a field, you can enter, edit, and delete data.

You can move through a scroll region by using function keys assigned to editing functions. These keys include [Search], which invokes the scroll region search function.

Scroll Region Search Function

The scroll region search function allows you to look for a specific value in a scroll region. To do so, press [Search] to call up the Search Criteria dialog box. From the dialog box, specify the following search criteria:

- the text string for which you want to search
- a text string with which to optionally replace the search string
- whether to search/replace forwards or backwards from the current cursor position
- whether or not to replace all found instances of the search string with the replace string

The search function is case-insensitive.

Spread Table

A spread table is a screen display that can span more than one screen, both horizontally and vertically. It is made up of the following elements

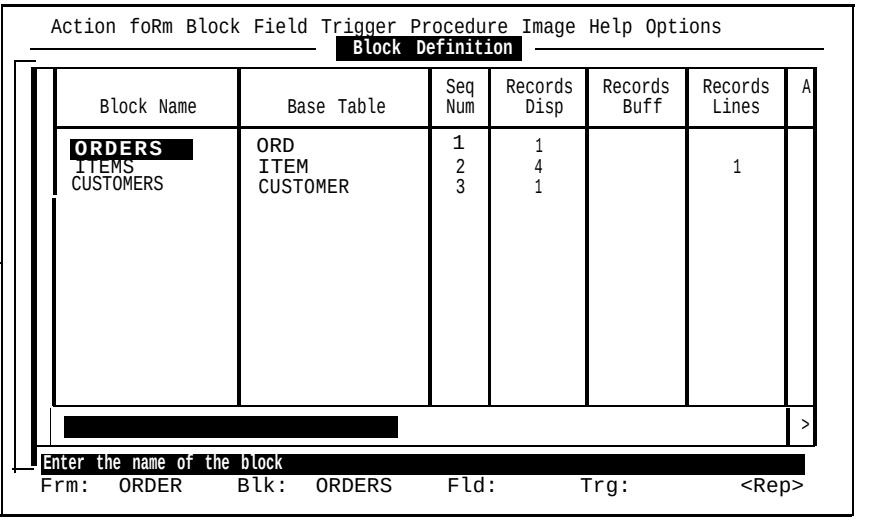
- buttons, check boxes, and fields
- a vertical scroll bar
- a horizontal scroll bar

The names, or titles, of the check boxes, buttons, and fields are arranged horizontally across the top of the spread table. The actual elements are arranged in vertical columns below the titles. Information in a spread table that applies to a single object appears in records. A record in a spread table is a horizontal row across all columns.

You navigate to the elements in a spread table using various function keys. When you press [Accept], SQL*Forms temporarily saves your choices and the spread table disappears.

Figure 5-13 shows part of the Block Definition spread table.

FIGURE 5-13
Spread Table Example



Spread Table Context

Spread tables always display records that are in context with the current object.

- Spread tables associated with blocks display all records that apply to the current form.
- Spread tables associated with fields display all records that apply to the current block.
- Spread tables associated with triggers display those records that apply to the current context object defined on the status line. The exception to this rule is when you invoke the spread table via the Trigger choice on the main menu.

Every spread table is associated with a form in the design interface. Use [Change Display Type] to switch from a spread table to a form and vice versa.

Related Topics: Context for forms, blocks, and fields 6-3
Context for triggers 6-5

Spread Table Scrolling

A spread table can scroll horizontally and vertically.

Horizontal Scrolling

A horizontal scroll bar appears on the bottom of a spread table page. (See Figure 5-13.) This scroll bar uses arrows to indicate if the spread table has additional fields "before" or "after" the displayed screen. If only part of the spread table is visible, the scroll bar also uses a bar graphic, known as an elevator, to indicate the relative horizontal position of the current screen display in the spread table.

If a spread table spans more than one screen, it scrolls horizontally a screen at a time. (In a bit-mapped environment, the scrolling is continuous, not by screen.) Spread tables always wrap from the last column of the spread table to the first column of the spread table and from the first column to the last column.

Vertical Scrolling

A vertical scroll bar appears on the left side of a spread table. (See Figure 5-13.) It uses arrows to indicate if there are more records of information "above" or "below" the current display. If only some of the records are visible, the scroll bar also uses a bar graphic, known as an elevator, to indicate the relative vertical position of the current screen display in the spread table.

If a spread table has more than one screen of records, it scrolls vertically, one record at a time.

Status Line

The status line is the last line of every SQL*Forms (Design) screen.

Note: This section explains the status line as it appears on design screens. The status line appears differently in the screen painter.

When the status line appears on a design screen, as in Figure 5-14, it displays the following information:

`Frm: form name`	Indicates the form that you are defining.
`Blk: block name`	Indicates the block that you are defining.
`Fld: field name`	Indicates the field that you are defining.
`Trg: trigger name`	Indicates the trigger that you are defining.
`<List>`	Indicates that there is a list of values for the current field. If there is no list of values for the current field this area of the status line appears blank.
`<Ins> or <Rep>`	`<Ins>`, or insert, indicates that SQL*Forms is in insert mode and will place the characters you type to the left of the cursor position. `<Rep>`, or replace, indicates that SQL*Forms is in replace mode and will overwrite existing characters with the characters you type, beginning at the current cursor position.

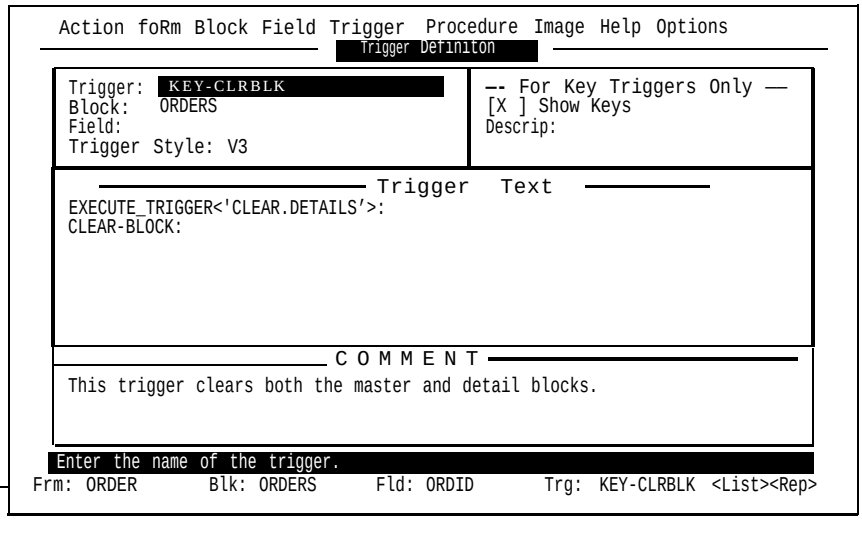


FIGURE 5-14
Status Line, on the Design Screen

CHAPTER

6 CONTEXT

In SQL*Forms, context is a concept that you use when you are working in the SQL*Forms (Design) interface. Context is your way of knowing what parts of a form definition you can currently access. Put another way, context answers the following two questions for you:

1. “If I go to the main menu right now, what SQL*Forms objects can I modify?”
2. "If I press [Zoom In] or [Zoom Out] right now, what information displays?”

SQL*Forms relays this information to you through the status line. Specifically the status line indicates the current context objects. There can be a current context for the form, block, and field objects.

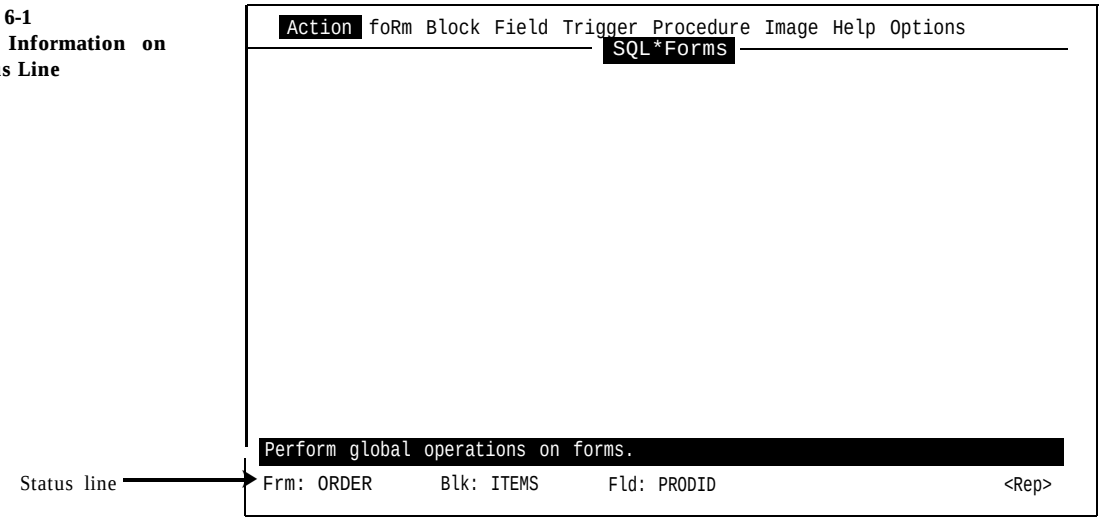
This chapter discusses the concept of context and how it relates to the SQL*Forms (Design) interface. Specifically, it covers the following topics:

- context and main menu choices
- context and zoom connections
- context for forms, blocks, and fields
- context for triggers
- context in the screen painter
- changing context with function keys

Context and Main Menu Choices

Context lets you know what object information you can currently access through the main menu. For example, assume that the status line displays as in Figure 6-1.

FIGURE 6-1
Context Information on the Status Line



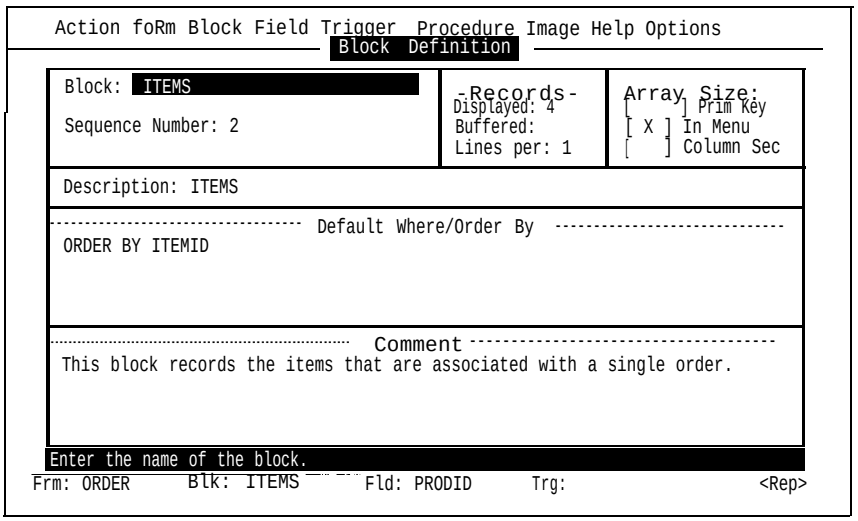
The status line in Figure 6-1 tells you that there is a current context form, block, and field. Therefore, if you went to the main menu and chose the Field option and then the Modify option, SQL*Forms would display information (records of data) on all of the fields in the ITEMS block and the current record would be the one associated with the PRODID field. Similarly, if you went to the main menu and chose the Block option and then the Modify option, SQL*Forms would display information on all of the blocks in the ORDER_ENTRY form and the current record would be the one associated with the ITEMS block.

Context and Zoom Connections

Context makes your zoom connections explicit. That is, when a form or spread table is displayed, the context information on the status line tells you what object information SQL*Forms will display if you press [Zoom In] or [Zoom Out].

For example, assume that your screen displays as the one in Figure 6-2.

FIGURE 6-2
Context and Zoom Connections



The status line indicates that there is a form, block, and field context. If you press [Zoom In] from the current screen, SQL*Forms displays the Field Definition form and makes the record associated with the PRODID field the current context field. Alternatively, if you press [Zoom Out] from the current screen, SQL*Forms displays the Form Definition form and displays information for the current form.

Related Topics: [Zoom In] 7-16
[Zoom Out] 7-17

Context for Forms, Blocks, and Fields

SQL*Forms establishes context from the top of the object hierarchy down. That is, SQL*Forms must have a form context before it can establish a block context and it must have a block context before it can establish a field context.

If you do not have a block context, SQL*Forms removes the highlight from the Field option on the main menu. This condition indicates that you cannot attempt to modify a field or zoom in to the field level until you have established a block context. Similarly, you cannot enter the screen painter unless you have established a block context.

Note that the form context is always the form that you have opened or created.

Default Context for Blocks and Fields

If there is block context but no field context, when you choose the Field option from the main menu (and then the Modify option), SQL*Forms moves the cursor to the first "field" record in the current block definition. This default action establishes the field associated with that record as the current context field, and this is reflected on the status line. Similarly, if there is form context but no block context, when you choose the Block option from the main menu (and then the Modify option), SQL*Forms moves the cursor to the first "block" record in the current form definition.

Lost Context for Blocks and Fields

When you change the block context, SQL*Forms does not remember what the last field context was for that block-the field context is lost. Depending on how you changed the block context, SQL*Forms provides no field context or it establishes the first field in the block definition as the current context field. Similarly, when you change the form context SQL*Forms does not remember the last block and field contexts.

Keeping Blocks and Fields in Context Order

SQL*Forms (Design) keeps track of the proper order of the records you create or bring into the object definition forms and spread tables. This tracking affects the way SQL*Forms behaves when you modify or create records associated with blocks and fields. Specifically, if you change the sequence number of a field or block, or create a new field or block, SQL*Forms moves the associated record to its correct (i.e., sequence) position in the list of records when you attempt to move the cursor out of the record. SQL*Forms keeps the cursor in the modified record, or new record, (changing the screen display if necessary to show the current record) and maintains the current context.

Context for Triggers

When you choose the Trigger option from the main menu (not from a sub-menu), SQL*Forms displays the Trigger Definition form or spread table with data for all triggers in the current form definition. Note that the display is the same despite the current block and field context. During this operation, SQL*Forms maintains the context you had before you chose the Trigger option unless you explicitly change it. This behavior exists because you will often want to look at information on all triggers in a form and still be able to go directly back to the form data with which you were working.

While you are displaying information for all triggers in a form, you can explicitly change the current context either by editing a trigger entry or by zooming in or out (via [Zoom In] or [Zoom Out]) from a trigger entry. In each case, SQL*Forms changes the current block and field context to correspond with the object that owns the individual trigger on which you have acted. For example, if you edit the record for a trigger that is attached to the PRODID field, SQL*Forms makes the following changes:

- The field context becomes the field to which the trigger is attached (i.e., PRODID).
- The block context becomes the block that owns the PRODID field (i.e., ITEMS).

Defining Triggers out of Context

SQL*Forms allows you to define a trigger out of context. For example, assume that your current context is the ORDER_ENTRY form and the ORDERS block. In addition, the Trigger Definition spread table is displaying information on triggers attached to the ORDERS block. You can insert a new record in this spread table and define a new trigger that is attached to the PRODID field in the ITEMS block. When you move to the main menu again (or zoom away from the current display), SQL*Forms will "move" the new record to where it belongs. Specifically, that record will now display only when you request the display of information on all triggers or triggers attached to the PRODID field. Note that this process does not change the current context.

Context in the Screen Painter

When you work in the screen painter, the status line indicates your form and block context. You can change the block context by navigating between blocks. Note that changing the block context eliminates any current field context. You can change the field context (or establish a new one) by taking any of the following actions:

- Move the cursor to an existing field and press [Define Field].
- Create a new field with [Define Field].
- Cut the current context field with [Cut].

Navigating from field to field does not affect the field context. Changing the page in the screen painter does not affect the block or field context.

Changing Context with Function Keys

In order to modify an object in a form definition, that object must be part of your current context. Therefore, you should be familiar with which SQL*Forms (Design) function keys allow you to change your context.

[Cut]	If you cut the current context field, you eliminate the field context.
[Define Field]	If you press [Define Field] from an existing field, SQL*Forms establishes the field you are defining as the current context field. If you press [Define Field] to create and define a new field, SQL*Forms eliminates any existing field context. SQL*Forms establishes the new field as the current context field, but does not reflect this change in the status line until the screen is updated.
[Down]	[Down] establishes the object associated with the next record as a current context object. For example, if you press [Down] from the Field Definition spread table, SQL*Forms makes the current context field the field that is associated with the next record in the spread table.

[Insert Record]	[Insert Record] eliminates any existing context for the object level you are defining and for any objects below that level. For example, if you press [Insert Record] while in the Block Definition spread table, SQL*Forms eliminates the current block context and any field context. If you define a new block at this point, SQL*Forms establishes the new block as the current context block, but does not reflect this change in the status line until the screen is updated.
[Navigate]	[Navigate] invokes the Navigate dialog box, which allows you to redefine the block and field context.
[Next Block]	[Next Block] establishes the next block (in screen painter sequence) as the current context block and eliminates any existing field context.
[Previous Block]	[Previous Block] establishes the previous block (in screen painter sequence) as the current context block and eliminates any existing field context.
[Up]	[Up] establishes the object associated with the previous record as a current context object. For example, if you press [Up] from the Field Definition spread table, SQL*Forms makes the current context field the field that is associated with the previous record in the spread table.
[Zoom In]	If there is a current context object for the object level that you zoom to, [Zoom In] maintains the current context. However, if there is no current context object for the object level that you zoom to, [Zoom In] establishes the object associated with the first record as a current context object. For example, if you zoom from the Block Definition spread table to the Field Definition spread table, and there is no current field context, SQL*Forms makes the field associated with the first record in the spread table the current context field.

CHAPTER

7 FUNCTION KEYS

SQL*Forms (Design) functions include executing actions and moving the cursor-everything except typing text. You perform these functions by pressing function keys.

SQL*Forms documentation refers to function keys by their function names (e.g., [Next Field]), not by physical key names. This is because the physical keys that function keys are mapped to vary between systems.

This chapter covers the following topics about SQL*Forms (Design) function keys:

- keyboard map
- function key descriptions
- function key groupings

This chapter does not discuss the keys that operators use in SQL*Forms (Run Form). For information on those, refer to the *SQL*Forms Operator's Guide.*

Keyboard Map

A keyboard map shows what physical keys correspond to what SQL*Forms function keys for your system. You can get a hard copy of the keyboard map for SQL*Forms (Design) from the chapter on SQL*Forms in your ORACLE Installation and User's Guide or from your DBA. This hardcopy map corresponds to the default mapping that Oracle Corporation supplies with the software for your system.

Note: There are at least two SQL*Forms keyboard maps for each system-one for designing a form (SQL*Forms (Design) keyboard map) and one for running or testing a form (SQL*Forms (Run Form) keyboard map).

Show Keys Screen

You can display a keyboard map on the terminal screen by pressing [Show Keys] from anywhere within SQL*Forms (Design). This action brings up the Show Keys screen.

The contents of the Show Keys screen pertain to the mapping that you are currently using. Therefore, no description appears if a particular key is not defined in the current mapping.

The contents of the Show Keys screen are also context-sensitive. That is, the screen displays descriptions for only those function keys that are valid for your current position or mode in the design interface. For example, if you press [Show Keys] while in the Open Form dialog box, the screen only displays descriptions for those keys that operate in that particular element.

Function Key Descriptions

This section provides descriptions for these SQL*Forms (Design) function keys:

[Accept]
[Beginning Of Line]
[Bookmark]
[Cancel]
[Change Display Type]
[Clear Field]
[Copy]
[Copy Object]
[Cut]
[Define Field]
[Delete Backward]
[Delete Character]
[Delete Line]
[Delete Record]
[Down]
[Draw Box/Line]
[End of Line]
[First Line]
[Help]
[Insert Line]
[Insert Record]
[Insert/ReplaceJ
[Last Line]
[Left]
[List]
[Menu]
[Navigate]
[Next Block]
[Next Field]
[Next Page]
[Next Record]
[Paste]
[Previous Block]
[Previous Field]
[Previous Page]
[Previous Record]
[Print]
[Refresh]
[Resize Field]
[Retrace]
[Right]
[Screen Painter]
[Scroll Down]
[Scroll Left]
[Scroll Right]
[Scroll Up]
[Search]
[Select]
[Show Keys]
[Undo]
[Up]
[Zoom In]
[Zoom Out]

The specific function that a function key performs varies depending on the context in which you use it. This section provides context-sensitive descriptions where applicable.

[Accept]

In a dialog box [Accept] closes the dialog box and acts upon your entry.

In a form or spread table [Accept] closes the current form or spread table and preserves any changes you made.

In the screen painter [Accept] preserves any changes you made in the screen painter. SQL*Forms displays the main menu.

[Beginning of Line]

In a field [Beginning of Line] moves the cursor to the first character space (displayed or undisplayed) of the current field.

In a scroll region [Beginning of Line] moves the cursor to the first character space of the current line.

[Bookmark]

In the help system [Bookmark] puts a placeholder on the current help screen.

In the design interface [Bookmark] displays the help screen on which you last put a placeholder.

Related Topics: Bookmark help feature 8-5

[Cancel]

In a form, spread table, or dialog box [Cancel]:

- cancels the current operation, or action (e.g., opening a form)
- reverses any changes made to the current object and returns the cursor to the main menu

In the screen painter [Cancel] preserves any changes you made in the screen painter. SQL*Forms displays the main menu.

[Change Display Type]

In a form [Change Display Type] displays the spread table that corresponds to the current form and moves the cursor to the corresponding element.

In a spread table [Change Display Type] displays the form that corresponds to the current spread table and moves the cursor to the corresponding element.

[Clear Field]

In a field [Clear Field] clears the entire field.

In a scroll region [Clear Field] clears to the end of the line, and rewraps text if possible.

[Copy]

[Copy]:

- functions in the screen painter and scroll regions
- copies a selected objector area from the screen and stores it in the paste buffer until you cut or copy another objector area

[Copy Object]

[Copy Object] displays the Source Object dialog box and the Target Object dialog box, which allow you to copy or reference objects that exist in the current form or that exist in other forms to which you have access privileges.

[Cut]

[Cut]:

- functions in the screen painter and scroll regions
- removes a selected objector area from the screen and stores it in the paste buffer until you cut or copy another objector area

Related Topics: Cutting an object 2-9

[Define Field]

[Define Field]:

- functions only in the screen painter
- displays the Field Definition spread table or the Field Definition form, depending on whether a spread table or a form is your default element

If you are modifying an existing field, [Define Field] makes the record that corresponds to the chosen field the current record in the spread table or form. If you are creating a new field, [Define Field] makes a new record the current record in the spread table or form. This new record will contain data that corresponds with the field you drew in the screen painter.

Related Topics: Creating a field, from the screen painter 2-5
Use Forms as Default design option 26-6

[Delete Backward]

[Delete Backward]:

- functions in fields and scroll regions
- deletes the character to the left of the current cursor position and moves any characters on the right one space to the left

[Delete Line]

[Delete Line]:

- functions only in scroll regions
- deletes the current line, and rewraps text if possible

[Delete Record]

[Delete Record] deletes the current record from the form or spread table. The record is deleted from the database upon the next database commit.

[Delete Character]

[Delete Character]:

- functions only in fields and scroll regions
- deletes the character at the current cursor position and moves any characters on the right one space to the left

[Down]

In a form [Down]:

- displays the data for the next record in all fields, toggle lists, check boxes, and scroll regions in the form
- displays a new record if the current record is the last record
- does nothing if the current record is a new record

In a spread table [Down]:

- moves the cursor down one line, to the next record or to a new record
- does nothing if the cursor is in a new (blank) record

In the screen painter [Down]:

- moves the cursor down one character position
- moves the cursor to the first line of the page if pressed when the cursor is in the last line of the page, scrolling the page vertically if necessary

[Draw Box/Line]

[Draw Box/line]:

- functions only in the screen painter
- draws a line between two selected points if the two points are in the same horizontal or vertical plane
- draws a box using two selected points as diagonally opposite corners if the two points are not on the same horizontal or vertical plane

Related Topics: Drawing lines and boxes 2-11

[End of Line]

In a field [End of Line] moves the cursor to the right of the last character in the current field, scrolling if necessary.

In a scroll region [End of Line] moves the cursor to the right of the last character in the current line.

[First Line]

[First Line]:

- functions only in scroll regions
- moves the cursor to the first character in the first line of data in the scroll region, scrolling the region if necessary.

[Help]

In the help system [Help] displays the help overview, titled "About the Help System."

In the design interface [Help] invokes the SQL*Forms help system, displaying help text for the current interface element. If an error message is displayed on the message line, [Help] displays help text for that error.

[Insert Line]

[Insert Line]:

- functions only in scroll regions
- creates a new (blank) line after the current line

[Insert Record]

[Insert Record]:

- functions in forms and spread tables
- creates a new (blank) record after the current record

[Insert/Replace]

[Insert/Replace]:

- functions in fields, scroll regions, and the screen painter
- toggles SQL*Forms from insert mode to replace mode or from replace mode to insert mode

In insert mode, SQL*Forms places the characters you type to the left of the cursor position. In replace mode, SQL*Forms overwrites existing characters with the characters you type, beginning at the current cursor position.

Note: The status line indicates whether SQL*Forms is in insert mode or replace mode.

Related Topics: Ins status line item 5-16
Rep status line item 5-16

[Last Line]

[Last Line]:

- functions only in scroll regions
- moves the cursor to the last character in the last non-blank line of data in the scroll region, scrolling the region if necessary.

[Left]

In a form or spread table [Left]:

- moves the cursor one character position to the left
- will not move the cursor out of a field or scroll region

In the main menu [Left]:

- highlights the menu choice to the left of the current choice
- highlights the right-most menu choice if the current choice is the left-most choice

In the screen painter [Left]:

- moves the cursor one character position to the left
- moves the cursor to the right-most character position on a page if pressed when the cursor is in the left-most position on the page, scrolling the page horizontally if necessary

[List]

[List] displays a list of values for the field if a list of values exists.

Note: The status line indicates if a list of values is available for the field.

Related Topics: List status line item 5-16

[Menu]

[Menu]:

- displays the main menu
- activates the main menu

[Navigate]

[Navigate] displays the Edit Object dialog box, which allows you to "go to" the spread table or form for a specific object (i.e., a page, block, field, or trigger in the form).

Note: Use [Navigate] as a shortcut for directly accessing the form or spread table for a particular object. This feature is especially useful for designers working on block mode terminals because it usually requires fewer screen interrupts than alternative methods.

[Next Block]

[Next Block]:

- functions only in the screen painter
- makes the next block, in block sequence, the current block
- purges the undo buffer

[Next Field]

In a form [Next Field]:

- moves the cursor in sequence, to the next field, check box, or scroll region
- moves the cursor from a field or check box into a scroll region
- makes current the first field or check box in a form if pressed in the last element

In a list of values [Next Field]:

- moves the cursor down, to the next value in the list of values
- moves the cursor to the first value in a list of values if the cursor is at the last value

In a menu [Next Field]:

- moves the cursor down or to the right, to the next option in a menu
- moves the cursor to the first menu option if the cursor is at the last option

In a spread table [Next Field]:

- moves the cursor to the right, to the next field or check box
- moves the cursor to the first field or check box in a spread table if the cursor is at the last one

In the screen painter [Next Field]:

- moves the cursor-from left to right and top to bottom-to the next field in the block (on the current page only), making that field the current field
- moves the cursor to, and makes current, the first field in the block if pressed in the last field

Note that the described cursor movement is independent of field sequence numbers.

[Next Page]

[Next Page]:

- functions only in the screen painter
- displays the next screen painter page, even if the page is blank
- moves the cursor to the upper left corner of the screen
- purges the undo buffer .

[Next Record]

In a form [Next Record]:

- displays the data for the next record in all fields, toggle lists, check boxes, and scroll regions in the form
- displays a new record if the current record is the last record
- does nothing if the current record is a new record

In a spread table [Next Record]:

- moves the cursor down one line, to the next record or to a new record
- does nothing if the cursor is in a new (blank) record

[Paste]

[Paste]:

- functions only in the screen painter and scroll regions
- places the contents of the paste buffer (i.e., the objector area that you cut most recently) in the screen, positioning the upper left corner of the pasted area at the cursor position

Related Topics: Pasting an object 2-10

[Previous Block]

[Previous Block]:

- functions only in the screen painter
- makes the previous block, in sequence, the current block
- purges the undo buffer

[Previous Field]

In a form [Previous Field]:

- moves the cursor in sequence to the previous field, check box, or scroll region in a spread table
- moves the cursor from a field or check box into a scroll region
- makes current the first field or check box in a form if pressed in the last element

In a list of values [Previous Field]:

- moves the cursor up to the previous value in the list of values
- moves the cursor to the last value in a list of values if the cursor is at the first value

In a menu [Previous Field]:

- moves the cursor up or to the left to the previous option in a menu
- moves the cursor to the last menu option if the cursor is at the first one

In a spread table [Previous Field]:

- moves the cursor to the left to the previous field or check box in a spread table
- moves the cursor to the last field or check box in a spread table if the cursor is at the first one

In the screen painter [Previous Field]:

- moves the cursor-from right to left and bottom to top-to the previous field in the block, making the field the current field
- moves the cursor to, and makes current, the last field in the block if pressed in the first field

[Previous Page]

[Previous Page]:

- functions only in the screen painter
- displays the previous screen painter page
- moves the cursor to the upper left corner of the screen
- purges the undo buffer
- does nothing if the current page is page one

[Previous Record]

In a form [Previous Record]:

- displays the data for the previous record in all fields, toggle lists, check boxes, and scroll regions in the form
- does nothing if the cursor is in the first record

In a spread table [Previous Record]:

- moves the cursor up one line to the previous record
- does nothing if the cursor is in the first record

[Print]

[Print]:

- writes the current screen to a file and gives you the option of printing it on the system printer
- gives you the option of printing the key map

[Refresh]

[Refresh]:

- refreshes the current screen
- redisplays any changes you have made to the screen if you are using a character mode terminal
- discards any changes you have made to the screen and re-displays previous values if you are using a block mode terminal. (On block mode devices this is a terminal function, not a SQL*Forms function.)

[Resize Field]

[Resize Field]:

- functions only in the screen painter
- changes the display length of the selected field by changing the right endpoint of the field to be the current cursor location

[Retrace]

[Retrace]:

- functions only in the help system
- displays the help screen that you were in prior to the current help screen (for the current help session)

Related Topics: Retrace help feature 8-4

[Right]

In a form or spread table [Right]:

- moves the cursor one character position to the right
- will not move the cursor out of a field or scroll region

In the main menu [Right]:

- highlights the menu choice to the right of the current choice
- highlights the left-most menu choice if the current choice is the right-most choice

In the screen painter [Right]:

- moves the cursor one character position to the right
- moves the cursor to the left-most character position on a page if pressed when the cursor is in the right-most position on the page, scrolling the page horizontally if necessary

[Screen Painter]

[Screen Painter] invokes the screen painter and places the cursor in the block, field, and page that corresponds to the current context.

For example, if you are in the Field Definition spread table and the current record defines the CLIENT_ID field, pressing [Screen Painter] places the cursor in the CLIENT_ID field in the screen painter.

[Scroll Down]

In a list of values [Scroll Down]:

- scrolls the contents of the list up by approximately 80 percent of the list's display, displaying values that were "below" the list's display
- navigates to the top-most displayed value once scrolling is complete

In a scroll region [Scroll Down]:

- scrolls the contents of the scroll region up by approximately 80 percent of the scroll region's display, displaying lines of information that were "below" the scroll region's display
- navigates to the same column in the top-most displayed line once scrolling is complete

In a spread table [Scroll Down]:

- scrolls the contents of the spread table up by approximately 80 percent of the spread table's display, displaying records that were "below" the spread table's display

- navigates to the instance of the cursor field in the displayed record with the lowest sequence number once scrolling is complete

[Scroll Left]

[Scroll Left]:

- functions only in a field
- scrolls approximately 80 percent of the displayed contents of a field to the right, effectively displaying data that was "to the left" of the field's display window

[Scroll Right]

[Scroll Right]:

- functions only in a field
- scrolls approximately 80 percent of the displayed contents of a field to the left, effectively displaying data that was "to the right" of the field's display window

[Scroll Up]

In a list of values [Scroll Up]:

- scrolls the contents of the list down by approximately 80 percent of the list's display, displaying values that were "above" the list's display
- navigates to the top-most displayed value once scrolling is complete

In a scroll region [Scroll Up]:

- scrolls the contents of the scroll region down by approximately 80 percent of the scroll region's display, displaying lines of information that were "above" the scroll region's display
- navigates to the same column in the top-most displayed line once scrolling is complete

In a spread table [Scroll Up]:

- scrolls the contents of the spread table down by approximately 80 percent of the spread table's display, displaying records that were "above" the spread table's display
- navigates to the instance of the cursor field in the displayed record with the lowest sequence number once scrolling is complete

[Search]

[Search]:

- functions only in a scroll region
- displays the Search Criteria dialog box, which allows you to search for, and optionally replace, a specified text string

[Select]

In a check box [Select] toggles the value in a check box, thereby activating the previously inactive value.

In a help system keyword [Select] chooses the keyword the cursor is on.

In a list of values [Select] chooses the value the cursor is on.

In a menu [Select] chooses the option the cursor is on.

In the screen painter [Select] marks a point on the screen in the screen painter. SQL*Forms uses this selection point for a number of screen painter activities.

[Show Keys]

[Show Keys] displays the keyboard map that corresponds to the mapping that you specified (or that the system defaulted to) when you logged into SQL*Forms.

[Undo]

[Undo]:

- functions only in the screen painter
- reverses the most recent operation that you performed in the screen painter
- reverses previous screen painter operations in succession if you press it repeatedly
- does not reverse the entering of constant text
- does not reverse an "undo" operation

Note: SQL*Forms flushes the undo buffer when you move between blocks or pages or when you leave the screen painter.

Related Topics: Limiting the undo buffer, -1 switch for SQL*Forms (Design) 23-12

[Up]

In a form [Up]:

- displays the data for the previous record in all fields, toggle lists, check boxes, and scroll regions in the form
- does nothing if the cursor is in the first record

In a spread table [Up]:

- moves the cursor up one line to the previous record
- does nothing if the cursor is in the first record

In the screen painter [Up]:

- moves the cursor up one character positions
- moves the cursor to the last line of the page if pressed when the cursor is in the first line of the page, scrolling the page vertically if necessary

[Zoom In]

[Zoom In]:

- displays the form or spread table for the object type that is one step lower in the object hierarchy than the current object type
- does nothing if the cursor is in the Trigger Step Definition form or spread table, or if the current trigger is a V3 trigger and the cursor is in the Trigger Definition form or spread table

When you use [Zoom In], SQL*Forms moves down the following hierarchy of objects, starting from the level of the current object:

- form
- block
- field
- trigger
- trigger step (only for a V2 trigger)

As SQL*Forms moves down the hierarchy, it maintains the current spread table or form display style. The exception to this is when you press [Zoom In] from the Form Definition form, in which case SQL*Forms moves to the Block Definition spread table or the Block Definition form depending on the default display style.

Related Topics: Use Forms as Default design option 26-6

[Zoom Out]

[Zoom Out]:

- displays the form or spread table for the object that owns the current object. The object to be displayed is usually one step higher in the object hierarchy than the current object
- does nothing if the cursor is in the Form Definition form

When you use [Zoom Out], SQL*Forms moves up the following hierarchy of objects, starting from the level of the current object:

- form
- block
- field
- trigger
- trigger step (only for a V2 trigger)

SQL*Forms moves up the hierarchy one level at a time, unless the current object is a trigger that is owned by a block or the form. As SQL*Forms moves up the hierarchy, it maintains the current spread table or form display style where possible.

Function Key Groupings

This section places all of the SQL*Forms (Design) function keys into groups by their basic characteristics:

- cursor movement functions
- editing functions
- general design functions
- screen painting functions

Use these groupings to identify a function key when you know what you want to do (e.g., move the cursor from field to field) or where you want to do it (e.g., working in the screen painter). Once you identify the function key, locate its description in this chapter.

Note: Some keys appear in more than one group.

Cursor Movement Functions

[Beginning of Line]
[Down]
[End of Line]
[First Line]
[Last Line]
[Left]
[Next Block]
[Next Field]
[Next Page]
[Next Record]
[Previous Block]
[Previous Field]
[Previous Page]
[Previous Record]
[Right]
[Scroll Down]
[Scroll Left]
[Scroll Right]
[Scroll Up]
[Up]

Editing Functions

[Beginning of Line]
[Clear Field]
[Copy]
[Cut]
[Delete Backward]
[Delete Line]
[Delete Record]
[Delete Character]
[Down]
[End of Line]
[First Line]
[Last Line]
[Insert Line]
[Insert Record]
[Insert/Replace]
[Left]
[Paste]
[Right]
[Search]

General Design Functions

[Accept]
[Cancel]
[Change Display Type]
[Copy Object]
[Help]
[Insert Record]
[List]
[Menu]
[Navigate]
[Next Record]
[Previous Record]
[Print]
[Refresh]
[Screen Painter]
[Select]
[Show Keys]
[Zoom In]
[Zoom Out]

Screen Painting Functions

[Copy]
[Cut]
[Define Field]
[Down]
[Draw Box/Line]
[Left]
[Next Block]
[Next Field]
[Next Page]
[Next Record]
[Paste]
[Previous Block]
[Previous Field]
[Previous Page]
[Previous Record]
[Resize Field]
[Right]
[Select]
[Undo]
[Up]

CHAPTER

8 HELP SYSTEM

In SQL*Forms, online help is product documentation that you access from the SQL*Forms (Design) interface. This documentation pertains only to the SQL*Forms (Design) component, and it primarily covers material about each individual element in the design interface (e.g., the Generate option, the Field menu, the Edit Object dialog box). This material is online because you will want to access it while you are actually using the product, and an online system provides the quickest and most convenient access.

Much of the material covered in the help system does not appear in the rest of the SQL*Forms documentation set. As a result, the SQL*Forms online help system is an integral part of the SQL*Forms documentation set with which you should be familiar.

This chapter discusses the following topics:

- organization of the help material
- features of the help system

This chapter also discusses online information that is separate from the help system. It does not cover how to invoke online help.

Related Topics: Help system, invoking 2-7
Documentation set ii

Organization of the Help Material

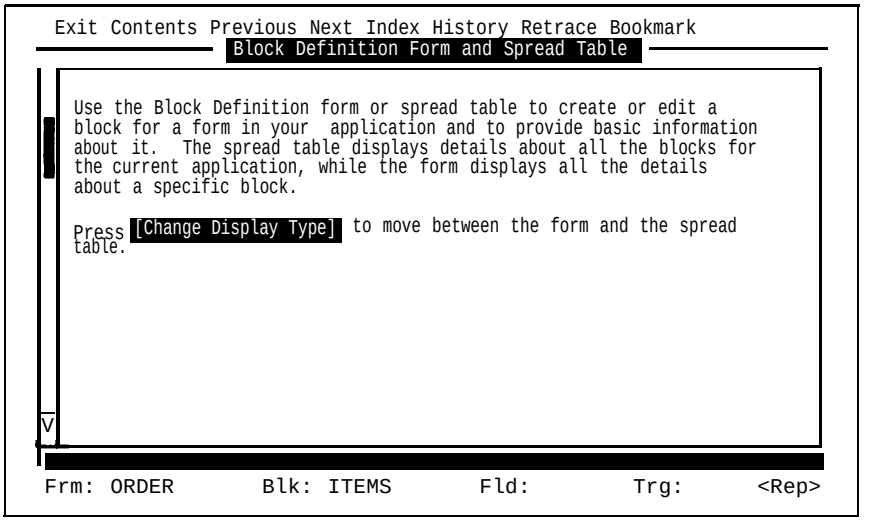
The information in the help system is arranged by topics. While there is a logical order to the way topics appear in the help system (as reflected in the table of contents for the help system), it does not affect the way you access the information. Certainly you can read through all of the help topics sequentially, but it is more likely that you will access screens of information randomly, as you need this information while building an application. The presence of an index and a keyword system further facilitate the random access of information.

The individual help screens that describe specific interface elements present information in a standard order. First, the help screen tells you where you are in the design interface and what you can do in that particular context. Then the help screen gives you a more general discussion of concepts related to the interface element.

The help system presents the information this way so that you can quickly access the information that will allow you to make a design decision and continue with your work. Only if you require more conceptual material do you need to go past the first screen or two.

Figure 8-1 shows the help screen for the Block Definition form and spread table.

FIGURE 8-1
Sample Help Screen



Features of the Help System

The online help system for SQL*Forms (Design) provides the following features to enhance the access of help information:

- help on help
- main menu for help
- contents for help
- index for help
- keywords in help screens
- history panel
- retrace feature
- bookmark feature
- error help

Help on Help

A general overview, called "About the Help System," describes how to use the help system. Read this information before you use the system for the first time.

Main Menu for Help

The main menu for the help system controls the major functions of the help system. It appears whenever you invoke the help system, and it remains on the screen as long as you are in the help system. This menu controls the system through the following options:

Exit	Returns you to the point at which you left SQL*Forms to enter the help system.
Contents	Displays the table of contents for the help system.
Previous	Displays the previous subtopics within each main topic.
Next	Displays the next subtopics within each main topic.
Index	Displays the index for the help system.
History	Displays the History panel.
Retrace	Displays the help screen that you were in prior to the current help screen.
Bookmark	Assigns a bookmark to the current help topic.

Choose a main menu option by moving the cursor to the option and pressing [Select].

Contents for Help

The table of contents in the help system lists the topics that are discussed in the body of the help system. Instead of using page numbers to access the information on these topics, you can use function keys to select any topic from the interface. You will see the requested information immediately.

Index for Help

The index in the help system lists all topics that are covered in the body of the help system. This listing is alphabetical and contains cross-references. Instead of using page numbers to access the information on the listed topics, you can use function keys to select the entry from the interface. You will see the requested information immediately.

Keywords in Help Screens

Keywords are highlighted words or groups of words that are embedded in the text of each help screen. These keywords identify topics in the help system that are related to the current topic. You can display the help screen for a keyword topic by moving the cursor to the keyword and pressing [Select].

History Panel

The History panel displays a list of help system topics that mark the path you have taken through the help system during the current help session. This feature not only allows you to see where you have been and in what order, but it allows you to select a topic in the History panel and immediately go back to the corresponding help screen.

When you use the retrace feature, SQL*Forms removes from the list the entry for the current screen (i.e., the screen you are leaving).

Retrace Feature

The retrace feature of the help system allows you to step back sequentially through the help screens that you have looked at during the current help session. When you select Retrace from the help system main menu or press [Retrace] from within the help system, SQL*Forms displays the help screen that you were in prior to the current help screen. The actual “backwards” path that you take is defined by the list in the History panel.

Related Topics: [Retrace] 7-12

Bookmark Feature

The bookmark feature of the help system allows you to assign a placeholder at the current help topic. If you select Bookmark from the help system main menu and then return to SQL*Forms (i.e., exit the help system), you can return to the marked screen immediately by pressing [Bookmark].

Related Topics: [Bookmark] 7-4

Error Help

Help screens that explain SQL*Forms (Design error messages are available through the help system. If you press [Help] when an error message is displayed on the message line, SQL*Forms displays help on that error and not on the interface item where the cursor is located.

Related Topics: Messages, SQL*Forms (Design) E-11

PART

III OBJECTS

CHAPTER

9 FORMS

This chapter discusses the SQL*Forms form object. Specifically, it covers the following topics:

- defining forms
- form characteristics
- non-enterable form type

This chapter does not discuss how to create or modify forms.

Related Topics: Creating a form 2-5

Defining Forms

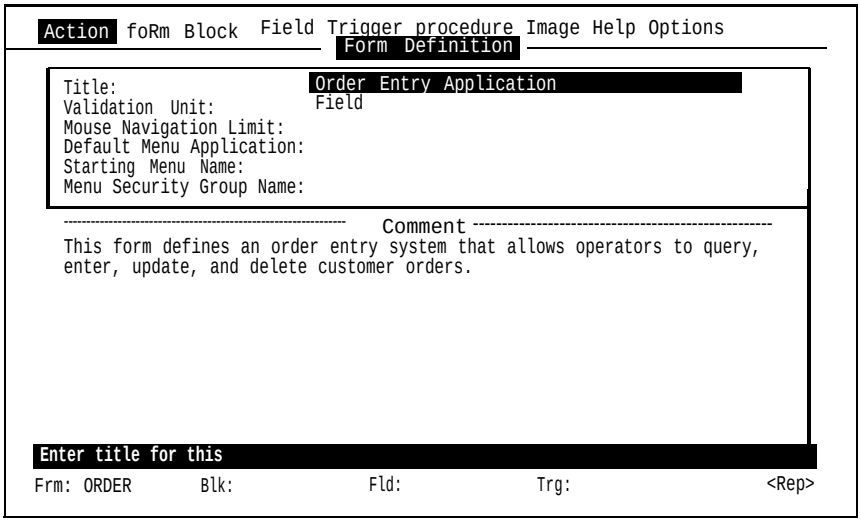
You create and modify forms in the SQL*Forms (Design) interface through the Form Definition form.

Form Definition Form

The Form Definition form is an element in the SQL*Forms (Design) interface where you define form characteristics. This form shows information for one form at a time.

Figure 9-1 shows the Form Definition form with sample data.

FIGURE 9-1
Form Definition Form



Location: To reach the Form Definition form from the main menu, complete the following steps:

1. Make sure you have form context (You can have block and field context, but it is not necessary.)
2. Select the foRm option.

You can also display the Form Definition form by pressing:

- [Zoom Out] from the Block Definition form
- [Navigate] from any form or spread table and specifying a form context

Form Characteristics

The process of defining a form consists of setting each of the form's characteristics. These characteristics indicate the following things about a form:

- basic information, including how ORACLE refers to the form
- how the form interacts with SQL*Menu upon execution
- how an operator can use a mouse with the form
- the validation unit

This section presents form characteristics in alphabetical order. Note that the names of the form characteristics correspond exactly with the names of the elements in the Form Definition form. For example, the Title field in the Form Definition form is the element that defines the Title form characteristic.

You assign form characteristics by typing values into field elements or scroll region elements.

Comment

The Comment form characteristic specifies general information about the form. Comment is your tool to comment on the purpose or contents of the form. This feature is especially useful to other designers who use your forms or applications and to you when you debug your applications.

Required/Optional: optional

Default Menu Application

The Default Menu Application form characteristic indicates the name of the SQL*Menu application that SQL*Forms should invoke whenever this form is executed in SQL*Forms (Run Form). You can replace this menu application during form execution through the REPLACE_MENU packaged procedure.

Default Value: The Default Menu Application characteristic defaults to DEFAULT, which specifies the default menu application that SQL*Forms provides.

Required/Optional: optional

Related Topics: REPLACE_MENU, packaged procedure 16-27
SQL*Menu, connecting to 25-3

Menu Security Group Name

The Menu Security Group Name form characteristic indicates the privilege group that SQL*Menu should use for the default menu application. SQL*Menu uses this setting to determine which menu options it should display when the starting menu displays.

Required/Optional: optional

Related Topics: SQL*Menu, connecting to 25-3

Mouse Navigation Limit

The Mouse Navigation Limit form characteristic specifies the mouse navigation limit that the form uses when an operator runs the form in SQL*Forms (Run Form). The mouse navigation limit determines what field(s) an operator can make current with a mouse. The available mouse navigation limits are field, record, block, and form:

Field	When you make the mouse navigation limit the field, an operator can only make current the current field. This setting has the effect of not using a mouse at all for navigation.
Block	When you make the mouse navigation limit the block, an operator can make any field in the current block the current field by clicking on that field.
Record	When you make the mouse navigation limit the record, an operator can make any field in the current record the current field by clicking on that field.
Form	When you make the mouse navigation limit the form, an operator can make any field in the current form the current field by clicking on that field.

Default Value: Form

Related Topics: Bit-mapped environment D-4

Starting Menu Name

The Starting Menu Name form characteristic indicates the name of the menu that SQL*Forms should display when this form is executed in SQL*Forms (Run Form). This menu must belong to the SQL*Menu application specified by the Default Menu Application form characteristic. The operator cannot go above the starting menu in the menu hierarchy for the application.

Required/Optional: optional

Related Topics: SQL*Menu, connecting to 25-3

Title

The Title form characteristic indicates the text that displays in the Title field of the Define Form form. You, or other designers, see the form title when modifying an existing form. Also, if the form has a menu application associated with it, this title appears in the menu's title bar. Use the Title characteristic to provide a clearer description of the form than the Form Name characteristic can indicate.

The field in the SQL*Forms (Design) interface that is associated with the Title characteristic accepts up to 255 characters and does accept blank characters.

Default Value: The Title characteristic defaults to the name that you assign to the form in the New Form dialog box.

Required/Optional: optional

Validation Unit

The Validation Unit form characteristic specifies the validation unit that the form uses when an operator runs the form in SQL*Forms (Run Form). The available validation units are form, block, record, and field.

Default Value: The Validation Unit characteristic defaults to NULL. This value specifies that SQL*Forms should use the default validation unit as determined by your system's environment. Specifically, if you are working in a block mode environment, NULL is equivalent to the block validation unit. In all other environments, NULL is equivalent to the field validation unit.

Related Topics: Setting the validation unit in block mode D-2
Validation unit 3-7

Non-enterable Form Type

A non-enterable form consists of all non-enterable fields (and, therefore, all non-enterable blocks). As a result, the cursor cannot enter the form.

A non-enterable form is populated only by the Pre-Form trigger. The Pre-Form trigger and the Post-Form trigger are the only triggers a non-enterable form can execute.

Designers usually use non-enterable forms to display data or messages or to calculate data.

Related Topics: Non-enterable block 11-19
Non-enterable field 12-27

CHAPTER

10 PAGES

This chapter discusses the SQL*Forms page object. Specifically, it covers the following topics:

- defining pages
- page characteristics

This chapter does not discuss how to modify or manipulate pages.

Defining Pages

You create and manipulate pages in the SQL*Forms (Design) interface through the Page Definition form and the Page Definition spread table.

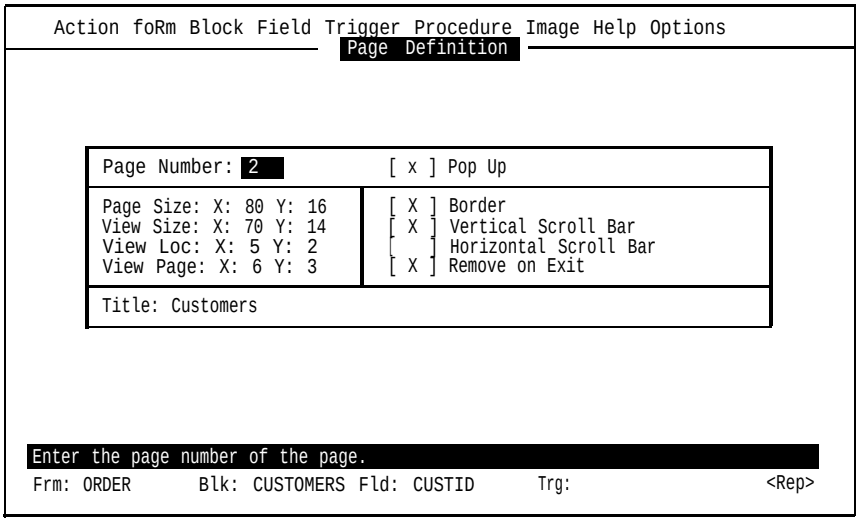
Page Definition Form

The Page Definition form is an element in the SQL*Forms (Design) interface where you create and modify pages. This form allows you to see all information for a single page at one time.

Note: Use the Page Definition spread table if you want to look at information on more than one page at the same time.

Figure 10-1 shows the Page Definition form with sample data.

FIGURE 10-1
Page Definition Form



Location: To reach the Page Definition form from the main menu, complete the following steps:

1. Make sure you have form context. (You can have block and field context, but it is not necessary.)
2. Select the Image option.
3. Select the Modify option from the Image menu.
4. If the Page Definition spread table displays, press [Change Display Type].

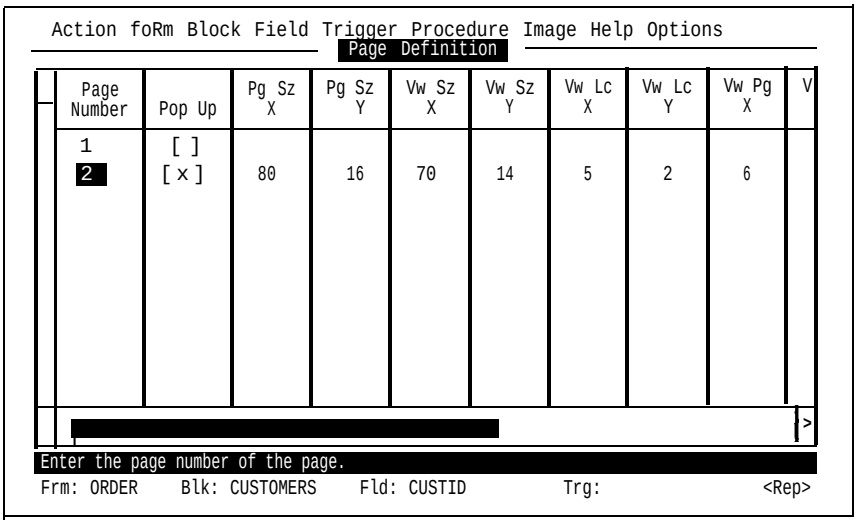
Page Definition Spread Table

The Page Definition spread table is an element in the SQL*Forms (Design) interface where you create or modify pages. This spread table allows you to see information on more than one page at a time. Note that all of the elements in this spread table do not appear at the same time.

Note: Use the Page Definition form if you want to look at all of the information for one page at the same time.

Figure 10-2 shows a portion of the Page Definition spread table with sample data.

FIGURE 10-2
Page Definition Spread Table



Location: To reach the Page Definition spread table from the main menu, complete the following steps:

1. Make sure you have form context. (You can have block and field context, but it is not necessary.)
2. Select the Image option.
3. Select the Modify option from the Image menu.
4. If the Page Definition form displays, press [Change Display Type].

Page Characteristics

The process of defining a page consists of setting each of the page's characteristics. These characteristics indicate the following things about a page:

- basic information, including the page number
- how a page appears and how it behaves

This section presents page characteristics in alphabetical order. Note that the names of the page characteristics correspond exactly with the names of the elements in the Page Definition form and Page Definition spread table. For example, the Page Number field in the Page Definition form is the element that defines the Page Number field characteristic.

You assign page characteristics through different interface elements and indifferent ways:

Interface Element	*To Assign a Field Characteristic*
check box	turn the characteristic on or off
field	type in a value

You should be aware of these assignment differences because they affect how this section describes the characteristics. For example, a characteristic assigned from a check box has a description for Default Setting (which will be either "on" or "off"). Alternatively, a characteristic assigned from a field might have a Required/Optional description, indicating whether you have to fill out this field element when defining a field object.

Related Topics: Interface elements 5-1

Border

The Border page characteristic specifies whether or not the view of a page should have a border.

If you assign the Border characteristic to a page, SQL*Forms makes the following adjustments to the view of the page.

- SQL*Forms includes the border characters within the specified page size. This ensures that the view of the page does not overlay any additional screen space. As a result, less of the page displays than would if you did not set the Border characteristic.
- SQL*Forms repositions the actual contents of the view one character space down and to the right on the screen. This ensures that the View Page coordinates-which define the initial location on the page of the upper left corner of the view on that page-are displayed and not overlayed by the border.

Note: SQL*Forms ignores the Border characteristic unless the Pop Up page characteristic is turned on for the page.

Required/Optional: optional

Default Setting: off

Horizontal Scroll Bar

The Horizontal Scroll Bar page characteristic specifies whether or not the view of a page should have a horizontal scroll bar on the bottom margin.

If you assign the Horizontal Scroll Bar characteristic to a page, SQL*Forms includes the scroll bar characters within the specified page size. This ensures that the view of the page does not overlay any additional screen space. As a result, less of the page displays than would if you did not set the Horizontal Scroll Bar characteristic.

Note: SQL*Forms ignores the Horizontal Scroll Bar characteristic unless the Pop Up and Border page characteristics are turned on for the page.

Required/Optional: optional

Default Setting: off

Page Number

The Page Number page characteristic specifies the number that SQL*Forms uses to identify the page.

Required/Optional: required

Page Size (Pg Sz)

The Page Size page characteristic consists of the following two settings

X	Determines the width of the page in character spaces.
Y	Determines the height of the page in character spaces.

Whatever you define as the size of a page, SQL*Forms always anchors the upper left corner of the page at the upper left corner of the screen (location 1,1).

Note: SQL*Forms ignores any value you specify for the Page Size characteristic unless the Pop Up page characteristic is turned on for the page. However, SQL*Forms does use a default value for Page Size even when the Pop Up characteristic is turned off.

Restrictions: A page can be a minimum of 1 character wide and 1 character high and a maximum of 255 characters wide and 255 characters high.

Required/Optional: The Page Size characteristic is required if the Pop Up characteristic is turned on.

Default Value: The X and Y settings default to the width and height of the screen as specified by the terminal definition that you used to create the form. These default settings do not display in the SQL*Forms (Design) interface.

Pop Up

The Pop Up page characteristic specifies whether or not a page should display as a pop-up page. When this characteristic is turned on, it makes the following page characteristics active:

- Border
- Horizontal Scroll Bar
- Remove on Exit
- Title
- Vertical Scroll Bar
- View Loc
- View Page
- View Size

Turning the Pop Up characteristic on also allows you to specify a value for the Page Size characteristic other than the default value.

Required/Optional: optional

Default Setting: off

Related Topics: Pop-up objects, page 13-5

Remove on Exit

The Remove on Exit page characteristic specifies whether or not the view of a page should disappear after the cursor navigates out of the page.

Note that a page with the Remove on Exit characteristic can be displayed when it is not the current page via the SHOW PAGE packaged procedure. The page disappears once the page has been the current page and the cursor leaves the page.

Note: SQL*Forms ignores the Remove on Exit characteristic unless the Pop Up page characteristic is turned on for the page.

Required/Optional: optional

Default Setting: off

Related Topics: SHOW_PAGE, packaged procedure 16-33

Title

The Title page characteristic specifies the title that appears centered at the top of a view on a pop-up page.

Restrictions: The following restrictions apply when you assign the Title page characteristic:

- You can specify a title that is up to 255 characters long.
- SQL*Forms ignores the Title characteristic unless the Pop Up and Border page characteristics are turned on for the page.

Required/Optional: optional

Default Value: NULL

Vertical Scroll Bar

The Vertical Scroll Bar page characteristic specifies whether or not the view of a page should have a vertical scroll bar on the left or right margin of the view. (Whether the scroll bar appears on the left or right depends on the device that an operator runs the form on).

If you assign the Vertical Scroll Bar characteristic to a page, SQL*Forms makes the following adjustments to the view of the page:

- SQL*Forms includes the scroll bar characters within the specified page size. This ensures that the view of the page does not overlay any additional screen space. As a result, less of the page displays than would if you did not set the Vertical Scroll Bar characteristic.
- SQL*Forms repositions the actual contents of the view one character space to the right on the screen. This ensures that the View Page coordinates-which define the initial location on the page of the upper left corner of the view on that page-are displayed and not overlayed by the border.

Note: SQL*Forms ignores the Vertical Scroll Bar characteristic unless the Pop Up and Border page characteristics are turned on for the page.

Required/Optional: optional

Default Setting: off

View Loc (VW Lc)

The View Loc page characteristic consists of two settings (X and Y) that determine the initial location on the screen of the upper left corner of a view on the page. That is, they determine where on the screen the operator sees the view of the page.

Both settings can be changed during form execution by the ANCHOR_VIEW packaged procedure.

Note: SQL*Forms ignores the View Loc characteristic unless the Pop Up characteristic is turned on for the page.

Restrictions: You must position the view of the page so that the entire view fits on the screen.

Required/Optional: The View Loc characteristic is required if the Pop Up page characteristic is turned on.

Default Value: 1 (X),1 (Y)

Related Topics: ANCHOR_VIEW, packaged procedure 16-5

View Page (VW Pg)

The View Page page characteristic consists of two settings (X and Y) that determine the initial location on the page of the upper left corner of a view on that page. That is, they determine what part of the page the operator sees through the view.

Both settings can be changed during form execution by the MOVE_VIEW packaged procedure.

Note: SQL*Forms ignores the View Page characteristic unless the Pop Up characteristic is turned on for the page.

Note: If a pop-up page does not contain the first enterable field, the form context will override the page settings of a pop-up when the operator navigates to page.

Restrictions: You must position the view so that the entire view is on the page that it is viewing.

Required/Optional: The View Page characteristic is required if the Pop Up page characteristic is turned on.

Default Value: 1 (x),1 (Y)

Related Topics: MOVE_VIEW, packaged procedure 16-23

View Size (VW Sz)

The View Size page characteristic consists of the following two settings:

X	Determines the initial width of the view on a page.
Y	Determines the initial height of the view on a page.

Both settings can be changed during form execution by the RESIZE_VIEW packaged procedure.

Note: SQL*Forms ignores the View Size characteristic unless the Pop Up characteristic is turned on for the page.

Restrictions: When you define a page with the View Size characteristic, the following restrictions apply:

- The width and height of the view of a page cannot be larger than the width and height of the page.
- The view of a page must be at least 1 character space wide and 1 character space high.
- The view of a page must be sized so that the entire view fits on the terminal screen.

Required/Optional: The View Size characteristic is required if the Pop Up page characteristic is turned on.

Default Value: The X and Y settings default to the width and height of the page.

Related Topics: RESIZE_VIEW, packaged procedure 16-28

CHAPTER

11

BLOCKS

This chapter discusses the SQL*Forms block object. Specifically, it covers the following topics:

- defining blocks
- block characteristics
- block types
- master-detail relationship

This chapter does not discuss how to create, modify, or manipulate blocks.

Related Topics: Creating a block, default block 2-4

Defining Blocks

You create and modify blocks in the SQL*Forms (Design) interface through the Block Definition form, the Block Definition spread table, and the Default Block form.

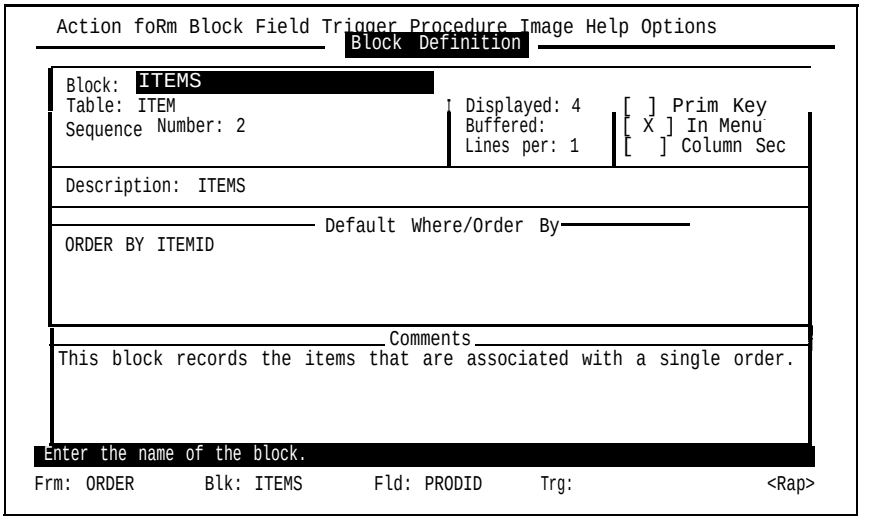
Block Definition Form

The Block Definition form is an element in the SQL*Forms (Design) interface where you create, modify, or delete blocks. This form allows you to see information for a single block at one time.

Note: Use the Block Definition spread table if you want to look at information on more than one block at the same time. To save time, use the Default Block form to create your blocks and then use the Block Definition form or Block Definition spread table to modify them.

Figure 11-1 shows the Block Definition form with sample data.

FIGURE 11-1
Block Definition Form



Location: To reach the Block Definition form from the main menu, complete the following steps:

1. Make sure you have form context (You can have block and field context, but it is not necessary.)
2. Select the Block option.

3. Select the Modify option from the Block menu.
4. If the Block Definition spread table displays, press [Change Display Type].

You can also display the Block Definition form by pressing:

- [Zoom In] from the Form Definition form
- [Zoom Out] from the Field Definition form
- [Navigate] from any form or spread table and specifying a block context. (This might display the Block Definition spread table.)

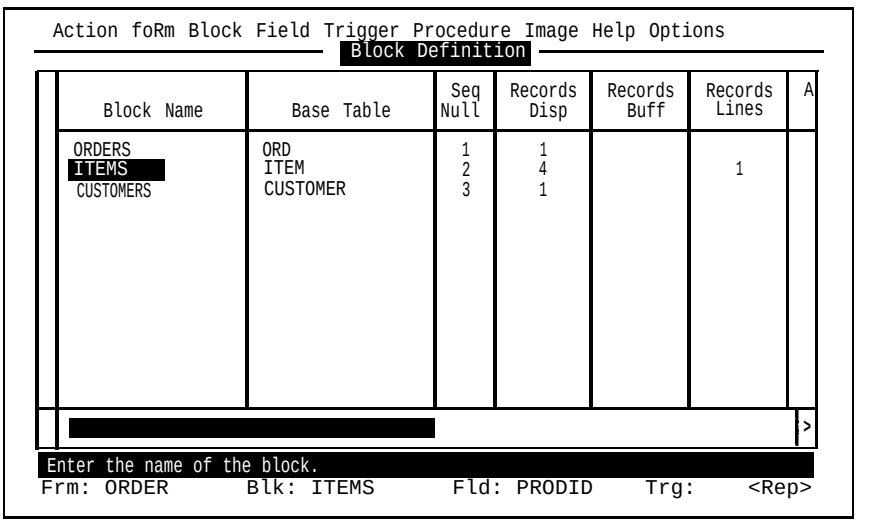
Block Definition Spread Table

The Block Definition spread table is an element in the SQL*Forms (Design) interface where you create, modify, or delete blocks. This spread table allows you to see information on more than one block at a time. Note that all of the elements in this spread table do not appear at the same time.

Note: Use the Block Definition form if you want to look at all of the information for one block at the same time. To save time, use the Default Block form to create your blocks and then use the Block Definition form or Block Definition spread table to modify them.

Figure 11-2 shows a portion of the Block Definition spread table with sample data.

FIGURE 11-2
Block Definition Spread Table



Location: To reach the Block Definition spread table from the main menu, complete the following steps:

1. Make sure you have form context (You can have block and field context, but it is not necessary.)
2. Select the Block option.
3. Select the Modify option.
4. If the Block Definition form displays, press [Change Display Type].

You can also display the Block Definition spread table by pressing:

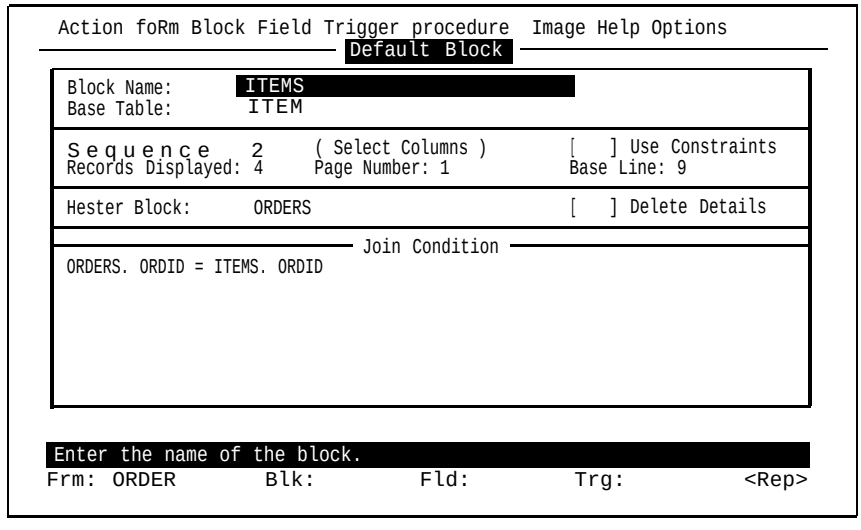
- [Zoom In] from the Form Definition form
- [Zoom Out] from the Field Definition spread table
- [Navigate] from any form or spread table and specifying a block context (This might display the Block Definition form.)

Default Block Form

The Default Block form is an element in the SQL*Forms (Design) interface where you create default blocks. Specifically, this form is your access to the default block facility. It allows you to enter information for a single block.

Figure 11-3 shows the Default Block form with sample data.

FIGURE 11-3
Default Block Form



The Default Block form is the only element in the SQL*Forms (Design) interface where you can automatically establish a master-detail relationship between blocks. That is why you can set the following block characteristics only from the Default Block form:

- Delete Details
- Join Condition
- Master Block

You can also explicitly set the Base Line and Use Constraints block characteristics only in the Default Block form.

You cannot explicitly set the following block characteristics from the Default Block form:

- Column Sec
- Comment
- In Menu
- Menu Description
- Array Size
- Default Where/Order By
- Primary Key
- Records Buffered
- Lines Per Record

Each of these characteristics assumes a default value or setting when you create a block through the default block facility.

Although its use is not required, you should use the Default Block form to create all blocks that contain base table fields. Then you can use the Block Definition form or Block Definition spread table to look at or modify information for existing blocks.

Location: To reach the Default Block Definition form from the main menu, complete the following steps:

1. Make sure you have form context. (You can have block and field context, but they will have no effect)
2. Select the Block option.
3. Select the Default option from the Block menu.

Default Block Facility

The SQL*Forms default block facility creates blocks based on a few criteria that you provide through the Default Block form. From the information you provide, this facility creates a block and the fields that make up that block, and gives the block and fields default characteristic settings. As a result, the default block facility saves you a great amount of time in defining the block and its fields. Once you create a default block, you can easily modify it to the exact specifications you desire.

The default block facility is also your means of automatically establishing a master-detail relationship between two blocks.

Block Characteristics

The process of defining a block consists of setting each of the block's characteristics. These characteristics indicate the following things about a block:

- basic information, including where the block is sequenced in a form
- how the block appears and how it behaves
- if the block is involved in a master-detail relationship

This section presents block characteristics in alphabetical order. Note that the names of the block characteristics correspond exactly with the names of the elements in the Block Definition form, Block Definition spread table, and Default Block form. For example, the Block Name field in the Block Definition form is the element that defines the Block Name field characteristic.

Note: The full names of the block characteristics sometimes are abbreviated in the design interface. In this section, these abbreviations follow the full names in parentheses (e.g., Sequence Number (Seq Num)).

You assign block characteristics through different interface elements and indifferent ways:

Interface Element	*To Assign a Block Characteristic*
button	press [Select] to view additional elements
check box	turn the characteristic on or off
field	type in a value
scroll region	type in a value

You should be aware of these assignment differences because they affect how this section describes the characteristics. For example, a characteristic assigned from a check box has a description for Default Setting (which will be either “on” or “off’). Alternatively, a characteristic assigned from a field might have a Required/Optional description, indicating whether you have to fill out this field element when defining a field object.

Related Topics: BLOCK_CHARACTERISTIC packaged function 17-4
Interface elements 5-1

Array Size

The Array Size block characteristic specifies the maximum number of records that SQL*Forms (Run Form) can fetch from the database at one time.

When SQL*Forms attempts to fetch a large number of rows from the database, an array size of 1 provides the fastest “perceived" response time. That is, SQL*Forms displays the first fetched record sooner than it would if the array size was greater than 1 because SQL*Forms fetches and displays 1 record at a time. By contrast, an array size of 10 fetches up to 10 records before displaying any of them. This improved response time when fetching a large number of rows is only a perceived improvement because a larger array size reduces overall processing time by making fewer calls to the database for records.

Restrictions: You cannot set the array size to less than 1. You can set it to a value greater than the number of records displayed for the block (as specified by the Records Displayed block characteristic).

Required/Optional: required

Default Value: The array size defaults to the number of records that the block can display at one time.

Base Line

The Base Line block characteristic indicates the page line number of the line directly above the line on which you want the block to begin. The actual line you indicate displays as a blank line separating the block information from existing page information.

For example, if you want the block to start online 10 (i.e., the tenth line from the top of the page), set the base line to 9. Alternatively, if line 10 is the first available line, set the baseline to 10.

Note: You only indicate a base line for a block when you create a block through the Default Block form.

Restrictions: The following restrictions apply when you assign the Base Line characteristic:

- You cannot set the baseline to be a line that an existing field or constant text uses.
- You cannot set the baseline so as to cause part of the block to display off of the page.

Required/Optional: The Base Line characteristic is required only when you create a block through the Default Block form.

Default Value: The Base Line Characteristic defaults to the number of the first blank line after all existing blocks and constant text on the page.

Base Table (Table)

The Base Table block characteristic specifies the name of the database table or view to which the block corresponds. You can choose a table or view name from the Table Name list of values.

Note: The block does not have to correspond to any table. Also, the block can contain fields that do not correspond to columns in the base table.

Required/Optional: If you are creating a block through the Default Block form, the Base Table characteristic is required. Otherwise, it is optional.

Block Name (Block)

The Block Name block characteristic specifies the name that SQL*Forms uses to identify the block. This is the name you use when you refer to the block within triggers.

The default block facility uses the block name as the constant text label for the block. Once SQL*Forms creates the block, you can change this label in the screen painter without affecting the block name.

Required/Optional: required

Related Topics: Naming conventions 20-2

Column Sec

The Column Sec (column security) block characteristic indicates whether or not SQL*Forms should enforce update privileges on a column-by-column basis for the block's base table.

When the Column Sec characteristic is turned on, SQL*Forms enforces the update privileges that are defined in the database for the current operator. If an operator does not have update privileges on a particular column in a base table, SQL*Forms makes the corresponding field non-updatable-for this operator only-by turning off the Update Allowed field characteristic.

When the Column Sec characteristic is turned off, SQL*Forms does not enforce the defined update privileges. Rather, the database enforces the privileges when SQL*Forms attempts to update the database.

Default Setting: off

Related Topics: Updates 319

Comment

The Comment block characteristic specifies general information about the block. Comment is your tool to describe the purpose or characteristics of the block. This feature is especially useful to other designers who use your objects or applications and to you when you maintain and debug your applications.

The comment you enter can be of any length.

Required/Optional: optional

Default Where/Order By

The Default Where/Order By block characteristic specifies conditions for the block and for retrieving all records in a query. Specifically, you can define one or both of the following clauses of a SQL SELECT statement:

WHERE clause	Sets the conditions, for the block, that a row must meet in order for SQL*Forms to retrieve it during a query. You can specify field names in this clause.
ORDER BY clause	Determines the order, for the block, in which SQL*Forms presents those records selected by a query. You cannot specify field names in this clause, only column names.

Note: You cannot reference a global variable in either clause.

For example, consider the following Default Where/Order By statement from an order tracking system:

```
WHERE SHIPDATE IS NULL
ORDER BY ORDERDATE
```

In this example, the WHERE clause specifies that SQL*Forms only retrieve records for which the SHIPDATE value is NULL; that is, it retrieves only those orders that have not been shipped. The ORDER BY clause arranges the selected records from the lowest (earliest) date to the highest (latest) date.

Note that the Default Where/Order By statement specifies the base conditions for record retrieval. An operator can further specify the conditions-or override the conditions completely-by doing a selected query and/or using complex search criteria. To continue the above example, if the operator specifies a particular CLIENT_ID, SQL*Forms retrieves all unshipped orders for that client only.

Note: If you do not specify a Default Where/Order By statement for a block, and an operator does not alter the search criteria, SQL*Forms retrieves all appropriate records in an unspecified order.

Required/Optional: optional

Related Topics: Record selection criteria 3-26

Default Where/Order By Syntax

Since WHERE and ORDER BY are standard SQL clauses, they must follow the rules for such constructs as specified in the *SQL Lanuage Reference Manual.*

In SQL*Forms, you can use either clause by itself or both clauses together:

- If you use both clauses, the WHERE clause must appear first.
- If you use only the ORDER BY clause, you do not need to specify the words ORDER BY. SQL*Forms will automatically append these words to the statement.

Delete Details

The Delete Details block characteristic specifies the form's behavior when a record is deleted or cleared from the block's master block:

- When this characteristic is turned on, it specifies that if a record in the block's master block is deleted then SQL*Forms deletes all records that correspond to that master record. The rows associated with the detail records are not actually deleted from the database until SQL*Forms verifies at commit time that the master record is deleted.
- When this characteristic is turned off, it specifies that SQL*Forms should not allow the deletion of a master record if any rows associated with its detail records exist (i.e., are not deleted).

The Delete Details characteristic is only effective if the block is a detail block in a master-detail relationship with another block.

Default Setting: off

Related Topics: Master-detail relationship 11-20

In Menu

The In Menu block characteristic indicates whether or not the block menu displays the menu description for this block (as specified by the Menu Description block characteristic). As a result, this characteristic determines if an operator can select this block from the block menu.

Default Setting: on

Related Topics: Block menu 11-13

Join Condition

The Join Condition block characteristic specifies the relationship that links a detail record in the block with a master record in the block's master block. You can specify a join condition with the following entries:

- a field name that exists in both the master block and the detail block
- an equating condition of two field names, where one field exists in the master block and the other field exists in the detail block
- a combination of field names and equating conditions

Note that you cannot specify a constant in the join condition.

For example, assume that you want to link the block to its master block through the ORDID field that is common to both blocks. To do this, specify the following join condition:

```
ORDID
```

If you wanted to link the block to its master block through a number of fields, you could write a join condition in the following form:

```
block1.field1  = block2.field1 AND block1.field2  = block2.field2
```

Note: The Join Condition characteristic is only effective if the block is a detail block in a master-detail relationship with another block. You can specify the Join Contition characteristic only when you create a block through the Default Block form.

Restrictions: When you specify the Join Condition characteristic for a block, you also must specify the Master Block block characteristic.

Required/Optional: optional

Related Topics: Master-detail relationship 11-20

Lines Per Record (Records Lines)

The Lines Per Record block characteristic specifies the number of lines SQL*Forms reserves for each record in the block. SQL*Forms only recognizes this characteristic for multi-record blocks. For single-record blocks, SQL*Forms automatically inserts a blank line after each line that contains a field.

Make the lines per record greater than 1 when either of the following conditions apply

- all of the fields in a record will not fit on one line

or

- you want to put one or more blank lines between records

If you modify a multi-record block by adding blank lines between records, and that change would cause all or part of a record not to display, SQL*Forms automatically adjusts the number of records displayed.

Optional/Required: optional

Default Value: 1

Master Block

The Master Block block characteristic specifies another block as the current block's master block, and thereby designates the current block as a detail block.

Note: You can specify the Master Block characteristic only when you create a block through the Default Block form.

Restrictions: When you specify the Master Block Characteristic for a block, you also must specify the Join Condition block characteristic.

Required/Optional: optional

Related Topics: Master-detail relationship 11-20

Menu Description (Description)

The Menu Description block characteristic indicates the text that displays in the block menu. The menu description does not display in the block menu if:

- the In Menu block characteristic is turned off

or

- you clear the menu description (i.e., define it as blank characters)

Optional/Required: optional

Default Value: The Menu Description default is the block name (as specified by the Block Name block characteristic).

Page Number

The Page Number block characteristic specifies the page on which you want to place a block's fields. You only specify a page number when you create a block through the Default Block form.

Note: You can modify a block so that some of its fields appear on other pages. However, when you create a block through the Default Block form, all of the fields must initially fit on one page.

Required/Optional: A page number is required when you create a block through the Default Block form.

Default Value: The default value of Page Number depends on whether or not you have entered the screen painter during the current SQL*Forms (Design) session:

- If you have not entered the screen painter, Page Number defaults to 1.
- If you have entered the screen painter, Page Number defaults to the page number of the last page that you displayed in the screen painter.

Primary Key (Prim Key)

The Primary Key block characteristic indicates that any record inserted or updated in the block must have a unique characteristic in order to be committed to the database. This characteristic ensures that the record does not duplicate any row already in the block's base table.

The Primary Key block characteristic is only effective if you use it in conjunction with the Primary Key field characteristic for one or more fields in the block.

Note: The Primary Key characteristic exists in SQL*Forms for compatibility with previous versions of SQL*Forms. Oracle Corporation recommends that you use a unique index or a UNIQUE column constraint on a column (or columns) in the database table instead of using the Primary Key field characteristic and the Primary Key block characteristic. The Primary Key characteristics are less efficient and only prevent duplicate rows from being committed via the form.

Restrictions: A block with the Primary Key characteristic must also have the Base Table block characteristic assigned.

Default Setting: off

Related Topics: Primary Key, field characteristic 12-19

Records Buffered (Records Buff)

The Records Buffered block characteristic specifies the maximum number of records that SQL*Forms buffers in memory when a query is performed on the block. SQL*Forms buffers any additional records to a temporary file on disk.

You can improve processing speed by increasing the number of records buffered. Likewise, you can save memory by decreasing the number of records buffered.

Restrictions: The lowest number of records buffered that you can specify is the number of records displayed (i.e., the value set for the Records Displayed characteristic) plus a small constant. If you set a lower number, SQL*Forms gives you an error when you attempt to accept the value.

Required/Optional: optional

Default Value: The number of records buffered defaults to the number of records displayed (i.e., the value set for the Records Displayed characteristic) plus a small constant. This default value does not display in the Records Buffered field (i.e., a blank field invokes the default value).

Related Topics: -b switch, SQL*Forms (Run Form) 23-16

Records Displayed (Records Disp)

The Records Displayed block characteristic specifies the maximum number of records that can display in the block at one time.

Restrictions: If you indicate a number of records displayed that would cause the block not to fit on the page, SQL*Forms warns you and does not let you modify or create the block.

Required/Optional: required

Default Value: 1

Select Columns

The Select Columns block characteristic specifies the columns in the base table that SQL*Forms should use to create the block. You only define this characteristic (via the Select Columns button) when you are creating a block through the Default Block form.

By selecting or deselecting from the available columns, you can choose to build a default block with all, some, or none of the columns in the base table:

all columns	By default, SQL*Forms selects all columns in a base table.
some columns	If you deselect a column, the values in that column are not changed when you update rows through the block. When you insert new rows through the block, the deselected column assumes a NULL value. **Note:** If operators need to insert records through a form, a deselected column must not be defined as NOT NULL in the database.
no columns	If you deselect all columns, the block you create will contain no fields.

Default Value: By default, the Select Columns characteristic selects all columns in the base table.

Sequence Number (Seq Num)

The Sequence Number block characteristic indicates where the block is in the block sequence of the form. Provided that the block can be entered (i.e., at least one field in the block can be entered), the sequence number indicates the order in which the cursor moves to the block during SQL*Forms (Run Form) navigation.

Required/Optional: required

Default Value: The sequence number defaults to the next available block sequence number in the form.

Use Constraints

The Use Constraints block characteristic indicates that SQL*Forms should enforce the column and table constraints that are specified in the database. These constraints are explained in the *SQL Language Reference Manual*.

SQL*Forms complies with the constraints by writing triggers that enforce the conditions at run time:

- If the constraint conditions specify only one column, SQL*Forms writes an On-Validate-Field trigger on the appropriate field that enforces the constraint.
- If the constraint conditions specify more than one column, SQL*Forms writes an On-Validate-Record trigger on the block that enforces the constraint.
- If the constraint conditions specify a default value for a column, SQL*Forms writes an On-New-Record trigger for the appropriate field.
- If the constraint conditions specify a primary key-foreign key relationship, SQL*Forms writes a Key-DELREC trigger on the block that has the primary key.

SQL*Forms also turns on the Primary Key field characteristic for the fields that correspond to primary key columns and turns on the Primary Key block characteristic for the block.

You can only specify the Use Constraints block characteristic when you are creating a block through the Default Block form.

Default Setting: off

Related Topics: Key-DELREC trigger 14-9
On-New-Record trigger 14-17
On-Validate-Field trigger 14-18
On-Validate-Record trigger 14-19

Block Types

"Block types" can refer to common combinations of block characteristics or to block compositions of all one field type.

It is important to note that SQL*Forms does not recognize block types per se, but rather it recognizes a block's individual characteristic settings and the settings of its fields. The following block types are explained here, and used throughout this document, as an abbreviated way of describing a block:

- control block
- detail block
- master block
- multi-record block
- non-enterable block
- single-record block

Control Block

A control block is not associated with any table in the database. It is made up of fields that are not base table fields, such as temporary data fields.

You can use control blocks to display information to operators, such as calculated values or messages. Such blocks can also be used to receive data from operators (through its fields) that the application uses to perform calculations, queries, or other actions.

Required Settings: The Base Table block characteristic must be unassigned, and all fields in the block must have the Base Table characteristic turned off.

Related Topics: Creating a block, control block 2-3

Detail Block

A detail block is associated with a master block in a master-detail relationship. The detail block displays detail records associated with master records in the master block.

Designers usually make detail blocks multi-record blocks so the block can show as many detail records as possible.

Required Settings: The Master Block block characteristic and the Join Condition block characteristic must be filled in.

Related Topics: Master-detail relationship 11-20

Master Block

A master block is associated with a detail block in a master-detail relationship. The master block displays master records associated with detail records in the detail block.

Related Topics: Master-detail relationship 11-20

Multi-record Block

A multi-record block can display more than one record at a time.

Required Settings: The Records Displayed block characteristic is set to a value greater than one.

Non-enterable Block

A non-enterable block consists of all non-enterable fields. As a result, the cursor cannot enter any field in the block.

A non-enterable block has the following traits:

- populated only by triggers
- ignored during navigation
- processed only at commit time

Note that a failure results if you reference a non-enterable block in a GO_BLOCK packaged procedure.

Related Topics: Non-enterable field 12-27

Single-record Block

A single-record block can display only one record at a time.

Required Settings: The Records Displayed block characteristic is set to 1.

Master-Detail Relationship

A master-detail relationship allows you to maintain, on your form, the connection between two business entities that have a "primary to foreign key" relationship between them. For example, a department has many employees. When you create a form that uses the DEPT and EMP tables, you can make DEPT the master block and EMP the detail block You can then relate the two blocks through a join condition that establishes a primary key in the master block's base table and a foreign key in the detail block's base table. This arrangement allows you to maintain the relationship on the form.

You can specify a master-detail relationship between blocks when you create a block through the Default Block form. Creating a master-detail relationship between blocks allows you to accomplish the following actions:

- coordinate querying between the blocks
- establish the conditions by which the master and the detail blocks are joined and how these blocks behave when records in either block are deleted
- make the detail block display records associated with a master record in the master block
- set the value of the Enforce Foreign Key field characteristic of the foreign key field in the detail block to the value of the primary key field of the master block

When you specify a master-detail relationship, SQL*Forms generates certain triggers automatically. These triggers are associated with a master block that has one detail block with which it is associated. If you add more detail blocks to a master block, SQL*Forms automatically edits these triggers to enforce the new relationships.

When you specify a master-detail relationship, SQL*Forms alters the default characteristics of any foreign key fields in the following ways:

- turns off the following field characteristics (that normally default to "on"):
  - Displayed
  - Input Allowed
  - Query Allowed
  - Update Allowed
- assigns a value to the Sequence Number characteristic so as to sequence the foreign key fields as the last fields in the block
- assigns a value to the Screen Position characteristic so as to position the foreign key fields as the last fields in the block

SQL*Forms also does not assign constant text labels (which are normally based on the Field Name characteristic) to foreign key fields. The intent of these changes to field characteristics is to hide foreign key fields from the operator since the information in the fields are most likely to be displayed in the master block.

Related Topics: Delete Details block characteristic 11-11
Join Condition block characteristic 11-11
Master Block block characteristic 11-13

Help for Defining Master-Detail Relationships

The SQL*Forms (Design) interface provides a list of values on the Master Block field that will help you define master-detail relationships in your forms. This list of values displays the name of all blocks in the form that are potential master blocks for the current block. (A potential master block is a block that has a foreign key relationship with the current block.) The list of values also specifies the relationship between the current block and the potential master block.

If you select a block from the list of values, SQL*Forms writes the block name into the Master Block field and also writes the appropriate join condition into the Join Condition scroll region.

Limits in a Master-Detail Relationship

The automatic master-detail relationship has the following limitations:

- There can be only one master block for a detail block.
- There can be more than one master-detail relationship in a form.
- There can be more than one detail block for a master block.

Note: The master-detail characteristics for the block do not appear in either the Block Definition form or spread table.

Triggers in a Master-Detail Relationship

When you specify a master-detail relationship between two blocks, SQL*Forms automatically builds the triggers and form-level procedures to enforce that relationship. The triggers are associated with the master block (i.e., they are written at the block level).

This section lists and describes each of the triggers and form-level procedures that is associated with a master-detail relationship.

Function Key Triggers:

KEY-CLRBLK

```
CLEAR_ master_block_name_ DETAILS;
CLEAR_BLOCK ;
EXCEPTION WHEN FORM_ TRIGGER_ FAILURE THEN NULL ;
```

This trigger clears both the master and detail blocks.

KEY-CLRREC

```
CLEAR_ master_block_name_ DETAILS;
CLEAR_RECORD ;
QUERY_ master_block_name_ DETAILS;
EXCEPTION WHEN FORM_TRIGGER_FAILURE THEN NULL ;
```

This trigger first clears the detail block and then the record. It then queries the detail block to show the details that correspond to the new current row in the master block.

KEY-CREREC

```
CLEAR_ master_block_name_ DETAILS;
CREATE_ RECORD ;
EXCEPTION WHEN FORM_ TRIGGER_ FAILURE THEN NULL ;
```

This trigger clears the detail block and creates a new record in the master block.

KEY-DELREC

If the Delete Details block characteristic is turned off, the KEY-DELREC trigger takes the following form:

```
DECLARE
        CURSOR DETAIL_ CUR IS SELECT * FROM
            detail_ base_ table
        WHERE primary_ key_ field = :master_block.
            primary_key_field;
        DETAIL_ROW    DETAIL_CUR%ROWTYPE;
BEGIN
        IF NOT DETAIL_CUR%ISOPEN THEN
            OPEN DETAIL_CUR;
        END IF;
        FETCH DETAIL_CUR INTO DETAIL_ROW;
        IF (DETAIL_ROW. primary_key_field  IS NOT
NULL)THEN      MESSAGE ('Cannot delete master
record when            matching detail records exist. ') ;
        RAISE  FORM_TRIGGER_FAILURE;
        END IF;
END ;
CLEAR_ master_block_name_ DETAILS;
DELETE_RECORD  ;
QUERY_ master_block_name_ DETAILS;
EXCEPTION WHEN FORM_TRIGGER_FAILURE THEN NULL;
```

This trigger first checks the detail block for any records that correspond to the master block. If it finds any rows that correspond to the master block, it displays a message. If there are no corresponding records, this trigger clears the detail block and deletes the record in the master block. It then queries the detail block again to show the details that correspond to the new current row in the master block.

If the Delete Details block characteristic is turned on, the KEY-DELREC trigger takes the following form:

```
CLEAR_ master_block_name_ DETAILS;
DELETE_RECORD  ;
QUERY_ master_block_name_ DETAILS;
```

This trigger clears the detail block and deletes the record in the master block. (The records in the detail block are deleted by the Pre-Delete trigger.) It then queries the detail block again to show the details that correspond to the new current row in the master block.

KEY-DOWN

```
CLEAR_ master_block_name_ DETAILS;
DOWN ;
QUERY_ master_block_name_ DETAILS;
EXCEPTION WHEN FORM_ TRIGGER_ FAILURE THEN NULL ;
```

This trigger first clears the detail block and then goes to the next master record, but the cursor stays in the same field. It then queries the detail block again to show the details that correspond to the new current row in the master block.

KEY-ENTQRY

```
CLEAR_ master_block_name_ DETAILS;
ENTER_QUERY ;
QUERY_ master_block_name_ DETAILS;
EXCEPTION WHEN FORM_ TRIGGER_ FAILURE THEN NULL ;
```

This trigger first clears the detail block and then puts the form in Enter Query mode. It then queries the detail block again to show the details that correspond to the new current row in the master block.

KEY-EXEQRY

```
CLEAR_ master_block_name_ DETAILS;
EXECUTE_QUERY ;
QUERY_ master_block_name_ DETAILS;
EXCEPTION WHEN FORM_ TRIGGER_ FAILURE THEN NULL ;
```

This trigger first clears the detail block and then executes the query. It then queries the detail block again to show the details that correspond to the new current row in the master block.

KEY-NXTREC

```
CLEAR_ master_block_name_ DETAILS;
NEXT_RECORD ;
QUERY_ master_block_name_ DETAILS;
EXCEPTION WHEN FORM_TRIGGER_FAILURE THEN NULL;
```

This trigger first clears the detail block and then goes to the next master record. It then queries the detail block again to show the details that correspond to the new current row in the master block.

KEY-NXTSET

```
CLEAR_ master_block_name_ DETAILS;
NEXT_SET;
QUERY_ master_block_name_ DETAILS;
EXCEPTION WHEN FORM_TRIGGER_FAILURE THEN NULL;
```

This trigger first clears the detail block and then goes to the next master record. It then queries the detail block again to show the details that correspond to the new current row in the master block.

KEY-PRVREC

```
CLEAR_ master_block_name_ DETAILS;
PREVIOUS_RECORD ;
QUERY_ master_block_name_ DETAILS;
EXCEPTION WHEN FORM_TRIGGER_FAILURE THEN NULL;
```

This tigger first clears the detail block and then goes to the previous master record. It then queries the detail block again to show the details that correspond to the new current row in the master block.

KEY-SCRDOWN

```
CLEAR_ master_block_name_ DETAILS;
SCROLL_DOWN;
QUERY_ master_block_name_ DETAILS;
EXCEPTION WHEN FORM_TRIGGER_FAILURE THEN NULL;
```

This trigger first clears the detail block and then perfom the SCROLL_DOWN packaged procedure. It then queries the detail block again to show the details that correspond to the new current row in the master block.

KEY-SCRUP

```
CLEAR_ master_block_name_ DETAILS;
SCROLL_UP;
QUERY_ master_block_name_ DETAILS;
EXCEPTION WHEN FORM_TRIGGER_FAILURE THEN NULL;
```

This trigger first clears the detail block and then performs the SCROLL_UP packaged procedure. It then queries the detail block again to show the details that correspond to the new current row in the master block.

KEY-UP

```
CLEAR_ master_block_name_ DETAILS;
UP;
QUERY_ master_block_name_ DETAILS;
EXCEPTION WHEN FORM_TRIGGER_FAILURE THEN NULL;
```

This trigger first clears the detail block and then goes to the previous master record, but the cursor stays in the same field. It then queries the detail block again to show the details that correspond to the new current row in the master block.

Pre-Delete Trigger:

PRE-DELETE

This trigger is used if you want details to be deleted along with the master (i.e.,you turn on the Delete Details block characteristic).

```
LOCK TABLE detail_base_table IN SHARE UPDATE
MODE ;
DELETE FROM detail_base_table
        WHERE detail_base_table.foreign_key =
            :master_block.primary_key_field;
```

This trigger deletes all the rows in the table associated with the detail block that correspond to the master row that is to be deleted.

Form-level Procedures:

CHECK_ PACKAGE_ FAILURE

```
PROCEDURE CHECK_ PACKAGE_ FAILURE IS
BEGIN
        IF NOT FORM_ SUCCESS THEN
            RAISE  FORM_TRIGGER_FAILURE;
        END IF;
END ;
```

This form-level procedure is called by the form-level procedures CLEAR_ *master_block_ name_* DETAILS and QUERY_ *master_block_ name_* DETAILS. It checks to see if a packaged procedure called by a master-detail trigger has failed. If no packaged procedure has failed, then it has no effect.

If a packaged procedure has failed, then it ensures that the trigger which called it also fails.

CLEAR_ *master_ block_name_* DETAILS

```
PROCEDURE CLEAR_ master_block_name_  DETAILS IS
BEGIN
        IF (NAME IN ('master_block.primary_key_field')
                IS NOT NULL) THEN
            GO_BLOCK ('detail-block') ;
            CHECK_ PACKAGE_ FAILURE;
            IF  :SYSTEM.BLOCK_STATUS = 'CHANGED' THEN
                CLEAR_BLOCK (ASK_COMMIT);
                IF  :SYSTEM.BLOCK_STATUS = 'CHANGED'THEN
                    GO_BLOCK ('master-block');
                    RAISE  FORM_TRIGGER_FAILURE;
                END IF;
            END IF;
            CLEAR_BLOCK ;
            GO_BLOCK ('master_block');
        END IF;
END ;
```

This user-named trigger is called by a number of key triggers. It first checks to see if the master block has a current row. If the master block does not have a current row, that is, it is empty, then this trigger does nothing.

If the master block does have a current row, the procedure navigates to the detail block, clears the detail block, and navigates back to the master block.

QUERY_ *master_block_name_* DETAILS

```
PROCEDURE QUERY_ master_block_name_ DETAILS IS
BEGIN
        IF (NAME_IN ('master_block.primary_key_field')
                IS NOT NULL) THEN
            GO_ BLOCK ('detail_block');
            CHECK_ PACKAGE_ FAILURE ;
            EXECUTE_QUERY ;
            GO_BLOCK ('master_block');
        END IF;
END ;
```

This user-named trigger is called by a number of key triggers. It first checks to see if the master block has a current row. If the master block does not have a current row, that is, it is empty, then this trigger does nothing.

If the master block does have a current row, the procedure navigates to the detail block, queries the detail block to show the details that correspond to the current row in the master block, and navigates back to the master block.

Deletion in a Master-Detail Relationship

When you delete a detail block, the following rules apply:

- If a detail block is deleted, SQL*Forms does not automatically delete the triggers that apply to that block, as they are associated with the master block.
- If the detail block is deleted, SQL*Forms does not automatically delete the master block-the block stays defined.

CHAPTER

12 FIELDS

This chapter discusses the SQL*Forms field object. Specifically, it covers the following topics:

- defining fields
- field characteristics
- field types

This chapter does not discuss how to create, modify, or manipulate fields.

Related Topics: Creating a field 2-4
Modifying a field 2-7

Defining Fields

You create and modify fields in the SQL*Forms (Design) interface through the Field Definition form and the Field Definition spread table.

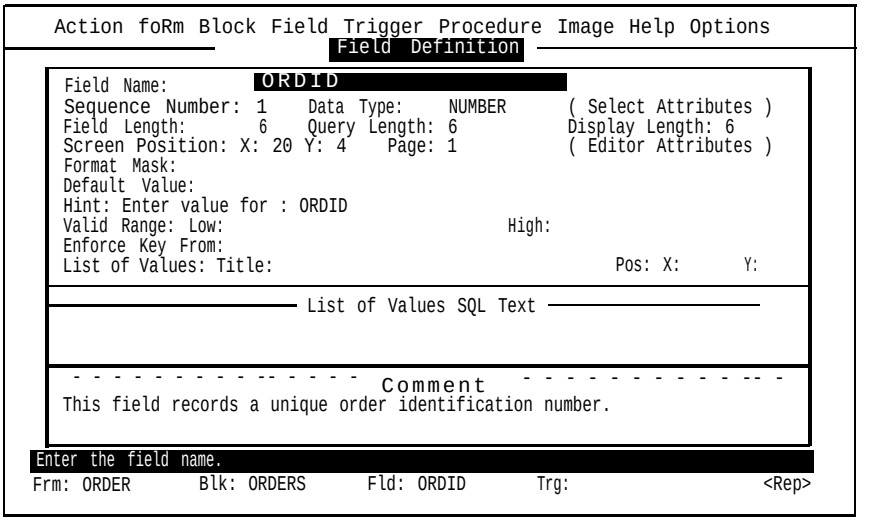
Field Definition Form

The Field Definition form is an element in the SQL*Forms (Design) interface where you create, modify, or delete fields. This form allows you to see all information for a single field at one time.

Note: Use the Field Definition spread table if you want to look at information on more than one field at the same time.

Figure 12-1 shows the Field Definition form with sample data.

FIGURE 12-1
Field Definition Form



Location: To reach the Field Definition form from the main menu, complete the following steps:

1. Make sure you have form and block context. (You can have field context, but it is not necessary.)
2. Select the Field option.
3. Select the Modify option from the Field menu.
4. If the Field Definition spread table displays, press [Change Display Type].

You can also display the Field Definition form by pressing:

- [Zoom In] from the Block Definition form
- [Zoom Out] from the Trigger Definition form (when you have field context)
- [Navigate] from any form or spread table and specifying a block and field context. (This might display the Field Definition spread table.)

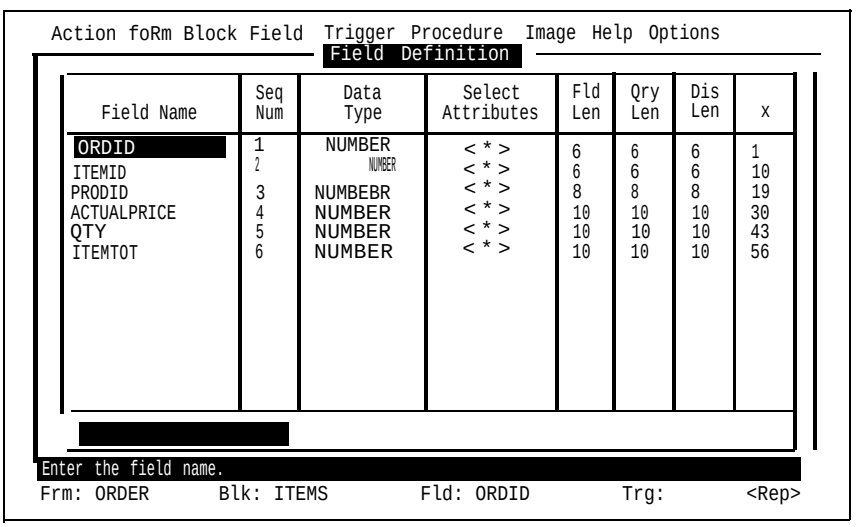
Field Definition Spread Table

The Field Definition spread table is an element in the SQL*Forms (Design) interface where you create, modify, or delete fields. This spread table allows you to see information on more than one field at a time. Note that all of the elements in this spread table do not appear at the same time.

Note: Use the Field Definition form if you want to look at all of the information for one field at the same time.

Figure 12-2 shows a portion of the Field Definition spread table with sample data.

FIGURE 12-2
Field Definition Spread Table



Location: To reach the Field Definition spread table from the main menu, complete the following steps:

1. Make sure you have form and block context. (You can have field context, but it is not necessary.)
2. Select the Field option.
3. Select the Modify option.
4. If the Field Definition form displays, press [Change Display Type].

You can also display the Field Definition spread table by pressing:

- [Zoom In] from the Block Definition spread table
- [Zoom Out] from the Trigger Definition spread table (when you have field context)
- [Navigate] from any form or spread table and specifying a block and field context. (This might display the Field Definition form.)

Field Characteristics

The process of defining a field consists of setting each of the field's characteristics. These characteristics indicate the following things about a field:

- basic information, including the field's location in a form and sequence in a block
- how an operator can interact with a particular field
- the type of data that an operator can enter in a field and the format in which data must be entered

This section presents field characteristics in alphabetical order. Note that the names of the field characteristics correspond exactly with the names of the elements in the Field Definition form and Field Definition spread table. For example, the Field Name field in the Field Definition form is the element that defines the Field Name field characteristic.

Note: The full names of the field characteristics sometimes are abbreviated in the SQL*Forms (Design) interface. In this section, these abbreviations follow the full names in parentheses (e.g., Sequence Number (Seq Num)).

You assign field characteristics through different interface elements and indifferent ways:

Interface Element	*To Assign a Field Characteristic*
check box	turn the characteristic on or off
field	type in a value
scroll region	type in a value

You should be aware of these assignment differences because they affect how this section describes the characteristics. For example, a characteristic assigned from a check box has a description for Default Setting (which will be either "on" or "off'). Alternatively, a characteristic assigned from a field might have a Required/Optional description, indicating whether you have to fill out this field element when defining a field object.

Related Topics: FIELD_CHARACTERISTIC packaged function 17-8
Interface elements 5-1

Automatic Hint

The Automatic Hint field characteristic indicates that SQL*Forms displays the field's help message, or hint, whenever the cursor enters the field (provided that you have defined hint text for the field).

In a block mode environment, SQL*Forms always acts as though this characteristic is turned off.

Default Setting: off

Note: In the design interface, the Automatic Hint check box is associated with the Select Attributes button.

Related Topics: Hint Text field characteristic l2-16

Automatic Skip

The Automatic Skip field characteristic indicates that SQL*Forms moves the cursor to the next enterable field when the last position in the field is changed or entered. This characteristic is primarily a keystroke saver. Designers often use it with the Fixed Length characteristic so that as soon as an operator has entered the last required character, the cursor moves to the next applicable field.

When the cursor moves to a field as a result of the Automatic Skip characteristic, SQL*Forms does not fire the Key-NXTFLD trigger. This behavior is consistent with the fact that the operator did not press [Next Field].

Default Setting: off

Restrictions: A field with the Automatic Skip characteristic must have the Displayed and Input Allowed characteristics.

Note: In the design interface, the Automatic Skip check box is associated with the Select Attributes button.

Base Table

The Base Table field characteristic indicates that the field corresponds to a column in the base table of the block that contains the field.

Default Setting: on

Note: In the design interface, the Base Table check box is associated with the Select Attributes button.

Comment

The Comment field characteristic specifies general information about the field. Comment is your tool to describe the purpose or characteristics of the field. This feature is especially useful to other designers who use your objects or applications and to you when you debug and maintain your applications.

Required/Optional: optional

Data Type

The Data Type field characteristic determines what kinds of values SQL*Forms allows as input in the field and how SQL*Forms displays those values on the screen.

Note: For all data types, SQL*Forms removes any trailing blanks in a field value. Therefore, if an operator types blank spaces after a field value, SQL*Forms removes those blanks when it accepts the value. If an operator types a field value of only blank spaces, SQL*Forms removes the blanks and changes the field's value to NULL.

Required/Optional: required

Default Value: The default data type depends on how you create the field.

- If you create a new field in the screen painter, the default data type is CHAR.
- If you create a new field through the default block facility, the default data type is the same data type as the corresponding column in the database table. Therefore, the default values can be CHAR, DATE, LONG, or NUMBER.

The following pages describe valid data types.

Related Topics: Data types, evaluating values 12-25

ALPHA

ALPHA, or alphabetic, fields can contain any combination of letters (upper and/or lower case).

CHAR

CHAR, or character, fields can contain any combination of the following characters:

- letters (upper and/or lowercase)
- digits
- blank spaces
- special characters ($, #, @, and _)

DATE

DATE fields can contain a date, which refers to a DATE column in the database and which SQL*Forms processes as a true date (not a character string). The default format is DD-MON-YY. The following examples represent DATE field values:

23-OCT-82
01-JUN-87

You can display a DATE field in any other valid format by changing the field's format mask.

Related Topics: Format Mask field characteristic 12-15

DATETIME

DATETIME fields can contain a date, which refers to a DATE column in the database and which SQL*Forms processes as a true date (not a character string). The default format is DD-MON-YY HH:MI[:SS]. The following examples represent DATETIME field values:

18-OCT-88 10:02
31-DEC-88 23:59:59

EDATE

EDATE, or European date, fields can contain a date, which refers to a NUMBER column in the database. The default format is DD/MM/YY. The following examples represent EDATE field values:

23/10/82
01/06/87

Note: The EDATE data type exists in SQL*Forms for historical reasons. Oracle Corporation recommends that you use the DATE data type and the Format Mask characteristic to produce the European data format. Using a format mask in this situation allows you to reference a DATE column in the database instead of a NUMBER column.

Related Topics: Format Mask field characteristic 12-15

INT

INT, or integer, fields can contain any integer (i.e., a number without a decimal point and with or without a sign).

JDATE

JDATE, or Julian date, fields can contain a date, which refers to a NUMBER column in the database. The default format is MM/DD/YY. The following examples represent JDATE field values:

10/23/82
06/01/87

Note: The JDATE data type exists in SQL*Forms for historical reasons. Oracle Corporation recommends that you use the DATE data type and the Format Mask characteristic to produce the Julian data format. Using a format mask in this situation allows you to reference a DATE column in the database instead of a NUMBER column.

LONG

LONG fields can contain any combination of characters and up to 65,534 characters. Data in LONG fields is stored in ORACLE as variable-length character strings.

Note: It is illegal to directly query LONG fields. That is, you cannot reference a LONG field in the WHERE or ORDER BY clauses of any SELECT statement. To prevent such a query, the SQL*Forms (Design) interface prevents you from setting the Query Allowed field characteristic to a field with the LONG data type.

MONEY

MONEY fields can contain a number, with or without a sign or a decimal point, that represents a sum of money. The following examples represent MONEY field values:

10.95
0.99
10000.00
-15.47

Note: You can use a format mask to format the values in a MONEY field with a monetary sign (e.g., $).

NUMBER

NUMBER fields can contain any number, with or without a sign or a decimal point, in regular or scientific notation. NUMBER fields refer to NUMBER columns in the database and SQL*Forms processes their values as true numbers (not character strings). The following examples represent NUMBER field values:

10963
-15
0.01
1018.996
1.85E3

SQL*Forms does not allow commas to be entered into a number field (e.g., 99,999). However, you can display a number field with commas by using a format mask.

Related Topics: Format Mask field characteristic 12-15

R I N T

RINT, or right-integer, fields have the same characteristics as INT fields, except that values in RINT fields display as right-justified (instead of left-justified).

Note: The RINT data type is supported in SQL*Forms only for upward compatibility. Oracle Corporation recommends that you use the NUMBER data type and an appropriate format mask (e.g., 999).

RMONEY

RMONEY, or right-money, fields have the same characteristics as MONEY fields, except that values in RMONEY fields display as right-justified (instead of left-justified). The following examples represent RMONEY field values:

10.95
0.99
1000.00
-15.47

Note: The RMONEY data type is supported in SQL*Forms only for upward compatibility. Oracle Corporation recommends that you use the NUMBER data type and an appropriate format mask (e.g., 9999.99).

RNUMBER

RNUMBER, or right-number, fields have the same characteristics as NUMBER fields, except that values in RNUMBER fields display as right-justified (instead of left-justified). The following examples represent RNUMBER field values:

```
10963
   -15
  0.01
1018.996
 1.85E3
```

Note: The RNUMBER data type is supported in SQL*Forms only for upward compatibility. Oracle Corporation recommends that you use the NUMBER data type and an appropriate format mask (e.g., 999.999).

TIME

TIME fields can contain numbers (and colons), which refer to NUMBER columns in the database, not to DATE columns. The default format is HH24:MI[:SS]. The following examples represent TIME field values:

```
10:23:05
21:07:13
```

Note: You can represent time information in a DATETIME field.

Related Topics: DATETIME data type 12-7
Format Mask field characteristic 12-15

Default Value (Default)

The Default Value field characteristic indicates the value that SQL*Forms gives to the field for every new record. You can set the default value to be any of the following types of values:

- raw value (e.g., 340, 'JONES')
- system variable (e.g., $$DATE$$, $$TIMES$$)
- reference to another field (e.g., :ORDER.ORDERID, :DEPTNO)

Note: Default values that are field references should be references to fields in other blocks. The referenced field will be populated at the time SQL*Forms evaluates the reference.

- reference to a sequence number object in the database to automatically generate a sequence number (e.g., :SEQUENCE.ORDER_NUMBER.NEXTVAL)

  Note: To prevent "wasted' sequence numbers, use a Pre-Insert trigger to fetch the next sequence number just prior to inserting the record in the database.

  Note: For information on creating sequence number objects in ORACLE, refer to the *SQL Language Reference Manual.*

Required/Optional: optional

Display Length (Dis Len)

The Display Length field characteristic determines the number of character spaces that SQL*Forms displays for the field.

Note that if you make the field's display length less than the field length, the field becomes a scrolled field. If you modify the display length of a field-via the Field Definition form or spread table-to be greater than the field length, SQL*Forms gives you an error when you attempt to generate the form.

Required/Optional: optional

Default Value: If you create the field in the screen painter, the default value is equal to the length of the field you drew.

If you create the field through the default block utility, the default value depends on the field's default data type:

- For fields with the CHAR (character) or LONG data type, the default value is the same as the field length up to a maximum of 60. If the field length is greater than 60, SQL*Forms sets the display length to 60 and the field becomes a scrolled field.
- For fields with the DATE data type, the default value is 9.
- For fields with the NUMBER data type, the default value depends on the width of the corresponding column in the database table:
  - If the corresponding column in the database table has an explicit width, the default value is two more than that width. The two added character spaces allow room for a decimal point and a sign.
  - If the corresponding column in the database table has no explicit width, the default value is 10.

Related Topics: Scrolled field 12-28

Displayed

The Displayed field characteristic indicates that the field is displayed, or visible, when invoked by SQL*Forms (Run Form).

Default Setting: The Displayed characteristic defaults to on. However, if the field is part of the foreign key in a master-detail relationship, SQL*Forms changes the default setting to off.

Note: In the design interface, the Displayed check box is associated with the Select Attributes button.

Related Topics: Hidden field 12-27

Echo Input

The Echo Input field characteristic indicates that SQL*Forms always displays the field's current value. Without the Echo Input characteristic, the field's value would always display as blank characters.

For example, the Password field on the SQL*Forms login screen has the Echo Input characteristic turned off. When you type your password, the characters you type do not display.

Default Setting: on

Restrictions: A field with the Echo Input characteristic must have the Displayed characteristic.

Note: In the design interface, the Echo Input check box is associated with the Select Attributes button.

Enforce Foreign Key (Enforce Key)

The Enforce Foreign Key field characteristic indicates the source of the value that SQL*Forms uses to populate the field. This characteristic consists of two items, which identify the name of the block (Block) and the field (Field) from which SQL*Forms reads the value.

SQL*Forms enforces the foreign key relationship when

- a new record that contains the field is created
- the block that contains the field is queried
- a record that contains the field is inserted

SQL*Forms does not enforce the foreign key relationship when an existing record is updated.

Note: A field with the Enforce Foreign Key characteristic should have the Input Allowed characteristic turned off. While this arrangement is not required, it does prevent operators from de-enforcing the foreign key relationship.

Required/Optional: optional

Field Editor Location

The Location field characteristic consists of two settings that specify the horizontal (X) and vertical (Y) coordinates of the upper left corner of the field editor when the SQL*Forms (Run Form) field editor is invoked for this field.

Required/Optional: optional

Default Value: 1 (X),1 (Y)

Note: In the design interface, the Field Editor Location fields (X and Y) are associated with the Editor Attributes button.

Related Topics: Field editor 13-2

Field Length (Fld Len)

The Field Length field characteristic determines the maximum number of characters that can be entered into the field.

Note that if you make the field length greater than the display length, the field becomes a scrolled field. If you modify the display length of a field-via the screen painter-to be greater than the field length, SQL*Forms ignores the Field Length value and uses the Display Length value.

Restrictions: The following recommendations and restrictions apply when you assign the Field Length characteristic:

- For fields with the Base Table characteristic, do not make the field length any more than the corresponding table column width plus any additional space you need for format mask characters. You do not want operators to attempt to commit a value that is longer than the column allows.
- For fields with the NUMBER data type that handle scientific notation, make the field length relatively wide. This prevents the possible truncation of values by field widths narrower than the desired precision. Note that if a value is truncated and the operator commits changes to other fields in the form, SQL*Forms returns the truncated value to the database.
- For all fields except LONG fields, the maximum field length is 255 characters.
- For fields with the LONG data type, the maximum field length is 65,534 characters.

Required/Optional: optional

Default Value: If you create the field in the screen painter, the default value is equal to the length of the field you drew.

If you create the field through the default block utility, the default value depends on the field's default data type:

- For fields with the CHAR (character) data type, the default value is the width of the associated column in the database table.
- For fields with the DATE data type, the default value is 9.
- For fields with the LONG data type, the default value is 255.
- For fields with the NUMBER data type, the default value depends on the width of the corresponding column in the database table:
  - If the corresponding column in the database table has an explicit width, the default value is two more than that width. The two added character spaces allow room for a decimal point and a sign.
  - If the corresponding column in the database table has no explicit width, the default value is 44.

Related Topics: Format Mask field characteristic 12-15
Scrolled field 12-28

Field Name

The Field Name field characteristic specific the name that SQL*Forms uses to identify the field. This is the name (minus any block name qualifier) that you must use to refer to the field in triggers and in Enforce Foreign Key relationships.

- For base table fields, the field name must be identical to the name of the corresponding column in the database table.
- For non-base table fields, the field name can be any name you choose that follows object naming conventions and that is unique to the block.

The default block facility uses the field name as the constant text label for the field. Once SQL*Forms creates the field, you can change this label in the screen painter without affecting the field name. Note that if the field is part of the foreign key in a master-detail relationship, SQL*Forms does not create a constant text label for the field.

Note: SQL*Plus allows you to create columns whose names contain blank spaces. However, SQL*Forms does not allow you to create fields whose names contain blank spaces. Therefore, you cannot create a field that is based on a column whose name contains blank spaces. If you want to base a field on such a column, create a view of the associated table that removes the blank spaces from the column name.

Required/Optional: required

Default Value: If you create the field through the default block facility, the default field name is the name of the corresponding column in the database table.

Related Topics: Base table field 12-26
Naming conventions 20-2

Fixed Length

The Fixed Length field characteristic indicates that the field must be filled with a value that is exactly as long as the field length. For example, you could assign the Fixed Length characteristic to a field that holds a six digit product code so that SQL*Forms would invalidate shorter-and therefore invalid-tries.

If a field with the Fixed Length characteristic does not also have the Required characteristic, SQL*Forms will accept a NULL value for the field.

Default Setting: off

Restrictions: The following restrictions apply to fields that have the Fixed Length characteristic turned on:

- The field must have the Displayed and Input Allowed characteristics.
- If the field has the NUMBER or RNUMBER data type, it cannot hold values with leading zeros. If an operator enters such a value, SQL*Forms automatically removes the leading zeros and interprets the field as "not filled."

Note: In the design interface, the Fixed Length check box is associated with the Select Attributes button.

Related Topics: Field Length field characteristic 12-13

Format Mask

The Format Mask field characteristic specifies the display format mask for the field. This characteristic allows you to specify that SQL*Forms display and accept embedded characters in the field.

When you define a format mask for a field, you enter the embedded characters in quotes within the desired pattern. The following examples demonstrate this convention:

999''-''99''-"9999	This format mask displays the field value in the format for United States social security numbers. After the operator enters the nine digit number, SQL*Forms formats the value correctly in the field. Alternatively, SQL*Forms accepts and displays the mask characters if the operator enters them.
MONTH" " DDth", "YYYY	This format mask displays the field value in the specified date pattern (e.g., JANUARY lst, 1988), including the appropriate blank spaces and comma.

SQL*Forms considers embedded characters as separate from field values-even when the operator enters them-and does not consider those elements when collating field values. However, when you set a field's length (via the Field Length field characteristic), you must allow room for embedded characters.

Note: For lists and descriptions of the available format mask elements, refer to the *SQL Language Reference Manual.*

Required/Optional: optional

Related Topics: Format mask, status 3-11

Hint Text (Hint)

The Hint Text field characteristic specifies the hint that displays on the SQL*Forms (Run Form) message line. This hint displays under any of the following conditions:

- an operator presses [Help] while the cursor is in the field
- a trigger processes the HELP packaged procedure for the field
- the cursor enters the field and the Automatic Hint characteristic is set on

Required/Optional: optional

Default Value: If you create the field through the default block utility, SQL*Forms creates the following default help text:

```
Enter value for: fieldname
```

where *fieldname* is the name of the current field.

Input Allowed

The Input Allowed field characteristic indicates that an operator can move the cursor to the field (outside of query mode) and enter a value.

Default Setting: The Input Allowed characteristic defaults to on. However, if the field is part of the foreign key in a master-detail relationship, SQL*Forms changes the default setting to off.

Restrictions: A field with the Input Allowed characteristic must have the Displayed characteristic.

Note: In the design interface, the Input Allowed check box is associated with the Select Attributes button.

List of Values Pos (List)

The List of Values Pos field characteristic consists of two settings that specify the horizontal (X) and vertical (Y) coordinates of the upper left corner of the pop-up list of values for the field.

Note: If you create a V2 style list of values, SQL*Forms ignores this characteristic.

Required/Optional: optional

Default Value: 1 (X),1 (Y)

Related Topics: List of values, pop-up 13-4
List of values, V2 style 12-17

List of Values SQL Text

The List of Values SQL Text field characteristic specifies the SQL statement that retrieves values into the list of values. From these values, an operator can choose a value to select into a field (or fields).

Note that the operator does not have to select a value from a list of values, nor does the list of values necessarily show all possible values.

The syntax that you use for this field characteristic determines whether SQL*Forms creates the list of values as a V2 list of values or as a V3 list of values.

- A V2 list of values functions identically to list of values in previous versions of SQL*Forms.
- A V3 list of values is a pop-up object and functions identically to list of values in the design interface.

Syntax for V2 List of Values SQL Text

The List of Values SQL text for a V2 list of values must consist only of the names of one table and column.

For example, assume that you want to create a list of values that lets the operator select from a list of employee IDS. You might write the following List of Values SQL Text:

```
EMP.EMPNO
```

SQL*Forms constructs the rest of the appropriate SQL text in order to build a list of values that functions identically to list of values in previous versions of SQL*Forms.

Syntax for V3 List of Values SQL Text

The List of Values SQL text for a V3 list of values:

- must be a SELECT statement
- can select data into one or more fields or into no fields
- can select data into the pop-up list of values from more than one column and more than one table
- cannot reference a global variable

For example, assume that you want to create a pop-up list of values that lets the operator select from a list of managers' employee IDs. You also want the list of values to provide the operator with a list of the managers' office locations (as a cross reference from another table). You might write the following List of Values SQL text:

```
SELECT EMPNO, LOC
    INTO : MGR
    FROM EMP , DEPT
    WHERE EMP.DEPTNO = DEPT.DEPTNO
        AND JOB = 'MANAGER'
```

The database columns you reference in the SELECT clause must appear in the order that the INTO clause selects them into the fields. In the example, the EMPNO column corresponds to the MGR field. If you also wanted to select the LOC column value into the LOC field, you would have to reference the LOC field after the MGR field in the INTO clause (i.e., `INTO :MGR , :LOC)`.

As the example statement shows, you can reference columns in the SELECT clause that do not map to corresponding fields in the INTO clause. This arrangement lets you display information that might be useful to the operator but that you do not want written to a field. Note that when you establish this kind of display, the unmapped column references can only appear in the SELECT statement *after* all mapped columns (i.e., the INTO clause cannot skip column names in the SELECT statement when mapping these columns to fields). The following example will not work correctly because it breaks this rule by referencing the DEPTNO column before the LOC column:

```
SELECT EMPNO , DEPTNO , LOC
    INTO  : MGR, :LOC
    FROM EMP , DEPT
    WHERE EMP . DEPTNO = DEPT. DEPTNO
        AND JOB = 'MANAGER '
```

Required/Optional: optional

Related Topics: List of values, pop-up 13-4

List of Values Title

The List of Values Title field characteristic specifies the title that SQL*Forms places at the top of the pop-up list of values for the field.

Note: If you create a V2 style list of values, SQL*Forms ignores this characteristic.

Required/Optional: optional

Related Topics: List of values, pop-up 13-4
List of values, V2 style 12-17

Primary Key

The Primary Key field characteristic indicates that the field is a unique characteristic for a record or part of a unique characteristic (one of a set of fields with the Primary Key characteristic). To ensure that an inserted or updated record does not duplicate an existing record, give the critical field(s) the Primary Key characteristic and give the block the Primary Key characteristic.

For example, you can ensure that a payroll application does not allow more than one record with the same employee ID. To do this, give the EMPLOYEE_ID field the Primary Key field characteristic and give the fields block the Primary Key block characteristic.

Note that when a field is part of a primary key, SQL*Forms might allow that particular field's value to be duplicated. This condition is allowed because SQL*Forms checks that all of the fields in the Primary Key, *in combination,* are not duplicated.

The Primary Key characteristic exists in SQL*Forms for historical reasons. Oracle Corporation recommends that you use a unique index on a column in the database table instead of using the Primary Key characteristic on the associated field. A unique index enforces uniqueness for all applications that access the column and provides faster processing than a Primary Key setting.

For information on creating and using unique indexes, refer to the *SQL Language Reference Manual* and the *ORACLE RDBMS Database Administrator's Guide*.

Default Setting: off

Restrictions: A field with the Primary Key characteristic must also have the Base Table characteristic.

Note: In the design interface, the Primary Key check box is associated with the Select Attributes button.

Related Topic: Primary Key, block characteristic 11-14

Query Allowed

The Query Allowed field characteristic indicates that an operator can move the cursor to the field (while in query mode) and enter a query selection condition.

Default Setting: The Query Allowed characteristic defaults to on. However, if the field is part of the foreign key in a master-detail relationship, SQL*Forms changes the default setting to off.

Restrictions: The following restrictions apply when you assign the Query Allowed characteristic:

- A field with the Query Allowed characteristic also must have the Displayed characteristic.
- A field with the LONG data type cannot have the Query Allowed field characteristic.

Note: In the design interface, the Query Allowed check box is associated with the Select Attributes button.

Related Topic: LONG data type 12-8

Query Length (Qry Len)

The Query Length field characteristic determines the maximum number of characters that can be entered into the field as a query condition.

You can make a field's query length greater than the field's display length to allow the operator to enter complex query conditions. For example, a query length of 5 allows an operator to enter the query condition `!=500` in a field with a display length of 3.

Required/Optional: optional

Default Value: The default value is equal to the field length.

Restrictions: The following restrictions apply when you assign the Query Length characteristic:

- A field's query length cannot be less than its field length. If you attempt to accept a query length value that is too low, SQL*Forms gives you an error.
- The maximum query length is 255 characters.

Related Topics: Scrolled field 12-28

Required

The Required field characteristic indicates that the field cannot be NULL. For example, if you wanted to assure that no record in a payroll application was entered (or changed) without an employee ID, you could give the EMPLOYEE_ID field the Required characteristic.

Note that you can assign the Required characteristic to a field even if that field does not also have the Displayed and Input Allowed characteristics. If a field has this combination of characteristics, it is your responsibility to guarantee that the field is populated (via a trigger or some other means).

Default Setting: SQL*Forms turns on the Required characteristic if you create the field through the default block facility *and* the corresponding column in the database is marked NOT NULL. Otherwise, SQL*Forms sets the Required characteristic to off.

Note: In the design interface, the Required check box is associated with the Select Attributes button.

Screen Position

The Screen Position field characteristic consists of the following three settings:

Page	Determines the page number of the form on which the field appears.
X	Determines the horizontal coordinate on the page of the field's left endpoint.
Y	Determines the vertical coordinate on the page of the field's left endpoint.

Required/Optional: required

Default Value: If you create a field in the screen painter, Page defaults to the number of the page on which you create the field, and X and Y default to the coordinates of the field you drew.

If you create a field through the default block facility, Page defaults to the page number that you specify for the block (via the Page Number block characteristic), and X and Y default to coordinates determined by SQL*Forms.

Sequence Number (Seq Num)

The Sequence Number field characteristic indicates where the field is within the field sequence of the block. Provided that the field is enterable, the sequence number indicates the order in which the cursor moves to the field during SQL*Forms (Run Form) navigation.

Required/Optional: required

Default Value: The default value of Sequence Number depends on how the field is placed in the form:

- When you create a default block, SQL*Forms sequences the fields, starting with the number 1. SQL*Forms sequences the fields in the order that they appear in the screen painter (i.e., left to right, top to bottom).

  Note: Fields that are part of a foreign key in a master-detail relationship receive the highest default sequence numbers in the block.

- When you create a field in the screen painter, the sequence number defaults to the next available sequence number in the block.
- When you copy or reference a field, the sequence number defaults to the next available sequence number in the block.

When you cut and paste a field, SQL*Forms changes the default value of Sequence Number:

- When you cut and paste a single field in a block of fields and paste it back into the same block, the field's sequence number defaults to the number after the highest sequence number in the block. Also, SQL*Forms decrements the sequence numbers of all other fields in the block that had higher sequence numbers than the cut field.
- When you cut and paste all of the fields in a block and then paste them back into the same block, SQL*Forms re-sets the sequence numbers for all fields in the block, starting with the number 1. SQL*Forms sequences the fields in the order that they appear in the screen painter (i.e., left to right, top to bottom).
- When you cut and paste fields from a block and paste those fields into another block, SQL*Forms resets the sequence numbers for the pasted fields, starting with the number after the highest sequence number in the block in which you are pasting. SQL*Forms sequences the pasted fields in the order that they appear in the screen painter.

Update Allowed

The Update Allowed field characteristic indicates that an operator can change the value of the field in any record.

Default Setting: The Update Allowed characteristic defaults to on. However, if the field is part of the foreign key in a master-detail relationship, SQL*Forms changes the default setting to off.

Under the following conditions, SQL*Forms sets the default to off only for the current operator:

- the block that owns the field has the Column Sec block Characteristic turned on

and

- the operator does not have update privileges on the corresponding column

Restrictions: A field with the Update Allowed characteristic must have the Displayed and Input Allowed characteristics, and it cannot have the Update if Null characteristic.

Note: In the design interface, the Update Allowed check box is associated with the Select Attributes button.

Related Topics: Column Sec block characteristic 11-9

Update if Null

The Update if Null field characteristic indicates that an operator can change the value of the field only in those records retrieved by a query where the current value of the field is NULL.

Default Setting: off

Restrictions: A field with the Update if Null characteristic must have the Displayed characteristic and cannot have the Update Allowed characteristic.

Note: In the design interface, the Update if Null check box is associated with the Select Attributes button.

Uppercase

The Uppercase field characteristic indicates that lower case letters are converted into upper case letters as they are entered into the field. For example, you could give the STATE field the Uppercase characteristic to guarantee that operators entered state abbreviations in all capital letters.

Note that the Uppercase characteristic only affects values entered into the field through the form. It does not affect values entered through other forms or through triggers. It also does not affect the display of queried records; queried records display as they are stored in the database.

Default Setting: SQL*Forms sets the Uppercase characteristic to on if you create the field through the default block facility *and* the corresponding column in the database has the DATE data type. Otherwise, SQL*Forms sets the Uppercase characteristic to off.

Restrictions: A field with the Uppercase characteristic is only effective for fields with CHAR or ALPHA data types.

Note: In the design interface, the Uppercase check box is associated with the Select Attributes button.

Valid Range

The Valid Range field characteristic consists of two settings that determine the maximum (High) and minimum (Low) values that SQL*Forms allows in the field. SQL*Forms allows only the named values and any valid values between them.

The following table shows how SQL*Forms evaluates the values in fields with particular data types:

Data Type	*Method of Evaluation*
ALPHA	alphabetical according to your system's collation sequence
CHAR	alphabetical according to your system's collation sequence
DATE	chronological
DATETIME	chronological
EDATE	numerical
INT	numerical
JDATE	numerical
MONEY	numerical
NUMBER	numerical
RINT	numerical
RMONEY	numerical
RNUMBER	numerical
TIME	numerical

Required/Optional: optional

Related Topics: Data Type field characteristic 12-6

Word Wrap

The Word Wrap field characteristic indicates that the SQL*Forms (Run Form) field editor automatically wraps the text that an operator enters into this field from line to line. When this characteristic is turned off, SQL*Forms only breaks lines when the operator presses [Insert Line].

Default Setting: on

Note: In the design interface, the Word Wrap check box is associated with the Editor Attributes button.

Related Topics: Field editor 13-2

Field Types

"Field types" refers to common combinations of field characteristics. It is important to note that SQL*Forms does not recognize field types per se, but rather it recognizes a field's individual characteristic settings. The following field types are explained here, and used throughout this document, as an abbreviated way of describing a field's important characteristic settings:

- base table field
- control field
- enterable field
- hidden field
- look-up field
- non-enterable field
- scrolled field

Base Table Field

A base table field is associated with a column in the base table of the block that contains it. Depending on what field characteristics you set, an operator or a trigger can use a base table field to perform the following actions:

- query values from the associated column
- update data in the associated column
- insert data into the associated column
- delete data from the associated column

Required Settings: The Base Table characteristic is turned on.

Control Field

A control field is not associated with any column in a database table.

Designers usually use control fields to hold calculated values or messages. Such fields can also be used to receive data from operators, which the application uses to perform a calculation, a menuing action, a query, or some other action.

You must write a trigger to write data to the control field. SQL*Forms can derive that data from the following sources:

- operator input
- non-base tables
- other fields in the form
- other fields in forms called by the current form

Required Settings: The Base Table characteristic is turned off.

Enterable Field

An enterable field is a field to which the operator can move the cursor in normal mode or Enter Query mode.

Required Settings: Either the Input Allowed field characteristic or the Update Allowed field characteristic (or both) is turned on.

Hidden Field

A hidden field is not displayed in SQL*Forms (Run Form). A hidden field does display in the screen painter on the page with which it is associated.

Use hidden fields to hide proprietary information.

Required Settings: The Displayed characteristic is turned off.

Look-up Field

A look-up field is associated with a column in a table other than the base table of the block that contains it. Use a look-up field to display data (from another table) that is associated with other information in the block. That data appears for informational purposes only and, by default, the operator cannot query, update, insert, or delete data through the field. However, you can modify this behavior through triggers to make a look-up field appear like a database field to operators.

You must write a trigger to retrieve data from the database table and into the look-up field.

Required Settings: The Base Table characteristic is turned off.

Non-enterable Field

The cursor cannot enter a non-enterable field outside of query mode.

Required Settings: The Input Allowed characteristic is turned off.

Scrolled Field

A scrolled field has a display length that is less than its field length. As a result, the field can "hold" more than it can display. This can cause part of a value to scroll out of view. Use scrolled fields to hold long values (e.g., an address) when there is no room, or reason, to display the entire value.

Required Settings: The Display Length characteristic is set to a lower value than the Field Length characteristic.

Right-Justified, Scrolled Field

Avoid defining fields with right-justified data types (i.e., RINT, RNUMBER, RMONEY) as scrolled fields. Right-justified, scrolled fields might confuse operators by displaying aseemingly blank field when data resides outside of the visible field area.

If you create a form that contains a right-justified, scrolled field, SQL*Forms displays a warning (but continues processing) when you generate the form.

CHAPTER

13 POP-UP WINDOWS

Pop-up windows are screen areas that overlay all or a portion of the display screen when a form is run. For some pop-up windows, you can define the size and the location of the windows and additional display characteristics.

Each type of pop-up window has unique functionality. However, all types give you the ability to display some information to the operator while maintaining context.

This chapter covers the general characteristics of pop-up windows and details on the following types of pop-up windows:

- the pop-up field editor
- pop-up lists of values
- pop-up pages

This chapter does not cover how to create pop-up windows.

Related Topics: Pop-up windows, user exit routines 19-10

Pop-up Field Editor

The pop-up field editor allows operators to use time-saving editing features when modifying field data. When the field editor is invoked (by an operator or by a trigger), SQL*Forms creates a pop-up window that associates the field editor and the field in which the editor was invoked. This window acts as a supplement to the field, where the field data can be viewed and edited. While the field editor is active, the form is in Edit mode. In this mode, SQL*Forms suspends processing and does not fire key triggers.

The pop-up field editor can be invoked for any displayed field.

Display Characteristics: The pop-up field editor displays whenever the operator presses [Edit] or a trigger invokes the EDIT_FIELD packaged procedure. Pressing [Edit] or [Exit/Cancel] makes the pop-up field editor disappear.

The width and height of the pop-up window associated with the field editor is established by default when you create or modify a field. These dimensions vary according to the size of the source field:

- If the source field has a field length of less than or equal to 60, the pop-up window has a width equal to the field length and a height of 1 character.
- If the source field has a field length of more than 60, the pop-up window has a width of 60 and a height that is needed to accommodate the field length, up to a maximum height of six lines.

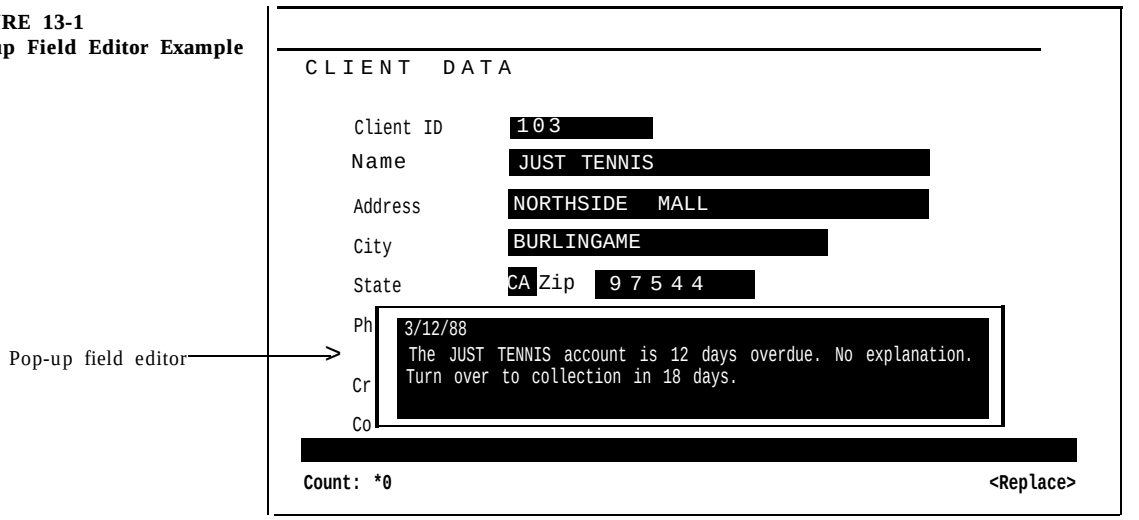
  For example, if the field length of a field is 150, the pop-up editor requires three 60 character lines. In this example, the pop-up editor displays more character spaces than are actually available for entry since multiple lines must have the same length. (If an operator enters information past the field length, SQL*Forms issues a warning that the data will be truncated.)

SQL*Forms positions the pop-up editor according to the Field Editor Location field characteristic. If the positioning would cause part of the pop-up editor to display outside of the screen, SQL*Forms automatically moves the pop-up window the minimum amount possible to legally position it.

You can modify the height, width, and location of the pop-up field editor via the EDIT_FIELD packaged procedure.

Figure 13-1 shows the pop-up field editor for a LONG field.

FIGURE 13-1
Pop-up Field Editor Example



Common Uses: The pop-up field editor is useful for displaying large amounts of data (especially text) to the operator and for facilitating the operator's entering and editing of that data. This data remains hidden, or partially displayed, until the pop-up editor is invoked. Therefore, the pop-up editor is particularly useful for displaying material, such as comments or detailed instructions, that you do not want the operator to see most of the time.

In addition, because the pop-up field editor invokes the Edit mode, the editor makes available field editing features that are not available to operators in normal mode. These features include search, cut, and paste functions.

Related Topics: EDIT_FIELD, packaged procedure 16-16
Field Editor Location field characteristic 12-13
Word Wrap field characteristic 12-25

Pop-up List of Values

A pop-up list of values has the identical features and functionality as a list of values that appears as an interface element in SQL*Forms (Design).

Any list of values that you define with a full SQL SELECT statement appears in a pop-up window. Any list of values that you define as a V2 style list of values does not appear in a pop-up window, but rather it performs as in previous versions of SQL*Forms.

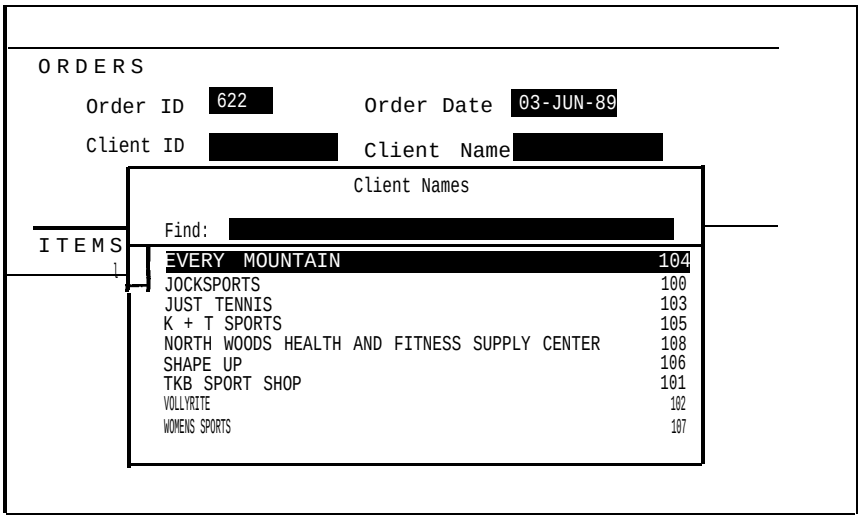
Display Characteristics: A pop-up list of values displays when the operator presses [List] or a trigger invokes the LIST_VALUES packaged procedure. Selecting any value or exiting the list of values makes the pop-up window disappear.

When you create a list of values for a field, you set the field characteristics that determine the following items:

- the title of the list of values
- the position of the list of values on the screen (i.e., X and Y coordinates for the upper left corner)
- the SQL text that retrieves values into the list of values

The list of values is designed to scroll vertically and to display no more than 10 values at a time. (See Figure 13-2.) Within limits, the contents and title of the list of values determine the width of the pop-up window. For example, if a list of values contains a number of values of varying widths, the list of values widens to accommodate the length of the longest value it displays. However, there are limits to the width of a list of values. If a list of values would be so wide as to not fit on the screen, SQL*Forms truncates (from the right) the data within the list of values.

FIGURE 13-2
Example Pop-up List of values



Common Uses: Pop-up lists of values are useful for showing operators a number of valid entries for a field. This feature can help prevent operators from entering invalid choices since they can see the valid entries and search for a correct value among those entries. It can also help eliminate some typing errors since operators can enter a listed value directly into a field.

Related Topics: List of Values Pos field characteristic 12-17
List of Values SQL Text field characteristic 12-17
List of Values Title field characteristic 12-19

Pop-up Page

A pop-up page is a view of a page. That page can belong to the current form or a called form. The view displays all of a page or some portion of the page, and its characteristics can be changed during form execution.

A page only appears as a pop-up page if you define it as one with the Pop Up page characteristic. Otherwise, a page display displaces the entire screen (even if the physical size of the page is not as large as the screen).

Display Characteristics: A pop-up page displays when the cursor navigates to a field on that page or when a trigger explicitly displays it with the SHOW_PAGE packaged procedure. Note however, that a pop-up page is not active until the cursor navigates to a field on that page.

The pop-up page disappears when the cursor navigates out of the page and the Remove on Exit page characteristic is turned on or when the HIDE_PAGE packaged procedure explicitly removes it.

When you define a page as a pop-up page (on the Page Definition form or spread table), you can specify page characteristics that affect how the page appears. These characteristics determine the following specifications:

- the initial size of the view (i.e., how much of the page you see)
- the initial location of the view on the page (i.e., what part of the page you see)
- the initial location of the view on the screen (i.e., where on the screen you see the view of the page)
- the title of the view
- whether the view should have a border
- whether the view should have scroll bars

Note that the size of the view, the location of the view on the page, and the location of the view on the screen are dynamic characteristics. That is, they can be changed during execution of the form by the RESIZE_VIEW, ANCHOR_VIEW, and MOVE_VIEW packaged procedures. The location of the view on the page can also be changed through navigational events during execution.

Figure 13-3 shows an example pop-up page as it would appear in the screen painter. Figure 13-4 then shows how a view of that page might appear on the screen.

FIGURE 13-3
Example Pop-up Page

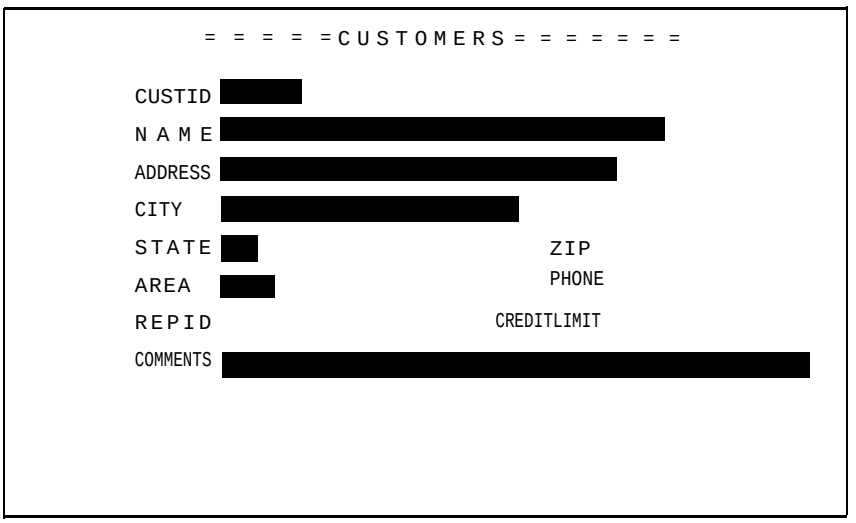
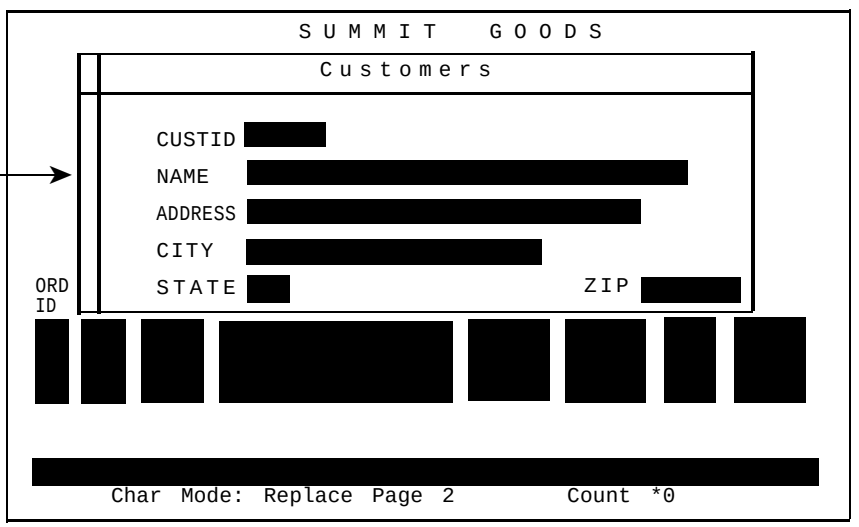


FIGURE 13-4
Example View of a Pop-up Page



While you can display more than one page at a time, you can have only one active page. The active page is always displayed. If the view of the active page overlaps with any other view, the active page displays “on top of" or “in front of" any other pages.

Common Uses: Pop-up pages are useful for displaying parts of a form while maintaining context with the current page and staying within the same commit unit. Use a pop-up page when you want to display an individual field or block in a form while maintaining context.

For example, use pop-up pages to perform the following actions:

- display an individual field or block of information in a pop-up window based on operator input to a field
- display a query-only block in a pop-up window which allows operators to lookup relevant information while maintaining context
- create your own dialog boxes to receive information from operators
- display help text for an application without obscuring all parts of the current display

Related Topics: ANCHOR_VIEW, packaged procedure 16-5
HIDE_PAGE, packaged procedure 16-21
MOVE_VIEW, packaged procedure 16-23
Pages, characteristics 10-4
Remove on Exit page characteristic 10-7
RESIZE_VIEW, packaged procedure 16-28
SHOW_PAGE, packaged procedure 16-33

CHAPTER

14 TRIGGERS

This chapter discusses the SQL*Forms trigger object. Specifically, it covers the following topics:

- defining triggers
- trigger characteristics
- trigger types

Related Topics: Triggers, concepts 4-1

Defining Triggers

You create and modify triggers in the SQL*Forms (Design) interface through the Trigger Definition form and the Trigger Definition spread table.

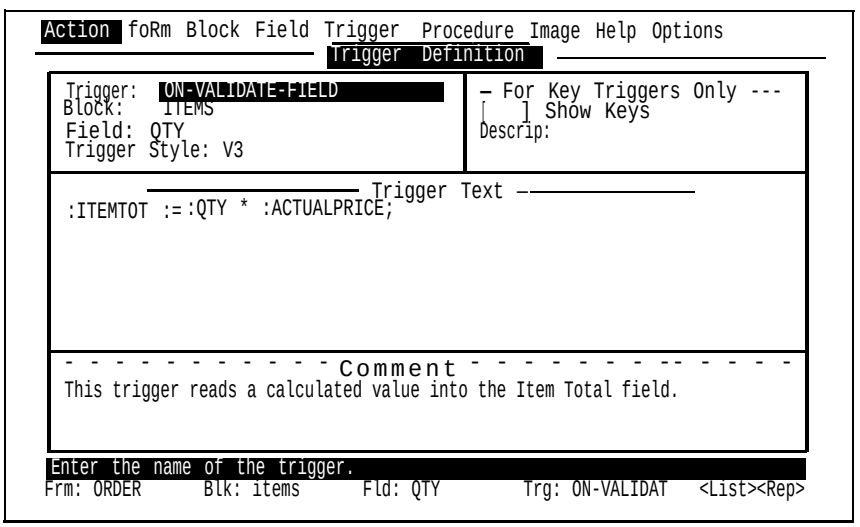
Trigger Definition Form

The Trigger Definition form is an element in the SQL*Forms (Design) interface where you create, modify, or delete triggers. This form allows you to see all information for a single trigger at one time.

Note: Use the Trigger Definition spread table if you want to look at information on more than one trigger at the same time.

Figure 14-1 shows the Trigger Definition form with sample data.

FIGURE 14-1
Trigger Definition Form



Location: To reach the Trigger Definition form from the main menu, complete the following steps:

1. Make sure you have form context.
2. Select the Trigger option.
3. If the Trigger Definition spread table displays, press [Change Display Type].

You can also display the Trigger Definition form in the following ways:

- Choose the Trigger option from the Form menu.
- Choose the Trigger option from the Block menu.
- Choose the Trigger option from the Field menu.
- Press [Zoom In] from the Field Definition form.
- If you are working with a V2 trigger, press [Zoom Out] from the Trigger Step Definition form.
- Press [Navigate] from any form or spread table and specify a trigger context. You can optionally specify a block and field context also.

Note that these methods might display the Trigger Definition spread table.

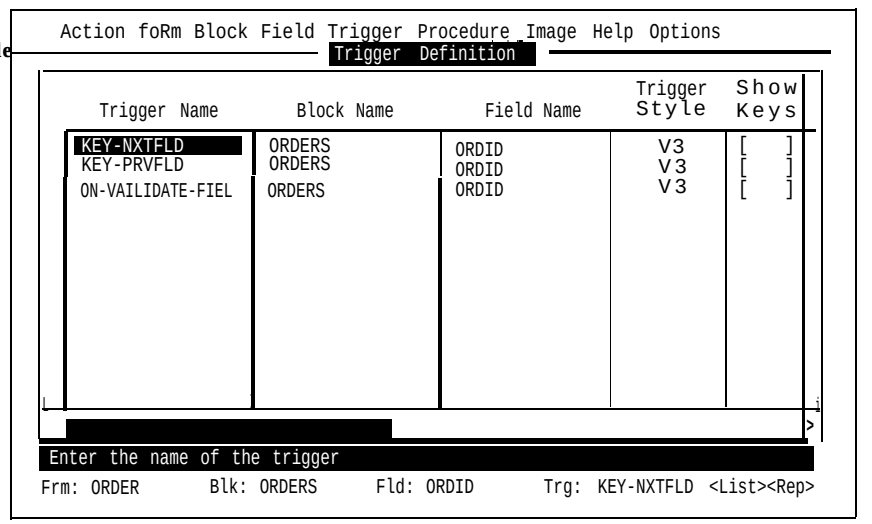
Trigger Definition Spread Table

The Trigger Definition spread table is an element in the SQL*Forms (Design) interface where you create, modify, or delete triggers. This spread table allows you to see information on more than one trigger at a time. Note that all of the elements in this spread table do not appear at the same time.

Note: Use the Trigger Definition form if you want to look at all of the information for one trigger at the same time.

Figure 14-2 shows a portion of the Trigger Definition spread table with sample data.

FIGURE 14-2
Trigger Definition Spread Table



Location: To reach the Trigger Definition spread table from the main menu, complete the following steps:

1. Make sure you have form context.
2. Select the Trigger option.
3. If the Trigger Definition form displays, press [Change Display Type].

You can also display the Trigger Definition spread table in the following ways:

- Choose the Trigger option from the Form menu.
- Choose the Trigger option from the Block menu.
- Choose the Trigger option from the Field menu.
- Press [Zoom In] from the Field Definition spread table.
- If you are working with a V2 trigger, press [Zoom Out] from the Trigger Step Definition form.
- Press [Navigate] from any form or spread table and specify a trigger context. You can optionally specify a block and field context also.

Note that these methods might display the Trigger Definition form.

Trigger Characteristics

The process of defining a trigger consists of setting each of the trigger's characteristics. These characteristics indicate the following things about a trigger:

- basic information, including the name of an object in a form to which the trigger is attached
- the trigger's style and what the trigger does

This section presents trigger characteristics in alphabetical order. Note that the names of the trigger characteristics correspond exactly with the names of the elements in the Trigger Definition form and the Trigger Definition spread table. For example, the Trigger Name field in the Trigger Definition spread table is the element that defines the Trigger Name trigger characteristic.

Note: The full names of the trigger characteristics are sometimes abbreviated in the design interface. In this section, these abbreviations follow the full names in parentheses (e.g., Block Name (Block)).

You assign trigger characteristics through different interface elements and in different ways:

Interface Element	*To Assign a Trigger Characteristic*
check box	turn the characteristic on or off
field	type in a value
scroll region	type in a value

You should be aware of these assignment differences because they affect how this section describes the characteristics. For example, a characteristic assigned from a check box has a description for Default Setting (which will be either “on” or “off"). Alternatively, a characteristic assigned from a field might have a description for Required/Optional, indicating whether you have to fill out this field element when defining a trigger object.

Related Topics: Interface elements 5-1

Block Name (Block)

The Block Name trigger characteristic indicates the name of the block to which the trigger is attached if you define the trigger at the block or field level.

Required/Optional: The Block Name characteristic is required if you want to attach the trigger to a particular block or field.

Default: The Block Name characteristic defaults to the current block, if there is one.

Related Topics: Trigger scope 43

Comment

The Comment trigger characteristic specifies general information about the trigger. Use Comment to describe the purpose or characteristics of the trigger. This feature is especially useful to other designers who use your objects or applications and to you when you debug your applications.

Required/Optional: optional

Description (Descrip)

The Description trigger characteristic indicates the text that displays on the Show Keys screen for a key trigger. You can only use the Description characteristic to describe key triggers.

The description text replaces the original text that was mapped to the indicated physical key. For example, if you define a Key-NXTBLK trigger that always moves the cursor to the next page of a form when an operator presses [Next Block], you might change the description text from Next Block to Next Page. The mapped keystroke for the function key (e.g., PF1) displays as before. Note that no description appears for Key-Fn triggers unless you specify description text through the Description characteristic.

If the Show Keys trigger characteristic is "off," the entry for the function key (including the description) does not display on the Show Keys screen.

Required/Optional: optional

Default Value: The default value for the Description characteristic is blank. This indicates that SQL*Forms should use the function key's default description.

Related Topics: Show Keys, trigger characteristic 14-6

Field Name (Field)

The Field Name trigger characteristic indicates the name of the field to which the trigger is attached if you define the trigger at the field level.

Required/Optional: The Field Name characteristic is required if you want to attach the trigger to a particular field.

Default Value: The Field Name characteristic defaults to the current field, if there is one.

Related Topics: Trigger scope 4-3

Show Keys

The Show Keys trigger characteristic indicates if the Show Keys screen should display a function key's entry. This characteristic only applies to key triggers and affects the entry for the key that a trigger defines.

An entry in the Show Keys screen includes a text description for the function and the physical keystroke associated with it (e.g., Exit PF4). By default, a redefined function key maintains its original description. You can replace that description through the Description trigger characteristic.

Default Setting: on

Related Topics: Description, trigger characteristic 14-5

Trigger Name (Trigger)

The Trigger Name trigger characteristic indicates the name that SQL*Forms uses to identify the trigger. For user-named triggers, the Trigger Name characteristic is any name you choose that follows the object naming conventions. For all other triggers, this name is identical to the trigger type name. For example, the Trigger Name characteristic of an On-Delete trigger is ON-DELETE.

Note: If you change the name of a user-named trigger in an existing application, be sure to change all references in other triggers to that user-named trigger.

Required/Optional: required

Related Topics: objects, naming conventions 20-2

Trigger Style

The Trigger Style trigger characteristic specifies whether the trigger is a V2 trigger or a V3 trigger.

Default Setting: V3

Related Topics: Triggers, types 14-8
V3 triggers versus V2 triggers 4-2

Trigger Text

The Trigger Text trigger characteristic specifies the PL/SQL routine or procedure that executes when the trigger fires. You can only specify the Trigger Text characteristic for V3 triggers.

Note: If you create a V2 trigger, SQL*Forms ignores any information you specify for the Trigger Text characteristic. The text of a V2 trigger is specified in trigger steps.

Required/Optional: The Trigger Text characteristic is required for V3 triggers.

Trigger Types

This section discusses trigger types. Specifically, it covers the behavior of each type of trigger and the rules for using each one, including the following information:

- restrictions on using the trigger
- the definition level that a trigger can have
- the legal commands you can include in the trigger text
- common uses for the trigger

Note: This book sometimes uses the terms "navigational triggers" and "transactional triggers." These terms do not designate separate types of triggers, but rather groups of related triggers.

Navigational triggers	Refers to Pre- and Post- Field, Record, Block, and Form triggers.
Transactional triggers	Refers to Pre- and Post- Commit, Delete, Insert, and Update triggers and On- Delete, Insert, and Update triggers.

Related Topics: Triggers, concepts 4-1

Key Triggers

There are four types of key triggers:

- function key trigger
- Key-Fn trigger
- Key-Others
- Key-Startup

The majority of key triggers are function key triggers. These key triggers have a one-to-one relationship with specific keys. That is, the trigger fires only when you press a specific key or a specific sequence of keys. All other key triggers are exceptions to this one-to-one relationship.

Function Key Triggers

Function key triggers are associated with individual SQL*Forms (Run Form) function keys. A function key trigger fires only when an operator presses the associated function key. The actions that a function key trigger defines replace the processing that the associated function key would invoke by default.

The following table shows all function key triggers and the SQL*Forms (Run Form) function keys to which they correspond.

Trigger Type	*Associated Function Key*
Key-CLRBLK	[Clear Block]
Key-CLRFRM	[Clear Form]
Key-CLRREC	[Clear Record]
Key-COMMIT	[Commit/Accept]
Key-CQUERY	[Count Query Hits]
Key-CREREC	[Insert Record]
Key-DELREC	[Delete Record]
Key-DOWN	[Down]
Key-DUPFLD	[Duplicate Field]
Key-DUPREC	[Duplicate Record]
Key-EDIT	[Edit]
Key-ENTER	[Enter]
Key-ENTQRY	[Enter Query]
Key-EXEQRY	[Execute Query]
Key-EXIT	[Exit]
Key-HELP	[Help]
Key-LISTVAL	[List]
Key-MENU	[Block Menu]
Key-NXTBLK	[Next Block]
Key-NXTFLD	[Next Field]
Key-NXTKEY	[Next Primary Key Fld]
Key-NXTREC	[Next Record]
Key-NXTSET	[Next Set of Records]
Key-PRINT	[Print]
Key-PRVBLK	[Previous Block]
Key-PRVFLD	[Previous Field]

Table continued on next page.

Trigger Type	*Associated Function Key*
Key-PRVREC	[Previous Record]
Key-SCRDOWN	[Scroll Down]
Key-SCRUP	[Scroll Up]
Key-UP	[UP]
Key-UPDREC	[Lock Record]

Note that you cannot redefine all SQL*Forms (Run Form) function keys with function key triggers. Specifically, you cannot ever redefine the following static function keys because they are often performed by the terminal or user interface management system and not by SQL*Forms.

[Beginning of Line]	[End of Line]	[Refresh]
[Clear Field]	[First Line]	[Right]
[Copy]	[Insert Line]	[Scroll Left]
[Cut]	[Insert/Replace]	[Scroll Right]
[Delete Backward]	[Last Line]	[Search]
[Delete Character]	[Left]	[select]
[Delete Line]	[Menu]	[Show Keys]
[Display Error]	[Paste]	[Transmit]

Restrictions: SQL*Forms ignores all function key triggers in Enter Query mode (including any Key-EXEQRY or Key-EXIT trigger) and in Edit mode. SQL*Forms ignores the Key-Commit trigger when an operator presses [Commit/Accept] in a dialog box.

Definition Level: field, block, or form

Legal Commands: SELECT statements, restricted packaged procedures, unrestricted packaged procedures

Common Uses: Use function key triggers to perform the following tasks:

- disable function keys dynamically
- replace the default behavior of function keys
- dynamically remap function keys
- perform complex or multiple functions with a single key or key sequence

Related Topics: Process the Function Key 21-105

Key-Fn Trigger

You can attach Key-Fn triggers to 10 keys or key sequences that normally do not perform any SQL*Forms operations. These keys are referred to as Key-F0 through Key-F9. Before you can attach key triggers to these keys, you or the DBA must use Oracle*Terminal to map the keys to the appropriate functions.

A Key-Fn trigger fires when an operator presses the associated key.

Restrictions: SQL*Forms ignores Key-Fn triggers in Enter Query mode and Edit mode.

Definition Level: field, block, or form

Legal Commands: SELECT statements, restricted packaged procedures, unrestricted packaged procedures

Common Uses: Use Key-Fn triggers to create additional function keys for custom functions.

Related Topics: Key-F0 through Key-F9, mapping 25-11

Key-Others Trigger

A Key-Others trigger is associated with all keys that *can* have key triggers associated with them but *are not* currently defined by function key triggers (at any level). That is, a Key-Others trigger is a catch for all remaining keys that do not have triggers (rather than a trigger to override all keys at its scope). A Key-Others trigger fires when an operator presses an associated key.

A Key-Others trigger overrides the default behavior of a SQL*Forms (Run Form) function key (unless one of the following restrictions applies). When this occurs, however, SQL*Forms still displays the function key's default entry in the Show Keys screen.

Restrictions: SQL*Forms ignores a Key-Others trigger under the following conditions:

- the form is in Enter Query mode
- a list of values, the Show Keys screen, a help screen, or an error screen is displayed
- the operator is responding to a SQL*Forms (Run Form) prompt
- the operator presses a static function key

Definition Level: field, block, or form

Legal Commands: SELECT statements, restricted packaged procedures, unrestricted packaged procedures

Common Uses: Use Key-Others triggers to limit an operator's possible actions. Specifically, Use Key-Others triggers to perform the following tasks:

- disable all keys that are not relevant in a particular situation
- perform one specific action whenever an operator presses any key

Key-Startup Trigger

A Key-Startup trigger fires when SQL*Forms runs a form. This trigger is considered a key trigger because it acts as if the operator has pressed a phantom "startup" key.

A Key-Startup trigger always fires after successful navigation into a form. Therefore, no matter at what level you define the Key-Startup trigger, it fires after the successful completion of any navigational triggers. If you define more than one Key-Startup trigger in a form, the one with the lowest applicable definition level takes precedence.

Definition Level: You can define a Key-Startup trigger at the field, block, or form level. If you define the trigger at the block level, it will only fire if you attach it to the first block that the cursor enters upon running the form. If you define the trigger at the field level, it will only fire if you attach it to the first field that the cursor enters upon running the form.

Legal Commands: SELECT statements, restricted packaged procedures, unrestricted packaged procedures

Common Uses: Use a Key-Startup trigger to set up a form for operator use. Note that a Key-Startup trigger can use restricted packaged procedures, which are not permitted in a Pre-Form, Pre-Block, or Pre-Field trigger.

Related Topics: Run the Form 21-113

On-Clear-Block Trigger

An On-Clear-Block trigger fires just before SQL*Forms flushes the current block. This action occurs during the following events:

- CLEAR_BLOCK
- COUNT_QUERY
- ENTER_QUERY
- Open the Query

Note that the On-Clear-Block trigger does not fire when SQL*Forms flushes the current block during the CLEAR_FORM event.

Definition Level: field, block, or form

Legal Commands: SELECT statements, unrestricted packaged procedures

Common Uses: Use an On-Clear-Block trigger to perform an action every time SQL*Forms flushes the current block. For example, you might want to perform an automatic commit whenever this condition occurs.

Related Topics: CLEAR_BLOCK, flow chart 21-18
COUNT_QUERY, flow chart 21-26
ENTER_QUERY, flow chart 21-47
Open the Query 21-88

On-Database-Record Trigger

An On-Database-Record trigger fires whenever SQL*Forms first marks a record as an insert or an update. That is, the trigger fires as soon as SQL*Forms determines that the record will be processed by the next post or commit as an insert or update. This action occurs during the following events:

- DUPLICATE_RECORD
- Mark Fields and Records as Changed

Definition Level: field, block, or form

Legal Commands: SELECT statements, unrestricted packaged procedures

Common Uses: Use an On-Database-Record trigger to perform an action every time a record is first marked as an insert or an update. For example, use this trigger to immediately update a running total.

Related Topics: DUPLICATE_RECORD, flow chart 21-38
Mark Fields and Records as Changed 21-76

On-Delete Trigger

An On-Delete trigger fires during the Post and Commit Transactions event. Specifically, it fires after the Pre-Delete trigger and before the Post-Delete trigger. An On-Delete trigger replaces the default SQL*Forms processing for handling deleted records during transaction posting. It fires once for each row that is marked for deletion from the database.

Definition Level: block or form

Legal Commands: SELECT statements, Data Manipulation Language (DML) statements (i.e., DELETE, INSERT, UPDATE), unrestricted packaged procedures

Common Uses: Use an On-Delete trigger to replace the default SQL*Forms processing for handling deleted records during transaction posting.

Related Topics: Post and Commit Transactions 21-92

On-Error Trigger

An On-Error trigger fires whenever SQL*Forms would normally cause an error message to display. The actions of an On-Error trigger replace the writing of an error message.

Definition Level: field, block, or form

Legal Commands: SELECT statements, unrestricted packaged procedures

Common Uses: Use an On-Error trigger for the following purposes:

- trap and recover from an error
- replace a standard error message with a custom message

Use the ERROR_CODE, ERROR_TEXT, or ERROR_TYPE packaged function in an On-Error trigger to identify a specific error condition.

Related Topics: Display a Message 21-31
Error messages, message type E-43
ERROR_CODE packaged function 17-5
ERROR_TEXT packaged function 17-6
ERROR_TYPE packaged function 17-7

On-Insert Trigger

An On-Insert trigger fires during the Post and Commit Transactions event. Specifically, it fires after the Pre-Insert trigger and before the Post-Insert trigger. An On-Insert trigger replaces the default SQL*Forms processing for handling inserted records during transaction posting. It fires once for each row that is marked for insertion into the database.

Definition Level: block or form

Legal Commands: SELECT statements, Data Manipulation Language (DML) statements (i.e., DELETE, INSERT, UPDATE), unrestricted packaged procedures

Common Uses: Use an On-Insert trigger to replace the default SQL*Forms processing for handling inserted records during transaction posting.

Related Topics: Post and Commit Transactions 21-92

On-Lock Trigger

An On-Lock trigger fires whenever SQL*Forms would normally attempt to lock a row. The actions of an On-Lock trigger replace the default SQL*Forms processing for locking rows.

Definition Level: block or form

Legal Commands: SELECT statements, unrestricted packaged procedures

Common Uses: Use an On-Lock trigger to replace the default SQL*Forms processing for locking rows. For example, if you are designing an application for use on a single-user system, you can use the On-Lock trigger to speed processing by bypassing all locking requirements.

Related Topics: Lock the Row 21-73

On-Message Trigger

An On-Message trigger fires whenever SQL*Forms would normally cause an informative message to display. The actions of an On-Message trigger replace the writing of a message.

Definition Level: field, block, or form

Legal Commands: SELECT statements, unrestricted packaged procedures

Common Uses: Use an On-Message trigger for the following purposes:

- trap and respond to an informative message
- replace a standard informative message with a custom message

Use the MESSAGE_CODE, MESSAGE_TEXT, or MESSAGE_TYPE packaged function in an On-Message trigger to identify the occurrence of a specific message condition.

Related Topics: Display a Message 21-31
Informative message type E-43
MESSAGE_CODE packaged function 17-13
MESSAGE_TEXT packaged function 17-14
MESSAGE_TYPE packaged function 17-15

On-New-Field-Instance Trigger

An On-New-Field-Instance trigger fires whenever SQL*Forms navigates to a field *and* readies to accept user input (i.e., typed values or function keys).

Note: The conditions for firing this trigger are *not* met under the following circumstances.

- a trigger causes SQL*Forms to navigate through a field instance without stopping to accept input
- the cursor appears in a dialog box, a prompt, or any part of a SQL*Menu application
- the form is in Enter Query mode

Definition Level: field, block, or form

Legal Commands: SELECT statements, restricted packaged procedures, unrestricted packaged procedures

Common Uses: Use an On-New-Field-Instance trigger to perform an action whenever an operator enters a field. For example, use this trigger to automatically invoke a list of values every time the operator enters a particular field. The On-New-Field-Instance trigger is especially useful for performing navigational packaged procedures (e.g., NEXT_FIELD, GO_BLOCK) outside of a key trigger.

Related Topics: Run the Form 21-113
SYSTEM.CURSOR_FIELD 18-8

On-New-Record Trigger

An On-New-Record trigger fires whenever SQL*Forms creates a new record. This action occurs during the following events:

- CREATE_RECORD
- Enter the Block (when the block only contains New records)

Definition Level: field, block, or form

Legal Commands: SELECT statements, unrestricted packaged procedures

Common Uses: Use an On-New-Record trigger to perform an action every time SQL*Forms attempts to create a new record. For example, use this trigger to populate the fields in new records with dynamic default values.

Related Topics: CREATE_RECORD, flow chart 21-27
Enter the Block 21-42

On-Remove-Record Trigger

An On-Remove-Record trigger fires whenever SQL*Forms clears or deletes a record. This action occurs during the following events:

- CLEAR_RECORD
- DELETE_RECORD
- PREVIOUS_RECORD (from the last, blank record in a block)

Definition Level: field, block, or form

Legal Commands: SELECT statements, unrestricted packaged procedures

Common Uses: Use an On-Remove-Record trigger to perform an action every time SQL*Forms clears or deletes a record. For example, use this trigger to immediately update a running total.

Related Topics: CLEAR_RECORD, flow chart 21-22
DELETE_RECORD, flow chart 21-30
PREVIOUS_RECORD, flow chart 21-102

On-Update Trigger

An On-Update trigger fires during the Post and Commit Transactions event. Specifically, it fires after the Pre-Update trigger and before the Post-Update trigger. An On-Update trigger replaces the default SQL*Forms processing for handling updated records during transaction posting. It fires once for each row that is marked for update in the database.

Definition Level: block or form

Legal Commands: SELECT statements, Data Manipulation Language (DML) statements (i.e., DELETE, INSERT, UPDATE), unrestricted packaged procedures

Common Uses: Use an On-Update trigger to replace the default SQL*Forms processing for handling updated records during transaction posting.

Related Topics: Post and Commit Transactions 21-92

On-Validate-Field Trigger

An On-Validate-Field trigger fires during the Validate the Field event. Specifically, it fires as the last part of field validation for fields with the New or Changed validation status.

Definition Level: field, block, or form

Legal Commands: SELECT statements, unrestricted packaged procedures

Common Uses: Use an On-Validate-Field trigger to supplement the SQL*Forms processing for field validation.

Note that it is possible to write an On-Validate-Field trigger that changes the value of a field that SQL*Forms is validating. If validation succeeds, SQL*Forms marks the changed field as Valid and does not re-validate it. While this behavior is necessary to avoid validation loops, it does allow you to commit an invalid value to the database.

Related Topics: Validate the Field 21-127

On-Validate-Record Trigger

An On-Validate-Record trigger fires during the Validate the Record event. Specifically, it fires as the last part of record validation for records with the New or Changed validation status.

Definition Level: block or form

Legal Commands: SELECT statements, unrestricted packaged procedures

Common Uses: Use an On-Validate-Record trigger to supplement the SQL*Forms processing for record validation.

Note that it is possible to write an On-Validate-Record trigger that changes the value of a field in the record that SQL*Forms is validating. If validation succeeds, SQL*Forms marks the record and all of the fields as Valid and does not re-validate. While this behavior is necessary to avoid validation loops, it does allow you to commit an invalid value to the database.

Related Topics: Validate the Record 21-130

Post-Block Trigger

A Post-Block trigger fires when the Leave the Block event occurs.

Restrictions: A Post-Block trigger only fires when the form is run with a validation unit of the field, record, or block

Definition Level: block or form

Legal Commands: SELECT statements, unrestricted packaged procedures

Common Uses: Use a Post-Block trigger to perform complex validation of all the records in a block's list of records.

Related Topics: Leave the Block 21-65

Post-Change Trigger

A Post-Change trigger fires when any of the following conditions occur:

- the Validate the Field event determines that a field is marked as Changed and is non-NULL
- an operator reads a value into a field from a list of values
- SQL*Forms reads a value into a field from a fetched record

Definition Level: field, block, or form

Legal Commands: SELECT statements, unrestricted packaged procedures

Common Uses: Use a Post-Change trigger to perform the following tasks:

- set global variables
- supplement the behavior of SQL*Forms when it is populating a field via a list of values or a fetch

Oracle Corporation suggests that you do not use a Post-Change trigger for validation purposes. While this practice was standard with previous versions of SQL*Forms-and does still work—you should use an On-Validate-Field trigger instead.

Note that it is possible to write a Post-Change trigger that changes the value of a field that SQL*Forms is validating. If validation succeeds, SQL*Forms marks the changed field as Valid and does not re-validate it. While this behavior is necessary to avoid validation loops, it does allow you to commit an invalid value to the database.

Related Topics: Validate the Field 21-127

Post-Commit Trigger

A Post-Commit trigger fires once during the Post and Commit Transactions event. Specifically, it fires after SQL*Forms processes any transactions but before the database commit occurs.

Definition Level: form

Legal Commands: SELECT statements, Data Manipulation Language (DML) statements (i.e., DELETE, INSERT, UPDATE), unrestricted packaged procedures

Common Uses: Use a Post-Commit trigger to perform an action anytime a database commit is going to occur, such as updating an audit trail.

Related Topics: Post and Commit Transactions 21-92

Post-Delete Trigger

A Post-Delete trigger fires during the Post and Commit Transactions event, after a row is deleted. It fires once for each row that is deleted from the database during the commit event.

Definition Level: block or form

Legal Commands: SELECT statements, Data Manipulation Language (DML) statements (i.e., DELETE, INSERT, UPDATE), unrestricted packaged procedures

Common Uses: Use a Post-Delete trigger to audit transactions.

Related Topics: Post and Commit Transactions 21-92

Post-Field Trigger

A Post-Field trigger fires when the Leave the Field event occurs.

Restrictions: A Post-Field trigger only fires when the form is run with the field validation unit.

Definition Level: field, block, or form

Legal Commands: SELECT statements, unrestricted packaged procedures

Common Uses: Use a Post-Field trigger to calculate or change field values.

Related Topics: Leave the Field 21-66

Post-Form Trigger

A Post-Form trigger fires when the Leave the Form event occurs.

Definition Level: form

Legal Commands: SELECT statements, unrestricted packaged procedures

Common Uses: Use a Post-Form trigger to "clean up" the form before exiting. For example, use a Post-Form trigger to erase any global variables that the form no longer requires. Also, use a Post-Form trigger to display a message to the operator upon form exit.

Related Topics: Leave the Form 21-67

Post-Insert Trigger

A Post-Insert trigger fires during the Post and Commit Transactions event, after a row is inserted. It fires once for each row that is inserted into the database during the commit event.

Definition Level: block or form

Legal Commands: SELECT statements, Data Manipulation Language (DML) statements (i.e., DELETE, INSERT, UPDATE), unrestricted packaged procedures

Common Uses: Use a Post-Insert trigger to audit transactions.

Related Topics: Post and Commit Transactions 21-92

Post-Query Trigger

A Post-Query trigger fires each time SQL*Forms fetches a record into a block.

Definition Level: block or form

Legal Commands: SELECT statements, unrestricted packaged procedures

Common Uses: Use a Post-Query trigger to perform the following task:

- change the field values of fetched records
- calculate statistics about the records retrieved by a query
- populate non-database fields or fields in other blocks

Related Topics: Fetch Records 21-53

Post-Record Trigger

A Post-Record trigger fires when the Leave the Record event occurs.

Restrictions: A Post-Record trigger only fires when the form is run with a validation unit of the field or record.

Definition Level: block or form

Legal Commands: SELECT statements, unrestricted packaged procedures

Common Uses: Use a Post-Record trigger to perform the following tasks:

- perform calculations
- keep a running total
- validate combinations of fields in a record

Related Topics: Leave the Record 21-68

Post-Update Trigger

A Post-Update trigger fires during the Post and Commit Transactions event, after a row is updated. It fires once for each row that is updated in the database during the commit event.

Definition Level: block or form

Legal Commands: SELECT statements, Data Manipulation Language (DML) statements (i.e., DELETE, INSERT, UPDATE), unrestricted function codes

Common Uses: Use a Post-Update trigger to audit transactions.

Related Topics: Post and Commit Transactions 21-92

Pre-Block Trigger

A Pre-Block trigger fires when the Enter the Block event occurs.

Restrictions: A Pre-Block trigger only fires when the form is run with a validation unit of the field, record, or block.

Definition Level: block or form

Legal Commands: SELECT statements, unrestricted packaged procedures

Common Uses: Use a Pre-Block trigger to perform the following tasks:

- authorize access to a block
- derive default values for a block's fields
- set global variables

Related Topics: Enter the Block 21-42

Pre-Commit Trigger

A Pre-Commit trigger fires once during the Post and Commit Transactions event, before SQL*Forms processes any transactions. Specifically, it fires after SQL*Forms determines that there are changes in the form to post or to commit.

Definition Level: form

Legal Commands: SELECT statements, Data Manipulation Language (DML) statements (i.e., DELETE, INSERT, UPDATE), unrestricted packaged procedures

Common Uses: Use a Pre-Commit trigger to perform an action anytime a database commit is going to occur, such as setting up special locking requirements.

Related Topics: Post and Commit Transactions 21-92

Pre-Delete Trigger

A Pre-Delete trigger fires during the Post and Commit Transactions event, before a row is deleted. It fires once for each row that is marked for deletion from the database.

Definition Level: block or form

Legal Commands: SELECT statements, Data Manipulation Language (DML) statements (i.e., DELETE, INSERT, UPDATE), unrestricted packaged procedures

Common Uses: Use a Pre-Delete trigger to prevent the deletion of a record if that record is the master record for detail records that still exist.

Related Topics: Post and Commit Transactions 21-92

Pre-Field Trigger

A Pre-Field trigger fires when the Enter the Field event occurs.

Restrictions: A Pre-Field trigger only fires when the form is run with a validation unit of the field.

Definition Level: field, block, or form

Legal Commands: SELECT statements, unrestricted packaged procedures

Common Uses: Use a Pre-Field trigger to derive a complex default value.

Related Topics: Enter the Field 21-43

Pre-Form Trigger

A Pre-Form trigger fires when the Enter the Form event occurs.

Definition Level: form

Legal Commands: SELECT statements, unrestricted packaged procedures

Common Uses: Use a Pre-Form trigger to perform the following tasks:

- restrict access to a form
- initialize global variables
- print messages

Related Topics: Enter the Form 21-44

Pre-Insert Trigger

A Pre-Insert trigger fires during the Post and Commit Transactions event, before a row is inserted. It fires once for each row that is marked for insertion into the database.

Definition Level: block or form

Legal Commands: SELECT statements, Data Manipulation Language (DML) statements (i.e., DELETE, INSERT, UPDATE), unrestricted packaged procedures

Common Uses: Use a Pre-Insert trigger to perform the following task:

- change field values
- validate field values in a record

Related Topics: Post and Commit Transactions 21-92

Pre-Query Trigger

A Pre-Query trigger fires when the EXECUTE_QUERY or COUNT_QUERY functions are invoked but before SQL*Forms creates the query SELECT statement that fetches or counts any records.

Definition Level: block or form

Legal Commands: SELECT statements, unrestricted packaged procedures

Common Uses: Use a Pre-Query trigger to modify the example record in order to determine what records the query retrieves.

Related Topics: COUNT_QUERY, flow chart 21-26
EXECUTE_QUERY, flow chart 21-50

Pre-Record Trigger

A Pre-Record trigger fires when the Enter the Record event occurs.

Restrictions: A Pre-Record trigger only fires when the form is run with a validation unit of the field or record.

Definition Level: block or form

Legal Commands: SELECT statements, unrestricted packaged procedures

Common Uses: Use a Pre-Record trigger to perform the following tasks:

- keep a running total
- derive a complex default value

Related Topics: Enter the Record 21-45

Pre-Update Trigger

A Pre-Update trigger fires during the Post and Commit Transactions event, before a row is updated. It fires once for each row that is marked for update in the database.

Definition Level: block or form

Legal Commands: SELECT Statements, Data Manipulation Language (DML) statements (i.e., DELETE, INSERT, UPDATE), unrestricted packaged procedures

Common Uses: Use a Pre-Update trigger to perform the following tasks:

- change field values
- validate field values in a record

Related Topics: Post and Commit Transactions 21-92

User-named Trigger

A user-named trigger fires when invoked from another trigger.

There are no set types of user-named triggers. Each user-named trigger you create has a name you assign to it, and you use that name to call the trigger from within other triggers.

As with all triggers, the scope of a user-named trigger is the definition level and below. If more than one user-named trigger has the same name, the one defined at the lowest level has precedence.

It is most practical to define user-named triggers at the form level.

Note: Form-level procedures can perform any task for which you might use a user-named trigger. User-named triggers exist only for upward compatibility.

Definition Level: field, block, or form

Legal Commands: The legal commands of a user-named trigger are the same as those of any trigger that calls the user-named trigger.

Common Uses: Use a user-named trigger to execute common trigger code that you otherwise would have to repeat in more than one trigger.

Related Topics: EXECUTE_TRIGGER, flow chart 21-51

CHAPTER

15 PL/SQL

PL/SQL is a database transaction processing language and an application development language. It combines extensive procedural capabilities-such as conditional control and looping-with the non-procedural capabilities of SQL.

This chapter discusses how to use PL/SQL in SQL*Forms. Specifically, it covers the following topics:

- how SQL*Forms uses PL/SQL
- PL/SQL blocks in SQL*Forms
- SQL*Forms objects in PL/SQL
- packaged procedures in PL/SQL
- PL/SQL restrictions within SQL*Forms
- executable statement failure
- PL/SQL compilation

This chapter does not discuss the details of how to write programs in PL/SQL. For this information, refer to the *PL/SQL User's Guide and Reference.*

Related Topics: Trigger processing 4-5

How SQL*Forms Uses PL/SQL

SQL*Forms uses PL/SQL as the trigger language for V3 triggers. The trigger text of a V3 trigger is a PL/SQL block.

A trigger written in PL/SQL provides end-users superior performance over triggers written in SQL. In addition, a V3 trigger can take advantage of PL/SQL to perform the following tasks:

- perform arithmetic, string, and Boolean operations
- specify branching with IF...THEN and GOTO statements
- specify exception handlers for error processing

PL/SQL enhances SQL*Forms triggers by providing the advantages of a procedural language in addition to the capabilities of SQL.

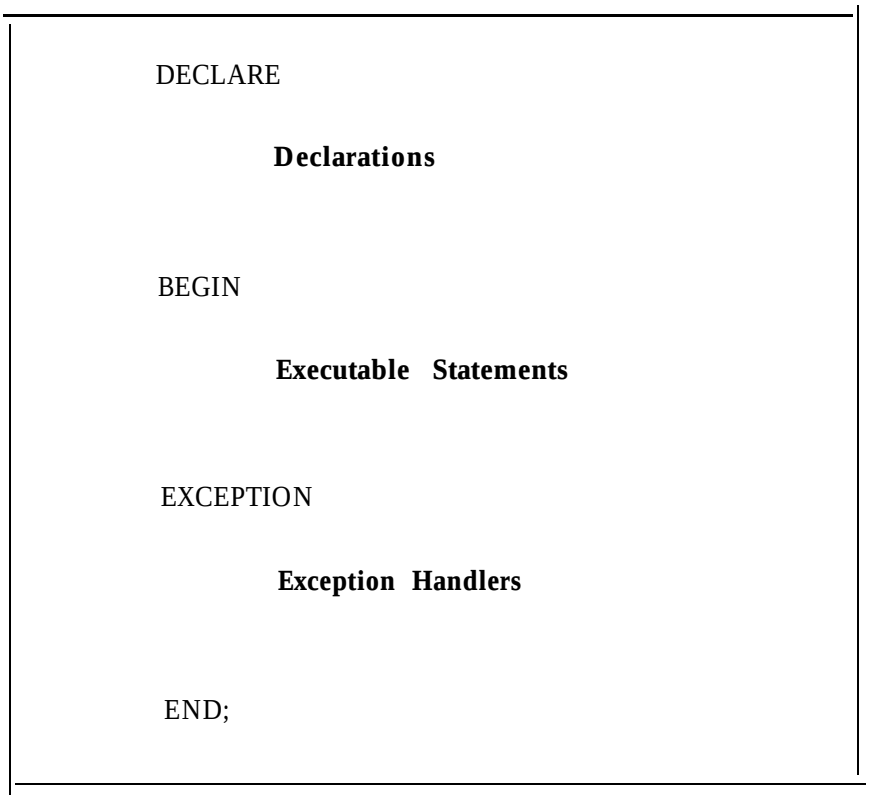
PL/SQL Blocks in SQL*Forms

A standard PL/SQL code segment is called a block. A block can consist of three parts, or sections:

- a declaration section for variables, constants, and exceptions, which is optional
- a section of executable statements, which is required
- a section of exception handlers, which is optional

Figure 15-1 illustrates the order in which these sections can appear in a block and the PL/SQL keywords associated with a standard PL/SQL block.

FIGURE 15-1
Parts of a PL/SQL Block



SQL*Forms uses PL/SQL blocks in two different constructs:

- anonymous blocks
- procedures (referred to as form-level procedures in SQL*Forms)

Anonymous Block

An anonymous block is a PL/SQL block that, unlike a form-level procedure, has no name. This means that you can only execute an anonymous block from the trigger in which it is defined.

Also, in SQL*Forms, an anonymous block does not require the explicit presence of the BEGIN and END keywords to enclose the executable statements. SQL*Forms automatically packages the BEGIN and END keywords with the trigger for compilation and execution.

There is an exception to the use of the BEGIN and END keywords. You must explicitly state BEGIN and END if the procedure includes a declaration section. In this case, PL/SQL needs the explicit BEGIN and END to separate the declaration section from the executable statements. An anonymous block requires a declaration section if the block uses local variables, cursors, or named exception handlers.

Note: You *can* always state the BEGIN and END keywords explicitly in an anonymous block. This practice affords syntactic consistency.

When you are defining a specific V3 trigger, enter anonymous blocks directly into the Trigger Text scroll region of the Trigger Definition form.

Examples

1. Assume that you want to keep track of the number of records that operators insert in the ORDERS block of a form. You can create a Post-Insert trigger that maintains a counter in a non-database field called NEW_ITEMS. The text of your trigger is simple:

```
:NEW_ITEMS : =  : NEW_ITEMS + 1 ;
```

2. Assume that you want to read the current value of the MINPRICE column into a global variable every time the cursor moves into the PRICE field. The following Pre-Field trigger performs this function:

```
SELECT MINPRICE INTO : GLOBAL.MINIMUM
    FROM PRICE
    WHERE PRICE.PRODID = :PRODID
        AND SYSDATE BETWEEN STARTDATE AND NVL (ENDDATE, SYSDATE ) ;
```

Form-Level Procedure

A form-level procedure is a PL/SQL block that requires a name. The full PL/SQL syntax, including declarations and the BEGIN and END keywords, are also required.

Form-level procedures in SQL*Forms must contain PL/SQL procedure definitions. They cannot contain an anonymous block.

You enter the procedure name and the procedure text of a form-level procedure in the Procedure Definition spread table or form. You can then invoke this form-level procedure in any V3 trigger in the current form.

You can use form-level procedures that are defined in other forms by copying them into the current form or referencing them. To do this, use [Copy Object] and fill the Source Object and Target Object dialog boxes.

The form-level procedure allows you to create generic procedures and use them in multiple places throughout a form. Form-level procedures can also take arguments, while anonymous blocks cannot.

Form-Level Procedure Name

The name of a form-level procedure must adhere to ORACLE naming conventions and be unique within a form. You must also ensure that this name matches the name of the procedure actually defined in the corresponding procedure text.

Form-Level Procedure Text

Enter form-level procedures in the following form:

```
PROCEDURE procedure_name[argument_list] IS
    [local_variable_declaration]
BEGIN
    procedure_statements
    [EXCEPTION exception_handlers]
END ;
```

where:

procedure_name Is the unique name of the form-level procedure.

argument_list Is *({var_name [mode] type [:=value])[, ...])*

where:

var_name	Is the unique name of a local variable.
mode	Is {IN |OUT |IN OUT}.
type	Is {BOOLEAN CHAR |NUMBER |DATE}.
value	Is a PL/SQL expression.

local_variable_declaration Is *{var_name type [:= value])[,. ..]*

procedure_statements Are the executable PL/SQL statements.

exception_handlers (See the *PL/SQL User's Guide and Reference.)*

Examples

1. Assume that you need to calculate an item total in more than one situation in a form. For efficiency, you create a form-level procedure called CALCULATE_ITEM_TOTAL that performs the calculation:

```
PROCEDURE CALCULATE_ITEM_TOTAL IS
BEGIN
    :ITEMTOTAL := :QUANTITY * :ACTUALPRICE;
END;
```

You can now call this procedure in any V3 trigger in the current form. (Example 2 demonstrates this feature.)

Note: The form-level procedure in this example references external objects (i.e., form objects). The trigger's function can be performed more generically by a form-level procedure that uses an argument list. The procedure in example 4 demonstrates this method.

2. Assume that you want a trigger to perform nothing else but the form-level procedure CALCULATE_ITEM_TOTAL. You simply enter the name of the form-level procedure as the trigger text and the semicolon terminator:

```
CALCULATE_ITEM_TOTAL  ;
```

3. Assume that you want to call the form-level procedure CALCULATE_ITEM_TOTAL in a trigger only if the operator has entered a product price (in the PRICE field) that is not below a minimum allowable price (stored in the MINPRICE column). You call the form-level procedure (shown here in bold) conditionally:

```
DECLARE
    MINIMUM NUMBER;
BEGIN
    SELECT PRICE.MINPRICE INTO MINIMUM
        FROM PRICE WHERE PRICE.PRODID = :ITEMS.PRODID
            AND SYSDATE BETWEEN STARTDATE AND NVL (ENDDATE, SYSDATE) ;
    IF : PRICE < MINIMUM
        THEN MESSAGE ( ' Price must be greater than'
            ||TO_CHAR (MINIMUM) ||'.');
            RAISE FORM_ TRIGGER_ FAILURE ;
        ELSE CALCULATE_ITEM_TOTAL;
    END IF;
END ;
```

Note: FORM_TRIGGER_FAILURE is a predefined PL/SQL exception (that you can only use from within SQL*Forms) that causes the trigger to fail when raised.

SQL*Forms Objects in PL/SQL

In a V3 trigger or a form-level procedure, refer to SQL*Forms forms, blocks, and fields by the names with which they are defined in the design interface or by legal variable names. Any additional syntax elements you use depend on whether you are referencing the value of a SQL*Forms objector the name of a SQL*Forms object.

- If you are referencing the value of a field (or the value of a variable that represents a field), prefix the object with a colon:

  ```
  :field
  : GLOBAL. variablename
  ```

  where *field* is fieldname or block.fieldname.

- If you are referencing the name of a form, block, or field, enclose the object in single quotes. Do not prefix the object with a colon:

  ```
  'formname'
  'blockname'
  'field'
  ```

  where *field* is fieldname or block.fieldname.

Examples 1. To set the value of the EMPNO field equal to 100, type:

```
:EMPNO := 100;
```

2. To call the PERSONNEL form from the current form, type:

```
CALL ('PERSONNEL') ;
```

3. To navigate to the EMPNO field, type:

```
GO_FIELD ('EMPNO') ;
```

Alternatively, if the name of the EMPNO field is represented by the GLOBAL.LASTFIELD variable, use the NAME_IN packaged function:

```
GO_FIELD (NAME_IN ('GLOBAL.LASTFIELD'));
```

Related Topics: NAME_IN packaged function 17-16

Packaged Procedures in PL/SQL

In a PL/SQL block (an anonymous block m a V3 trigger or form-level procedure), you can invoke or call any packaged procedure as an executable statement. Use the specified syntax for that packaged procedure.

PL/SQL Restrictions within SQL*Forms

Remember the following restrictions when you create PL/SQL blocks that refer to SQL*Forms objects or functions:

- You can use packaged procedures in PL/SQL blocks only when running PL/SQL within SQL*Forms.
- You can reference SQL*Forms Objects in PL/SQL blocks only when running PL/SQL within SQL*Forms. When you attempt to execute or compile a PL/SQL block from within the ORACLE RDBMS, you will receive errors if the block references SQL*Forms objects.

Executable Statement Failure

In SQL*Forms triggers, you trap, or isolate, success and failure differently depending on whether the executable statement is a SQL statement or a packaged procedure. Use the appropriate error processing method when the success or failure of an executable statement is important to the execution of a trigger.

SQL Statement Failure

Use exception handling to trap the successor failure of a SQL statement. The PL/SQL functions SQLCODE and SQLERRM are available to identify specific errors. Refer to the *PL/SQL User's Guide and Reference* for information on how to use these functions.

Exception Handling

Exception handlers define statements that are to be executed when certain conditions, or exceptions, are raised in a PL/SQL block. Some exceptions are pre-defined and are raised internally by PL/SQL (e.g., ZERO.DIVIDE), while other exception handlers are user-defined and must be raised explicitly with the RAISE statement. When an exception is raised, PL/SQL stops the normal execution of the PL/SQL block and passes control to the appropriate exception handler. The execution of a V3 trigger stops when the exception handler completes execution.

Refer to the *PL/SQL User's Guide and Reference* for an explanation of exception handling.

Packaged Procedure Failure

Use any of the following packaged functions to trap the successor failure of packaged procedures:

- FORM_SUCCESS
- FORM_FAILURE
- FORM_FATAL
- ERROR_TYPE
- ERROR_CODE
- ERROR_TEXT

Each of these functions tests the value of a specific error variable that is set whenever a packaged procedure executes. These error variables record the following information:

- whether the most recent statement succeeded, failed, or caused a fatal error
- the type, number, and text of any associated error message

Because each packaged procedure sets the return values of these functions when it executes, the functions only report information on the most recently executed packaged procedure during a SQL*Forms (Run Form) session.

You can alter the flow of trigger processing by testing the value of any of the six error variables through the appropriate packaged function and then raising an exception or taking the appropriate action.

Example The following example demonstrates how a trigger can trap a failure condition for a packaged procedure. Notice that the example trigger tests for failure immediately after executing a packaged procedure. While this practice is recommended, it is not necessary. You can test the results of the last packaged procedure executed up until the time the application executes another packaged procedure or the SQL*Forms (Run Form) session ends.

1. Assume you want to read a default value into the ORDER_DATE field if the DUPLICATE_FIELD function fails during a key trigger because there is no previous record from which to copy a field value. Since DUPLICATE_FIELD can fail for more than one reason, you need to test for a particular error. The following key trigger performs this test (in the ELSIF clause) by checking for the desired error text.

   This trigger also tests for another possible failure message in order to demonstrate how you can test for more than one condition.

```
DUPLICATE_ FIELD;
IF ERROR_TEXT =  'Field is protected against update. '
    THEN
        SYNCHRONIZE;
ELSIF ERROR_TEXT =  'No previous record to copy value from. '
    THEN
        SELECT SYSDATE INTO : ORDER_DATE FROM DUAL;
END IF;
```

   Note: If your application is supported in multiple languages, avoid testing with the ERROR_TEXT packaged function. Instead, use the ERROR_TYPE and ERROR_CODE functions to test for a specific error.

Related Topics: ERROR_CODE packaged function 17-5
ERROR_TEXT packaged function 17-6
ERROR_TYPE packaged function 17-7
FORM_FAILURE packaged function 17-11
FORM_FATAL packaged function 17-12
FORM_SUCCESS packaged function 17-13

PL/SQL Compilation

PL/SQL blocks must be compiled before they can be executed. When a PL/SQL block is compiled, the compiler detects syntax errors, semantic errors, and references to non-existent objects or procedures.

When you use anonymous blocks and form-level procedures in V3 triggers, SQL*Forms compiles them immediately after you create or modify a V3 trigger and when you generate a form that contains V3 triggers.

SQL*Forms discards all of a form's existing compilation information when you save the form or before you create an INP file, retaining only trigger and form-level procedure source.

Compiling at Design Time

SQL*Forms first compiles the PL/SQL anonymous block in a V3 trigger after you write the trigger in the Trigger Definition form and press [Accept]. If the compiler identifies any errors in the block, SQL*Forms displays the Compilation Errors form. This form highlights the first error in the trigger and displays explanatory text for all of the errors found during compilation. SQL*Forms does not compile a trigger from the Trigger Definition form if you select the Disable Trigger Compilation design option.

SQL*Forms first compiles a form-level procedure after you write the procedure in the Procedure Definition form and press [Accept]. If the compiler identifies any errors in the procedure, SQL*Forms displays the Compilation Errors form. This form highlights the first error in the procedure and displays explanatory text for all of the errors found during compilation. SQL*Forms does not compile a form-level procedure from the Procedure Definition form if you select the Disable Trigger Compilation design option.

Related Topics: Disable Trigger Compilation design option 26-5

Compiling at Generation Time

SQL*Forms compiles all of the PL/SQL blocks in all of a form's triggers and form-level procedures when you generate the form.

Specifically, SQL*Forms compiles the following constructs in the following order:

1. SQL*Forms scans all form-level procedures and compiles each one's specification, or "spec."

   The procedure spec is the part of the procedure text that includes the procedure name and any associated argument list. SQL*Forms needs to compile this information before it can go on to compile procedures. This is because SQL*Forms must be able to verify that the referenced procedure exists and if its parameters allow it to be referenced in such a way.

2. SQL*Forms compiles each form-level procedure.
3. SQL*Forms identifies all V3 triggers and compiles each anonymous block.

How SQL*Forms notifies you of errors during this compilation depends on whether you generate the form from the command line or from SQL*Forms (Design).

Compiling PL/SQL from the Command Line

If you generate a form from the command line, SQL*Forms compiles all of the triggers and form-level procedures in the form. This compilation generates messages that identify all form-level procedures and triggers and that report any errors and error locations. SQL*Forms displays the messages, one at a time, on your terminal screen. You can suppress the display with the -b command line switch.

Related Topics: -b switch, SQL*Forms (Generate) 23-14

Compiling PL/SQL from SQL*Forms Design)

If you generate a form from SQL*Forms (Design) (i.e., via the Generate option on the Action menu), SQL*Forms compiles all of the triggers and form-level procedures in the form. This compilation generates messages that identify all form-level procedures and triggers and that report any errors and error locations:

- SQL*Forms displays hte message `Errors and warnings written to file` *`formname. err,`* where *`formname`* is the name of the current form.
- SQL*Forms writes the messages to the designated file.

If the compilation detects no errors, SQL*Forms erases the formname.err file.

Compilation Error Messages

As SQL*Forms compiles each trigger, it displays compilation error messages one at a time. These messages take the following form:

```
Compiling object_name object_type on owner_name owner_ type. . .
Error error_num at line line_num, column col_num:
    error_ text
```

where:

object_name	Is the unambiguous name of the trigger or form-level procedure that SQL*Forms is compiling.
object_type	Is “trigger”, “procedure”, or “procedure spec.”
owner_name	Is the unambiguous name of the form, block, or field to which the trigger is attached.
owner_ type	Is the word “form”, “block”, or “field.”
error_num	Is the documented error number.
line_num	Is the line number of the trigger or form-level procedure where the error occurred. This is the line number within the Trigger Text scroll region or the Procedure Text scroll region.
col_num	Is the column number of the trigger or form-level procedure where the error occurred. This is the column number within the Trigger Text scroll region or the Procedure Text scroll region.
error_text	Is a complete description of the error.

For example, your terminal screen might display the following message:

```
Compiling KEY-COMMIT trigger on ORDER_ ENTRY form . . .
Error 313 at line 1, column 4:
    ‘COMIT_FORM‘  not declared in this scope
```

The *ORACLE Error Messages and Codes Manual* documents compilation error numbers and text.

CHAPTER

16

PACKAGED PROCEDURES

A packaged procedure is a built-in PL/SQL procedure that is available in all forms. Each packaged procedure executes a SQL*Forms function, such as moving to a field or executing a query. By using packaged procedures, you can invoke through triggers most of the functions that are invoked by operator function keys and some functions that can only be invoked through a trigger.

Use packaged procedures to build triggers that perform the following tasks:

- reduce the amount of repetitive or complex data entry by operators
- control the flow of an application by forcing the operator to navigate along a predetermined path
- ensure that operators always follow the same sequence of actions when they use a form
- modify the SQL*Forms default processing for processes such as validation and locking

You can only use packaged procedures in V3 triggers and form-level procedures. There are different types of packaged procedures and restrictions on the types you can use in specific triggers.

Note: If you have used previous versions of SQL*Forms, you might have referred to the equivalent of a packaged procedure as a "macro" (e.g., the CALL macro).

This chapter covers the following topics:

- SQL*Forms functions without packaged procedures
- packaged procedure definitions
- unrestricted packaged procedures
- restricted packaged procedures

This chapter does not cover packaged procedure processing. It also does not cover V2 function codes. V2 function codes are the equivalent of packaged procedures in previous versions of SQL*Forms and are used in V2 triggers.

Related Topics: Trigger processing 4-5
V2 function codes F-12

Functions without Packaged Procedures

Some SQL*Forms (Run Form) functions do not have corresponding packaged procedures. These functions are not practical for trigger execution because they are often performed by the terminal or user interface management system and not by SQL*Forms.

[Beginning of Line]
[Delete Backward]
[Delete character]
[Delete Line]
[End of Line]
[First Line]
[Insert Line]
[Insert/Replace]
[Last Line]
[Menu]
[Move Left]
[Move Right]
[Scroll Left]
[Scroll Right]
[Search]

Packaged Procedure Definitions

This section presents each packaged procedure in alphabetical order. The syntax of the entire packaged procedure and a description are provided.

For detailed information on the processing of each packaged procedure (including special cases and failure), refer to the appropriate flow charts in Chapter 21, "Event Processing."

Related Topics: Event processing 21-1

Packaged Procedure Arguments

Some packaged procedures require character string or number arguments. For conciseness, the syntax diagrams for the packaged procedures represent these arguments with names that describe what the argument must evaluate to. For example, the GO_FIELD packaged procedure has an argument called field_name. This syntax says that the GO_FIELD packaged procedure requires a character string argument that evaluates to the name of a field-in this case, the name of the field to which you want to navigate.

There are many different ways to construct a valid character string or number argument. To use packaged procedures correctly, you should be aware of all of the forms that these arguments can take. The following information details the types of expressions that you can use for packaged procedure arguments that must evaluate to character strings or numbers.

literal reference	*'argument_string'* or *number* **where:** argument_ string — Is the character string that satisfies the argument. *number* — Is the integer that satisfies the argument.
field value reference	*:field_name* where *field_name* is the name of a field whose value is the character string or integer that satisfies the argument.
SQL*Forms variable reference	*:variable_name* or TO-NUMBER(: *variable_name*) where *variable_name* is the name of a SQL*Forms variable whose value is the character string or integer that satisfies the argument. The named variable can be a system variable or a global variable. Use the TO_NUMBER function to convert a character string value to the desired integer value.
PL/SQL variable reference	*variable_name* where *variable_name* is the name of a PL/SQL local variable whose value is the character string or integer that satisfies the argument. Refer to the *PL/SQL User's Guide and Reference* for information on PL/SQL local variables.
indirect reference	NAME_IN *(argument_reference)* where *argument_reference* can be any kind of argument reference (including another indirect reference) that evaluates to the name of a field or variable whose value is the character string that satisfies the argument. Do not use indirect reference for number arguments.

Arguments can also be constructed with combinations of these expressions. For example, you could use a field value reference that refers to a field that contains a SQL*Forms variable reference. You can also use other general PL/SQL expressions such as concatenation for character string arguments and addition for number arguments.

Note: Any argument called field.name can evaluate to a string that includes a block identifier in the form blockname.fieldname. Any argument called form_name can evaluate to a string that includes a system-specific path name for the form.

Related Topics: Global variables 18-2
NAME_IN packaged function 17-16
System variables 18-3

ABORT_QUERY

Syntax: `ABORT_QUERY;`

Description: ABORT_QUERY closes a query that is open in the current block.

Related Topics: ABORT_QUERY, flow chart 21-4

ANCHOR_VIEW

Syntax: `ANCHOR_VIEW (page_number, x_coordinate, y_coordinate);`

where:

page_ number	Specifies the integer number of the page in the current form of which the view consists.
x_coordinate	Specifies the integer number of the X coordinate on the screen where you want to place the upper left corner of a view.
y_coordinate	Specifies the integer number of the Y coordinate on the screen where you want to place the upper left corner of a view.

Description: ANCHOR_VIEW moves a view of a page to a new location on the screen. This procedure effectively changes where on the screen the operator sees the view.

Note:

- You cannot move a view so that any part of it would display outside of the screen area. Such a move causes an error.
- The location that ANCHOR_VIEW specifies holds for as long as the current form is active or until another ANCHOR_VIEW changes the location of the view.
- You can use ANCHOR_VIEW for pop-up pages only.

Related Topics: ANCHOR_VIEW, flow chart 21-5
MOVE_VIEW, packaged procedure 16-23
Pages, characteristics 10-4
Pages, pop-up 13-5

BELL

Syntax: `BELL;`

Description: BELL sets the terminal bell to ring the next time the terminal screen synchronizes with the internal state of the form.

Related Topics: BELL, flow chart 21-6

BLOCK_MENU

Syntax: `BLOCK_MENU;`

Description: BLOCK_MENU displays the block menu and accepts operator input. If the operator enters a valid selection, BLOCK_MENU navigates to the appropriate block.

Related Topics: BLOCK_MENU, flow chart 21-7

BREAK

Syntax: `BREAK;`

Description: If the current form is running in debug mode, BREAK halts form execution and displays the Debug Mode Options menu. If the current form is not running in debug mode, BREAK does nothing.

Related Topics: BREAK, flow chart 21-8
Debug facility 28-1

CALL

Syntax: `CALL (form_name[, {HIDE|NO_HIDE} [,{ DO_REPLACE | NO REPLACE }]];`

Description: CALL runs an indicated form while keeping the parent form active. SQL*Forms runs the called form with the same SQL*Forms (Run Form) options as the parent form. When the called form is exited-through the EXIT function or as a result of navigational failure-processing resumes in the parent format the point the CALL occurred.

By default, CALL uses the HIDE and NO_REPLACE parameters. The HIDE parameter causes SQL*Forms to clear the calling form from the screen before drawing the called form. If you use the NO_HIDE parameter, SQL*Forms does not clear the calling form before drawing the called form. This means that if you call a form with NO_HIDE, and some page of the called form is smaller than the screen area, the calling form displays in the background.

The NO_REPLACE parameter causes SQL*Forms to keep the default menu application of the calling form. If you use the DO_REPLACE parameter, SQL*Forms replaces the default menu application of the calling form with the default menu application of the called form.

When you call a form, SQL*Forms issues a savepoint for the called form. If the CLEAR_FORM function causes a rollback when the called form is current, SQL*Forms rolls back uncommitted changes to this savepoint.

Related Topics: CALL, flow chart 21-10
Default Menu Application form characteristic 9-3
Page Size page characteristic 10-6
Savepoints, issued when calling a form 3-18

CALL_INPUT

Syntax: `CALL_INPUT;`

Description: CALL_INPUT accepts and processes function key input from the operator. When CALL_INPUT is terminated, processing resumes in the format the point CALL_INPUT occurred.

Note: CALL_INPUT is included in Version 3.0 of SQL*Forms for compatibility with previous versions. Oracle Corporation recommends that you do not use this packaged procedure in new applications.

Related Topics: CALL_INPUT, flow chart 21-11

CALL_QUERY

Syntax: `CALL_QUERY (form_name[, { HIDE | NO_HIDE) [,{DO_REPLACE | NO REPLACE }]]);`

Description: CALL_QUERY runs an indicated form while keeping the parent form active. SQL*Forms runs the called form with the same options as the parent form *and* as a query-only form. In a query-only form, operators (and triggers) can query data, but cannot perform inserts, updates, or deletes.

When the called form is exited-through the EXIT function or as a result of navigational failure-processing resumes in the parent form at the point the CALL_QUERY occurred.

By default, CALL_QUERY uses the HIDE and NO_REPLACE parameters. The HIDE parameter causes SQL*Forms to clear the calling form from the screen before drawing the called form. If you use the NO_HIDE parameter, SQL*Forms does not clear the calling form before drawing the called form. This means that if you call a form with NO_HIDE, and some page of the called form is smaller than the screen area, the calling form displays in the background.

The NO_REPLACE parameter causes SQL*Forms to keep the default menu application of the calling form. If you use the DO_REPLACE parameter, SQL*Forms replaces the default menu application of the calling form with the default menu application of the called form.

When you call a form, SQL*Forms issues a savepoint for the called form. If the CLEAR_FORM function causes a rollback when the called form is current, SQL*Forms rolls back uncommitted changes to this savepoint.

Related Topics: CALL_QUERY, flow chart 21-12
Default Menu Application form characteristic 9-3
Page Size page characteristic 10-6
Savepoints, issued when calling a form 3-18

CLEAR_BLOCK

Syntax: `CLEAR_ BLOCK [ (ASK COMMIT | DO_ COMMIT | NO_ COMMIT | NO_VALIDATE)];`

Description: CLEAR_BLOCK flushes the current block. If there are changes in the current block that have not been posted or committed, the behavior of CLEAR_BLOCK depends on the parameters you supply:

Parameter	*Effect*
no parameters	SQL*Forms prompts the operator to commit the changes during CLEAR_BLOCK processing.
ASK_COMMIT	SQL*Forms prompts the operator to commit the changes during CLEAR_BLOCK processing.
DO_COMMIT	SQL*Forms validates the changes, performs a commit, and flushes the current block without prompting the operator.
NO_COMMIT	SQL*Forms validates the changes and flushes the current block without performing a commit or prompting the operator.
NO_VALIDATE	SQL*Forms flushes the current block without validating the changes, committing the changes, or prompting the operator.

Related Topics: CLEAR_BLOCK, flow chart 21-18

CLEAR_EOL

Syntax: `CLEAR_EOL;`

Description: CLEAR_EOL clears the current field's value from the current cursor position to the end of the line, or field.

Related Topics: CLEAR_EOL, flow chart 21-19

CLEAR_FIELD

Syntax: `CLEAR_FIELD;`

Description: CLEAR_FIELD clears the entire value of the current field-regardless of the current cursor position-changing the field's value to NULL.

Related Topics: CLEAR_FIELD, flow chart 21-20

CLEAR_FORM

Syntax:
```
CLEAR_FORM[( ASK_COMMIT | DO_COMMIT | NO_COMMIT |
NO_VALIDATE[, TO_SAVEPOINT | FULL_ ROLLBACK] )] ;
```

Description: CLEAR_FORM flushes the current form. If there are changes in the current form or any called form that have not been posted or committed, the behavior of CLEAR_FORM depends on the parameters you supply:

Parameter	*Effect*
no parameters	SQL*Forms prompts the operator to commit the changes during CLEAR.FORM processing.
ASK_COMMIT	SQL*Forms prompts the operator to commit the changes during CLEAR_FORM processing.
DO_COMMIT	SQL*Forms validates the changes, performs a commit, and flushes the current form without prompting the operator.
NO_COMMIT	SQL*Forms validates the changes and flushes the current form without performing a commit or prompting the operator.
NO_VALIDATE	SQL*Forms flushes the current form without validating the changes, committing the changes, or prompting the operator.
TO_SAVEPOINT	SQL*Forms rolls back all uncommited changes (including posted changes) to the current form's savepoint.
FULL_ ROLLBACK	SQL*Forms rolls back all uncommitted changes (including posted changes) which were made during the current SQL*Forms (Run Form) session. You cannot specify a FULL_ROLLBACK from a form that is running in post-only mode.

Note: If you use a PL/SQL ROLLBACK statement in an anonymous block or a form-level procedure, SQL*Forms interprets that statement as a CLEAR_FORM packaged procedure with no parameters.

Related Topics: CLEAR_FORM, flow chart 21-21
Savepoints, issued when posting 3-17

CLEAR_RECORD

Syntax: `CLEAR_RECORD;`

Description: CLEAR_RECORD flushes the current record from the block. If a query is open in the block, SQL*Forms fetches a record to refill the block.

Related Topics: CLEAR_RECORD, flow chart 21-22

COMMIT_FORM

Syntax: `COMMIT.FORM;`

Description: COMMIT_FORM updates data in the database to match data in the form. SQL*Forms first validates the form. If there are changes to commit to the database, for each block in the form SQL*Forms posts deletes, inserts, and updates to the database and performs a database commit. When the commit completes, SQL*Forms releases all table and row locks held by the operator.

If you have posted data to the database during the current SQL*Forms (Run Form) session, COMMIT_FORM commits this data to the database.

Note: If you use a PL/SQL COMMIT statement in an anonymous block or a form-level procedure, SQL*Forms interprets that statement as a COMMIT_FORM packaged procedure.

Related Topics: COMMIT_FORM, flow chart 21-23
POST, packaged procedure 16-26

COPY

Syntax: `COPY (value_string,field_name);`

Description: COPY writes a value into a field. COPY exists specifically to write a value into a field that is referenced through the NAME_IN packaged function. This feature exists because you cannot use standard PL/SQL syntax to set a referenced field equal to a value.

Related Topics: COPY, flow chart 21-24

COPY_REGION

Syntax: `COPY_REGION;`

Description: COPY_REGION functions in scroll regions. It copies a selected region from the screen and stores it in the paste buffer until you cut or copy another selected region.

Related Topics: COPY_REGION, flow chart 21-25

COUNT_QUERY

Syntax: `COUNT_QUERY;`

Description: COUNT_QUERY identifies the number of rows that a query will retrieve for the current block. If there are changes to commit in the block, SQL*Forms prompts the operator to commit them during COUNT_QUERY processing.

Related Topics: COUNT_QUERY, flow chart 21-26

CREATE_RECORD

Syntax: `CREATE_ RECORD ;`

Description: CREATE_RECORD creates a new record in the current block after the current record. SQL*Forms then navigates to the new record.

Related Topics: CREATE_RECORD, flow chart 21-27

CUT_REGION

Syntax: `CUT_REGION;`

Description: CUT_REGION functions in scroll regions. It removes a selected region from the screen and stores it in the paste buffer until you cut or copy another selected region.

Related Topics: CUT_REGION, flow chart 21-28

DEFAULT_VALUE

Syntax: `DEFAULT_VALUE` *`(value_string, variable_name);`*

Description: DEFAULT_VALUE copies an indicated value to an indicated variable if the variable's current value is NULL. If the variable's current value is not NULL, DEFAULT_VALUE does nothing. (Therefore, for fields this packaged procedure works identically to using the COPY packaged procedure on a NULL field.) If the variable is an undefined global variable, SQL*Forms creates the variable.

Note: The DEFAULT_VALUE packaged procedure *does not* perform the same function as the Default Value field characteristic.

Related Topics: DEFAULT_VALUE, flow chart 21-29
Default Value field characteristic 12-10

DELETE_RECORD

Syntax: `DELETE_RECORD;`

Description: DELETE_RECORD removes the current record from the block. If the record corresponds to a row in the database, SQL*Forms locks the record before removing it and marking it as a delete. If a query is open in the block, SQL*Forms fetches a record to refill the block if necessary.

Related Topics: DELETE_RECORD, flow chart 21-30

DISPLAY_ERROR

Syntax: `DISPLAY_ERROR;`

Description: DISPLAY_ERROR displays the Display Error screen if there is a logged error. When the operator presses a function key, SQL*Forms redisplays the form.

Related Topics: DISPLAY_ERROR, flow chart 21-32

DISPLAY_FIELD

Syntax: `DISPLAY_FIELD ( variable_name, display_attribute_name);`

Description: DISPLAY_FIELD modifies the way a field appears on the terminal by assigning a specified display attribute to the field. The names of available display attributes are stored in the Oracle*Terminal mapping with which the form is running. If you specify an invalid display attribute, SQL*Forms assigns the default display attribute for the field as specified in the current mapping.

You can reference any field in the current form. Note that DISPLAY_FIELD only affects the display of the current instance of the field; other instances of the specified field are not affected. This means that if you specify a display change for a field that exists in a multi-record block, DISPLAY_FIELD only changes the instance of that field that belongs to the block's current record.

Any change made by a DISPLAY_FIELD packaged procedure is effective until another DISPLAY_FIELD references the same field or the current form is exited.

Related Topics: DISPLAY_FIELD, flow chart 21-34
Oracle*Terminal 25-10

DISPLAY_PAGE

Syntax: `DISPLAY_PAGE (page_number, display_attribute_name) ;`

Description: DISPLAY_PAGE modifies the way a page appears on the terminal by assigning a specified display attribute to the page. The names of available display attributes are stored in the Oracle*Terminal mapping with which the form is running. If you specify an invalid display attribute, SQL*Forms assigns the default display attribute for the page as specified in the current mapping.

You can reference any page in the current form. Any change made by a DISPLAY_PAGE packaged procedure is effective until another DISPLAY_PAGE references the same page or the current form is exited.

Related Topics: DISPLAY_PAGE, flow chart 21-35
Oracle*Terminal 25-10

DO_KEY

Syntax: `DO_KEY (packaged_procedure_name) ;`

Description: DO_KEY executes the key trigger that corresponds to the specified packaged procedure. If no such key trigger exists, then the specified packaged procedure executes. This behavior is analagous to the user's pressing the corresponding function key.

The following table lists all packaged procedures that have corresponding key triggers. The table also shows the SQL*Forms (Run Form) function keys that are associated with the key triggers:

Packaged Procedure	*Key Trigger*	*Associated Function Key*
ABORT_QUERY	Key-EXIT	[Exit/Cancel]
BLOCK_MENU	Key-MENU	[Block Menu]
CLEAR_BLOCK	Key_CLRBLK	[Clear Block]
CLEAR_FORM	Key—CLRFRM	[Clear Form]
CLEAR_RECORD	Key-CLRREC	[Clear Record]
COMMIT_FORM	Key-COMMIT	[Commit]
COUNT_QUERY	Key-CQUERY	[Count Query Hits]
CREATE_RECORD	Key-CREREC	[Insert Record]
DELETE_RECORD	Key-DELREC	[Delete Record]
DOWN	Key-DOWN	[Down]
DUPLICATE_FIELD	Key-DUPFLD	[Duplicate Field]
DUPLICATE_RECORD	Key-DUPREC	[Duplicate Record]
EDIT_FIELD	Key-EDIT	[Edit]
ENTER	Key-ENTER	[Enter]
ENTER_QUERY	Key-ENTQRY	[Enter Query]
EXECUTE_QUERY	Key-EXEQRY	[Execute Query]
EXIT_FORM	Key-EXIT	[Exit/Cancel]
HELP	Key-HELP	[Help]
LIST_VALUES	Key-LISTVAL	[List]
LOCK_RECORD	Key-UPDREC	[Lock Record]
NEXT_BLOCK	Key-NXTBLK	[Next Block]
NEXT_FIELD	Key-NXTFLD	[Next Field]
NEXT_KEY	Key-NXTKEY	[Next Primary Key Field]
NEXT_RECORD	Key-NXTREC	[Next Record]
NEXT_SET	Key-NXTSET	[Next Set of Records]
PREVIOUS_BLOCK	Key-PRVBLK	[Next Block]
PREVIOUS_FIELD	Key-PRVFLD	[Previous Field]
PREVIOUS_RECORD	Key-PRVREC	[Previous Record]
PRINT	Key-PRINT	[Print]
SROLL_DOWN	Key-SCRDWN	[Scroll Down]
SCROLL_UP	Key-SCRUP	[Scroll Up]
UP	Key-UP	[Up]

Related Topics: Do_KEY, flow chart 21-33
Key Triggers 14-8

DOWN

Syntax: `DOWN;`

Description: DOWN navigates to the instance of the current field in the record with the next higher sequence number. If necessary, SQL*Forms fetches a record to which to navigate. If SQL*Forms has to create a record to which to navigate, DOWN navigates to the first enterable field in the record.

Related Topics: DOWN, flow chart 21-37

DUPLICATE_FIELD

Syntax: `DUPLICATE_FIELD;`

Description: DUPLICATE_FIELD assigns the current field the same value as the instance of this field in the previous record.

Related Topics: DUPLICATE_FIELD, flow chart 21-36

DUPLICATE_RECORD

Syntax: `DUPLICATE_RECORD;`

Description: DUPLICATE_RECORD copies the value of each field in the record with the next lower sequence number to the corresponding fields in the current record. The current record must not correspond already to a row in the database. If it does, an error occurs.

Related Topics: DUPLICATE_RECORD, flow chart 21-38

EDIT_FIELD

Syntax: `EDIT_FIELD[ (x_coordinate, y_coordinate, width, height) ] ;`

where:

x_coordinate	Specifies the integer number of the X coordinate on the screen where you want to place the upper left corner of the pop-up field editor.
y_coordinate	Specifies the integer number of the Y coordinate on the screen where you want to place the upper left corner of the pop-up field editor.
width	Specifies the integer number of the new width, in characters, of the pop-up field editor.
height	Specifies the integer number of the new height, in characters, of the pop-up field editor. If you specify a height less than 5, SQL*Forms will set the height equal to 5.

Description: EDIT_FIELD invokes the SQL*Forms (Run Form) field editor for the current field and puts the form in Edit mode.

You can use the optional EDIT_FIELD parameters to specify the location and dimensions of the pop-up window with which the field editor is associated. If you do not use these parameters, SQL*Forms invokes the field editor with its default location and dimensions.

Related Topics: EDIT_FIELD, flow chart 21-39
Field editor 13-2

ENTER

Syntax: `ENTER;`

Description: ENTER validates data in the current validation unit.

Related Topics: ENTER, flow chart 21-40
Validation unit 3-7

ENTER_QUERY

Syntax:
```
ENTER_QUERY [ ( {ALL_RECORDS [ , FOR-UPDATE [ , NOWAIT]] |
FOR_UPDATE[ , NOWAIT ] } ) ] ;
```

Description: The behavior of ENTER_QUERY varies depending on any parameters you supply.

- `ENTER_QUERY;` flushes the current block and puts the form in Enter Query mode. If there are changes to commit, SQL*Forms prompts the operator to commit them during the ENTER_QUERY event.
- `ENTER_QUERY (ALL_RECORDS);` performs the same action as `ENTER_QUERY;`, except that when EXECUTE_QUERY is invoked, SQL*Forms fetches all of the selected records.
- `ENTER_QUERY (FOR_UPDATE );` perform the same actions as `ENTER_QUERY;` , except that when EXECUTE_QUERY is invoked, SQL*Forms attempts to lock all of the selected records immediately.
- `ENTER_QUERY (ALL_RECORDS , FOR_UPDATE);` performs the same actions as `ENTER QUERY;`, except that when EXECUTE_QUERY is invoked, SQL*Forms attempts to lock all of the selected records immediately and fetches all of the selected records.

- Anytime that you use the FOR_UPDATE parameter, you can also use the NOWAIT parameter. When you use NOWAIT, SQL*Forms displays the following message if it cannot immediately obtain a lock on a record:

  ```
  Attempting to reserve record for update or delete (CTRL-C
  to cancel) . . .
  ```

  Without the NOWAIT parameter, SQL*Forms keeps trying to obtain the locks without letting the operator cancel the process.

Note: Oracle Corporation recommends that you use the ALL_RECORDS and FOR_UPDATE parameters with caution. Fetching a very large number of rows could cause a long delay. Locking a very large number of rows at once requires many resources.

Related Topics: ENTER_QUERY, flow chart 21-47
EXECUTE_QUERY, flow chart 21-50

ERASE

Syntax: `ERASE ( global_variable_name);`

Description: ERASE removes an indicated global variable, so that it no longer exists, and releases the memory associated with the global variable.

Related Topics: ERASE, flow chart 21-48

EXECUTE_QUERY

Syntax:
```
EXECUTE_QUERY [ ( { ALL_RECORDS [ , FOR_UPDATE [ , NOWAIT] ] |
FOR_UPDATE [ , NOWAIT] } ) ] ;
```

Description: The behavior of EXECUTE_QUERY varies depending on the parameters you supply.

- `EXECUTE_QUERY;` flushes the current block, opens a query, and fetches a number of selected records. If there are changes to commit, SQL*Forms prompts the operator to commit them during EXECUTE_QUERY processing.
- `EXECUTE_QUERY (ALL_RECORDS);` performs the same actions as `EXECUTE_ QUERY ; /` except that SQL*Forms fetches all of the selected records.
- `EXECUTE_QUERY (FOR_UPDATE);` performs the same actions as `EXECUTE_QUERY;,` except that SQL*Forms attempts to lock all of the selected records immediately.

- `EXECUTE_QUERY(ALL_RECORDS, FOR.UPDATE);` Performs the same actions as `EXECUTE_QUERY;`, except the SQL*Forms attempts to lock all of the selected records immediately and fetches all of the selected records.
- Anytime that you use the FOR_UPDATE parameter, you can also use the NOWAIT parameter. When you use NOWAIT, SQL*Forms displays the following message if it cannot immediately obtain a lock on a record:

  ```
  Attempting to reserve record for update or delete (CTRL-C
  to cancel) . . .
  ```

  Without the NOWAIT parameter, SQL*Forms keeps trying to obtain the locks without letting the operator cancel the process.

Note: Oracle Corporation recommends that you use the ALL_RECORDS and FOR_UPDATE parameters with caution. Fetching a very large number of rows could cause a long delay. Locking a very large number of rows at once requires many resources.

Related Topics: EXECUTE_QUERY, flow chart 21-50

EXECUTE_TRIGGER

Syntax: `EXECUTE_TRIGGER` *`(trigger_name);`*

Description: EXECUTE_TRIGGER executes an indicated user-named trigger.

Related Topics: EXECUTE_TRIGGER, flow chart 21-51

EXIT_FORM

Syntax:
```
EXIT_FORM[( ASK COMMIT | DO_COMMIT | NO_COMMIT |
NO_VALIDATE [ , NO_ROLLBACK])]
```

Description: During a CALL_INPUT, EXIT_FORM terminates the CALL_INPUT function. In all other contexts, EXIT_FORM navigates to "outside" of the form. If there are changes in the current form that have not been posted or committed, the behavior of EXIT_FORM depends on the parameters you supply:

Parameter	*Effect*
no parameters	SQL*Forms prompts the operator to commit the changes during EXIT_FORM processing.
ASK_COMMIT	SQL*Forms prompts the operator to commit the changes during EXIT_FORM processing.
DO_COMMIT	SQL*Forms validates the changes, performs a commit, and exits the current form without prompting the operator.
NO_COMMIT	SQL*Forms validates the changes and exits the current form without performing a commit or prompting the operator.
NO_VALIDATE	SQL*Forms exits the current form without validating the changes, committing the changes, or prompting the operator.
NO_ROLLBACK	SQL*Forms exits the current form, preserving uncommitted changes.

Related Topics: EXIT_FORM, flow chart 21-52

FIRST_RECORD

Syntax: `FIRST_RECORD;`

Description: FIRST_RECORD navigates to the first record in the block's list of records.

Related Topics: FIRST_RECORD, flow chart 21-56

GO_BLOCK

Syntax: `GO_BLOCK ( block_name) ;`

Description: GO_BLOCK navigates to an indicated block. If the target block is non-enterable, an error occurs.

Related Topics: GO_BLOCK, flow chart 21-57

GO_FIELD

Syntax: `GO_FIELD (field_name);`

Description: GO_FIELD navigates to an indicated field.

Related Topics: GO_FIELD, flow chart 21-58

GO_RECORD

Syntax: `GO_RECORD (record_number);`

where *record.number* specifies any integer value that PL/SQL can evaluate to a number. This includes values such as `TO_NUMBER ( : SYSTEM. TRIGGER_ RECORD ) + 8.`

Description: GO_RECORD uses a sequence number to navigate to an indicated record.

Note: You can use the SYSTEM.CURSOR_RECORD or SYSTEM.TRIGGER_RECORD system variables to determine a record's sequence number.

Related Topics: GO_RECORD, flow chart 21-59
SYSTEM.CURSOR_RECORD 18-8
SYSTEM.TRIGGER_RECORD 18-13

HELP

Syntax: `HELP;`

Description: HELP displays the current field's hint message on the message line. If the hint message is already displayed, HELP displays the detailed help screen for the field.

Related Topics: HELP, flow chart 21-60

HIDE_MENU

Syntax: `HIDE_MENU;`

Description: HIDE_MENU makes the current SQL*Menu menu disappear if it is currently displayed, uncovering any part of the form display that the menu had covered. The menu will redisplay if the SHOW_MENU packaged procedure is invoked or the operator presses [Menu].

Related Topics: HIDE_MENU, flow chart 21-61
SHOW_MENU, packaged procedure 16-33
SQL*Menu, connecting to 25-3

HIDE_PAGE

Syntax: `HIDE_PAGE (page_number) ;`

where *page_ number* is the integer number of the page in the current form that you want to hide.

Description: HIDE_PAGE makes an indicated page disappear from the screen.

Note:

- You cannot hide the current page of a form.
- You should use HIDE_PAGE for pop-up pages only.

Related Topics: HIDE_PAGE, flow chart 21-62
Pages, characteristics 10-4
Pages, pop-up 13-5

HOST

Syntax: `HOST ( system_command_ string[ , NO_ SCREEN] ) ;`

Description: HOST executes an indicated operating system command. If you use the NO_SCREEN parameter, SQL*Forms does not clear the screen or prompt the operator to return from the command.

Related Topics: HOST, flow chart 21-63

LAST_RECORD

Syntax: `LAST_RECORD;`

Description: LAST_RECORD navigates to the last record in the block's list of records. If a query is open in the block, SQL*Forms fetches the remaining selected records into the block's list of records.

Related Topics: LAST_RECORD, flow chart 21-64

LIST_VALUES

Syntax: `LIST_VALUES (NO RESTRICT|RESTRICT) ;`

Description: LIST_VALUES displays the list of values for the current field. The list of values remains displayed until the operator presses [Exit/Cancel] or selects a value into a field.

By default, LIST_VALUES uses the NO_RESTRICT parameter. This parameter causes SQL*Forms not to use the automatic search and complete feature. If you use the RESTRICT parameter, SQL*Forms uses the automatic search and complete feature.

Automatic Search and Complete

With the automatic search and complete feature, a list of values evaluates a field's current value as a search value. That is, if an operator presses [List] in a field that has a list of values, SQL*Forms checks to see if the field contains a value. If the field does contain a value, SQL*Forms automatically uses that value as if the operator entered the value into the list of value's search field and pressed [List] to narrow the list.

If the field value would narrow the list to only one value, SQL*Forms does not display the list of values, but automatically reads the correct value into the field.

Related Topics: LIST_VALUES, flow chart 21-71

LOCK_RECORD

Syntax: `LOCK_RECORD;`

Description: LOCK_RECORD attempts to lock the row in the database that corresponds to the current record.

Related Topics: LOCK_RECORD, flow chart 21-75

MESSAGE

Syntax: `MESSAGE (message_string);`

Description: MESSAGE displays specified text on the message line.

The character string may be up to 200 characters long. If you want to display a single quote character in a message, use two single quotes in the String (e.g., `Record isn' 't available.`). Maximum display length is display device-dependent.

Related Topics: MESSAGE, flow chart 21-77

MOVE_VIEW

Syntax: `MOVE VIEW (page_number, x_coordinate, y-coordinate) ;`

where:

page_number	Specifies the integer number of the page in the current form of which the view consists.
x_coordinate	Specifies the integer number of the X coordinate on the page where you want to place the upper left corner of a view.
y_coordinate	Specifies the integer number of the Y coordinate on the page where you want to place the upper left corner of a view.

Description: MOVE_VIEW moves a view of a page to a new location on that page. This procedure effectively changes what part of the page the operator sees in the view. It does not change the position of the view on the screen.

Note:

- You cannot move a view so that it would display an area outside of the page. Such a move causes an error.
- You can use MOVE_VIEW for pop-up pages only.

Related Topics: ANCHOR_VIEW, packaged procedure 16-5
MOVE_VIEW, flow chart 21-78
Pages, characteristics 10-4
Pages, pop-up 13-5

NEW_FORM

Syntax: `NEW_FORM ( form_name [ , NO_ROLLBACK ] ) ;`

Description: NEW_FORM exits the current form and enters the indicated form. If you use the NO_ROLLBACK option, SQL*Forms preserves uncommitted changes when leaving the form. The calling form is terminated as the parent form. If the calling form had been called by a higher level form, SQL*Forms keeps the higher level call active and treats it as a call to the new form. SQL*Forms releases memory (such as database cursors) that the terminated form was using.

SQL*Forms runs the new form with the same SQL*Forms (Run Form) options as the parent form. If the parent form was a called form, SQL*Forms runs the new form with the same parameters (e.g., HIDE and NO_REPLACE) as the parent form.

Related Topics: CALL, packaged procedure 16-7
NEW_FORM, flow chart 21-80

NEXT_BLOCK

Syntax: `NEXT_BLOCK;`

Description: NEXT_BLOCK navigates to the enterable block with the next higher sequence number than the current block. If there is no such block, NEXT_BLOCK navigates to the enterable block with the lowest sequence number. If there is no such block, NEXT_BLOCK navigates to the current block.

Related Topics: NEXT_BLOCK, flow chart 21-81

NEXT_FIELD

Syntax: `NEXT_FIELD;`

Description: NEXT_FIELD navigates to the enterable field with the next higher sequence number than the current field. If there is no such field, NEXT_FIELD navigates to the enterable field with the lowest sequence number. If there is no such field, NEXT_FIELD navigates to the current field.

If the validation unit is the field, NEXT_FIELD validates any fields with sequence numbers greater than the current field or less than the target field.

Related Topics: NEXT_FIELD, flow chart 21-82

NEXT_KEY

Syntax: `NEXT.KEY;`

Description: NEXT_KEY navigates to the enterable, primary key field with the next higher sequence number than the current field. If there is no such field, NEXT_KEY navigates to the enterable, primary key field with the lowest sequence number. If there is no primary key field in the current block, an error occurs.

If the validation unit is the field, NEXT_KEY validates any fields with sequence numbers greater than the current field or less than the target field.

Related Topics: NEXT_KEY, flow chart 21-84
Primary Key, field characteristic 12-19

NEXT_RECORD

Syntax: `NEXT_RECORD;`

Description: NEXT_RECORD navigates to the record with the next higher sequence number than the current record. If there is no such record, SQL*Forms might fetch or create a record. If the current record is a new record, NEXT_RECORD fails.

Related Topics: NEXT_RECORD, flow chart 21-86

NEXT_SET

Syntax: `NEXT_SET;`

Description: NEXT_SET fetches another set of records from the database and navigates to the first record that the fetch retrieves. NEXT_SET succeeds only if a query is open in the current block.

Related Topics: NEXT_SET, flow chart 21-87

PASTE_REGION

Syntax: `PASTE_REGION;`

Description: PASTE_REGION functions in scroll regions. It places the contents of the paste buffer (i.e., the selected region that was cut or copied most recently) in the screen, positioning the upper left corner of the pasted area at the cursor position.

Related Topics: PASTE_REGION, flow chart 21-89

PAUSE

Syntax: `PAUSE;`

Description: PAUSE suspends processing until the operator presses a function key. PAUSE might display an alert.

Related Topics: PAUSE, flow chart 21-90

POST

Syntax: `POST;`

Description: POST writes data in the form to the database, but does not perform a database commit. SQL*Forms first validates the form. If there are changes to post to the database, for each block in the form SQL*Forms writes deletes, inserts, and updates to the database.

Any data that you post to the database is committed in the database by the next COMMIT_FORM that executes during the current SQL*Forms (Run Form) session. Alternatively, this data is rolled back by the next CLEAR_FORM.

Related Topics: COMMIT_FORM, packaged procedure 16-11
POST, flow chart 21-91
Savepoints 3-17

PREVIOUS_BLOCK

Syntax: `PREVIOUS_BLOCK;`

Description: PREVIOUS_BLOCK navigates to the enterable block with the next lower sequence number than the current block. If there is no such block, PREVIOUS_BLOCK navigates to the enterable block with the highest sequence number. If there is no such block, PREVIOUS_BLOCK navigates to the current block.

Related Topics: PREVIOUS_BLOCK, flow chart 21-100

PREVIOUS_FIELD

Syntax: `PREVIOUS_ FIELD;`

Description: PREVIOUS_FIELD navigates to the enterable field with the next lower sequence number than the current field. If there is no such field, PREVIOUS_FIELD navigates to the enterable field with the highest sequence number. If there is no such field, PREVIOUS_FIELD navigates to the current field.

Related Topics: PREVIOUS_FIELD, flow chart 21-101

PREVIOUS_RECORD

Syntax: `PREVIOUS_RECORD;`

Description: PREVIOUS_RECORD navigates to the record with the next lower sequence number than the current record.

Related Topics: PREVIOUS_RECORD, flow chart 21-102

PRINT

Syntax: `PRINT;`

Description: PRINT writes the contents of the current screen to a file. SQL*Forms prompts the operator to indicate the filename to write the screen capture to, and whether to send the file to the system printer.

Related Topics: PRINT, flow chart 21-103

REDISPLAY

Syntax: `REDISPLAY;`

Description: REDISPLAY redraws the screen. This eliminates any extraneous system messages.

Related Topics: REDISPLAY, flow chart 21-110

REPLACE_MENU

Syntax:

```
REPLACE_MENU (menu_application-name
  [ , { BAR |FULL_ SCREEN |PULL_DOWN}
[, starting_menu_name [, group_name]]] ) ;
```

where:

menu_ application name	Specifies the SQL*Menu application that should replace the current menu application. This parameter replaces the application specified in the Default Menu Application form characteristic.
BAR | FULL_ SCREEN | PULL_DOWN	Specifies the display type of the menu. This parameter replaces the default menu type (PULL_DOWN). If you do not specify a menu type, SQL*Menu uses the PULL_DOWN type. Refer to the *SQL*Menu User's Guide and Reference* for a description of the available menu types.

starting_menu name — Specifies the menu within the menu application that SQL*Forms should use as the starting menu. This parameter replaces the menu specified in the Starting Menu Name form characteristic. If you do not specify a starting menu, SQL*Menu uses the menu with the same name as the menu application.

group_name — Specifies the security group that SQL*Menu should use. This parameter replaces the security group specified in the Menu Security Group form characteristic. If you do not specify a group name, SQL*Menu uses the current username to determine the group.

Description: REPLACE_MENU replaces the current SQL*Menu menu with the specified menu. It displays the new menu, but it does not make the new menu active. REPLACE_MENU also allows you to change the way the menu displays and the security group that SQL*Menu should use when determining which menu options to display.

Because REPLACE_MENU does not make the new menu active, SQL*Forms does not allow the menu to obscure any part of the active page. Therefore, all or part of the menu does not appear on the screen if the active page would cover it.

Related Topics: REPLACE_MENU, flow chart 21-111
SQL*Menu, connecting to 25-3

RESIZE_VIEW

Syntax: `RESIZE_VIEW (page_number, width, height) ;`

where:

page_ number — Specifies the integer number of the page in the current form of which the view consists.

width — Specifies the integer number of the new width, in characters, of the view.

height — Specifies the integer number of the new height, in characters, of the view.

Description: RESIZE_VIEW changes the size of a view of a page.

Note:

- You cannot resize a view so that it is larger than the page of which the view consists.
- When you resize a view, the upper left corner of the view usually remains fixed. However, resizing a view does affect this reference point if the procedure would cause the view to display an area outside of the page. In such a case, SQL*Forms automatically moves the view the minimum amount possible to legally position the view.

Related Topics: RESIZE_VIEW, flow chart 21-112
Pages, characteristics 10-4
Pages, pop-up 13-5

SCROLL_DOWN

Syntax: `SCROLL_DOWN;`

Description: SCROLL_DOWN scrolls the current block's list of records up by approximately 80 percent of the block's display. This action displays records that were "below" the block's display. If there are available records and a query is open in the block, SQL*Forms fetches records during SCROLL_DOWN processing.

After scrolling, SCROLL_DOWN navigates to the instance of the cursor field in the displayed record with the lowest sequence number.

Related Topics: SCROLL_DOWN, flow chart 21-114

SCROLL_UP

Syntax: `SCROLL_UP;`

Description: SCROLL_UP scrolls the current block's list of records down by approximately 80 percent of the block's display. This action displays records that were "above" the block's display.

After scrolling SCROLL_UP navigates to the instance of the cursor field in the displayed record with the lowest sequence number.

Related Topics: SCROLL_UP, flow chart 21-116

SET_FIELD

Syntax:

```
SET_ FIELD ( field_name,
{AUTO_HELP |AUTO_SKIP |DISPLAYED |ECHO |ENTERABLE |
FIXED_LENGTH |QUERYABLE |REQUIRED |UPDATE_NULL |
UPDATEABLE |UPPER_CASE}[, { ATTR ON |ATTR_OFF}]);
```

Description: SET_FIELD modifies a field by changing a specified field characteristic. Chracteristics are mapped to the following parameters:

Parameter	*Characteristic*
AUTO_HELP	Automatic Hint
AUTO_SKIP	Automatic Skip
DISPLAYED	Displayed
ECHO	Echo Input
ENTERABLE	Input Allowed
FIXED_LENGTH	Fixed Length
QUERYABLE	Query Allowed
REQUIRED	Required
UPDATE_NULL	Update if Null
UPDATEABLE	Update Allowed
UPPERCASE	Uppercase

By default, SET_FIELD uses the ATTR_ON parameter. This parameter causes SQL*Forms to set the specified characteristic ''on." If you use the ATTR_OFF parameter, SQL*Forms sets the specified characteristic "off."

Related Topics: Fields, characteristics 12-4
SET_FIELD, flow chart 21-118

Validation of Characteristic Changes

When you specify a change through the SET_FIELD packaged procedure, SQL*Forms validates the change before it adjusts the characteristic. If the change passes validation, SQL*Forms makes the change and leaves it in effect until another SET_FIELD changes the same characteristic or the current form is exited.

If the change fails validation SQL*Forms issues an error message. The following changes are restricted via the SET_FIELD packaged procedure under the specified conditions:

SET_FIELD Parameter	*Restricted Setting*	*Target Field Condition*
(All)	on/off	• located on page 0
ENTERABLE	on	• current field • Displayed field characteristic is turned off
	off	• current field
DISPLAYED	on/off	• current field
UPDATEABLE	on	• Input Allowed field characteristic is turned off
UPDATE_NULL	on	• Input Allowed field characteristic is turned off
QUERYABLE	on	• Displayed field characteristic is turned off
REQUIRED	on	• Input allowed field characteristic is turned off • Update Allowed and Update if Null field characteristics are turned off

Note: SQL*Forms does not consider the current contents of a field before allowing a characteristic change. If SET_FIELD changes a field characteristic that would affect how SQL*Forms validates the data in a field (i.e., FIXED_LENGTH, REQUIRED), the validation consequences are not retroactive. The new validation rules do not apply to the field until SQL*Forms next validates it under normal circumstances.

Propagation of Characteristic Changes

You can only specify a change to one field characteristic at a time through the SET_FIELD packaged procedure. However, one SET_FIELD statement can cause changes to more than one field characteristic if the additional changes are necessary to complete, or propagate, the intended change.

For example, if you turn off the Input Allowed field characteristic (via the ENTERABLE parameter), SQL*Forms also turns off the Update Allowed, Update if Null, and Required field characteristics.

The following table shows the SET_FIELD parameter settings that cause SQL*Forms to propagate changes across field characteristics:

SET_FIELD Parameter	***Specified Setting***	***Propagated Changes***
ENTERABLE	off	• turns off the Update Allowed field characteristic • turns off the Update if Null field characteristic • turns off the Required field characteristic
DISPLAYED	off	• turns off the Update Allowed field characteristic • turns off the Update if Null field characteristic • turns off the Required field characteristic • turns off the Query Allowed field characteristic • turns off the Input Allowed field characteristic
UPDATEABLE	on	• turns off the Update if Null field characteristic
UPDATE_NULL	on	• turns off the Update Allowed field characteristic

SET_INPUT_FOCUS

Syntax: `SET_INPUT_FOCUS(MENU);`

Description: SET_INPUT_FOCUS sets the input focus on the menu of the current form (character mode or block mode only); once trigger processing is completed, the menu will be active.

Related Topics: SET_INPUT_FOCUS, flow chart 21-119

SHOW_KEYS

Syntax: `SHOW_KEYS;`

Description: SHOW_KEYS displays the Show Keys screen. When the operator presses a function key, SQL*Forms redisplays the form as it was before SHOW_KEYS was invoked.

Related Topics: SHOW_KEYS, flow chart 21-120

SHOW_MENU

Syntax: `SHOW_MENU;`

Description: SHOW_MENU displays the current SQL*Menu menu if it is not currently displayed. It does not make the menu active.

Because SHOW_MENU does not make the menu active, SQL*Forms does not allow the menu to obscure any part of the active page. Therefore, all or part of the menu does not appear on the screen if the active page would cover it.

Related Topics: SHOW_MENU, flow chart 21-121
SQL*Menu, connecting to 25-3

SHOW_PAGE

Syntax: `SHOW_PAGE (page_number) ;`

where *page_number* is the integer number of the page in the current form that you want to display.

Description: SHOW_PAGE displays a specified page on the screen. It does not make the specified page active. If the pages on the screen overlap, the page you most recently displayed with SHOW/PAGE appears under the active page but on top of any other displayed pages.

Note: You should use SHOW_PAGE for pop-up pages only.

Related Topics: SHOW_PAGE, flow chart 21-122
Pages, characteristics 10-4
Pages, pop-up 13-5

SYNCHRONIZE

Syntax: `SYNCHRONIZE;`

Description: SYNCHRONIZE synchronizes terminal screen with the internal state of the form. That is, SYNCHRONIZE updates the screen display to reflect the information that SQL*Forms has in its internal representation of the screen.

Related Topics: SYNCHRONIZE, flow chart 21-123

UP

Syntax: `UP;`

Description: UP navigates to the instance of the current field in the record with the next lowest sequence number.

Related Topics: Up, flow chart 21-124

USER_EXIT

Syntax: `USER_EXIT ( user_exit_string[ , error_string]) ;`

where:

user_exit_string	Specifies the name of the user exit you want to call and any parameters.
error_ string	Specifies an error message that SQL*Forms makes accessible if the user exit fails.

Description: USER_EXIT calls the user exit named in the user_exit_string.

Related Topics: User exits 19-1
USER_EXIT, flow chart 21-125

Unrestricted Packaged Procedures

Any packaged procedure that does not interfere with the basic functions of SQL*Forms is an unrestricted packaged procedure. You can use unrestricted packaged procedures in any type of trigger.

The following list shows all unrestricted packaged procedures

ABORT_QUERY
ANCHOR_VIEW
BELL
BREAK
CALL
CALL_QUERY
COPY
DEFAULT_VALUE
DISPLAY_ERROR
DISPLAY_FIELD
DISPLAY_PAGE
ERASE
EXECUTE_TRIGGER (if the user-named trigger uses only unrestricted packaged procedures)
HIDE_MENU
HIDE_PAGE
HOST
LOCK_RECORD
MESSAGE
MOVE_VIEW
PAUSE
PRINT
REDISPLAY
REPLACE/MENU
RESIZE_VIEW
SET_FIELD
SET_INPUT_FOCUS
SHOW_KEYS
SHOW_MENU
SHOW_PAGE
SYNCHRONIZE
USER_EXIT

Restricted Packaged Procedures

Any packaged procedure that affects the basic functions of SQL*Forms is a restricted packaged procedure. You should use restricted packaged procedures only in key triggers, user-named triggers that are invoked by key triggers, and On-New-Field-Instance triggers.

You should not use restricted packaged procedures in any of the following types of triggers:

- On-Error, On-Database-Record, On-Delete, On-Insert, On-Lock, On-Message, On-New-Record, On-Remove-Record, On-Update, On-Validate-Field, and On-Validate-Record triggers
- Post-Change triggers
- Pre- and Post-Field, Pre- and Post-Record, Pre- and Post-Block, and Pre- and Post-Form triggers
- Pre- and Post-Query triggers
- Pre- and Post-Insert, Pre- and Post-Update, Pre- and Post-Delete, and Pre- and Post-Commit triggers
- User-named triggers that are invoked by any of the above triggers

SQL*Forms (Design) allows you to write any V3 trigger using any packaged procedure-restricted or unrestricted. However, SQL*Forms (Run Form) issues an error if it attempts to execute a trigger that illegally uses a restricted packaged procedure. Follow the recommendations on legal commands for each trigger type.

The following packaged procedures are restricted:

BLOCK_MENU
CALL_INPUT
CLEAR_BLOCK
CLEAR_EOL
CLEAR_FIELD
CLEAR_FORM
CLEAR_RECORD
COMMIT_FORM
COPY_REGION
COUNT_QUERY
CREATE_RECORD
CUT_REGION
DELETE_RECORD
DO_KEY
DOWN
DUPLICATE_FIELD
DUPLICATE_RECORD
EDIT_FIELD
ENTER
ENTER_QUERY
EXECUTE_QUERY
EXECUTE_TRIGGER (if the user-named trigger uses any restricted packaged procedures)
EXIT_FORM
FIRST_RECORD
GO_BLOCK
GO_FIELD
GO_RECORD
HELP
LAST_RECORD
LIST_VALUES
NEW_FORM
NEXT_BLOCK
NEXT_FIELD
NEXT_KEY
NEXT_RECORD
NEXT_SET
PASTE_REGION
POST
PREVIOUS_BLOCK
PREVIOUS_FIELD
PREVIOUS_RECORD
SCROLL_DOWN
SCROLL_UP
UP

Related Topics: Triggers, types 14-8

CHAPTER

17

PACKAGED FUNCTIONS

A packaged function is a built-in PL/SQL function that is available in all forms. Each packaged function evaluates some aspect of the current SQL*Forms (Run Form) session and returns a value. By using packaged functions in triggers, you can determine information about SQL*Forms objects and processing during run time. You can use this information in any type of trigger to determine further processing in an application and to help debug your working applications.

This chapter presents definitions of the SQL*Forms packaged functions.

Packaged Function Definitions

This section presents each packaged function in alphabetical order. The syntax of the packaged function and a description are provided.

Packaged Function Arguments

Some packaged functions require character string arguments. For conciseness, the syntax diagrams for the packaged functions represent these arguments with names that describe what the argument must evaluate to. For example, the FIELD_CHARACTERISTIC packaged function has an argument called field_name. This syntax says that the FIELD_CHARACTERISTIC packaged function requires an argument that evaluates to the name of a field-in this case, the name of the field on which you want information.

There are many different ways to construct a valid character string argument. To use packaged functions correctly, you should be aware of all of the forms that these arguments can take. The following information details the types of expressions that you can use for packaged function arguments that must evaluate to character strings.

literal reference	*'argument_ string'* where *argument_string* is the character string that satisfies the argument.
field value reference	: *field_name* where *field_name* is the name of a field whose value is the character string that satisfies the argument.
SQL*Forms variable reference	: *variable_name* where *variable_name* is the name of a SQL*Forms variable whose value is the character string that satisfies the argument. The named variable can be a system variable or a global variable.
PL/SQL variable reference	*variable_name* where *variable_name* is the name of a PL/SQL local variable whose value is the character string that satisfies the argument. Refer to the *PL/SQL User's Guide and Reference* for information on PL/SQL local variables.

indirect reference

`NAME_IN (argument_reference)`

Where *argument_reference* can be any kind of argument reference (including another indirect reference) that evaluates to the name of a field or variable whose value is the character string that satisfies the argument.

Arguments can also be constructed with combinations of these expressions. For example, you could use a field value reference that refers to a field that contains a SQL*Forms variable reference. You can also use other general PL/SQL expressions such as concatenation.

Note: Any argument called field.name can evaluate to a string that includes a block identifier in the form blockname.fieldname.

Related Topics: Global variables 18-2
NAME_IN packaged function 17-16
System variables 18-3

APPLICATION_ CHARACTERISTIC

Syntax:

```
APPLICATION_ CHARACTERISTIC ( {CALLING_FORM |
CURRENT_FORM}  )
```

Description: The APPLICATION_CHARACTERISTIC packaged function returns information about the current SQL*Forms application.

CALLING_ FORM	Returns the name of the form that called the current form. Returns NULL if there is no calling form.
CURRENT_ FORM	Returns the name of the current form, including file specification.

Return Type: CHAR

Example Assume that you want to write a trigger that calls one of two forms. Which form the trigger calls is determined by the name of the current form's parent. The following trigger text performs this function

```
IF APPLICATION_ CHARACTERISTIC (CALLING_ FORM) = ' FORM1 '
    THEN CALL ( ' FORM4 ' ) ;
ELSIF APPLICATION_ CHARACTERISTIC ( CALLING_ FORM) = ' F0RM2 '
    THEN CALL ( ' FORM5 ' ) ;
ELSE EXIT_ FORM;
END IF;
```

BLOCK_CHARACTERISTIC

Syntax: `BLOCK_CHARACTERISTIC (block_name, {BASE_TABLE | ENTERABLE | FIRST_FIELD | LAST_FIELD | NEXTBLOCK | PREVIOUSBLOCK | RECORDS_DISPLAYED | TOP_RECORD})`

Description: The BLOCK_CHARACTERISTIC packaged function returns information about a specified block. Block characteristics are mapped to the following parameters:

BASE_TABLE	Returns the name of the base table for the indicated block. Returns NULL if the block does not have a base table. Corresponds to the Base Table block characteristic.
ENTERABLE	Returns TRUE if the indicated block is an enterable block. Returns FALSE if the block is not enterable.
FIRST_FIELD	Returns the name of the first field in the indicated block (i.e., the field with the lowest sequence number).
LAST_FIELD	Returns the name of the last field in the indicated block (i.e., the field with the highest sequence number).
NEXTBLOCK	Returns the name of the block that has the next higher sequence number. Returns NULL if the indicated block is the last block in the form.
PREVIOUSBLOCK	Returns the name of the block that has the next lower sequence number. Returns NULL if the indicated block is the first block in the form.
RECORDS_DISPLAYED	Returns the number of records that the indicated block can display. Corresponds to the Records Displayed block characteristic.
TOP_RECORD	Returns the record number of the topmost visible record in the indicated block.

Return Type: CHAR

Example Assume that you want to write a trigger that redefines [Next Field] for the last field in a block. You can perform this function with a field level key trigger on the last field. However, you might add fields to this block frequently enough that you would rather not have to know which field is the last field. Therefore, you can redefine a block level key trigger that would perform the required function without prior knowledge of the specific field name:

```
IF : SYSTEM. CURSOR_ FIELD = ' ORDERS ' ||
    BLOCK_ CHARACTERISTIC ( ' ORDERS. ' , LAST_ FIELD)
    THEN NEXT_ BLOCK ;
ELSE NEXT_ FIELD ;
END IF;
```

Note that this example concatenates the name of the block to the return value of the BLOCK_CHARACTERISTIC statement. This is because SYSTEM.CURSOR_FIELD returns a blockname.fieldname value and BLOCK_CHARACTERISTIC only returns a fieldname value.

Related Topics: Blocks, characteristics 11-6

ERROR_CODE

Syntax: `ERROR_CODE`

Description: The ERROR_CODE packaged function returns an error number for the action most recently performed during the current SQL*Forms (Run Form) session. If the action succeeds, ERROR_CODE returns 0.

Use ERROR_CODE for the following purposes:

- testing the outcome of a user action (e.g., pressing a key) to determine processing within an On-Error trigger
- testing the outcome of a packaged procedure to determine further processing within any trigger

To get the correct results in either type of test, you must perform the test immediately after the action executes. That is, another action should not occur prior to the test.

Refer to the section on "SQL*Forms (Run Form) Messages" for a list of messages and error numbers.

Return Type: NUMBER(5)

Example In the following On-Error trigger, the IF statement uses ERROR_CODE to trap a SQL*Forms (Run Form) error message caused by a user action and replace that error message with a designer-defined message.

```
IF ERROR-CODE = 41000
    THEN MESSAGE ( 'You pressed the wrong key.  Press ESC-K
        if you need help. ') ;
END IF;
```

Note that this trigger identifies the generated error message *(41000) before* SQL*Forms displays that message on the message line, allowing the rest of the trigger to replace the message.

Related Topics: Messages, SQL*Forms (Run Form) E-42
On-Error trigger 14-14
Packaged procedures, failure 15-9

ERROR_TEXT

Syntax: `ERROR_TEXT`

Description: The ERROR_TEXT packaged function returns error text for the action most recently performed during the current SQL*Forms (Run Form) session. If the action succeeds, ERROR_TEXT returns NULL.

Use ERROR_TEXT for the following purposes:

- testing the outcome of a user action (e.g., pressing a key) to determine processing within an On-Error trigger
- testing the outcome of a packaged procedure to determine further processing within any trigger

To get the correct results in either type of test, you must perform the test immediately after the action executes. That is, another action should not occur prior to the test.

Note: If your applications must be supported in more than one language, use the ERROR_CODE packaged function instead of the ERROR_TEXT packaged function. Oracle Corporation provides these separate functions to facilitate national language support.

Return Type: CHAR(78)

Example In the following On-Error trigger, the IF statement uses ERROR_TEXT to trap a SQL*Forms (Run Form) error message caused by a user action and replace that error message with a designer-defined message.

```
IF ERROR_TEXT = 'You pressed an undefined function key. '
    THEN MESSAGE ( 'You pressed the wrong key.  Press ESC-K
        if you need help.');
END IF;
```

Note that this trigger identifies the generated error message (`YOU pressed an undefined function key. `) *before* SQL*Forms displays that message on the message line, allowing the rest of the trigger to replace the message.

Related Topics: Messages, SQL*Forms (Run Form) E-42
On-Error trigger 14-14
Packaged procedures, failure 15-9

ERROR_TYPE

Syntax: `ERROR_TYPE`

Description: The ERROR_TYPE packaged function returns an error message type for the action most recently performed during the current SQL*Forms (Run Form) session.

ERROR_TYPE returns one of three values for the error message type:

FRM	Indicates that a SQL*Forms error occurred.
MNU	Indicates that a SQL*Menu error occurred.
ORA	Indicates that an ORACLE error occurred.

If the action does not generate an error message, ERROR_TYPE returns NULL.

Use ERROR_TYPE for the following purposes:

- testing the outcome of a user action (e.g., pressing a key) to determine processing within an On-Error trigger
- testing the outcome of a packaged procedure to determine further processing within any trigger

To get the correct results in either type of test, you must perform the test immediately after the action executes. That is, another action should not occur prior to the test.

Return Type: CHAR(3)

Example In the following anonymous block, the IF statement uses ERROR_TYPE to test the outcome of the CALL packaged procedure.

```
CALL (FORM_2) ;
IF ERROR_TYPE = 'ORA'
    THEN CLEAR_FORM ;
END IF;
```

Related Topics: On-Errror trigger 14-14
Packaged procedures, failure 15-9

FIELD_ CHARACTERISTIC

Syntax:

```
FIELD_ CHARACTERISTIC ( field_name,
{AUTO_HELP |AUTO_SKIP |BASE_ TABLE |DATATYPE |
DISPLAYED |DISPLAY_LENGTH |ECHO |ENTERABLE |
FIELD_LENGTH |FIXED_LENGTH |LIST |NEXTFIELD |
PAGE |PREVIOUSFIELD |PRIMARY_KEY |QUERY_LENGTH |
QUERYABLE |REQUIRED |UPDATEABLE |UPDATE_NULL |
UPPERCASE |X_POS |Y_POS})
```

Description: The FIELD_CHARACTERISTIC packaged function returns information about a specified field. Field characteristics are mapped to the following parameters:

AUTO_HELP	Returns TRUE if the indicated field has the Automatic Hi+nt field characteristic turned on. Returns FALSE if the characteristic is turned off.
AUTO_SKIP	Returns TRUE if the indicated field has the Automatic Skip field characteristic turned on. Returns FALSE if the characteristic is turned off.
BASE_TABLE	Returns TRUE if the indicated field has the Base Table field characteristic turned on. Returns FALSE if the characteristic is turned off.
DATATYPE	Returns the data type for the indicated field. Corresponds to the Data Type field characteristic.
DISPLAYED	Returns TRUE if the indicated field has the Displayed field characteristic turned on. Returns FALSE if the characteristic is turned off.
DISPLAY_ LENGTH	Returns the display length for the indicated field. Corresponds to the Display Length field Characteristic.

ECHO	Returns TRUE if the indicated field has the Echo Input field characteristic turned on. Returns FALSE if the characteristic is turned off.
ENTERABLE	Returns TRUE if the indicated field has the Input Allowed field characteristic turned on. Returns FALSE if the characteristic is turned off.
FIELD_LENGTH	Returns the field length for the indicated field. Corresponds to the Field Length field characteristic.
FIXED_LENGTH	Returns TRUE if the indicated field has the Fixed Length field characteristic turned on. Returns FALSE if the characteristic is turned off.
LIST	Returns TRUE if the indicated field has a list of values specified. Returns FALSE if a list of values is not specified. Corresponds to the List of Values SQL Text field characteristic.
NEXTFIELD	Returns the name of the field that has the next higher sequence number. Returns NULL if the indicated field is the last field in the block.
PAGE	Returns the page number for the indicated field. Corresponds to the Page setting of the Screen Position field characteristic.
PREVIOUSFIELD	Returns the name of the field that has the next lower sequence number. Returns NULL if the indicated field is the first field in the block.
PRIMARY_KEY	Returns TRUE if the indicated field has the Primary Key field characteristic turned on. Returns FALSE if the characteristic is turned off.
QUERY_ LENGTH	Returns the query length for the indicated field. Corresponds to the Query Length field characteristic.
QUERYABLE	Returns TRUE if the indicated field has the Query Allowed field characteristic turned on. Returns FALSE if the characteristic is turned off.

REQUIRED	Returns TRUE if the indicated field has the Required field characteristic turned on. Returns FALSE if the characteristic is turned off.
UPDATEABLE	Returns TRUE if the indicated field has the Update Allowed field characteristic turned on. Returns FALSE if the characteristic is turned off.
UPDATE_NULL	Returns TRUE if the indicated field has the Update if Null field characteristic turned on. Returns FALSE if the characteristic is turned off.
UPPER_CASE	Returns TRUE if the indicated field has the Uppercase field characteristic turned on. Returns FALSE if the characteristic is turned off.
X_POS	Returns the horizontal coordinate of the field's position on the page. Corresponds to the X setting of the Screen Position field characteristic.
Y_POS	Returns the vertical coordinate of the field's position on the page. Corresponds to the Y setting of the Screen Position field characteristic.

Return Type: CHAR

Example Assume that you want to create a form-level procedure that toggles the Automatic Hint field characteristic. The following procedure performs this task:

```
PROCEDURE  TOGGLE_FIELD_HINT  IS
BEGIN
    IF FIELD_ CHARACTERISTIC ( : SYSTEM. CURSOR_ FIELD , 'AUTO_ HELP) = ' TRUE '
        THEN SET_FIELD ( : SYSTEM. CURSOR_ FIELD , AUTO_HELP , ATTR_OFF ) ;
        ELSE SET_FIELD ( : SYSTEM. CURSOR_ FIELD , AUTO_HELP , ATTR_ON) ;
    END IF;
END ;
```

If the first FIELD_CHARACTERISTIC packaged function returns TRUE-indicating that the field has the Automatic Hint characteristic turned on-then the procedure turns the characteristic off. If the function returns FALSE-indicating that the field has the Automatic Hint characteristic turned off-then the procedure turns the characteristic on.

Related Topics: Fields, characteristics 12-4

FORM_ CHARACTERISTIC

Syntax: `FORM_CHARACTERISTIC (form_name {FIRST_BLOCK | LAST-BLOCK})`

Description: The FORM_CHARACTERISTIC packaged function returns information about the current form.

FIRST_BLOCK	Returns the name of the first block in the indicated form (i.e., the block with the lowest sequence number).
LAST_BLOCK	Returns the name of the last block in the indicated form (i.e., the block with the highest sequence number).

Return Type: CHAR

Example Assume that you want to write a trigger that defines the Key-F1 function key to always navigate to the first block (EMP) of a form. After the DBA created the Key-F1 key, you could write the following form level key trigger:

```
GO_ BLOCK (FORM_ CHARACTERISTIC ( ' EMP ' , FIRST_ BLOCK) ) ;
```

Because this trigger uses the FORM_CHARACTERISTIC function to identify the first block of the form, you can copy this trigger to all of your forms and use it without modification.

FORM_FAILURE

Syntax: `FORM_FAILURE`

Description: The FORM_FAILURE packaged function returns a value that indicates the outcome of the action most recently performed during the current SQL*Forms (Run Form) session.

Outcome	*Returned Value*
success	FALSE
failure	TRUE
fatal error	FALSE

If no action has executed in the current SQL*Forms (Run Form) session, FORM_FAILURE returns FALSE.

Use FORM_FAILURE to test the outcome of a packaged procedure to determine further processing within any trigger. To get the correct results, you must perform the test immediately after the action executes. That is, another action should not occur prior to the test.

Return Type: BOOLEAN

Example In the following anonymous block, the IF statement uses FORM_FAILURE to test the outcome of the NEXT_RECORD packaged procedure.

```
NEXT_RECORD;
IF FORM_ FAILURE
    THEN FIRST_RECORD ;
END IF;
```

Related Topics: Packaged procedures, failure 15-9

FORM_FATAL

Syntax: `FORM_FATAL`

Description: The FORM_FATAL packaged function returns the outcome of the action most recently performed during the current SQL*Forms (Run Form) session.

Outcome	*Returned Value*
success	FALSE
failure	FALSE
fatal error	TRUE

Use FORM_FATAL to test the outcome of a packaged procedure to determine further processing within any trigger. To get the correct results, you must perform the test immediately after the action executes. That is, another action should not occur prior to the test.

Return Type: BOOLEAN

Example In the following anonymous block, the IF statement uses FORM_FATAL to test the outcome of the USER_EXIT packaged procedure.

```
USER_EXIT ( 'CALC_TOTAL' ) ;
IF FORM_ FATAL
    THEN MESSAGE ( 'Fatal error occurred.  Contact the DBA. ' ) ;
END IF;
```

Related Topics: Packaged procedures, failure 15-9

FORM_SUCCESS

Syntax: `FORM_ SUCCESS`

Description: The FORM_SUCCESS packaged function returns the outcome of the action most recently performed during the current SQL*Forms (Run Form) session.

Outcome	*Returned Value*
success	TRUE
failure	FALSE
fatal error	FALSE

Use FORM_SUCCESS to test the outcome of a packaged procedure to determine further processing within any trigger. To get the correct results, you must perform the test immediately after the action executes. That is, another action should not occur prior to the test.

Return Type: BOOLEAN

Example In the following anonymous block, the IF statement uses FORM_SUCCESS to test the outcome of the COMMIT_FORM packaged procedure.

```
COMMIT_FORM;
IF FORM_ SUCCESS
    THEN SYNCHRONIZE ;
END IF:
```

Related Topics: Packaged procedures, failure 15-9

MESSAGE_CODE

Syntax: `MESSAGE_CODE`

Description: The MESSAGE_CODE packaged function returns a message number for the message that SQL*Forms most recently generated during the current SQL*Forms (Run Form) session. MESSAGE_CODE returns 0 at the beginning of a session, before SQL*Forms generates any messages.

Use MESSAGE_CODE to test the outcome of a user action (e.g., pressing a key) to determine processing within an On-Message trigger.

Refer to the section on "SQL*Forms (Run Form) Messages" for a list of messages and message numbers.

Return Type: NUMBER(5)

Example In the following On-Message trigger, the IF statement uses MESSAGE_CODE to trap a SQL*Forms (Run Form) message caused by a user action and replace that message with a designer-defined message.

```
IF MESSAGE_CODE = 40350
    THEN MESSAGE('No orders match the search criteria
        that you specified.');
END IF;
```

Note that this trigger identifies the generated message (40350) *before* SQL*Forms displays that message on the message line, allowing the rest of the trigger to replace the message.

Related Topics: Messages, SQL*Forms (Run Form) E-42
On-Message trigger 14-15
Packaged procedures, failure 15-9

MESSAGE_TEXT

Syntax: MESSAGE_TEXT

Description: The MESSAGE_TEXT packaged function returns message text for the message that SQL*Forms most recently generated during the current SQL*Forms (Run Form) session. MESSAGE_TEXT returns NULL at the beginning of a session, before SQL*Forms generates any messages.

Use MESSAGE_TEXT to test the outcome of a user action (e.g., pressing a key) to determine processing within an On-Message trigger.

Note: If your applications must be supported in more than one language, use the MESSAGE_CODE packaged function instead of the MESSAGE_TEXT packaged function. Oracle Corporation provides these separate functions to facilitate national language support.

Return Type: CHAR(78)

Example In the following On-Message trigger, the IF statement uses MESSAGE_TEXT to trap a SQL*Forms (Run Form) message caused by a user action and replace that message with a designer-defined message.

```
IF MESSAGE_TEXT =  'Query caused no records to be retrieved. '
    THEN MESSAGE ( 'No orders match the search criteria
        that you specified. ' ) ;
END IF;
```

Note that this trigger identifies the generated message (`Query caused no records to be retrieved.`) *before* SQL*Forms displays that message on the message line, allowing the rest of the trigger to replace the message.

Related Topics: Messages, SQL*Forms (Run Form) E-42
On-Message trigger 14-15
Packaged procedures, failure 15-9

MESSAGE_TYPE

Syntax: `MESSAGE_TYPE`

Description: The MESSAGE_TYPE packaged function returns a message type for the message that SQL*Forms most recently generated during the current SQL*Forms (Run Form) session.

MESSAGE_TYPE returns one of three values for the message type:

F R M	Indicates that a SQL*Forms message was generated.
MNU	Indicates that a SQL*Menu message was generated.
ORA	Indicates that an ORACLE message was generated.

MESSAGE_TYPE returns NULL at the beginning of a session, before SQL*Forms generates any messages.

Use MESSAGE_TYPE to test the outcome of a user action (e.g., pressing a key) to determine processing within an On-Message trigger.

Return Type: CHAR(3)

Example In the following On-Message trigger, the IF statement uses MESSAGE_TYPE to trap informative messages from SQL*Menu. In effect, this trigger suppresses those messages.

```
IF MESSAGE_TYPE = 'MNU'
    THEN NULL ;
END IF;
```

Note that this trigger identifies any SQL*Menu informative message *before* SQL*Forms displays that message on the message line, allowing the rest of the trigger to suppress those messages.

Related Topics: On-Message trigger 14-15
Packaged procedures, failure 15-9

NAME_IN

Syntax: `NAME_IN ( variable_ name)`

Description: The NAME_IN packaged function returns the contents of the variable to which you apply it.

The returned value is in the form of a character string. However, you can use NAME_IN to return numbers and dates as character strings and then convert those strings to the appropriate data types. You can use the returned value as you would use any value within an executable statement.

If you nest the NAME_IN function, SQL*Forms evaluates the individual NAME_IN functions from the innermost one to the outermost one.

Note: The NAME_IN packaged function provides the functionality that is provided in V2 style triggers by variable references. The PL/SQL language does not support variable references.

Return Type: CHAR

Example In the following executable statement, NAME_IN evaluates the current value in the BLOCK_CHOICE field.

```
GO_ BLOCK (NAME_IN ( ' BLOCK_ CHOICE ' ) ) ;
```

For example, if the current value of the BLOCK_CHOICE field is ORD, the preceding statement instructs SQL*Forms to navigate to the ORD block.

In the following executable statement, the innermost NAME_IN evaluates the current value in the CUSTOMER field. Assuming that the returned value is the name of another field, the outermost NAME_IN then evaluates the current value in that field.

```
IF NAME_IN (NAME_IN ( 'COMMENT' ) ) = 'TKB SPORT SHOP'
    THEN NEXT_FIELD ;
END IF;
```

For example, if the current value of the COMMENT field is CUSTOMER and the current value of the CUSTOMER field is TKB SPORT SHOP, the preceding statement instructs SQL*Forms to navigate to the next field.

Related Topics: Objects, referencing in PL/SQL 15-7
Variable references F-28

CHAPTER

18 VARIABLES

This chapter discusses SQL*Forms variables. Specifically, it covers the following topics:

- PL/SQL local variables
- global variables
- system variables

Related Topics: NAME_IN packaged function 17-16

PL/SQL Local Variables

A local variable is a PL/SQL variable that is active only within the anonymous block or form-level procedure in which you declare it.

Refer to the *PL/SQL User's Guide and Reference* for information on how to declare and reference local variables.

Global Variables

Syntax: `GLOBAL. variablename`

where *variablename* is a name that follows ORACLE naming conventions.

Description: A global variable is a SQL*Forms variable that is active in any trigger within a form and is active throughout a SQL*Forms (Run Form) session. This variable stores a character string value of up to 255 characters in length.

Before a global variable can be active during a SQL*Forms (Run Form) session, a trigger statement must initialize the global variable. SQL*Forms initializes a global variable the first time you assign a value to it. You can delete an active global variable (and release the memory associated with it) with the ERASE packaged procedure.

Because global variables are active throughout a SQL*Forms (Run Form) session, a global variable declared in one form can be used by any forms that are called during the session.

How to Assign: In a V3 trigger, assign a value to a global variable with standard PL/SQL syntax for SQL*Forms objects:

```
: GLOBAL. variablename : = value;
```

Alternatively, use the DEFAULT_VALUE packaged procedure to assign a value to a global variable.

Common Usage: Use global variables to store data values that you do not want to store inside a block or that you want to share between forms during a SQL*Forms (Run Form) session.

Related Topics: DEFAULT_VALUE, packaged procedure 16-12
ERASE, packaged procedure 16-18
Global variables, constraints 22-2
Objects, referencing in PL/SQL 15-7

System Variables

A system variable is a SQL*Forms variable that keeps track of some internal SQL*Forms state. In general, you can reference the value of a system variable in order to control the way an application behaves.

SQL*Forms maintains the values of system variables on a per form basis. That is, the values of all system variables correspond only to the current form.

The following list presents the names of the available system variables:

- SYSTEM.BLOCK_STATUS
- SYSTEM.CURRENT_BLOCK
- SYSTEM.CURRENT_FIELD
- SYSTEM.CURRENT_FORM
- SYSTEM.CURRENT_VALUE
- SYSTEM.CURSOR_BLOCK
- SYSTEM.CURSOR_FIELD
- SYSTEM.CURSOR_RECORD
- SYSTEM.CURSOR_VALUE
- SYSTEM.FORM_STATUS
- SYSTEM.LAST_QUERY
- SYSTEM.LAST_RECORD
- SYSTEM.MESSAGE_LEVEL
- SYSTEM.RECORD_STATUS
- SYSTEM.TRIGGER_BLOCK
- SYSTEM.TRIGGER_FIELD
- SYSTEM.TRIGGER_RECORD

All system variables, except SYSTEM.MESSAGE_LEVEL, are read-only variables. SYSTEM.MESSAGE_LEVEL is the only system variable to which you can explicitly assign a value.

SQL*Forms also supplies three special system variables-$$DATE$$, $$DATETIME$$, and $$TIME$$-that supply date and time information and have special restrictions on their use.

$$DATE$$

Syntax: `$$DATE$$`

Description: The $$DATE$$ system variable records the current date in the following format:

DD-MON-YY HH:MI[:SS]

You can only use $$DATE$$ to designate a default value for a field via the Default Value field characteristic. The field must be of the CHAR, DATE, or DATETIME data type.

Example Assume that you want the value of a DATE field, called ORDERDATE, to default to the current date. When you define the ORDERDATE field, specify $$DATE$$ in the Default Value field.

Related Topics: Default Value field characteristic 12-10

$$DATETIME$$

Syntax: `$$DATETIME$$`

Description: The $$DATETME$$ system variable records the current date in the following format:

DD-MON-YY HH:MI[:SS]

You can only use $$DATETIME$$ to designate a default value for a field via the Default Value field characteristic. The field must be of the CHAR or DATETIME data type.

Example Assume that you want the value of a DATETIME field, called ORDERDATE, to default to the current date and time. When you define the ORDERDATE field, specify $$DATETME$$ in the Default Value field.

Related Topics: Default Value field characteristic 12-10

$$TIME$$

Syntax: `$$TIME$$`

Description: The $$TIME$$ system variable records the current time in the following format:

HH:MI[:SS]

You can only use $$TIME$$ to designate a default value for a field via the Default Value field characteristic. The field must be of the CHAR or TIME data type.

Example Assume that you want the value of a TIME field, called ORDERTIME, to default to the current time. When you define the ORDERTIME field, specify $$TIME$$ in the Default Value field.

Related Topics: **Default** Value field characteristic 12-10

SYSTEM. BLOCK_STATUS

Syntax: `SYSTEM.BLOCK_STATUS`

Description: The SYSTEM.BLOCK_STATUS system variable records the status of the block that the cursor is in. The recorded value can be one of three character strings:

CHANGED	Indicates that the block contains at least one Changed record.
N E W	Indicates that the block contains only New records.
QUERY	Indicates that the block contains only Valid records that have been retrieved from the database.

Example Assume that you want to create a trigger that performs a commit before clearing a block if there are changes to commit within that block. The following Key-CLRBLK trigger performs this function.

```
IF  : SYSTEM. BLOCK_ STATUS = ' CHANGED '
    THEN COMMIT ;
END IF;
CLEAR_ BLOCK;
```

Related Topics: Record validation states 3-10

SYSTEM. CURRENT_BLOCK

Syntax: `SYSTEM.CURRENT_BLOCK`

Description: The value that the SYSTEM.CURRENT_BLOCK system variable records depends on the current navigation unit.

- If the current navigation unit is the block, record, or field (as in the Pre- and Post- Field, Record, and Block triggers), the value of SYSTEM.CURRENT_BLOCK is the name of the block that SQL*Forms is processing or that the cursor is in.
- If the current navigation unit is the form (as in the Pre- and Post-Form triggers), the value of SYSTEM.CURRENT_BLOCK is NULL

The recorded value is always in the form of a character string.

Note: SYSTEM.CURRENT_BLOCK is included in Version 3.0 of SQL*Forms for compatibility with previous versions. Oracle Corporation recommends that you use SYSTEM.CURSOR_BLOCK and SYSTEM.TRIGGER_BLOCK instead.

Related Topics: Navigation unit 3-2

SYSTEM. CURRENT_FIELD

Syntax: `SYSTEM.CURRENT.FIELD`

Description: The value that the SYSTEM.CURRENT_FIELD system variable records depends on the current navigation unit.

- If the current navigation unit is the field (as in the Pre- and Post-Field triggers), the value of SYSTEM.CURRENT_FIELD is the name of the field that SQL*Forms is processing or that the cursor is in. The returned field name does not include a block name prefix.
- If the current navigation unit is the record, block, or form (as in the Pre- and Post-Record, Block, and Form triggers), the value of SYSTEM.CURRENT_FIELD is NULL.

The recorded value is always in the form of a character string.

Note: SYSTEM.CURRENT_FIELD is included in Version 3.0 of SQL*Forms for compatibility with previous versions. Oracle Corporation recommends that you use SYSTEM.CURSOR_FIELD and SYSTEM.TRIGGER_FIELD instead.

Related Topics: Navigation unit 3-2

SYSTEM. CURRENT_FORM

Syntax: `SYSTEM.CURRENT.FORM`

Description: The SYSTEM.CURRENT_FORM system variable records the name of the form that SQL*Forms is executing.

The recorded value is always in the form of a character string.

Example Assume that you want any called form to be able to identify the name of the form that called it. You can invoke the following form-level procedure before SQL*Forms issues a call. This procedure stores the name of the current form in a global variable named CALLING_FORM.

```
PROCEDURE  STORE_FORMNAME IS
BEGIN
    : GLOBAL.CALLING_FORM := : SYSTEM. CURRENT_FORM ;
END ;
```

SYSTEM. CURRENT_VALUE

Syntax: `SYSTEM. CURRENT_VALUE`

Description: The SYSTEM.CURRENT_VALUE system variable records the value of the field that is registered in SYSTEM.CURRENT_FIELD.

The recorded value is always in the form of a character string.

Note: SYSTEM.CURRENT_VALUE is included in Version 3.0 of SQL*Forms for compatibility with previous versions. Oracle Corporation recommends that you use SYSTEM.CURSOR_FIELD and SYSTEM.CURSOR_VALUE instead.

SYSTEM. CURSOR_BLOCK

Syntax: `SYSTEM.CURSOR_BLOCK`

Description: The SYSTEM.CURSOR_BLOCK system variable records the name of the block that the cursor is in.

The recorded value is always in the form of a character string.

Example Assume that you want to create a Key-NXTBLK trigger at the form level that navigates depending on what the current block is. The following trigger performs this function.

```
IF  : SYSTEM. CURSOR_ BLOCK = ' ORDERS '
    THEN GO_ BLOCK ( ' ITEMS ' ) ;
ELSIF  : SYSTEM. CURSOR_ BLOCK = ' ITEMS '
    THEN GO_ BLOCK ( ' CUSTOMERS ' ) ;
ELSIF  : SYSTEM. CURSOR_ BLOCK = ' CUSTOMERS '
    THEN GO_ BLOCK ('ORDERS') ;
END IF;
```

SYSTEM. CURSOR_FIELD

Syntax: `SYSTEM.CURSOR_FIELD`

Description: The SYSTEM.CURSOR_FIELD system variable records the name of the block and field (i.e., block.field) that the cursor is in.

The recorded value is always in the form of a character string.

Example Assume that you want to create a form-level procedure that takes the value of the field that the cursor is in (recorded in the SYSTEM.CURSOR_VALUE system variable), multiplies the value times a constant, and then reads the modified value into the same field. The following form-level procedure uses the COPY packaged procedure to perform this function.

```
PROCEDURE  CALC_VALUE  IS
    NEW_VALUE  NUMBER;
BEGIN
    NEW_VALUE  :=  TO_NUMBER ( : SYSTEM.CURSOR_VALUE ) * .06 ;
    COPY (TO_CHAR (NEW_VALUE) ,  : SYSTEM. CURSOR_ FIELD) ;
END ;
```

SYSTEM. CURSOR_RECORD

Syntax: `SYSTEM.CURSOR_RECORD`

Description: The SYSTEM.CURSOR_RECORD system variable records the sequence number of the record that the cursor is in. This number represents the record's current physical order in the block's list of records.

The recorded value is always in the form of a character string.

Example Assume that you want to redefine [Previous Field] on the first field of the ITEMS block so that it navigates to the last field of the ORDERS block if the current record is the first record. The following Key-PRVFLD trigger on the ITEMS.ORDERID field performs this function.

```
IF : SYSTEM.CURSOR_RECORD = '1'
    THEN GO_ FIELD ( ' ORDERS. TOTAL ' ) ;
    ELSE PREVIOUS_ FIELD ;
END IF;
```

SYSTEM. CURSOR_VALUE

Syntax: `SYSTEM.CURSOR_VALUE`

Description: The SYSTEM.CURSOR_VALUE system variable records the value of the field that the cursor is in.

The recorded value is always in the form of a character string.

Example Assume that you want to create a form-level procedure that takes the value of the field that the cursor is in, multiplies the value times a constant, and then reads the modified value into the same field. The following form-level procedure uses the COPY packaged procedure to perform this function.

```
PROCEDURE CALC_VALUE IS
    NEW_VALUE NUMBER;
BEGIN
    NEW_VALUE  : = TO_NUMBER ( : SYSTEM.CURSOR_VALUE) * . 06;
    COPY (TO_CHAR (NEW_VALUE ) , : SYSTEM . CURSOR_FIELD ) ;
END :
```

SYSTEM FORM_STATUS

Syntax: `SYSTEM. FORM_ STATUS`

Description: The SYSTEM.FORM_STATUS system variable records the status of the form that the cursor is in. The recorded value can be one of three character strings:

CHANGED	Indicates that the form contains at least one Changed record.
N E W	Indicates that the form contains only New records.
QUERY	Indicates that a query is open. The value of SYSTEM.FORM_STATUS becomes CHANGED only after at least one record in the form has been changed and the associated navigation unit has also changed.

Note: If validation fails while the status of the form is either NEW or QUERY, the status becomes CHANGED.

Example Assume that you want to create a trigger that performs a commit before clearing a form if there are changes to commit within that form. The following Key—CLRFRM trigger performs this function.

```
IF  : SYSTEM. FORM_ STATUS = 'CHANGED'
    THEN  COMMIT;
END IF;
CLEAR_FORM;
```

Related Topics: Record validation states 3-10

SYSTEM. LAST_QUERY

Syntax: `SYSTEM.LAST_QUERY`

Description: The SYSTEM.LAST_QUERY system variable records the query SELECT statement that SQL*Forms most recently used to populate a block (during the current SQL*Forms (Run Form) session). The recorded value is always in the form of a character string.

Example Assume that you want to generate a report in SQL*Plus that retrieves information identical to a query you perform in SQL*Forms. You can write a Key-EXEQRY trigger that uses the HOST packaged procedure to pass the value in SYSTEM.LAST_QUERY to SQL*Plus. SQL*Plus can then generate the desired report based on the passed SELECT statement.

Related Topics: Query SELECT statement 3-27

SYSTEM. LAST_RECORD

Syntax: `SYSTEM.LAST_RECORD`

Description: The SYSTEM.LAST_RECORD system variable records whether the current record is the last record in a block's list of records. The recorded value is one of the following two values, in the form of a character string:

TRUE	Indicates that the current record is the last record in the current block's list of records.
FALSE	Indicates that the current record is not the last record in the current block's list of records.

Example Assume that you want to create a form-level procedure that displays a custom message when an operator navigates to the last record in a block's list of records. The following form-level procedure performs the basic function.

```
PROCEDURE LAST_ RECORD_MESSAGE IS
BEGIN
    IF : SYSTEM.LAST_RECORD = ' TRUE '
        THEN MESSAGE ( 'This is the last record fetched. ' ) ;
    END IF;
END ;
```

You can then redefine [Down], [Next Record], and [Scroll Down] to call this form-level procedure in addition to their normal processing.

SYSTEM. MESSAGE_LEVEL

Syntax: SYSTEM.MESSAGE_LEVEL

Description: The SYSTEM.MESSAGE_LEVEL system variable records one of the following message severity levels: 0,5,10,15,20, or 25.

The recorded value is always in the form of a character string.

During a SQL*Forms (Run Form) session, SQL*Forms suppresses all messages with a severity level that is the same or lower (less severe) than the indicated severity level.

Assign a value to the SYSTEM.MESSAGE_LEVEL system variable with standard PL/SQL syntax:

```
:SYSTEM. MESSAGE_ LEVEL : = value;
```

The legal values for SYSTEM.MESSAGE_LEVEL are 0,5,10,15,20, and 25.

Note: SQL*Forms does not suppress prompts or vital error messages no matter what severity level you select.

Default: 0

Example Assume that you want SQL*Forms only to display the most severe messages (level 25). The following Pre-Form trigger suppresses all messages at levels 20 and below.

```
:SYSTEM.MESSAGE_LEVEL : = 20;
```

Related Topics: Message severity levels E-42

SYSTEM. RECORD_STATUS

Syntax: `SYSTEM.RECORD_STATUS`

Description: The SYSTEM.RECORD_STATUS system variable records the status of the record that the cursor is in. The recorded value can be one of four character strings:

CHANGED	Indicates that the record's validation status is Changed.
INSERT	Indicates that the record's validation status is Changed and that the record does not exist in the database.
NEW	Indicates that the record's validation status is New.
QUERY	Indicates that the record's validation status is Valid and that it was retrieved from the database, or was just inserted or updated and subsequently queried..

Example Assume that you want to create a trigger that performs a commit before clearing a Changed record. The following Key-CLRREC trigger performs this function.

```
IF : SYSTEM. RECORD_ STATUS = ' CHANGED'
    OR : SYSTEM. RECORD_STATUS = ' INSERT '
    THEN  COMMIT;
END IF;
CLEAR_RECORD  ;
```

Related Topics: Record validation states 3-10

SYSTEM. TRIGGER_BLOCK

Syntax: `SYSTEM. TRIGGER_ BLOCK`

Description: The value of SYSTEM.TRIGGER_BLOCK is the name of the block where the cursor was when the current trigger initially fired. The value is NULL if the current trigger is a Pre- or Post-Form trigger.

The recorded value is always in the form of a character string.

Example Assume that you want to write a form-level procedure that navigates to the block where the cursor was when the current trigger initially fired. The following statement performs this function.

```
GO_ BLOCK (NAME_IN ('SYSTEM. TRIGGER_ BLOCK ' ) ) ;
```

SYSTEM. TRIGGER_FIELD

Syntax: `SYSTEM.TRIGGER_FIELD`

Description: SYSTEM.TRIGGER_FIELD records the field (block.field) that is being processed. When referend in a key trigger, it records the field location of the cursor when the trigger began. The value is NULL if the current trigger is a Pre- or Post-Record, Block, or Form trigger.

The recorded value is always in the form of a character string.

Example Assume that you want to write a form-level procedure that navigates to the field where the cursor was when the current trigger initially fired. The following statement performs this function.

```
GO_ FIELD (NAME_IN ( ‘ SYSTEM. TRIGGER_ FIELD ‘ ) ) ;
```

SYSTEM. TRIGGER_RECORD

Syntax: `SYSTEM .TRIGGER_RECORD`

Description: The SYSTEM.TRIGGER_RECORD system variable records the sequence number of the record that SQL*Forms is processing. This number represents the record’s current physical order in the block’s list of records.

The recorded value is always in the form of a character string.

Example In the following anonymous block, the IF statement uses SYSTEM.TRIGGER_RECORD to identify the current record before processing continues.

```
IF  : SYSTEM.TRIGGER_RECORD = ‘ 1 ‘
    THEN MESSAGE ( ‘First item  in this order. ‘ ) ;
END IF;
```

CHAPTER

19

USER EXITS

A user exit is a subroutine that you write and link into the SQL*Forms executable files. When you call a user exit from a SQL*Forms trigger, SQL*Forms temporarily passes processing control to the user exit. When a user exit is done executing, it returns an integer value to SQL*Forms, which indicates success, failure, or fatal error.

User exits cart perform the following tasks:

- perform complex data manipulation
- pass data to SQL*Forms from operating system text files
- manipulate LONG RAW data
- support PL/SQL blocks
- control real time devices, such as a printer or a robot

You can use user exits for other tasks, such as mathematical processing. However, Oracle Corporation recommends that you perform such tasks with PL/SQL in triggers.

Note: Not all types of user exits can perform all of the described tasks. You can accomplish most of these tasks only with ORACLE Precompiled user exits.

This chapter covers the following topics:

- types of user exits
- creating a user exit
- pop-up window routines

The section on creating a user exit covers an ORACLE Precompiler user exit-the type of user exit you will use most often. This section does not cover all of the details on how to actually write and implement a user exit. These details vary with the operating system on which you are working and the language in which you choose to write the user exit.

Read this chapter to get an understanding of user exits and use the appropriate ORACLE Installation and User's Guide to guide your actual work. Note that before you attempt to implement a user exit, you should have a thorough knowledge of operating system concepts. Previous exposure to the ORACLE Precompilers is also very useful.

Types of User Exits

You can write the following types of user exits:

- ORACLE Precompiler user exits
- OCI (ORACLE Call Interface) user exits
- non-ORACLE user exits

You can also write a user exit that combines both the ORACLE Precompiled interface and the OCI.

ORACLE Precompiled User Exits

An ORACLE Precompiler user exit incorporates the ORACLE Precompiled interface. This interface allows you to write a subroutine in one of the following host languages and embed SQL commands:

- Ada
- C
- COBOL
- FORTRAN
- Pascal
- PL/I

With embedded SQL commands, an ORACLE Precompiled user exit can access ORACLE databases. Such a user exit can also access SQL*Forms variables and fields. Because of these features, you will write most of your user exits as ORACLE Precompiled user exits.

For more information on the ORACLE Precompiled interface, refer to the *Programmer's Guide to the ORACLE Precompilers.*

Note: Not all operating systems support all of the listed languages. For details, refer to the ORACLE Installation and User's Guide for your system.

OCI (ORACLE Call Interface) User Exits

An OCI user exit incorporates the ORACLE Call Interface. This interface allows you to write a subroutine that contains calls to ORACLE databases. A user exit that incorporates only the OCI (and not the ORACLE Precompiled interface) cannot access SQL*Forms variables and fields.

For more information on the OCI, refer to the *Programmer's Guide to the ORACLE Call Interface.*

Non-ORACLE User Exits

A non-ORACLE user exit does not incorporate either the ORACLE Precompiler interface or the OCI. For example, a non-ORACLE user exit might be written entirely in C.

By definition, a non-ORACLE user exit cannot access ORACLE databases or SQL*Forms variables and fields.

Creating a User Exit

This section explains the basic steps of writing an ORACLE Precompiled user exit and calling that user exit from a SQL*Forms trigger. This section applies specifically to ORACLE Precompiler user exits because it is the type of user exit that you will use most often and it is the type that Oracle Corporation recommends for most tasks.

Writing a User Exit

All ORACLE Precompiled user exits can use host language statements to perform procedural operations. These user exits can also use the following types of statements to perform additional functions.

Statement Types	*Use*
EXEC SQL	Performs SQL commands.
EXEC IAF GET	Passes values from SQL*Forms to a user exit.
EXEC IAF PUT	Passes values from a user exit to SQL*Forms.
EXEC ORACLE	Executes ORACLE Precompiler options.

EXEC SQL Statements

An EXEC SQL statement is a SQL command, prefixed with "EXEC SQL." EXEC SQL statements allow you to peform any SQL command in an ORACLE Precompiled user exit. It is your means of selecting or manipulating data in the database from a user exit.

Syntax: `EXEC SQL` *`sql_statement;`*

where *`sql_statement`* is any valid ORACLE SQL statement, except for the restricted commands noted in this section.

Note that you do not need to perfom an explicit CONNECT in an ORACLE Precompiler user exit because SQL*Forms establishes the connection automatically. However, ORACLE does support concurrent connects. For more information, refer to the *Programmer's Guide to the ORACLE Precompilers.*

Restricted Commands: Oracle Corporation strongly recommends that you adhere to the following restrictions when you are using SQL commands in an ORACLE Precompiler user exit:

- Do not issue a SQL COMMIT or ROLLBACK statement from within a user exit if there are changes in a form that have not been posted to the database when the user exit is called.
- Do not issue any command that would implicitly cause a database commit, such as a DDL command.

EXEC IAF GET

An EXEC IAF GET statement passes a value from SQL*Forms to an ORACLE Precompiler user exit. Specifically, it places the value of a form field or SQL*Forms variable into a host language variable. Once SQL*Forms passes a value to a user exit, the user exit can use the value for calculation, manipulation, or update.

Syntax:

```
EXEC IAF GET form_variable_1 [, form_variable_2, . . .]
INTO : host_variable_l [ , : host_variable_.2, . ..];
```

where:

form_variable_n	Specifies the name of the form field or SQL*Forms variable from which you are reading a value.
host_variable_n	Specifies the name of the host language variable into which you are reading a value.

Note that *form_variable_n* can be a reference to any of the following items:

- an unambiguous field name (i.e., name or block.name)
- a SQL*Forms system variable
- a SQL*Forms global variable
- a host language variable (prefixed with a colon) whose value is any of the above choices

Refer to the *Programmer's Guide to the ORACLE Precompilers* for any restrictions on host language variables.

EXEC IAF PUT

An EXEC IAF PUT statement passes a value from an ORACLE Precompiler user exit to SQL*Forms. Specifically, it places the value of a constant or the value of a host language variable into a form field or SQL*Forms variable.

Any value that an EXEC IAF PUT statement passes to a form field displays after the user exit returns processing control to the form that called the user exit (providing, of course, that the field has the Displayed field attribute turned on).

Syntax:

```
EXEC IAF PUT form_ variable [ , . . . ]
  VALUES ({:host_variable |constant} [, . . .] ) ;
```

where:

form_variable	Specifies the name of the form field or SQL*Forms variable into which you are reading a value.
host_ variable	Specifies the name of the host language variable from which you are reading a value.
constant	Specifies the constant that you are reading. Do not precede a constant with a colon.

Notes:

1. The form_variable can be a reference to any of the following items:
   - an unambiguous field name (i.e., name or block.name)
   - a SQL*Forms system variable
   - a SQL*Forms global variable
   - a host language variable (prefixed with a colon) whose value is any of the above items
2. Refer to the *Programmer's Guide to the ORACLE Precompilers* for any restrictions on host language variables.
3. You must represent host variables and constants in standard SQL format. The following table provides examples:

Value	*Result*
:holder1	Inserts the value of the host variable, holder1.
'Summit Sporting Goods'	Inserts the constant string value, Summit Sporting Goods.
413.60	Inserts the constant numeric value, 413.60.

EXEC ORACLE Statements

An EXEC ORACLE statement is a statement that is not standard SQL and is used to execute ORACLE Precompiler options. For more information, refer to the *Programmer's Guide to the ORACLE Precompilers.*

Passing Arguments to a User Exit

In a call to an ORACLE Precompiler user exit, you can optionally pass arguments to the user exit. You do this when you define the user exit string and the error string in the USER_EXIT packaged procedure.

When you define these two strings, SQL*Forms automatically passes the following arguments to the user exit:

Argument	***Definition***
Command Line	Specifies the user exit string.
Command Line Length	Specifies the length (in characters) of the user exit string.
Error Message	Specifies the error string.
Error Message Length	Specifies the length of the error string.
In-Query	Specifies a Boolean value that reflects whether the user exit was invoked when the form was in Query mode. (Query mode occurs between the time the fetch occurs and the time the Post-Query trigger fires, exclusive of the Post-Query trigger execution time.)

In addition to the standard arguments that SQL*Forms passes automatically, you can pass any number of parameters to a user exit from the user exit string of the USER_EXIT packaged procedure. For example, to pass two parameters, PARAM1 and PARAM2, from the CALCULATE_VALUES user exit, you could specify the following statement:

```
USER_EXIT ( ' CALCULATE_VALUES PARAM1 PARAM2 ' ) ;
```

Use this feature to pass information such as field names.

Note: It is the responsibility of the user exit-not SQL*Forms-to parse the user exit string.

Related Topics: USER_EXIT, packaged procedure 16-34

Passing Return Values from a User Exit

After a user exit (of any type) executes, it passes a return value that is available to SQL*Forms. That return value indicates whether the user exit ended with success, failure, or a fatal error. These results are analogous to the results that a SQL statement passes to SQL*Forms when it executes from within a trigger. The trigger that calls the user exit determines how to handle the return condition.

Related Topics: Processing, SQL statements 4-5

Restrictions on Writing User Exits

The following restrictions apply to all user exits:

- When you specify the name of a user exit (in the user exit string of the USER_EXIT packaged procedure) that name must follow the rules of your operating system and host language. Be aware that these rules might cover case sensitivity.
- You cannot perform host language screen I/O from an ORACLE Precompiler user exit. This restriction exists because the run-time routines that a host language uses to perform screen I/O conflict with the routines that SQL*Forms uses to perform its I/O.

  Note: You can perform host language *file* I/O from a user exit.

Implementing a User Exit

The process of implementing a user exit is specific to each operating system. Refer to the appropriate ORACLE Installation and User's Guide for the steps you need to follow.

Note that before you can implement a user exit (on any operating system), you should have available the GENXTB program and GENXTB form. For details on how to make these facilities available, refer to your ORACLE Installation and User's Guide.

GENXTB Program

The GENXTB program has two functions in the process of creating user exits. Specifically, you use the GENXTB program to perform the following actions:

- create the IAPXTB table (the base table used by the GENXTB form)
- read the information in the IAPXTB table to create the entry point file needed to link your user exits

GENXTB Form

You should use the GENXTB form to insert descriptions of your user exits into the IAPXTB table.

IAPXTB Table

The IAPXTB table lists all of the user exits that you will link with SQL*Forms. You create the IAPXTB table by running the GENXTB program. You should update the table only by making changes through the GENXTB form.

The following table describes each column in the IAPXTB table:

Column	*Contents*
NAME	This column specifies the name of the user exit. (This is not necessarily the name of the file that contains the user exit.) Note that some host languages are case sensitive.
TYPE	This column specifies the language in which the user exit is written.
REMARKS	This column specifies a text string that describes the purpose of the user exit. The string can have a maximum length of 80 characters.
CREATION	This column specifies the date that you created a record in the IAPXTB table. When you commit a record, the GENXTB form automatically enters a value in this column.
MODIFY	This column specifies the date on which you last modified a record in the IAPXTB table. When you commit a record, the GENXTB form automatically enters a value in this column.

You must enter one row in the IAPXTB table for every user exit name. (This condition is true for all ORACLE Precompiled user exits, whether a user exit is in a file that is precompiled and compiled by itself or with several other user exits.) You should maintain all production user exits for a system in one IAPXTB table.

Note: You should keep test versions of your user exits in a separate IAPXTB table.

Refer to the appropriate ORACLE Installation and User's Guide for details on creating the IAPXTB table.

Calling a User Exit from a Trigger

The last step in implementing a user exit is calling the user exit from a SQL*Forms trigger. In V3 triggers, you call a user exit with the USER_EXIT packaged procedure.

Any user exit calls in V2 trigger steps work without modification in Version 3.0 of SQL*Forms. There is no V2 function code that is equivalent to the USER_EXIT packaged procedure.

Related Topics: USER_EXIT, packaged procedure 16-34

Pop-up Window Routines

The pop-up windows that SQL*Forms provides can satisfy the needs of most designers. However, advanced users might want to use pop-up windows for other purposes. Therefore, Oracle Corporation provides the following pop-up routines, or code, that you can use in C language user exits to make pop-up windows:

- UXWACT
- UXWADD
- UXWGXY
- UXWHID
- UXWMOV
- UXWPOP
- UXWPUC
- UXWPUS
- UXWREM
- UXWSHO

This section describes the prototype for each pop-up window routine. The prototypes are located in the iapuxw.h file on the medium that contains your SQL*Forms code.

Note that if you use the pop-up window routines, you have the responsibility for the following items:

- handling any return code that a routine returns if it encounters problems while manipulating a window (The iapuxw.h file specifies the return codes.)
- maintaining pointers to the pop-windows that you create
- providing a damage control function that will redraw the necessary portion of a screen when a routine changes, or . damages, the display

Note: Coordinates for windowing routines must be stated relative to the window's origin. The window's origin is at (0,0).

Related Topics: Pop-up windows 13-1

UXWACT

UXWACT makes a pop-up window the active window. On most platforms, UXWACT will also move the pop-up window to the top of all displayed windows. When you activate a window in a character mode environment, UXWACT sends the cursor to the window.

Prototype:

```
int uxwact (window_pointer)
    uxwind window_pointer;
```

where `window_pointer` specifies a pointer to the window that you want to activate.

UXWADD

UXWADD creates a pop-up window. UXWADD does not display the window.

Note that any pop-up window that you create with UXWADD exists until you explicitly remove it or until SQL*Forms exits the form in which it was created; SQL*Forms does not automatically delete the pop-up window when the user exit that created it completes. If you create a pop-up window in a form and then call another form, the window can only be referenced while the calling form is current.

Prototype:

```
uxwind uxwadd(x, y, h, v, border, damage_pointer,
user_parameter)
    int x;
    int y;
    int h;
    int v;
    bool border;
    void (*damage_pointer) (/* char *, int, int,int, int,char
    char * user_parameter;
```

where:

`x`	Specifies the X coordinate of the upper left corner of the pop-up window.
`y`	Specifies the Y coordinate of the upper left corner of the pop-up window.
`h`	Specifies the fixed height of the pop-up window in character spaces.
`V`	Specifies the fixed width of the pop-up window in character spaces.
`border`	Specifies if the pop-up window should have a border (non-zero) or not (zero).
`damage_pointer`	Specifies a pointer to this window's damage control function.
`user_parameter`	Specifies a parameter that can specify information about what the function should draw. This parameter can be NULL.

UXWGXY

UXWGXY moves the cursor position in the pop-up window. The cursor displays at the specified location if the specified pop-up window is active. If the window is not active, the location specifies where the cursor will display when the window does become active.

Note: UXWGXY is only effective in applications that are run in a character mode environment.

Prototype:

```
int uxwgxy(window_pointer, x, y)
    uxwind window_pointer
    int * x;
    int * y;
```

where:

`window_pointer`	Specifies a pointer to the pop-up window in which you want to move the cursor.
`x`	Specifies the X coordinate (relative to the window) of the cursor's intended location.
`y`	Specifies the Y coordinate of the cursor's intended location.

UXWHID

UXWHID hides a window. You can apply UXWHTD to the active window.

UXWHID might call the damage control functions of other windows.

Prototype:

```
int uxwhid (window_ pointer)
    uxwind window_pointer;
```

where `window_pointer` specifies a pointer to the pop-up window that you want to hide.

UXWMOV

UXWMOV moves a pop-up window to a location. UXWMOV might call the window's damage control function, and it might also call the damage functions of other windows.

Prototype:

```
int uxwmov (window_pointer, x, y)
    uxwind window_pointer;
    int * x;
    int * y;
```

where:

`window_pointer`	Specifies a pointer to the pop-up window that you want to move.
`x`	Specifies the X coordinate for the upper left corner of the window's intended location.
`y`	Specifies the Y coordinate for the upper left corner of the window's intended location.

UXWPOP

UXWPOP moves a pop-up window to the top of all displayed windows. Note that, in a character mode environment, UXWPOP does not make a window the active window. In some bit-mapped environments UXWPOP may make a window the active window.

UXWPOP might call the window's damage function and the damage control functions of other windows.

Prototype:

```
int UXWPOP (window_pointer)
    uxwind  window_pointer;
```

where `window.pointer` specifies a pointer to the pop-up window that you want to move.

UXWPUC

UXWPUC puts a character at a location in the pop-up window.

Prototype:

```
int UXWPUC (window_ pointer, window_ char, x, y )
    uxwind  window_pointer;
    char window_ char;
    int  x;
    int  y;
```

where:

`window_pointer`	Specifies a pointer to the pop-up window in which you want to put a character.
`window_char`	Specifies the character you want to insert.
`x`	Specifies the X coordinate (relative to the window) of the character's intended location.
`y`	Specifies the Y coordinate of the character's intended location.

USWPUS

UXWPUS puts a character string at a location in the pop-up window. You can put a character string in any pop-up window, even if it is not active or if it is hidden.

Note: The character string does not need to be NULL terminated.

Prototype:

```
int uxwpus(window_pointer, char_string, x, y, 1)
    uxwind window_pointer
    char * char_string;
    int x;
    int y;
    int 1;
```

where:

`window_pointer`	Specifies a pointer to the pop-up window in which you want to put a character string.
`char_string`	Specifies the character string you want to insert.
`x`	Specifies the X coordinate (relative to the window) of the character string's intended location.
`y`	Specifies the Y coordinate of the character string's intended location.
`1`	Specifies the length of the character string in character spaces.

UXWREM

UXWREM deletes, or "removes," a pop-up window. You can remove any pop-up window, even if it is active.

UXWREM might call the damage control functions of other windows.

Prototype:

```
int uxwrem(window_pointer)
    uxwind window_pointer;
```

where `window_pointer` specifies a pointer to the pop-up window that you want to remove.

UXWSHO

UXWSHO displays a window. If succeeded by UXWPOP, UXWSHO puts the indicated window over any other pages or windows on the screen, but it does not make the window the active window.

UXWSHO might call the window's damage control function.

Prototype:

```
int uxwsho (window_pointer)
    uxwind window_pointer;
```

where `window_pointer` specifies a pointer to the pop-up window that you want to display.

Damage Control Function

If you create and manipulate pop-up windows through user exit routines, you must create a damage control function that will redraw the necessary portion of a screen when a routine changes, or damages, the display.

You assign a damage control function to a pop-up window when you create the window with the UXWADD routine. Each pop-up window can have its own damage control function, or any number of pop-up windows can share the same damage control function.

When SQL*Forms judges that damage might have occurred to a user exit pop-up window, it calls the damage control function for the damaged window and supplies the information about the damaged region that the function requires.

Note: The damage control function only redraws the area of the window that needs repair.

Prototype:

```
char * damage_ function (window_pointer, x, y, h, v,
user_parameter)
    uxwind  window_pointer
    int x
    int y
    int h
    int v
    char * user_parameter;
```

where:

`window_pointer` Specifies a pointer to the pop-up window that you want to repair.

`x` Specifies the X coordinate of the upper left corner of the damaged region.

`y` Specifies the Y coordinate of the upper left corner of the damaged region.

`h` Specifies the fixed height of the damaged region in character spaces.

`v` Specifies the fixed width of the damaged region in character spaces.

`user_parameter` Specifies a parameter that can specify information about what the function should draw. This parameter can be NULL.

Pop-Up Window Example Program

The following C language program creates two pop-up windows (window_l and window_2) and manipulates them through pop-up window routines. Note that both windows use the same damage control function, but they send different parameters to the function.

Example

```
#include <iapuxw.h>

#define TRUE 1
#define FALSE 0

/*The following function gets a character from the keyboard.”/
/* It might vary depending on your operating system.         */
void a_pause()
{
    getchar ( ) ;
}
/* The following function is the damage control function. */
/* Note that the message that is displayed in the window */
/* depends on the parameter that you pass to the function. */
char * on_damag(a_window, x, y, h, v, a_param)
    uxwind a_window;
    int x, y, h, v;
    char * a_param
{
    char tempstr1[8];
    char tempstr2[8];

    sprintf(tempstr1,  “This is”);
    sprintf(tempstr2,  "window %d",  * a_param) ;

    uxwpus(a_window, tempstr1, 7, 1, 2);
    uxwpus(a_window, tempstr2, 8, 1, 4);
}
```

```
/* The following code is a simple example of creating       */
/* and manipulating pop-up windows.                         */
int win_exam()
{
    uxwind window_1, window_2;
    char a;
    char b;

    /* Create window_1.                                     */
    a = '1';
    window_1 = uxwadd(5, 5, 10, 10, TRUE, on_damag, &a);

    /* Create window_2.  */
    b= '2';
    window_2 = uxwadd(8, 8, 10, 10, TRUE, on_damag, &b);

    /* Show window_1.                                       */
    uxwsho(window_1) ;

    a_pause() ;

    /* Show window_2.                                       */
    uxwsho(window_2) ;

    a_pause() ;

    /* Move window_1 to the top of all windows.             */
    uxwpop(window_1) ;

    a_pause() ;

    /* Make window_1 the active window.                     */
    uxwact(window_1) ;

    a_pause() ;

    /* Hide window_1.                                       */
    uxwhid(window_1) ;
    /* Note that this routine calls the damage control      */
    /* function for window_2.                               */

    a_pause() ;

    /* Hide window_2.                                       */
    uxwhide(window_2) ;

    a_pause() ;

    return 0;
}
```

CHAPTER

20

WORKING WITH OBJECTS

This chapter provides information on how to work with SQL*Forms objects. Specifically, it covers the following topics:

- object naming conventions
- object referencing

Object Naming Conventions

The rules for naming SQL*Forms objects are the same as those for naming ORACLE objects. Specifically, SQL*Forms object names:

- can be up to 30 characters long
- must begin with a letter
- can contain letters, numbers, and the special characters $, #, @, and _
- cannot duplicate the complete name of another object of the same type (Note that SQL*Forms does not distinguish between uppercase and lowercase letters in object names.)
- cannot duplicate an ORACLE or SQL*Forms reserved word

Note: SQL*Forms does not support the ORACLE naming convention of double-quoted names.

Form Names

Forms are stored in two operating system files. You will usually store the files as the name of the form plus an extension (INP or FRM). Therefore, in addition to following the standard object naming conventions, the names of forms should follow the file naming rules of your operating system.

Related Topics: Title, form characteristic 9-5

Reserved Words

Do not use any of the following ORACLE, PL/SQL, or SQL*Forms reserved words to name an object:

ABORT	BEGIN	COMPRESS	DECLARE
ACCEPT	BETWEEN	CONNECT	DEFAULT
ACCESS	BLOCK	CONSTANT	DEFINITION
ADD	BODY	CONTAIN	DELAY
ALL	BOOLEAN	CONTAINS	DELETE
ALTER	BY	COUNT	DELTA
AND	CASE	CRASH	DESC
ANY	CHAR	CREATE	DIGITS
APPEND	CHAR_BASE	CURRENT	DISPOSE
ARRAY	CHECK	CURSOR	DISTINCT
AS	CLOSE	DATABASE	DO
ASC	CLUSTER	DATA_BASE	DOES
ASSERT	CLUSTERS	DATAPAGES	DROP
ASSIGN	COLAUTH	DATE	EACH
AT	COLUMN	DBA	ELSE
AUDIT	COLUMNS	DEBUGOFF	ELSIF
AUTHORIZATION	COMMENT	DEBUGON	END
AVG	COMMIT	DECIMAL	ENTRY

Table continued on next page.

ERASE
EVALUATE
EXCEPTION
EXCEPTION_INIT
EXCLUSIVE
EXISTS
EXIT
FALSE
FETCH
FIELD
FILE
FLOAT
FOR
FORM
FORMAT
FROM
FUNCTION
GENERIC
GLOBAL
GOTO
GRANT
GRAPHIC
GROUP
HAVING
IDENTIFIED
IF
IMAGE
IMMEDIATE
IN
INCREMENT
INDEX
INDEXED
INDEXES
INDEXPAGES
INDICATOR
INTITIAL
INSERT
INTEGER
lNTERSECT
INTO
IS
LEVEL
LIKE
LIMITED
LIST
LOCK
LONG
LOOP
M A X
MAXEXTENTS
MIN
MINUS
MOD
MODE
MODIFY
MOVE
NEW
NOAUDIT
NOCOMPRESS
NOLIST
NOSYSSORT
NOT
NOWAIT
NULL
NUMBER
NUMBER_BASE
OF
OFFLINE
OLD
ON
ONLINE
OPEN
OPTIMIZE
OPTION
OR
ORDER
OTHERS
OUT
PACKAGE
PARTITION
PCTFREE
PRAGMA
PRIOR
PRIVATE
PRIVLEGES
PROCEDURE
PUBLIC
RAISE
RANGE
RAW
RECORD
RELEASE
REM
RENAME
RENAMES
RESOURCE
RETURN
REVERSE
REVOKE
ROLLBACK
ROW
ROWID
ROWNUM
ROWS
ROWTYPE
RUN
SAVEPOINT
SCHEMA
SELECT
SEPARATE
SESSION
SET
SHARE
SIZE
SMALLINT
SPACE
SQL
SQLCODE
SQLERRM
START
STATEMENT
STDDEV
SUBTYPE
SUCCESSFUL
SUM
SYNONYM
SYSDATE
SYSSORT
SYSTEM
TABAUTH
TABLE
TABLES
TASK
TEMPORARY
TERMINATE
THEN
TO
TRIGGER
TRUE
TYPE
UID
UNION
UNIQUE
UPDATE
USE
USER
USING
VALIDATE
VALUES
VARCHAR
VARGRAPHIC
VARIANCE
VIEW
VIEWS
WHEN
WHENEVER
WHERE
WHILE
WITH
WORK
XOR

Object Referencing

Referencing an object allows you to create and maintain an object in a form and to reference that object in other forms. When you modify the "master" object, you effectively modify all references to that object in other form definitions. For example, you might have an On-Validate-Field trigger that you want to use in a number of forms, but only maintain in one.

You can reference blocks, fields, triggers, and form-level procedures that are owned by other forms in the current database. You cannot reference constant text, any object that is owned by the current form, or any object that is owned by a form that does not exist in the current database.

Note: This section does not cover the specific steps for referencing objects.

Related Topics: Referencing objects 2-3

Modifying Referenced Objects

When you reference an object, SQL*Forms limits the modifications that you can make to the referenced object and to any objects owned by the referenced object.

Referenced Blocks

When you reference a block, you also reference the following objects:

- all fields owned by the referenced block
- all triggers owned by the block and all triggers owned by the block's fields
- all objects referenced by the block and all objects referenced by the block's fields

Once a referenced block exists in a form definition, you can only modify the following block characteristics:

- Block Name
- Sequence Number

  Note: When you reference a block, SQL*Forms automatically changes the sequence number of the referenced block to the next available block sequence number in the current form definition.

- Comment

Use caution when you modify the name of a referenced block. You can interfere with the functionality of the application if you change the name of a block that is referred to in a trigger or a form-level procedure.

You cannot modify any characteristics for the objects that are owned or referenced by the referenced block.

Note that when you reference a block, you do not reference any constant text that is visually associated with the block. (Constant text belongs to the page that it appears on.) You can copy constant text, but you cannot reference it.

Referenced Fields

When you reference a field, you also reference the following objects:

- all triggers owned by the field
- all objects referenced by the field

Once a referenced field exists in a form definition, you can only modify the following field characteristics:

- Field Name
- Screen Position (X, Y, and Page settings)
- Sequence Number

  Note: When you reference a field, SQL*Forms automatically changes the sequence number of the referenced field to the next available field sequence number in the current form definition.

- Comment

Use caution when you modify the name of a referenced field. You can interfere with the functionality of the application if you change the name of a field that is referred to in a trigger or a form-level procedure. Note that you can change the screen position of a referenced field when you specify the reference in the Target Object dialog box.

You cannot modify any characteristics for the objects that are owned or referenced by the referenced field.

Note that when you reference a field, you do not reference any constant text that is visually associated with the field. (Constant text belongs to the page that it appears on.) You can copy constant text, but you cannot reference it.

Referenced Triggers	Once a referenced trigger exists in a form definition, you can only modify the following trigger characteristics: • Trigger Name • Block Name • Field Name • Comment Use caution when you modify the name of a referenced trigger. You can interfere with the functionality of the application if you change the trigger type or the name of a user-named trigger that is referred to in a trigger or a form-level procedure.
Referenced Form-level Procedures	Once a referenced form-level procedure exists in a form definition, you cannot modify any of its characteristics.

PART

IV PROCESSING

CHAPTER

21 EVENT PROCESSING

This chapter discusses SQL*Forms (Run Form) event processing. Specifically, it covers the following topics:

- event flow charts
- events and sub-events

The majority of this section consists of the event flow charts. These flow charts diagram the major internal and external processes that occur when an operator or a trigger invokes a SQL*Forms (Run Form) function.

This chapter does not cover the fundamentals of internal processing for SQL*Forms (Run Form). For example, while this section does present the flow charts for navigational events, it does not discuss the general concepts of navigation. You should be familiar with these fundamental concepts before attempting to use this chapter.

Related Topics: Fundamental processing 3-1

Event Flow Charts

The event flow charts illustrate what happens during each SQL*Forms event. Use the flow charts to determine trigger placement when designing complex applications.

In this chapter, events that are based on SQL*Forms functions are titled by the name of the function. For example, what happens when the NEXT_FIELD function is invoked-either by executing the NEXT_FIELD packaged procedure or pressing [Next Field]-is represented by the NEXT_FIELD flow chart.

The event flow charts appear in this chapter in alphabetical order.

Flow Chart Sub-events

Events that occur within other events are called sub-events. Sub-events appear in boxes in the flow charts so you can easily identify them. For example, the NEXT_FIELD flow chart contains the following element:

This element tells you to branch to the Validate the Field flow chart for the details of this part of NEXT_FIELD processing. Validate the Field has its own flow chart because it is a common sub-event of other events.

Related Topics: Events, sub-events 21-131

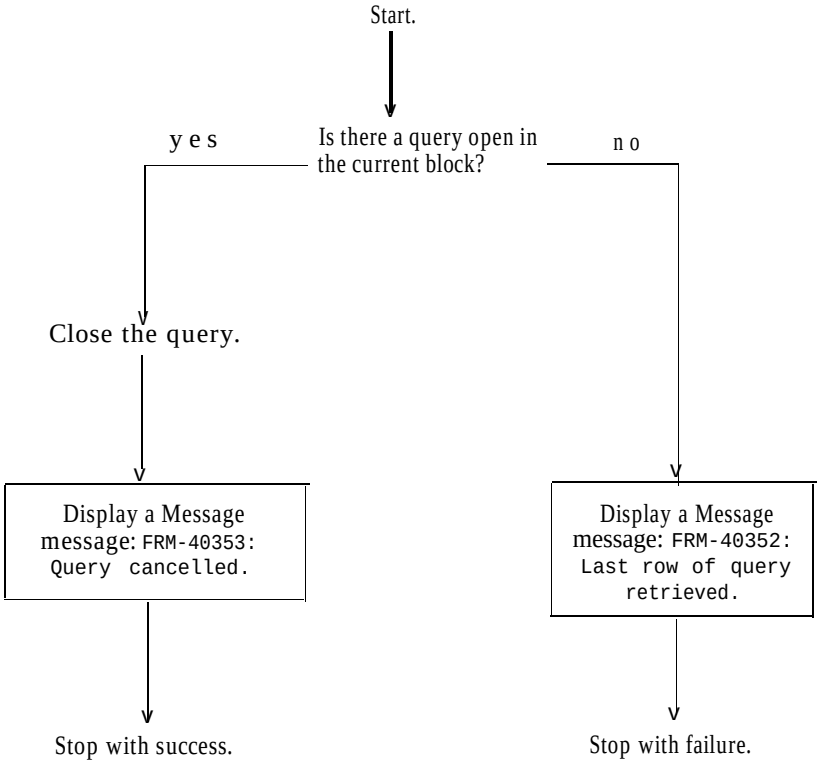
Flow Chart Assumptions

When you use the event flow charts, remember the following assumptions:

- Every event flow chart begins with a "Start" item and ends with one or more "Stop" items to clearly mark the endpoints of the event. Every "Stop" endpoint also indicates if the event's outcome resulted in success, failure, or a fatal error. The flow charts assume that when a sub-event stops, SQL*Forms passes processing control back to the event that contains the sub-event.
- When a flow chart indicates that a trigger fires (e.g., Fire the Post-Field trigger), the flow chart assumes that such a trigger exists and that the trigger at the applicable definition level fires. If a trigger does not exist, assume the success branch of processing.

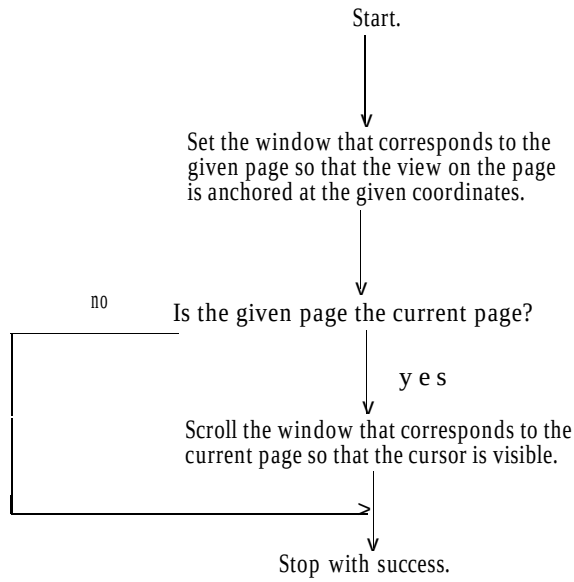
Related Topics: Trigger scope 4-3

ABORT_QUERY



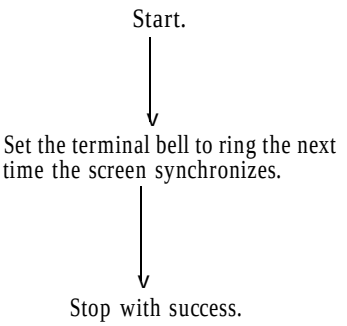
Related Topics: ABORT_QUERY, packaged procedure 16-5

ANCHOR_VIEW



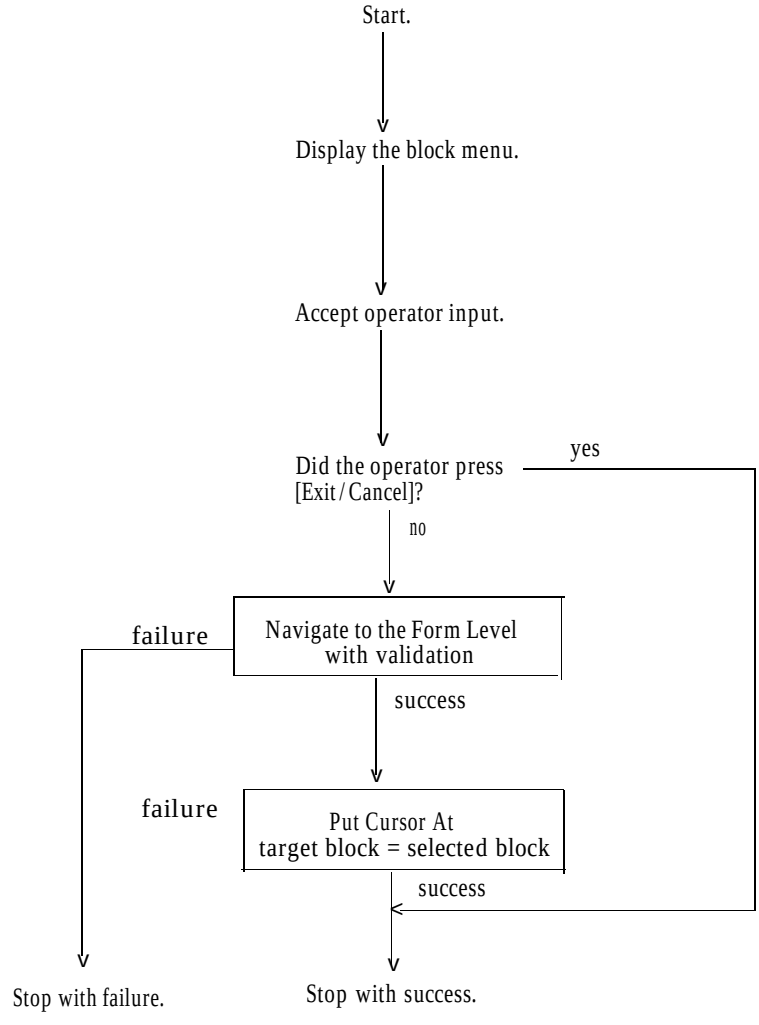
Related Topics: ANCHOR_VIEW, packaged procedure 16-5

BELL



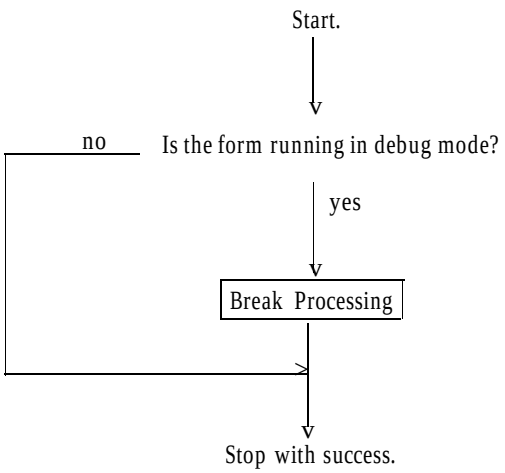
Related Topics: BELL, packaged procedure 16-6

BLOCK_MENU



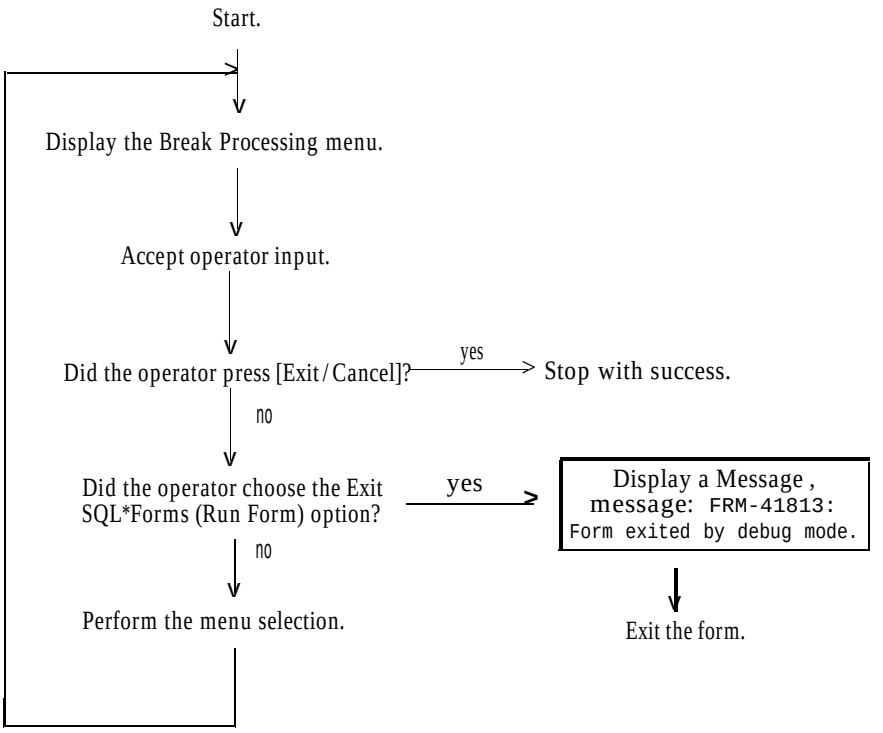
Related Topics: BLOCK_MENU, packaged procedure 16-6

BREAK



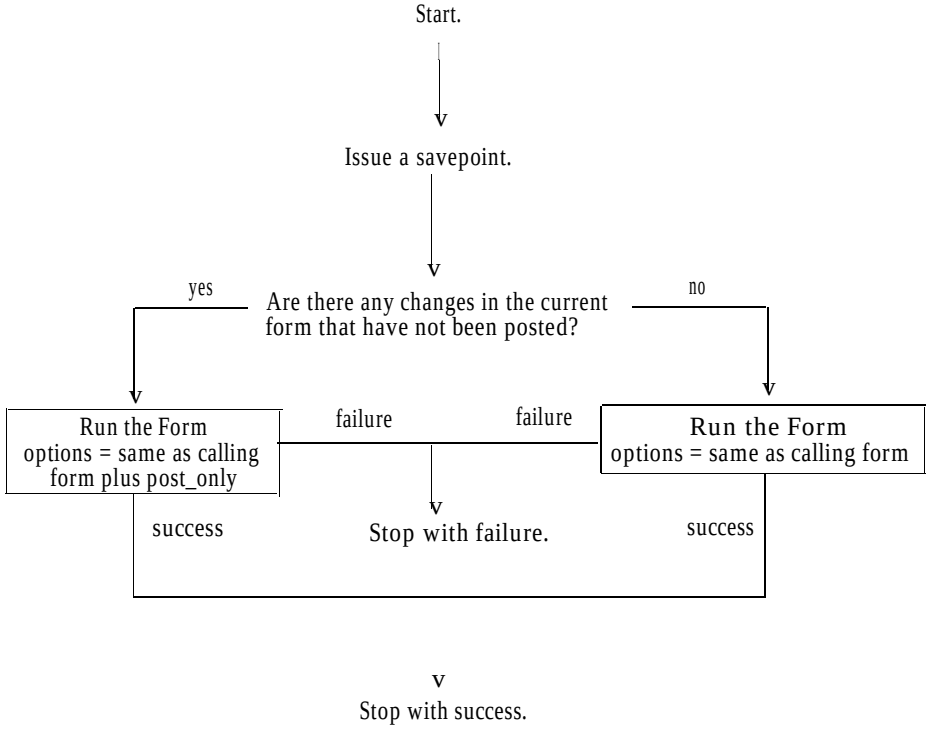
Related Topics: BREAK, packaged procedure 16-6

Break Processing



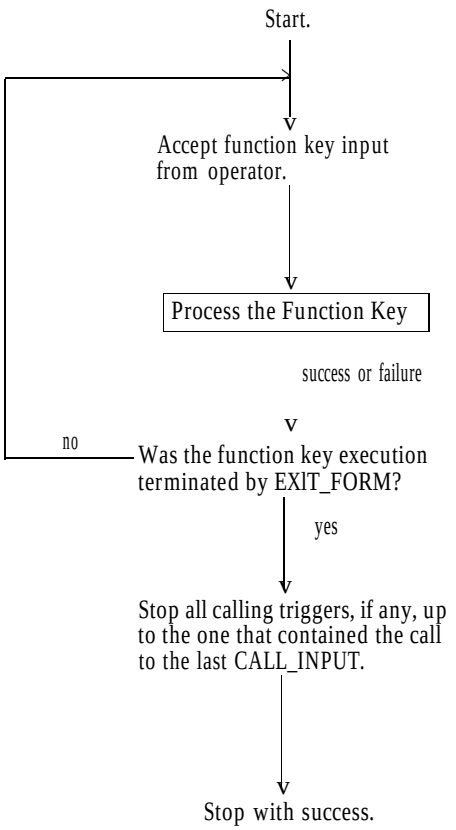
Related Topics: Break processing 28-2

CALL



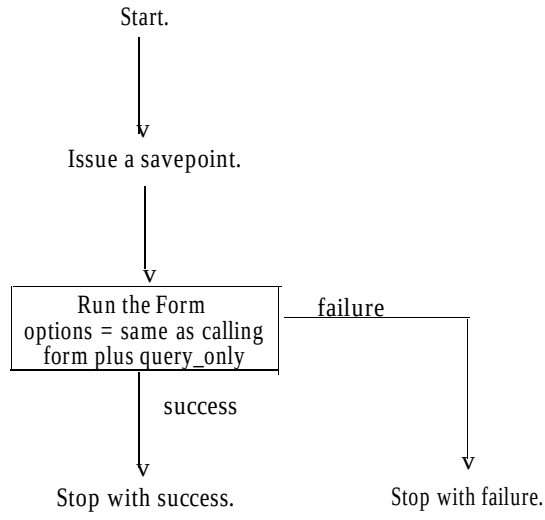
Related Topics: CALL, packaged procedure 16-7

CALL_INPUT



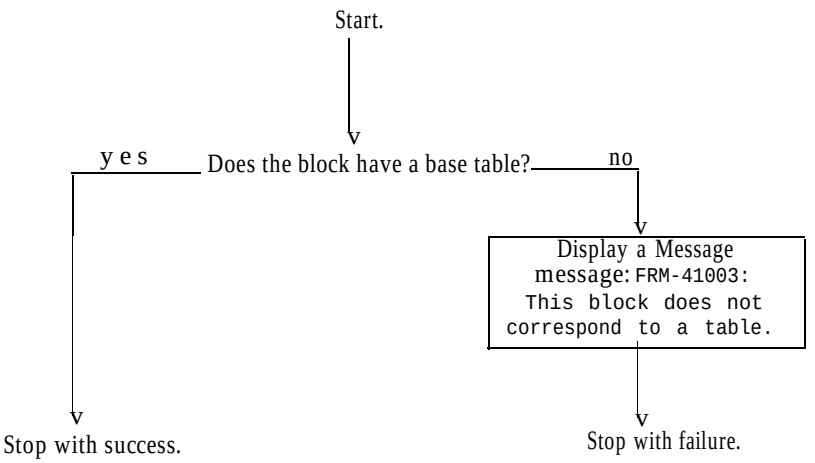
Related Topics: CALL_INPUT, packaged procedure 16-7

CALL_QUERY

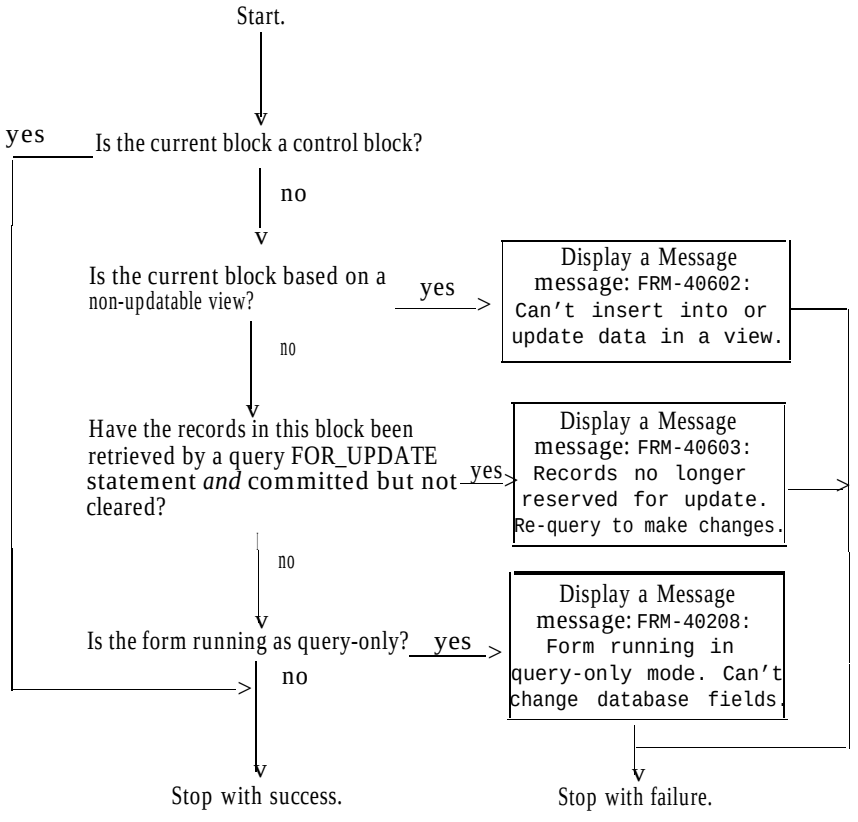


Related Topics: CALL_QUERY, packaged procedure 16-8

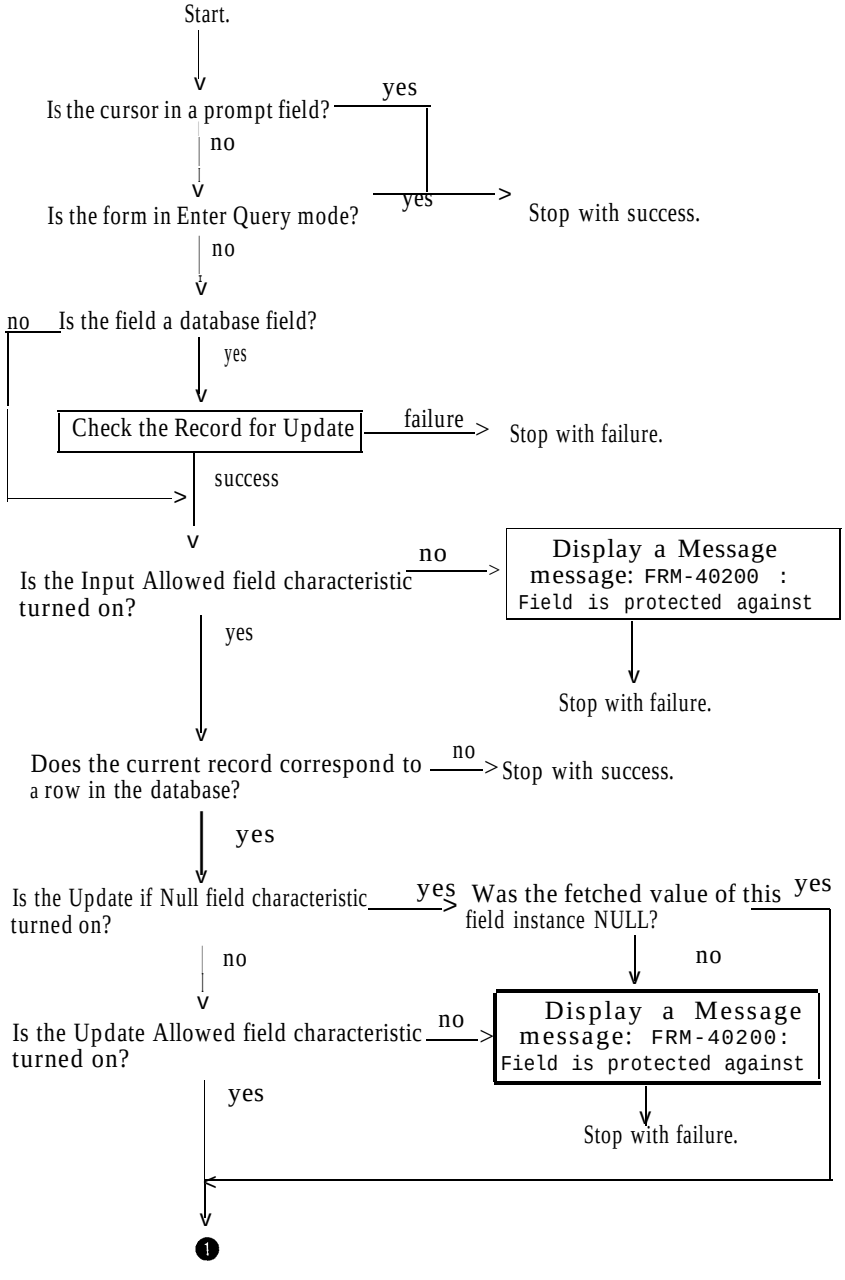
Check the Block for Query

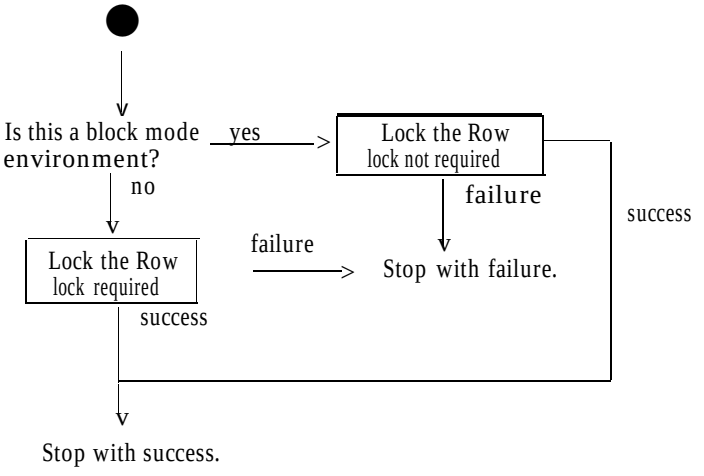


Check the Block for Update



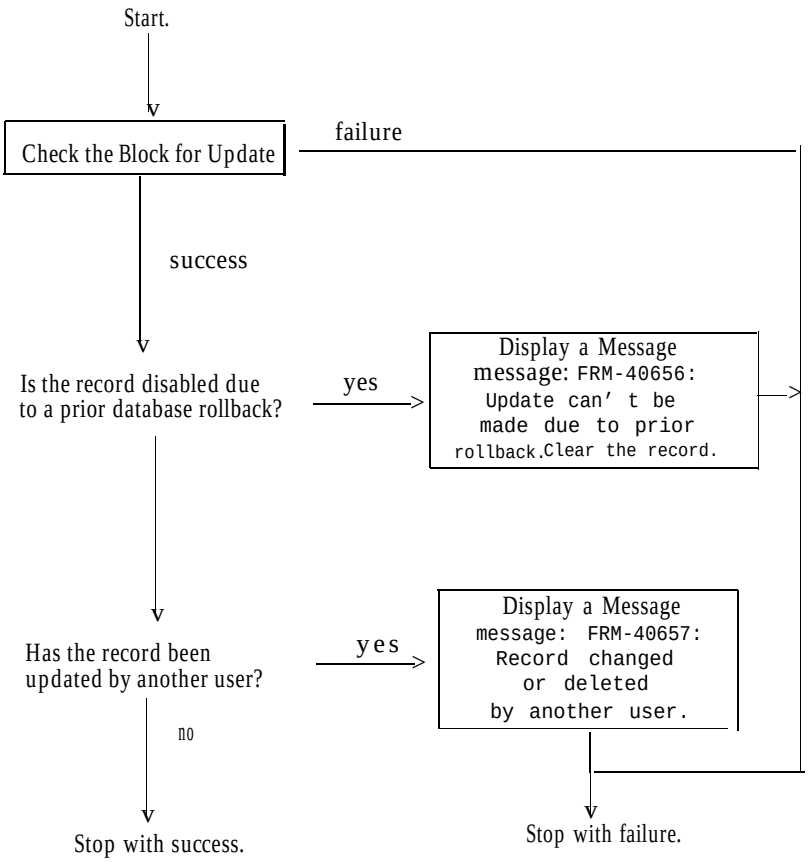
Check the Field for Edit





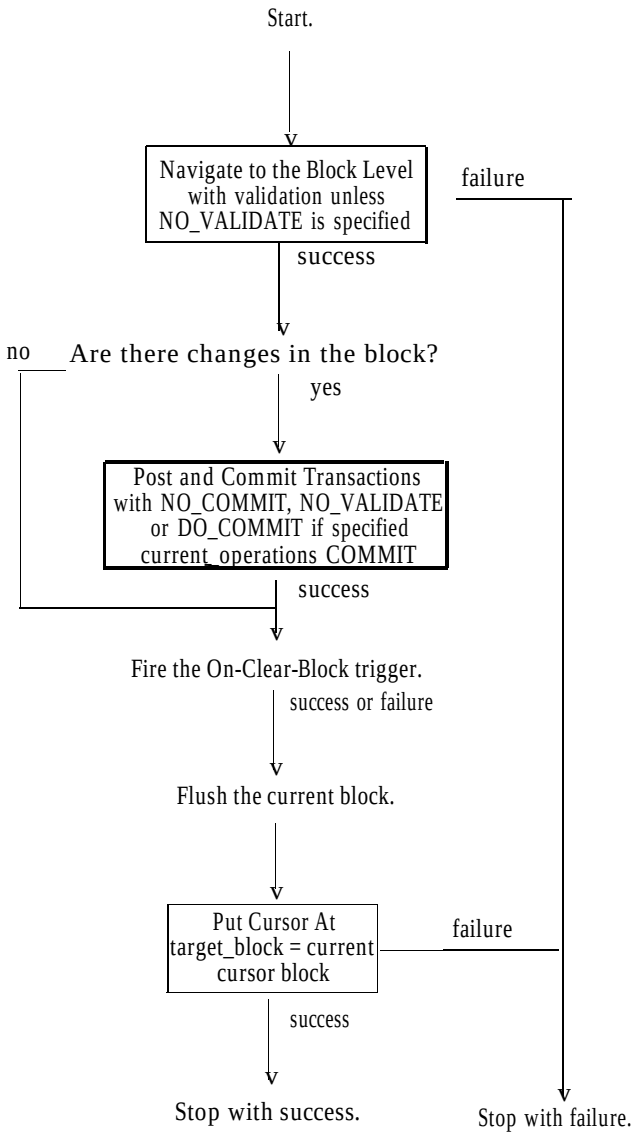
Related Topics: Locking 3-11

Check the Record for Update



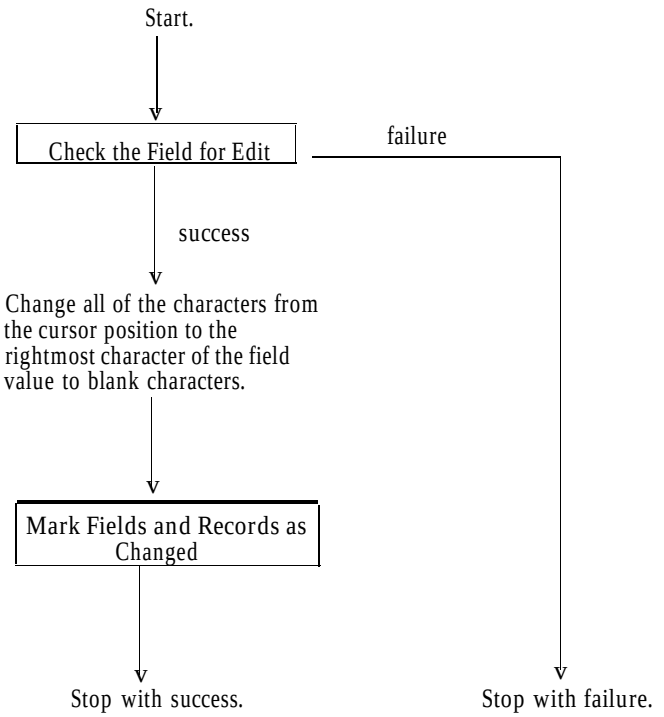
Related Topics: Locking 3-11

CLEAR_BLOCK



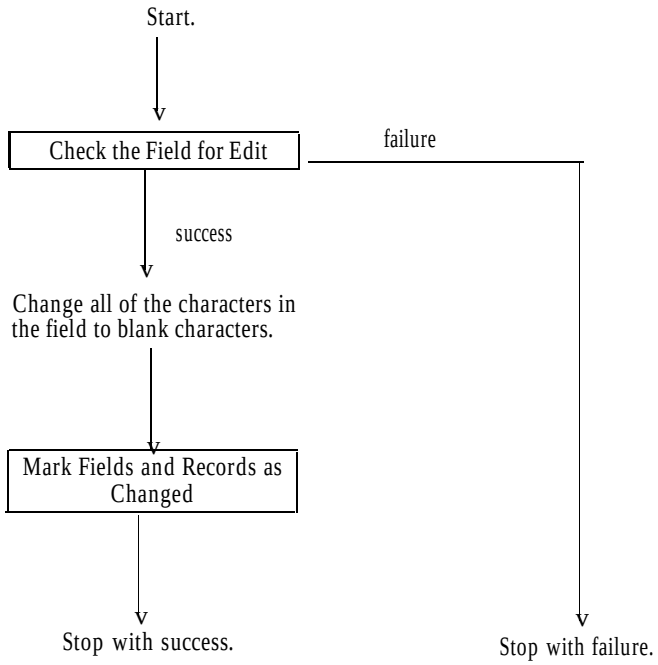
Related Topics: CLEAR_BLOCK, packaged procedure 16-8

CLEAR_EOL



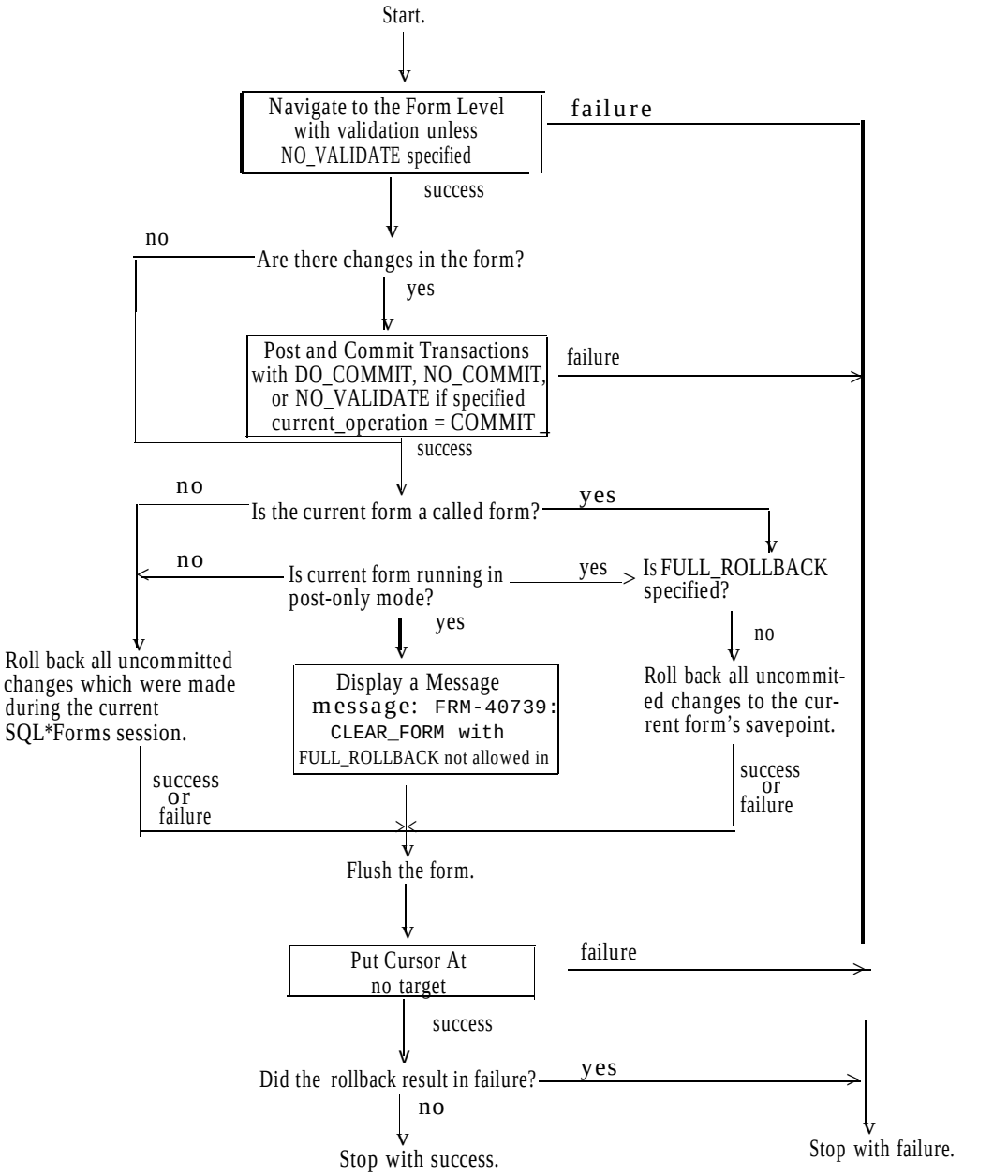
Related Topics: CLEAR_EOL, packaged procedure 16-9

CLEAR_FIELD



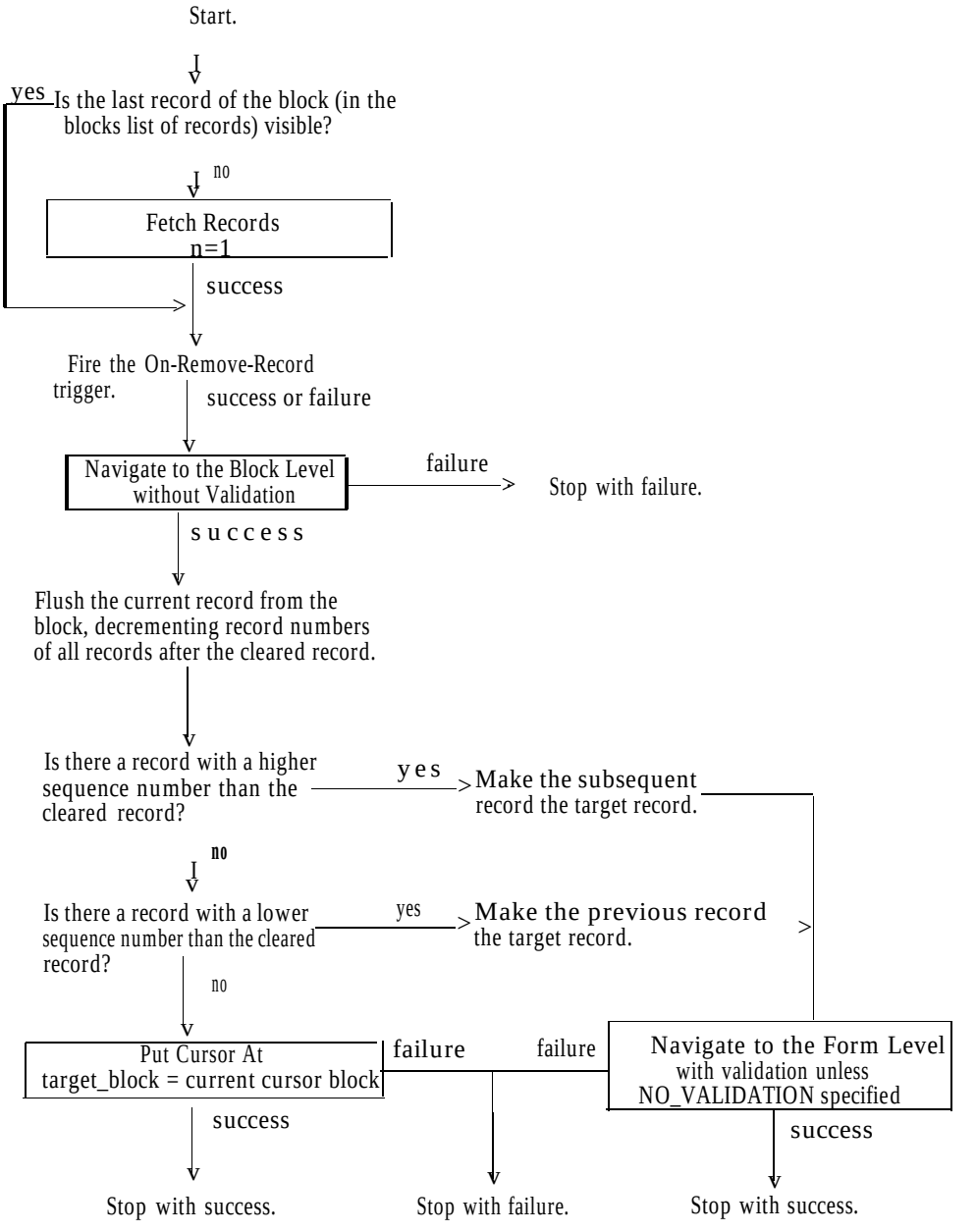
Related Topics: CLEAR_FIELD, packaged procedure 16-9

CLEAR_FORM



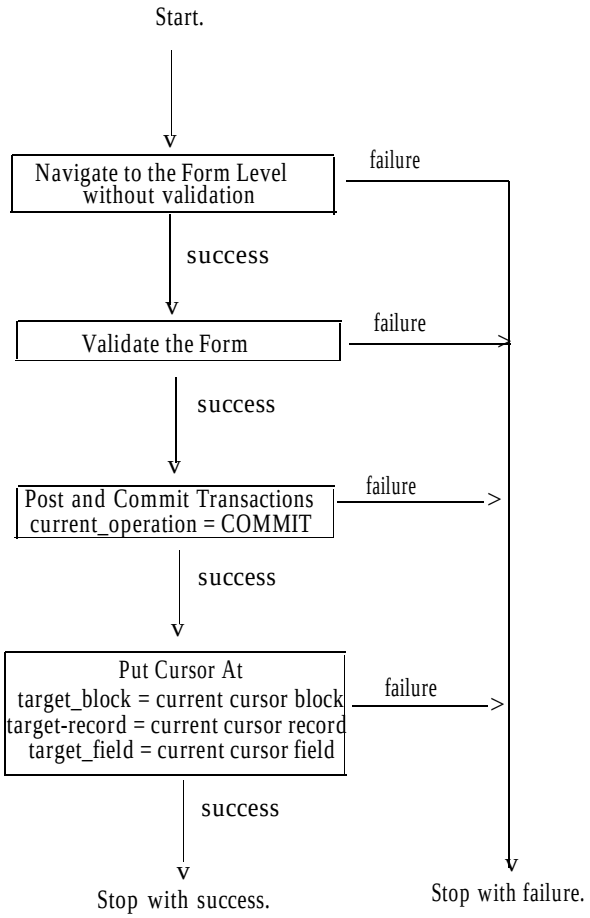
Related Topics: CLEAR_FORM, packaged procedure 16-9

CLEAR_RECORD



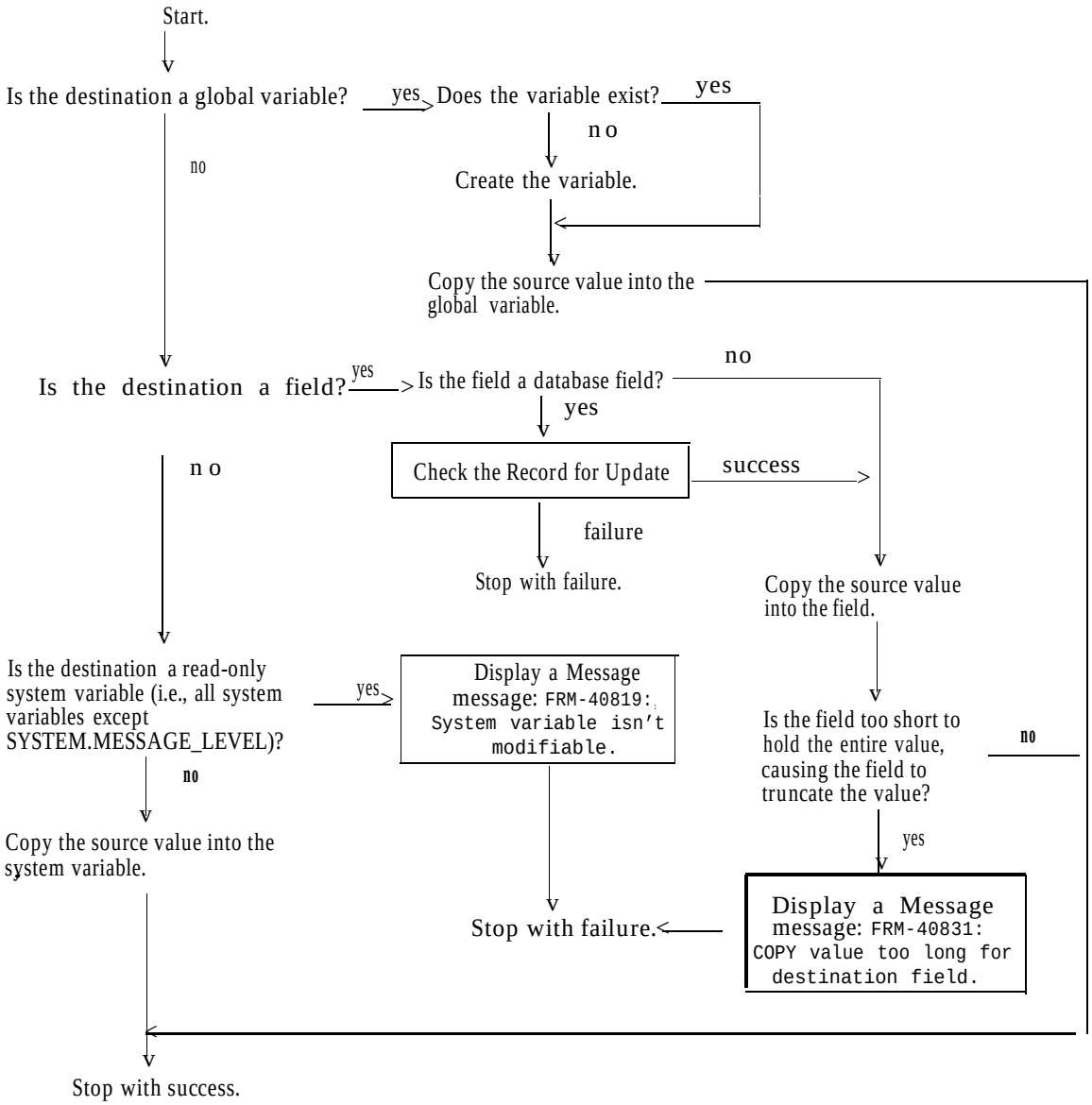
Related Topics: CLEAR_RECORD, packaged procedure 16-10

COMMIT_FORM



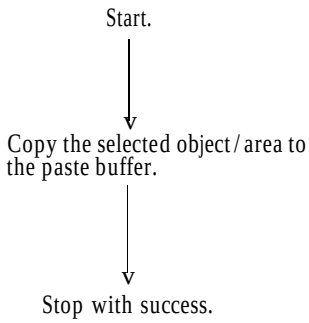
Related Topics: COMMIT_FORM, packaged procedure 16-11

COPY



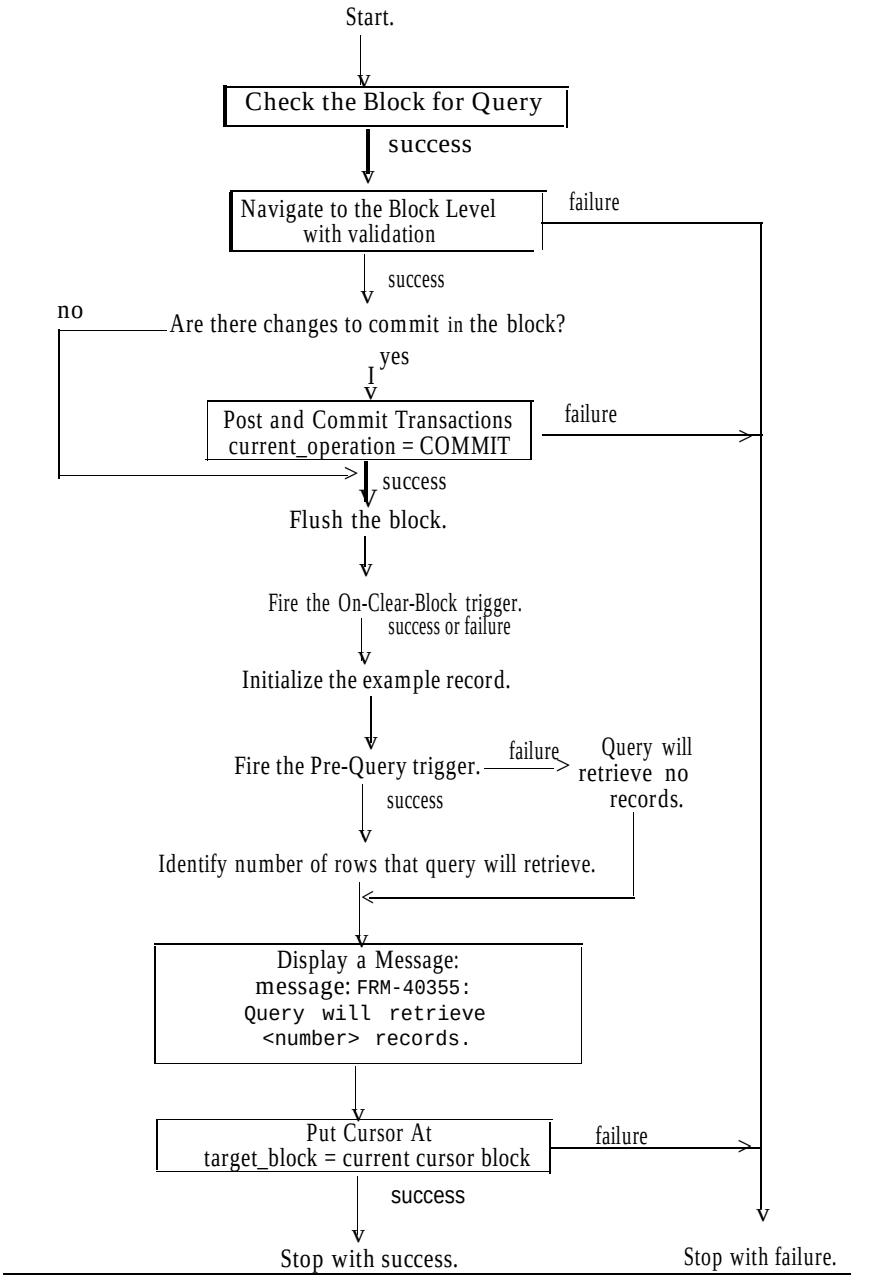
Related Topics: COPY, packaged procedure 16-11

COPY_REGION



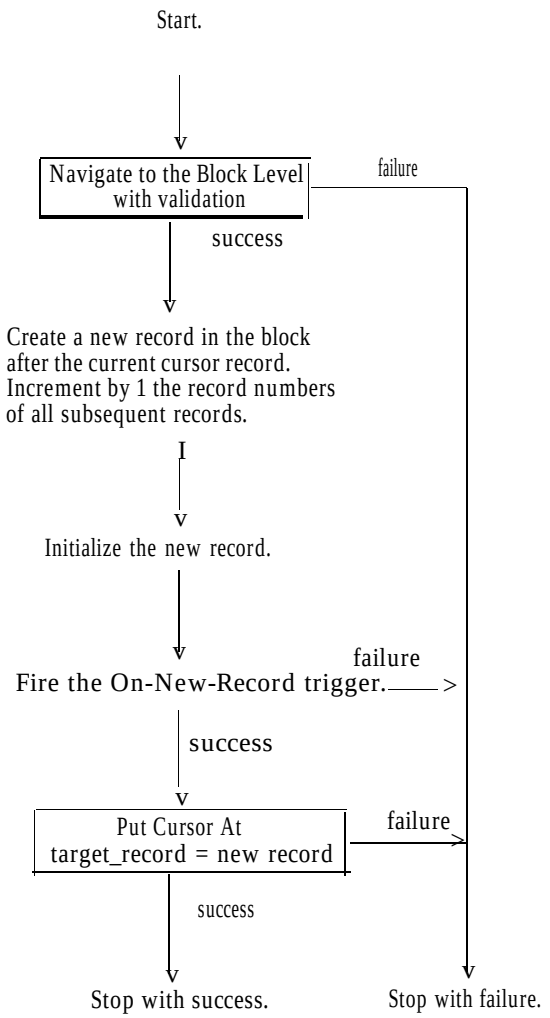
Related Topics: COPY_REGION, packaged procedure 16-11

COUNT_QUERY



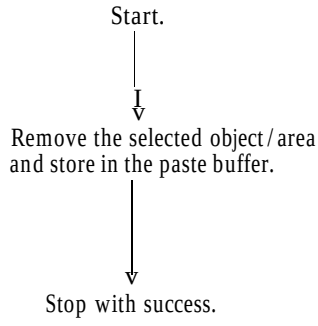
Related Topics: COUNT_QUERY, packaged procedure 16-11
Counting query hits 3-30

CREATE_RECORD



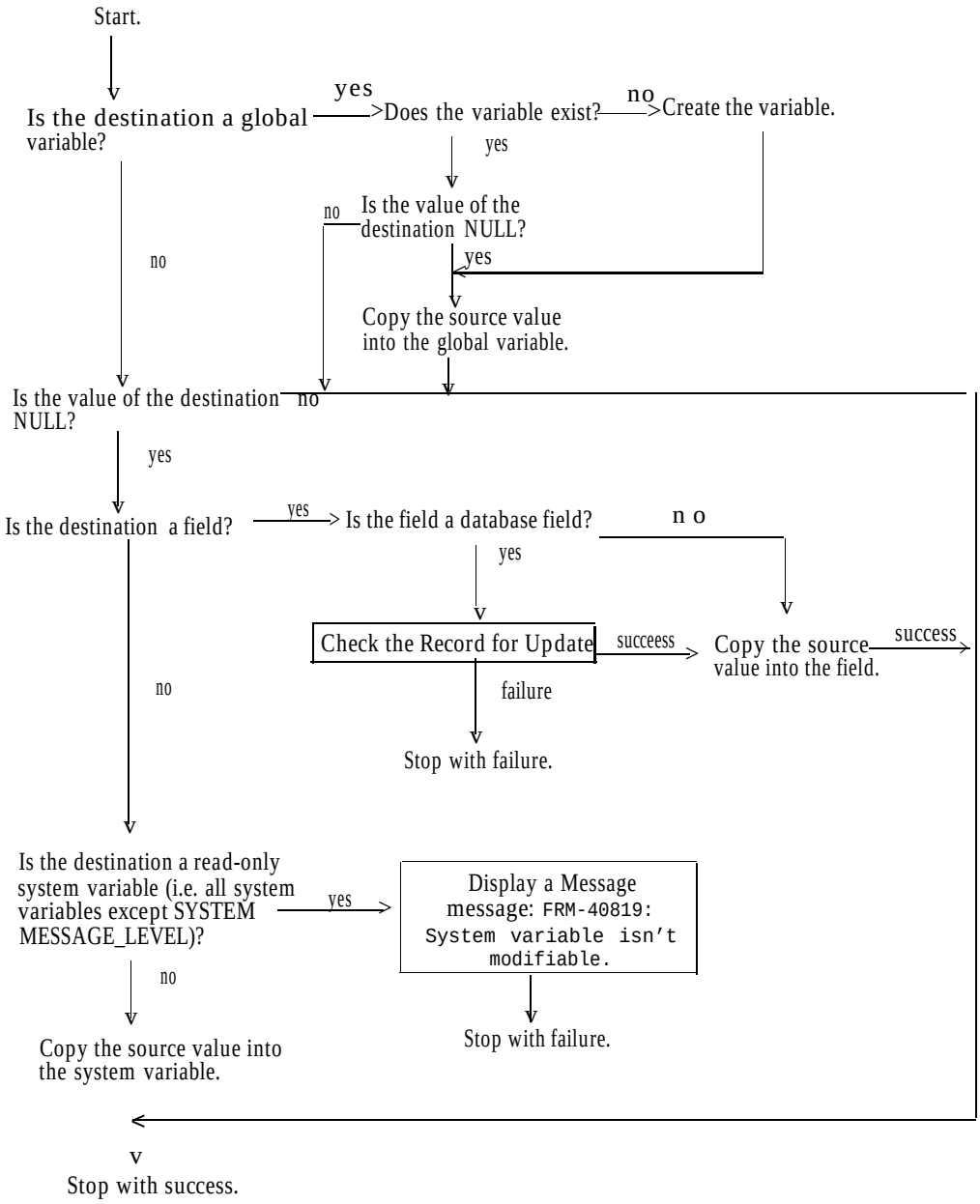
Related Topics: CREATE_RECORD, packaged procedure 16-l2

CUT_REGION



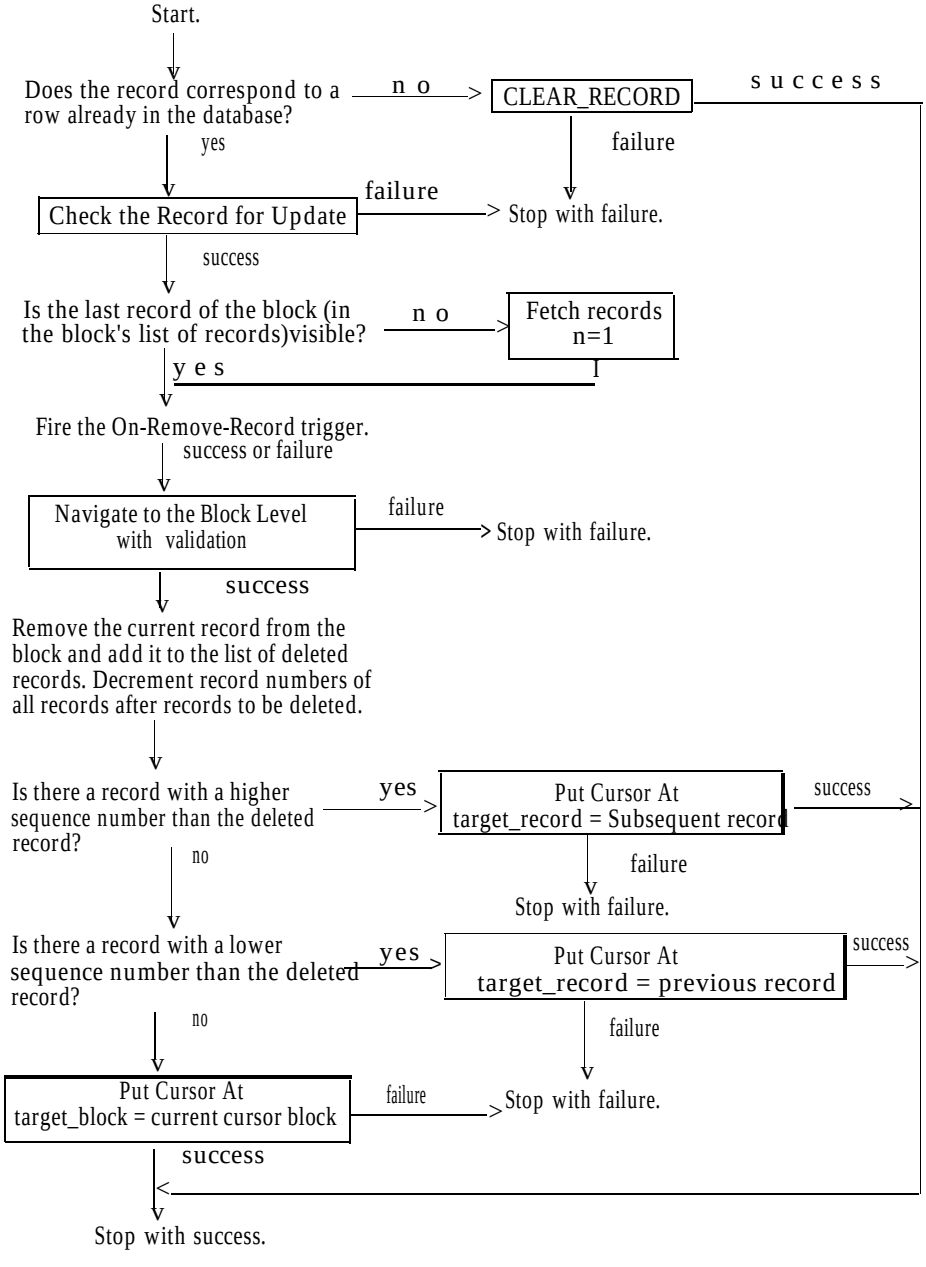
Related Topics: CUT_REGION, packaged procedure 16-12

DEFAULT_VALUE



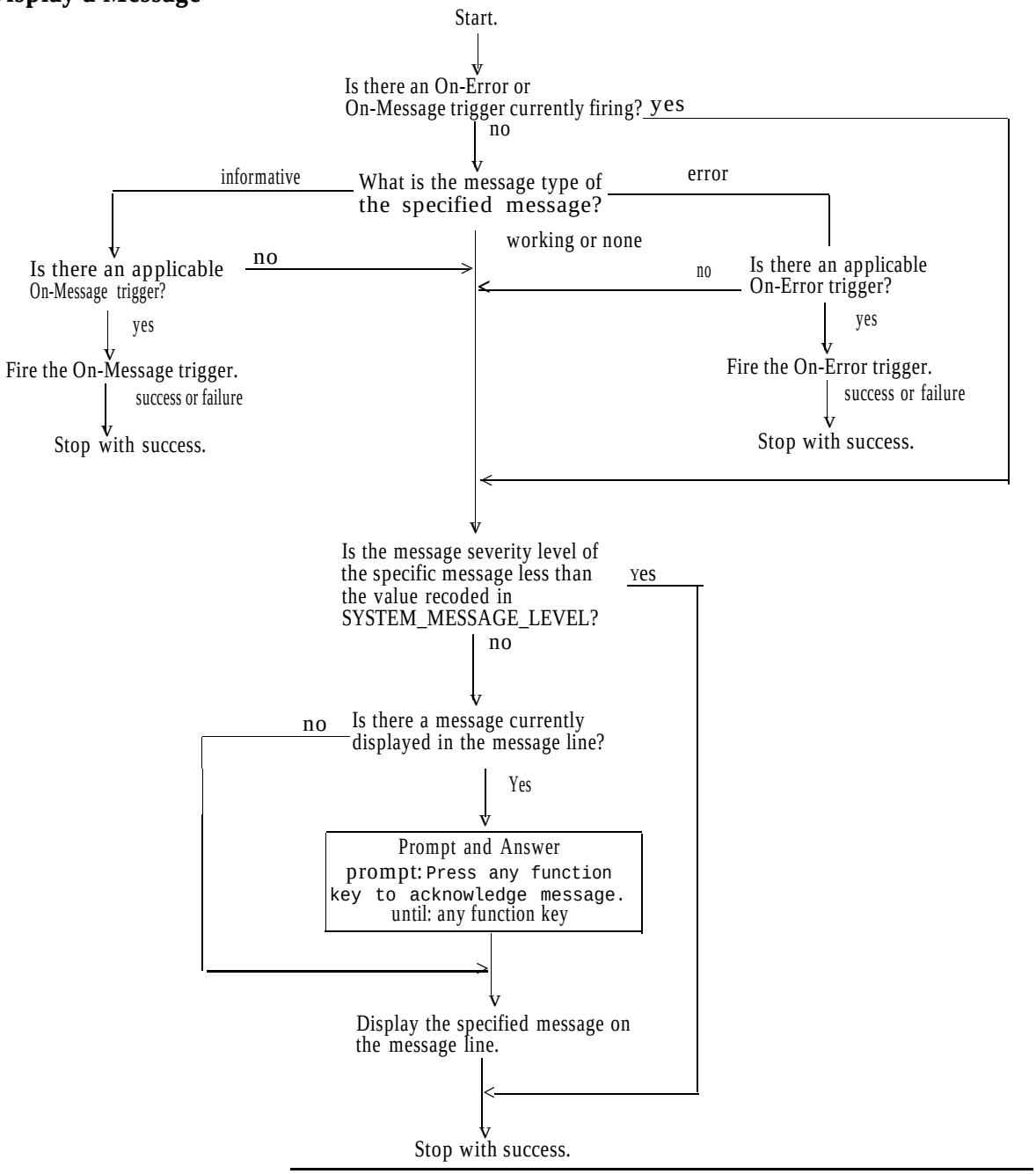
Related Topics: DEFAULT_VALUE, packaged procedure 16-12

DELETE_RECORD

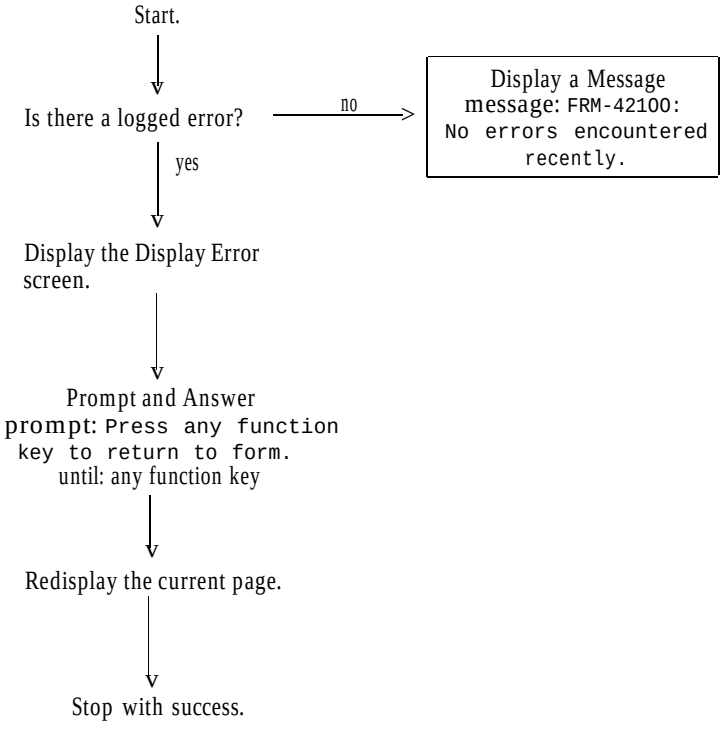


Related Topics: DELETE_RECORD, packaged procedure 16-12

Display a Message

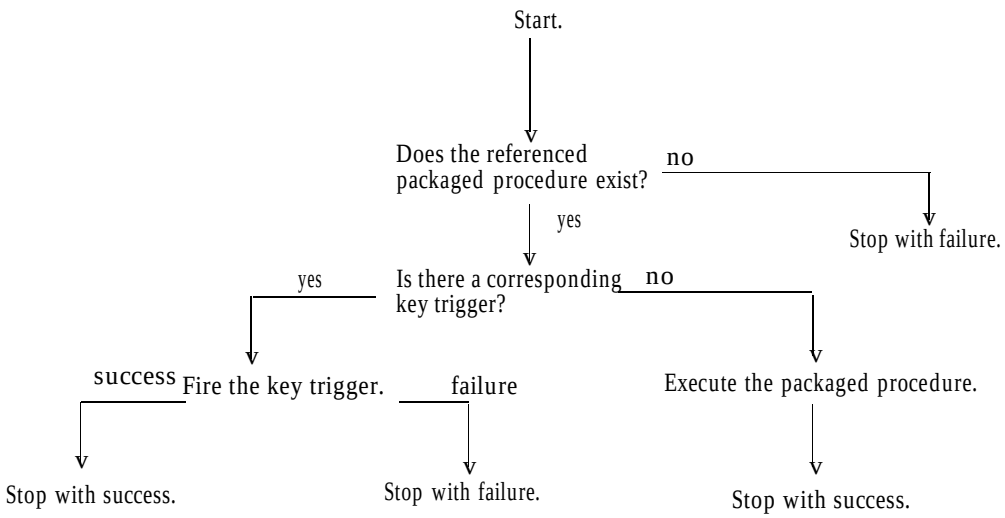


DISPLAY_ERROR



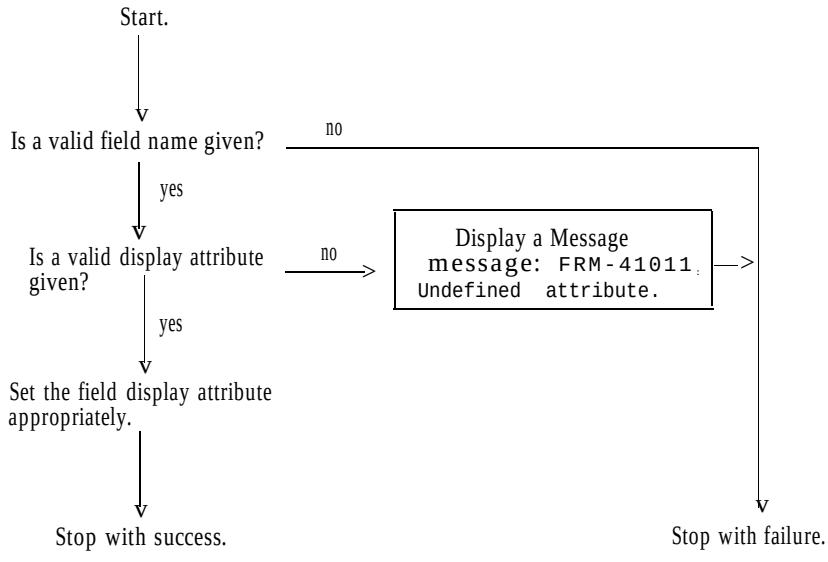
Related Topics: DISPLAY_ERROR packaged procedure 16-13

DO_KEY



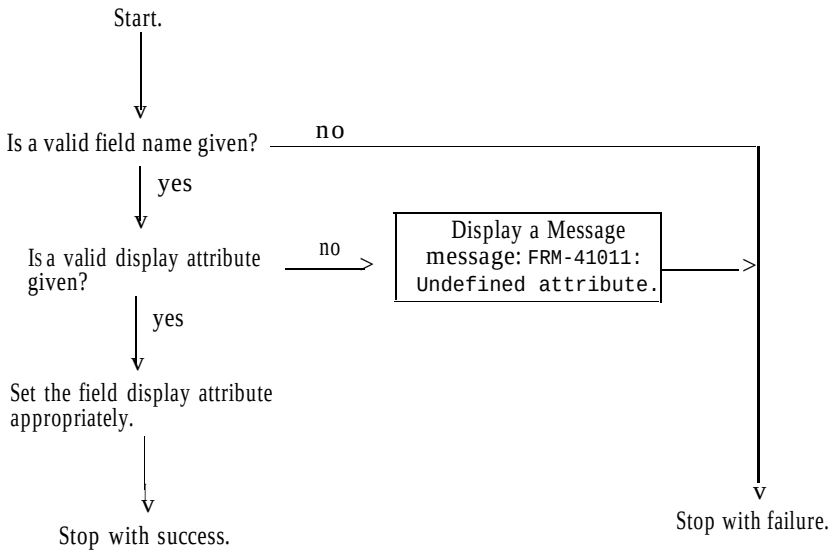
Related Topics: DO_KEY, packaged procedures 16-14

DISPLAY_FIELD



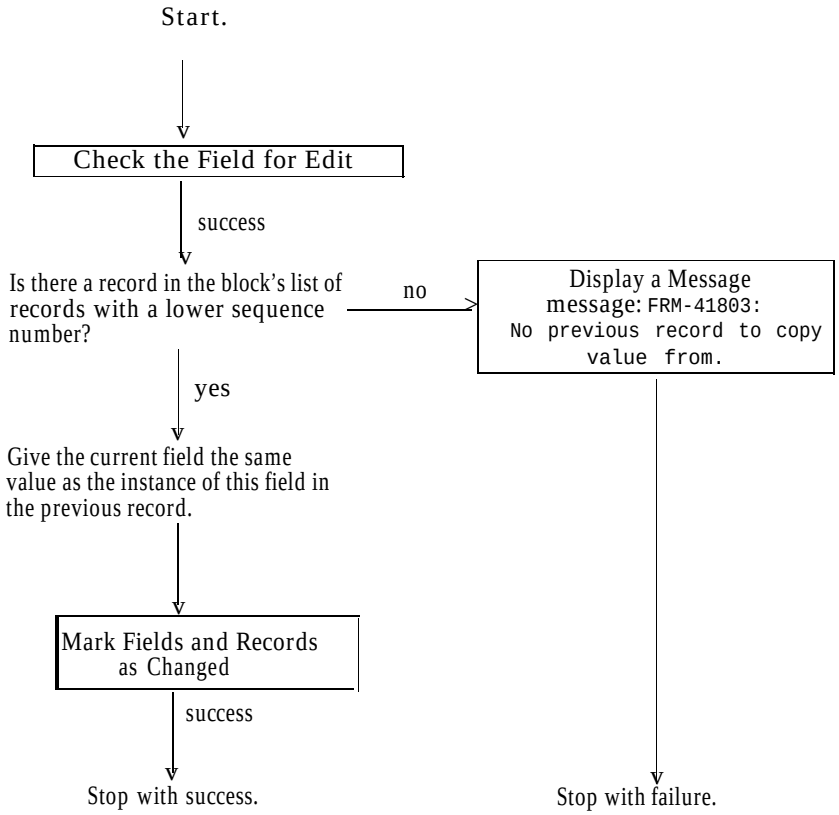
Related Topics: DISPLAY_FIELD, packaged procedure 16-13

DISPLAY_PAGE



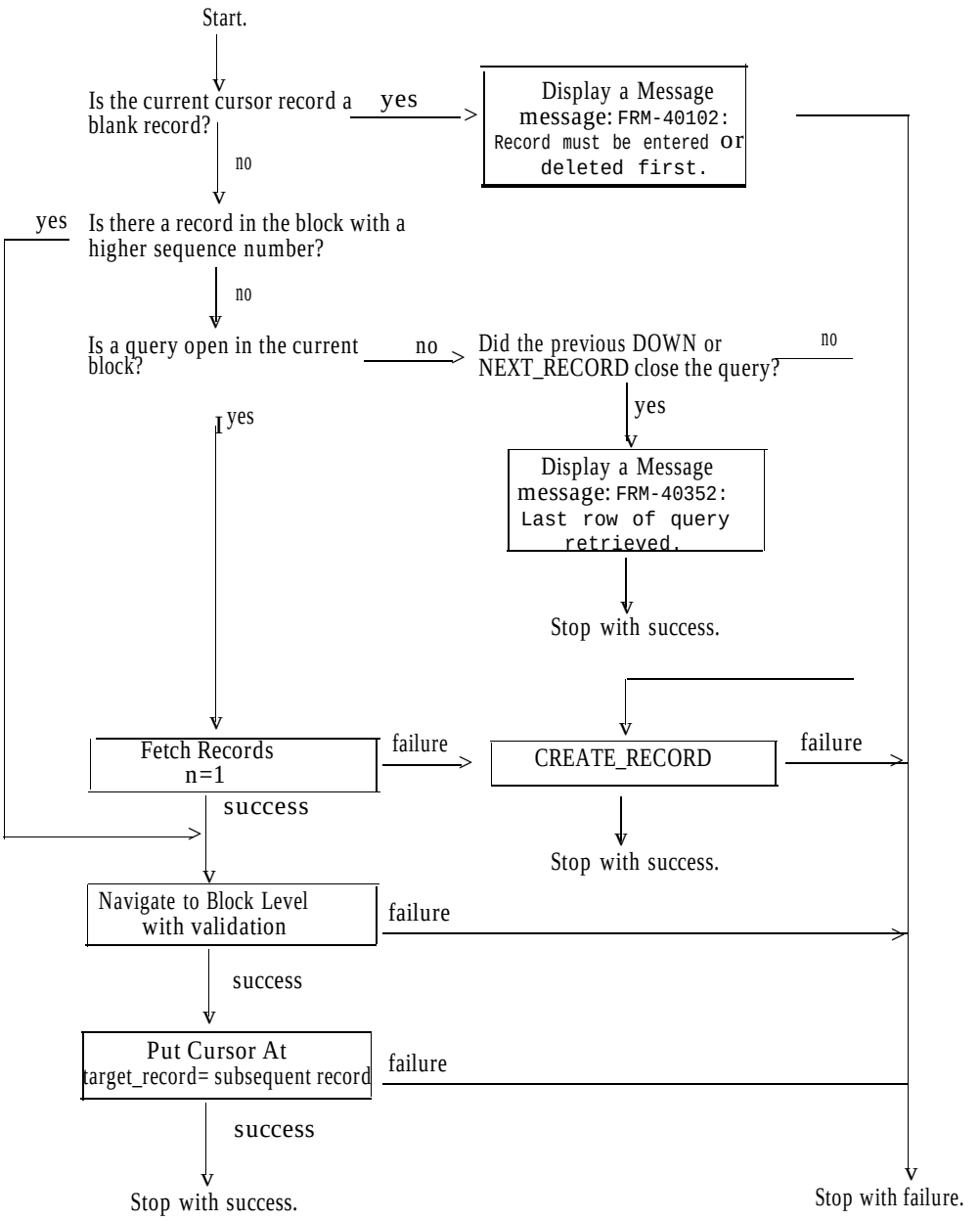
Related Topics: DISPLAY_PAGE, packaged procedure 16-13

DUPLICATE_FIELD



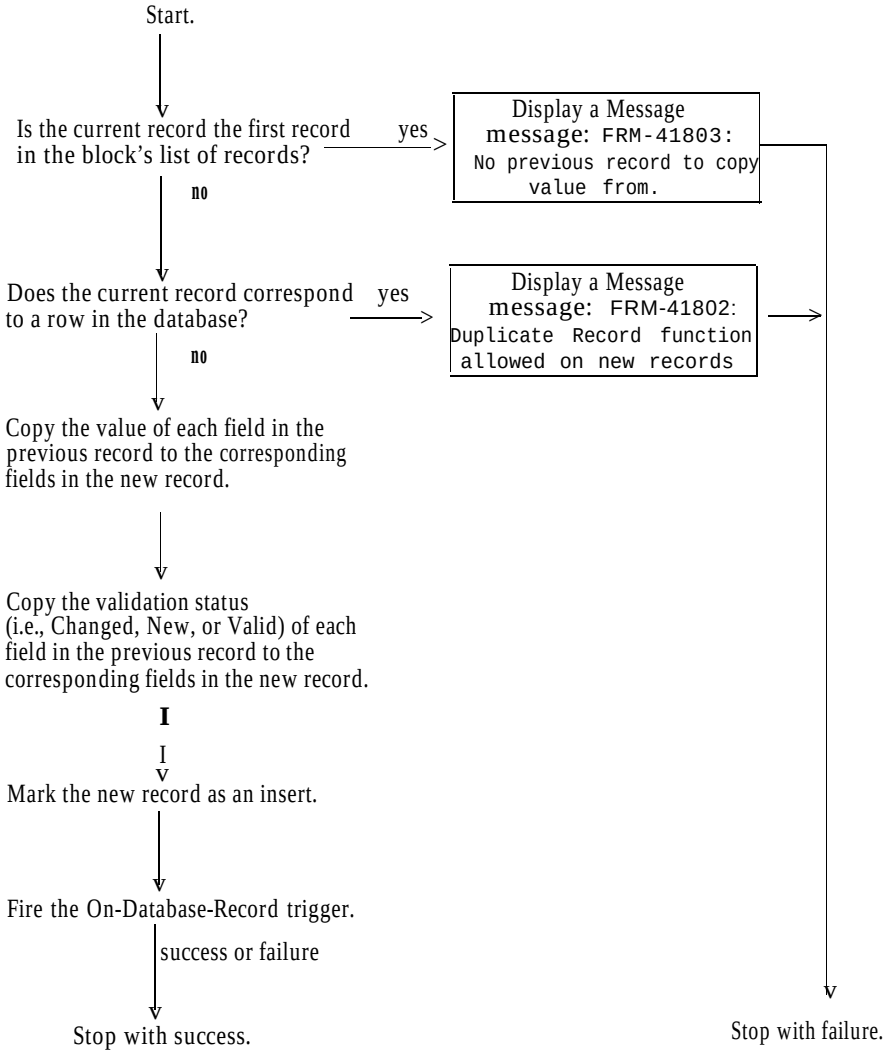
Related Topics: DUPLICATE_FIELD, packaged procedure 16-16

DOWN



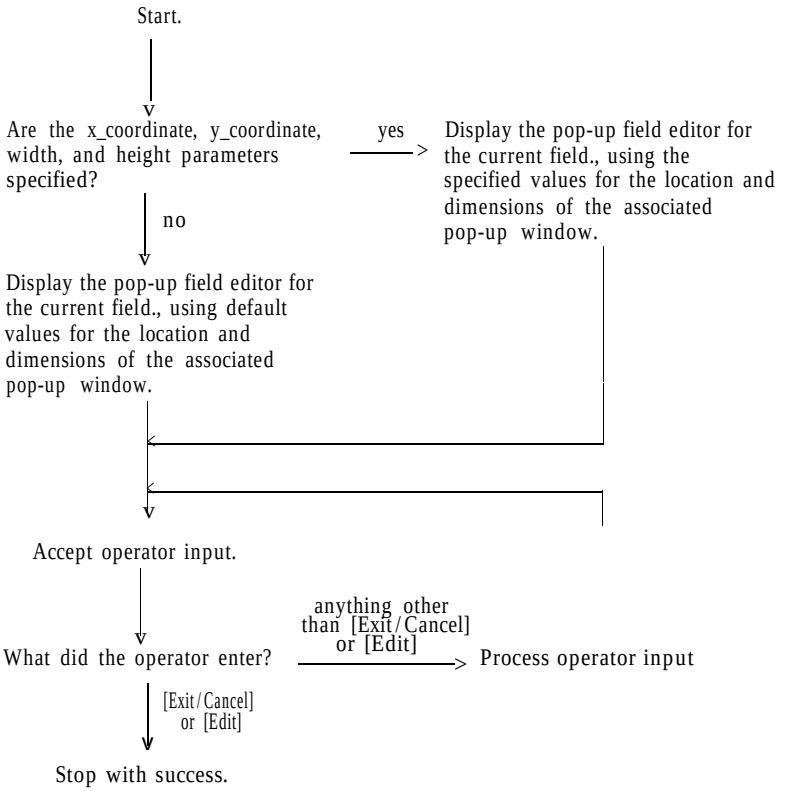
Related Topics: DOWN, packaged procedure 16-16

DUPLICATE_RECORD



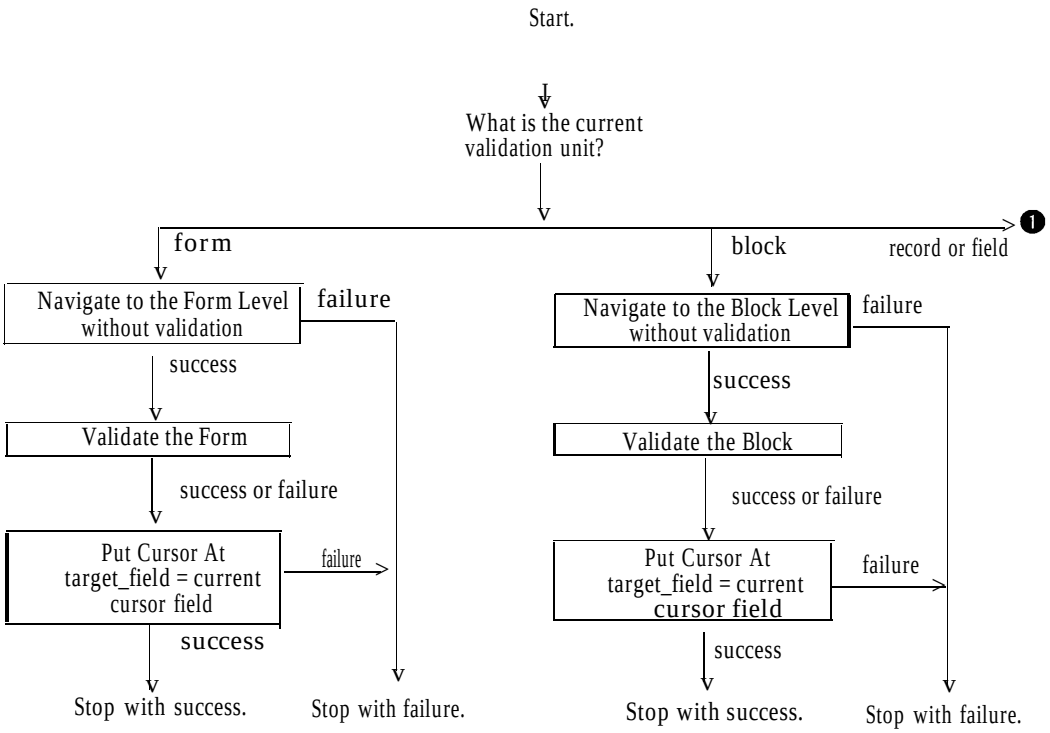
Related Topics: DUPLICATE_RECORD, packaged procedure 16-16

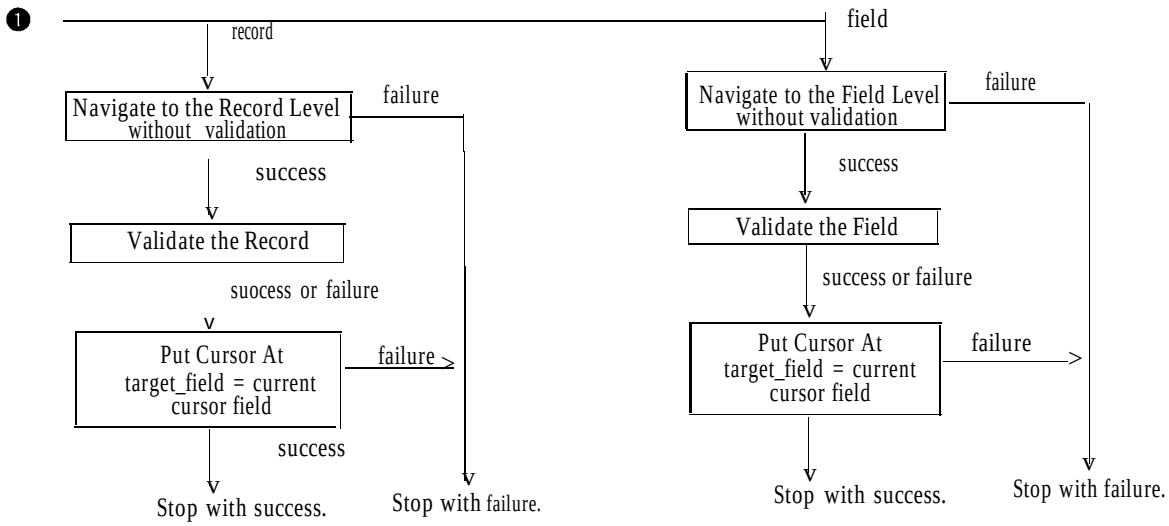
EDIT_FIELD



Related Topics: EDIT_FIELD, packaged procedure 16-16

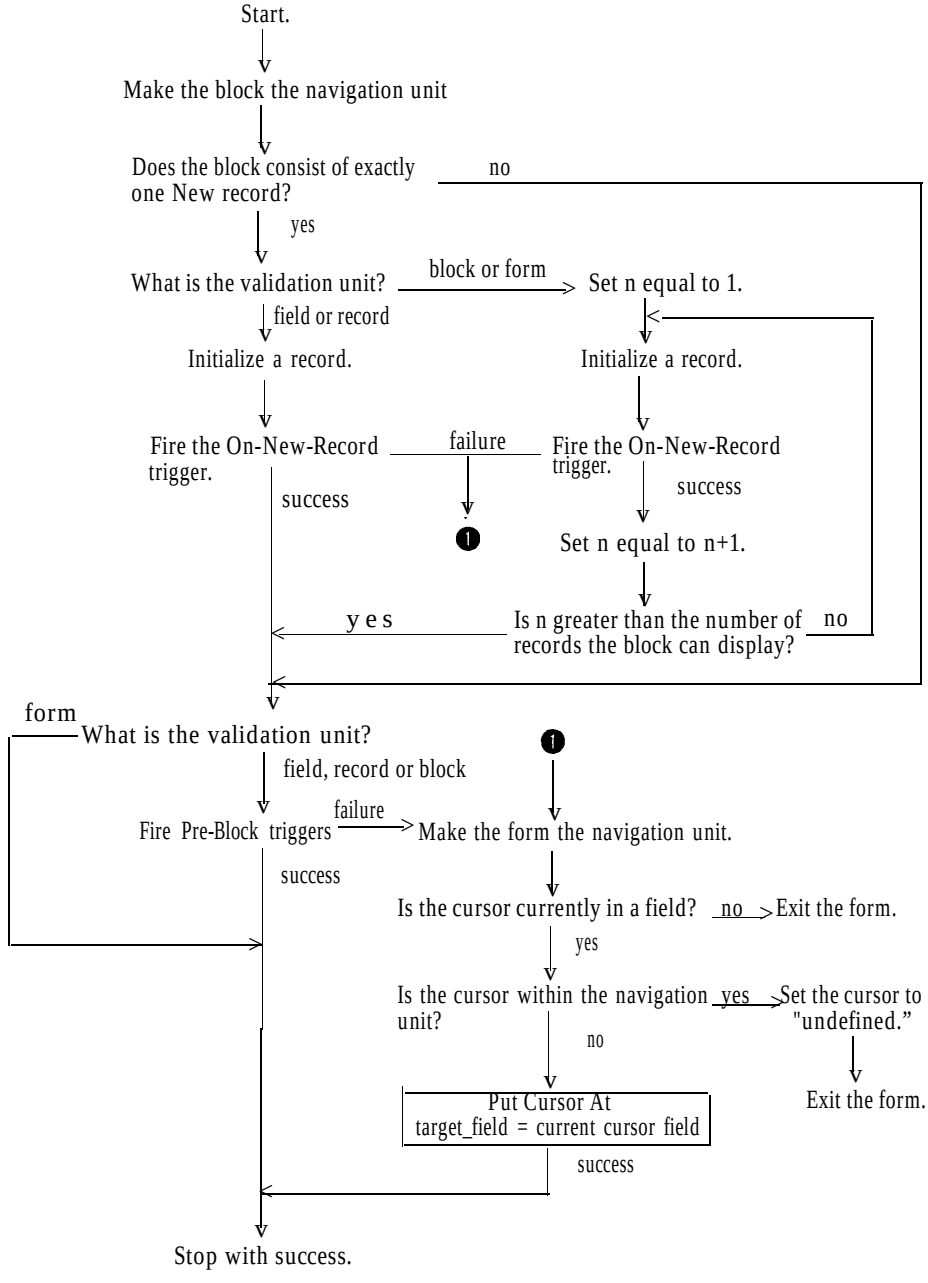
ENTER


Start.
What is the current validation unit?
form
block
record or field
1
Navigate to the Form Level without validation
failure
success
Validate the Form
success or failure
Put Cursor At target_field = current cursor field
failure
success
Stop with success.
Stop with failure.
Navigate to the Block Level without validation
failure
success
Validate the Block
success or failure
Put Cursor At target_field = current cursor field
failure
success
Stop with success.
Stop with failure.



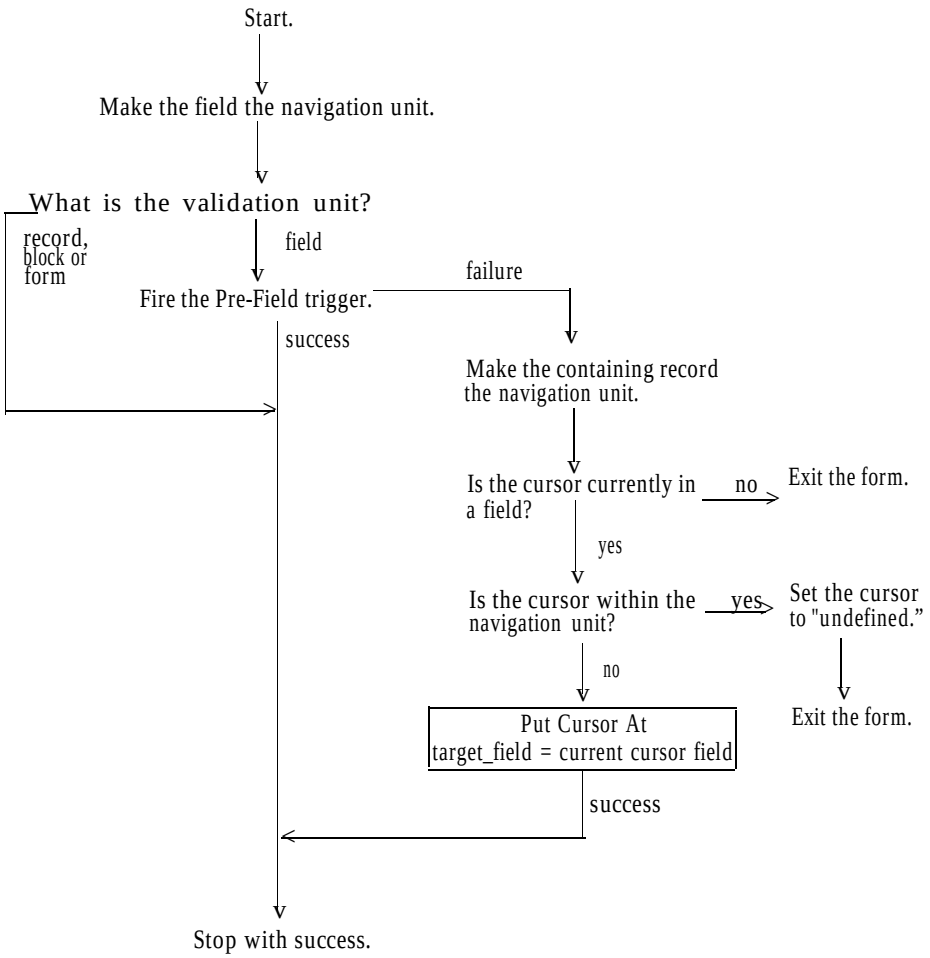
Related Topics: ENTER, packaged procedure 16-17

Enter the Block



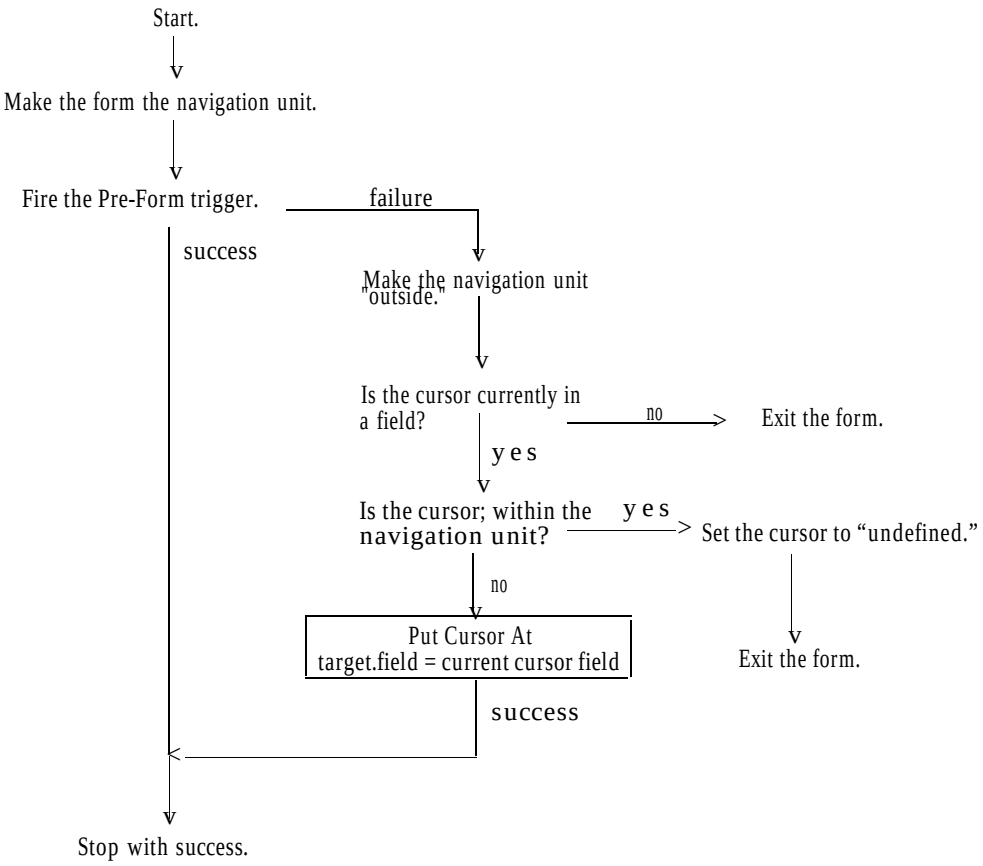
Related Topics: Navigation 3-2

Enter the Field



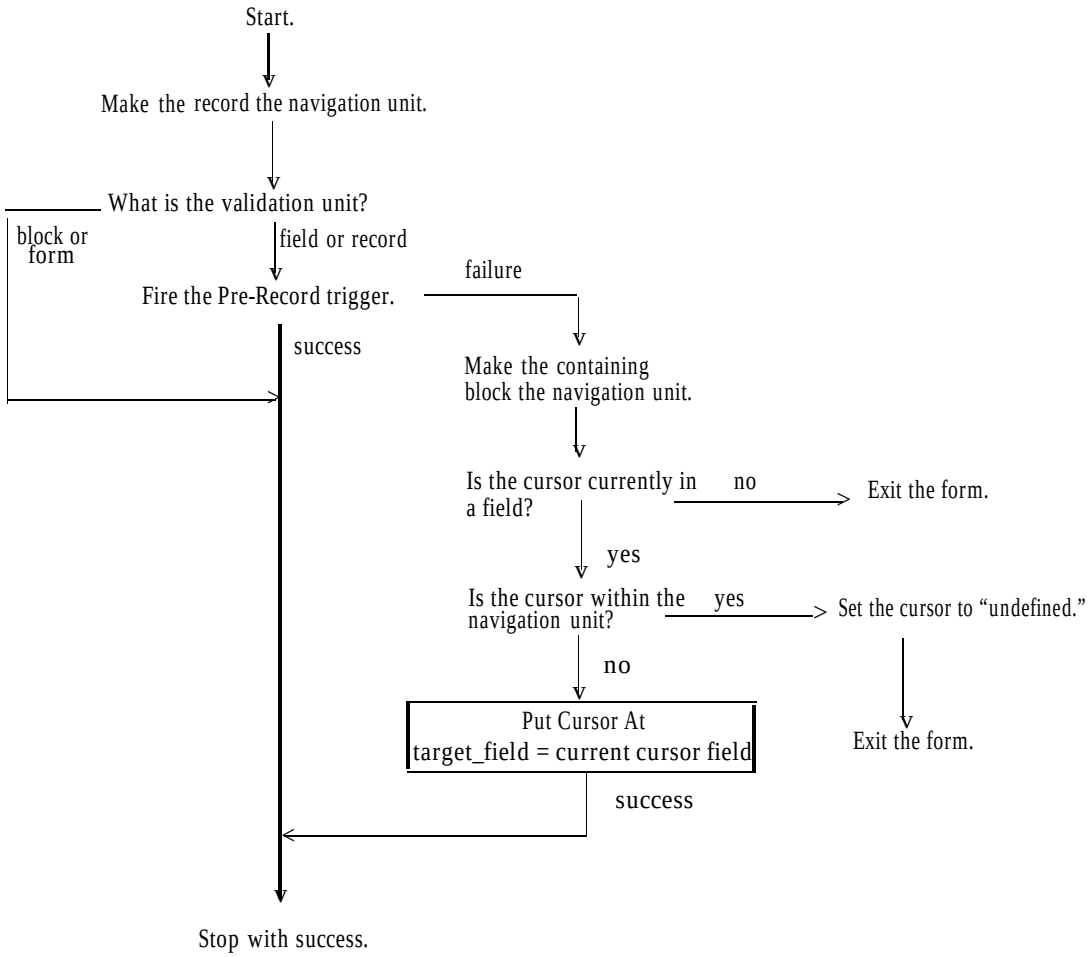
Related Topics: Navigation 3-2

Enter the Form



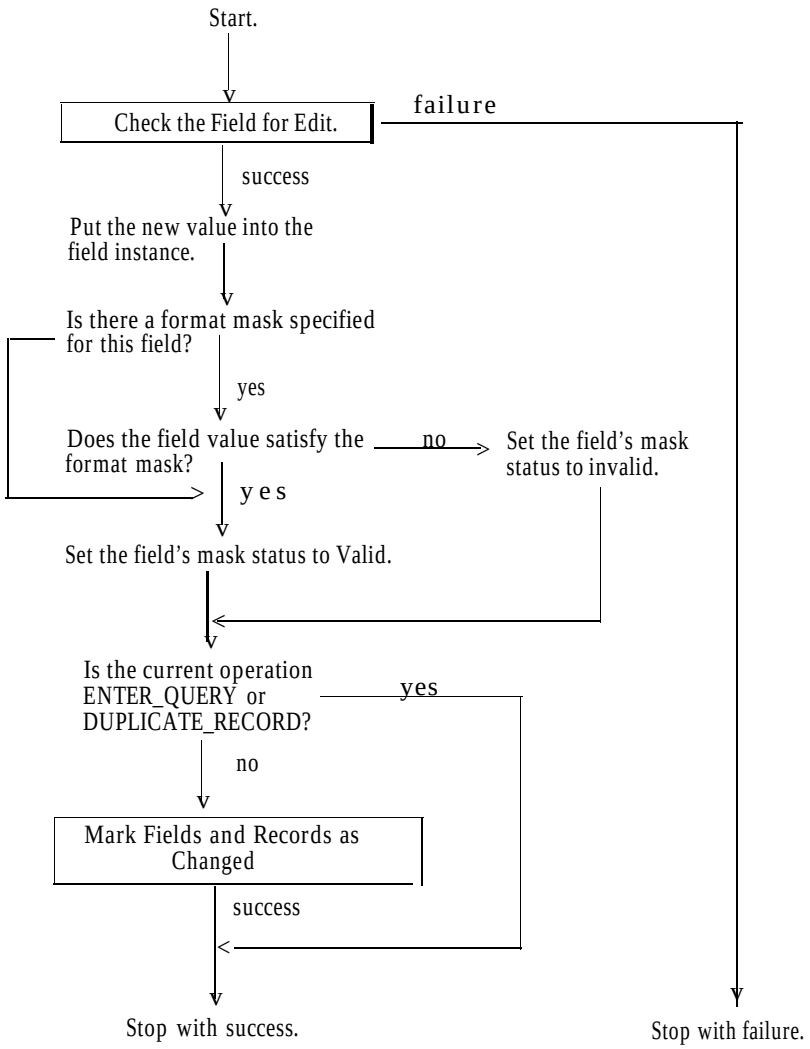
Related Topics: Navigation 3-2

Enter the Record

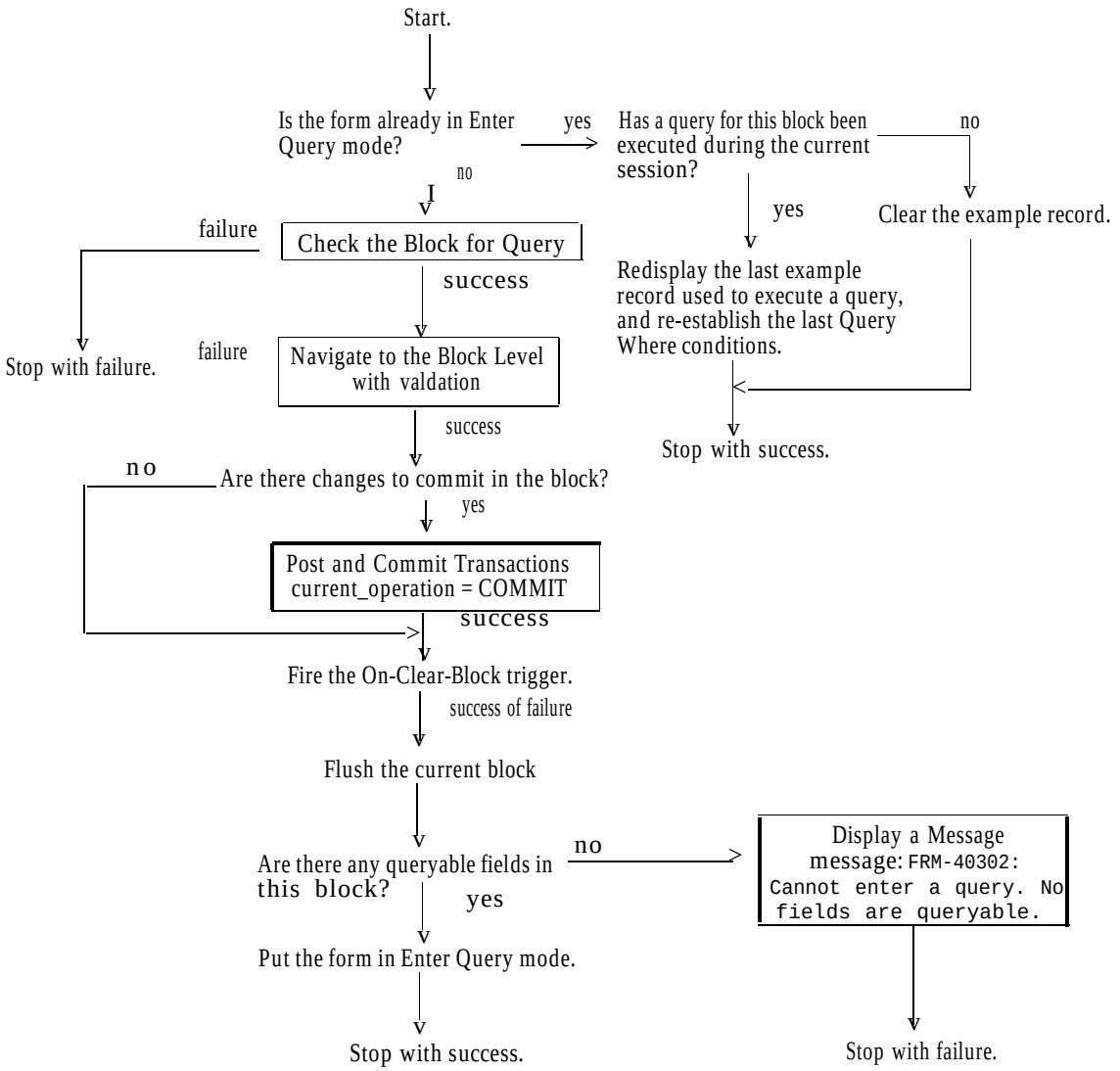


Related Topics: Navigation 3-2

Enter the Value Into a Field

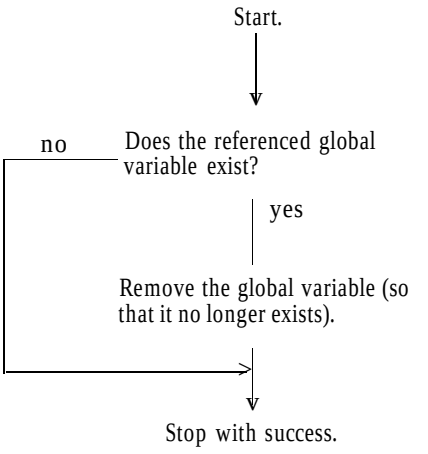


ENTER_QUERY



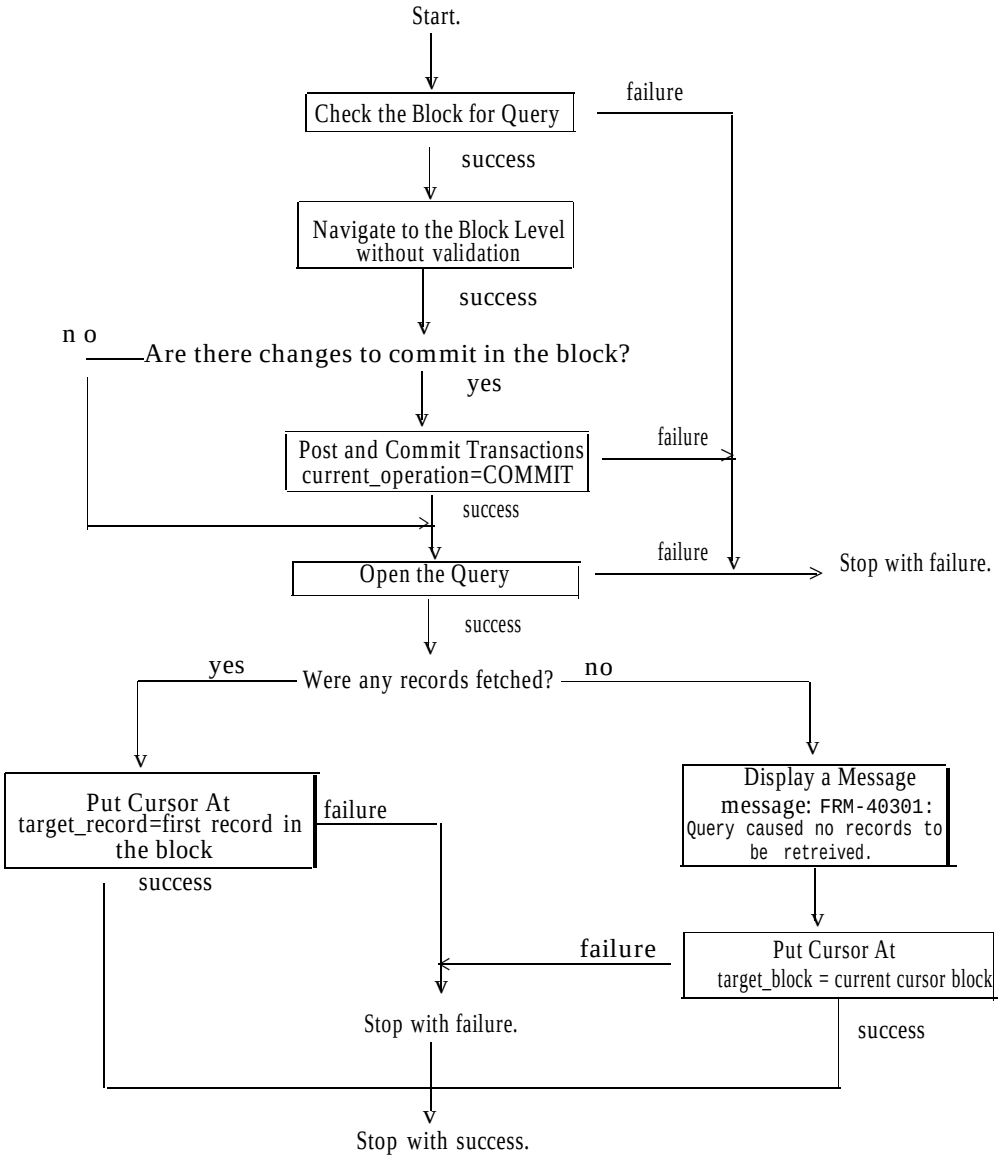
Related Topics: Query processing 3-24
ENTER_QUERY, packaged procedure 16-17

ERASE



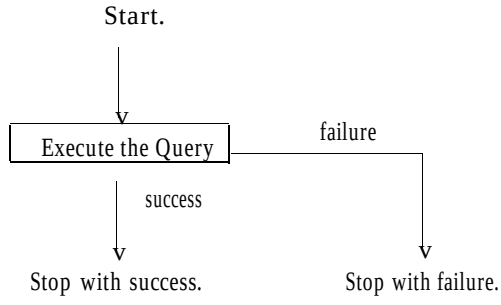
Related Topics: ERASE, packaged procedure 16-18

Execute the Query



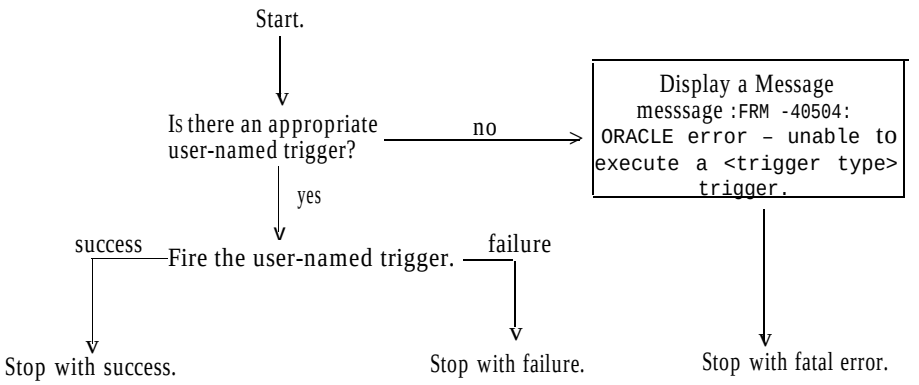
Related Topics: Query processing 3-24

EXECUTE_QUERY



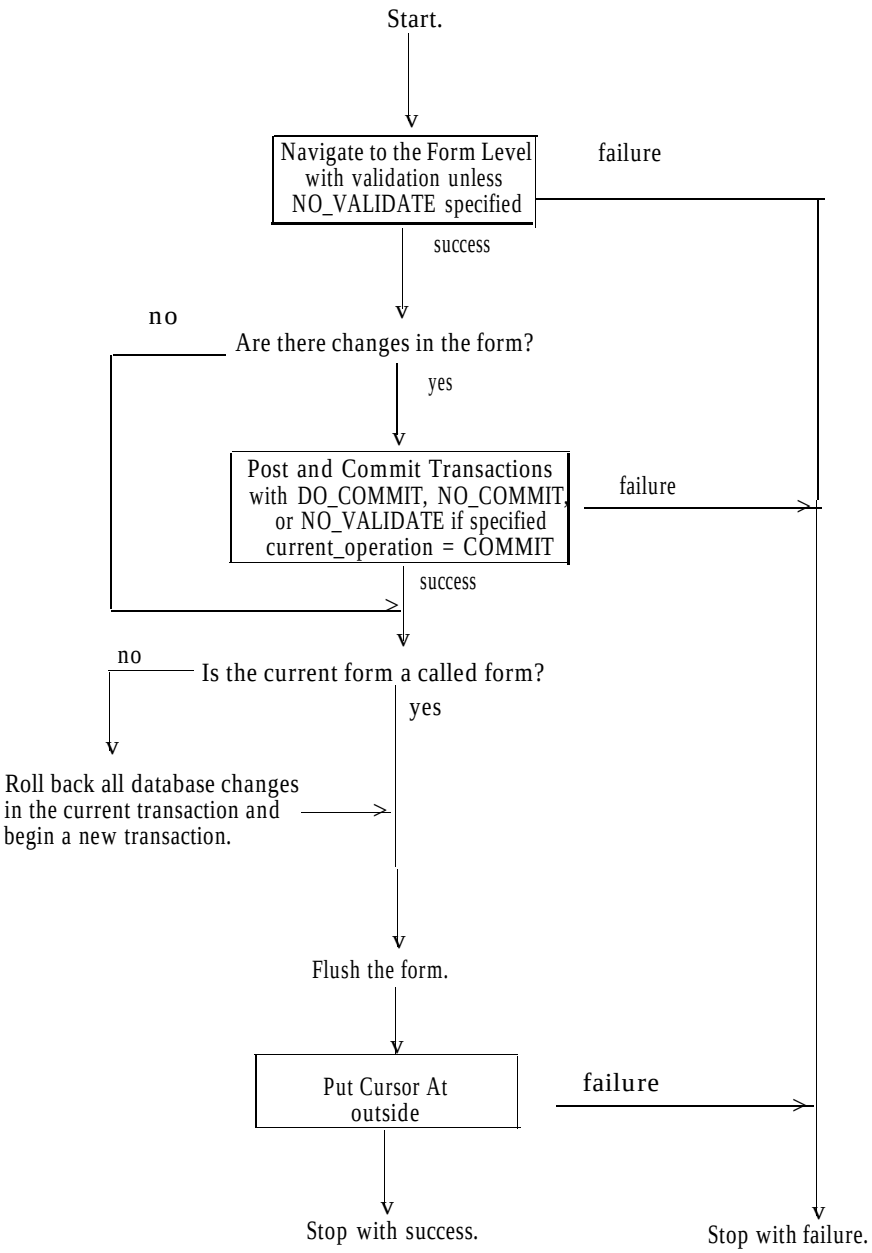
Related Topics: EXECUTE_QUERY, packaged procedure 16-18

EXECUTE_TRIGGER



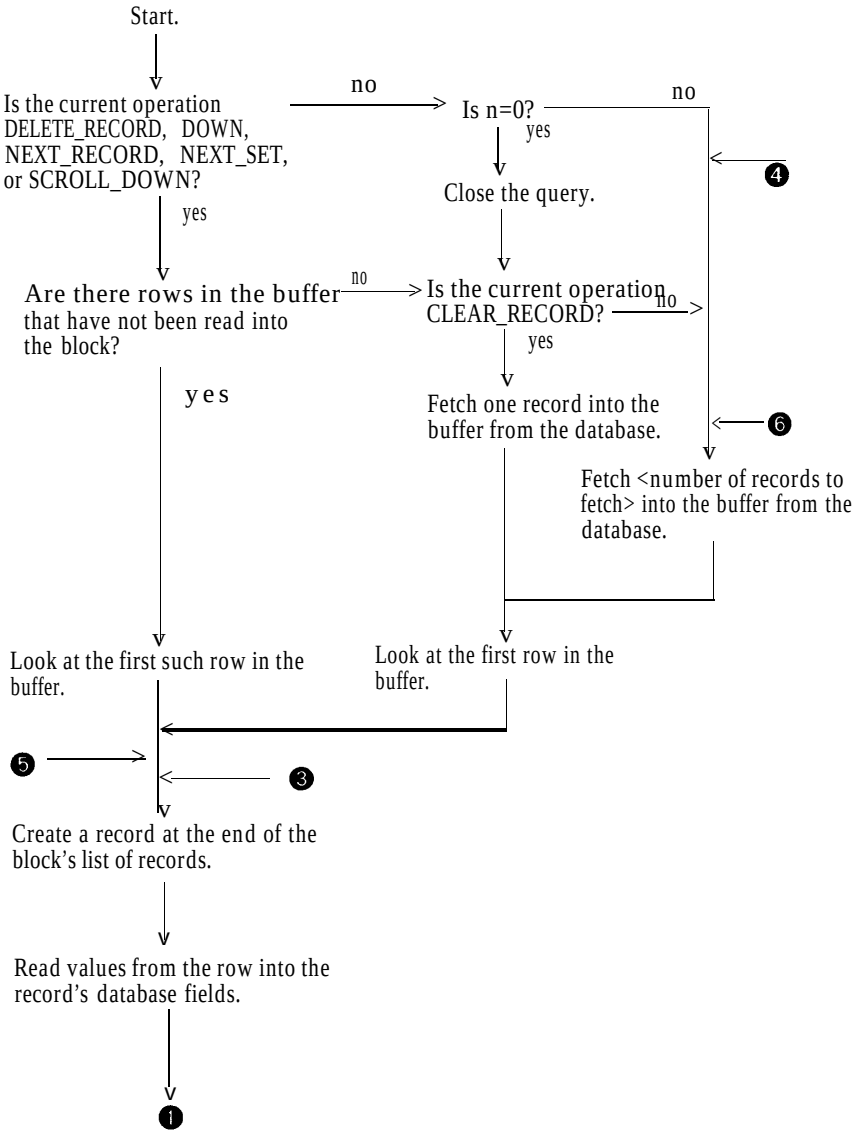
Related Topics: EXECUTE_TRIGGER, packaged procedure 16-19

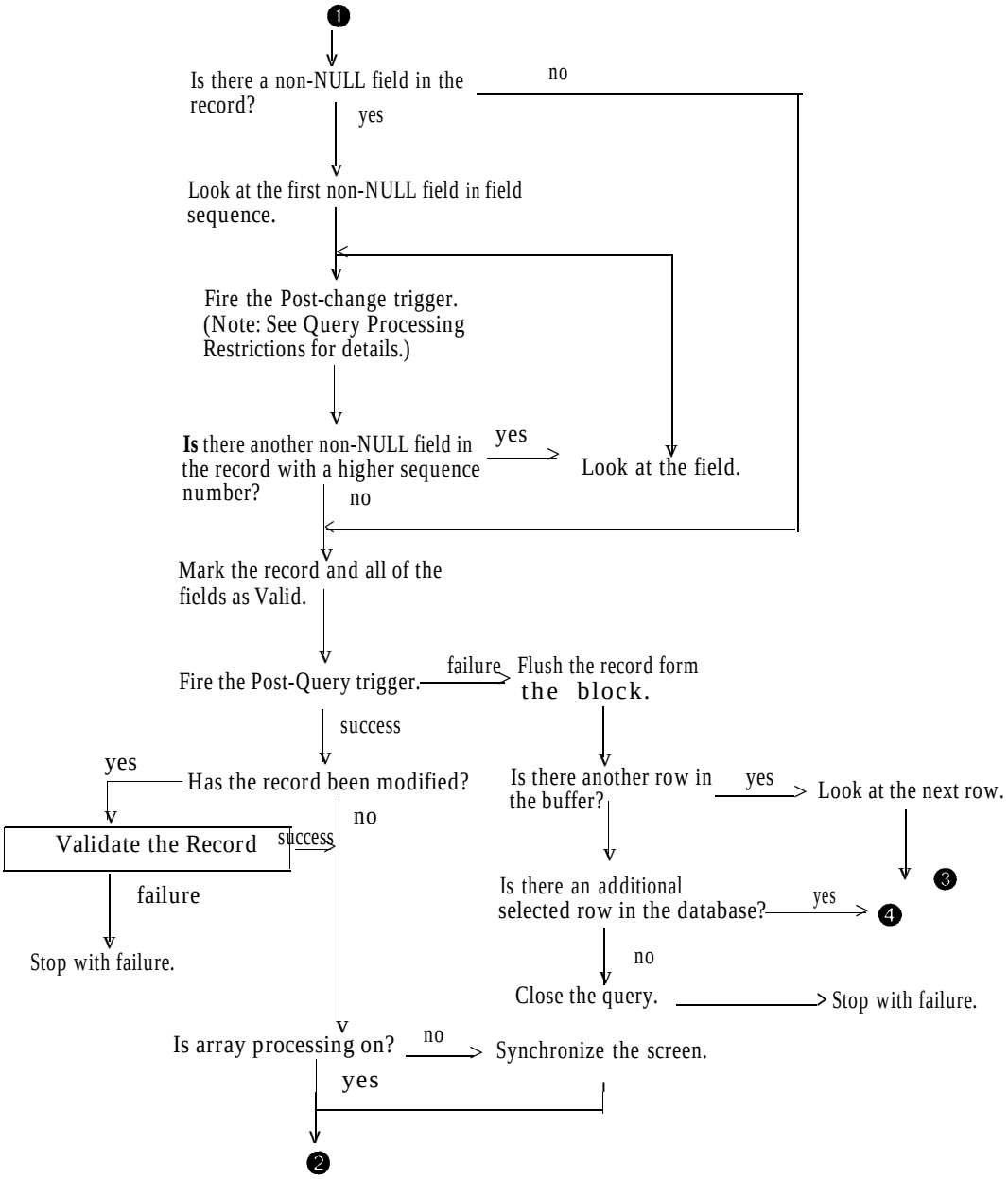
EXIT_FORM

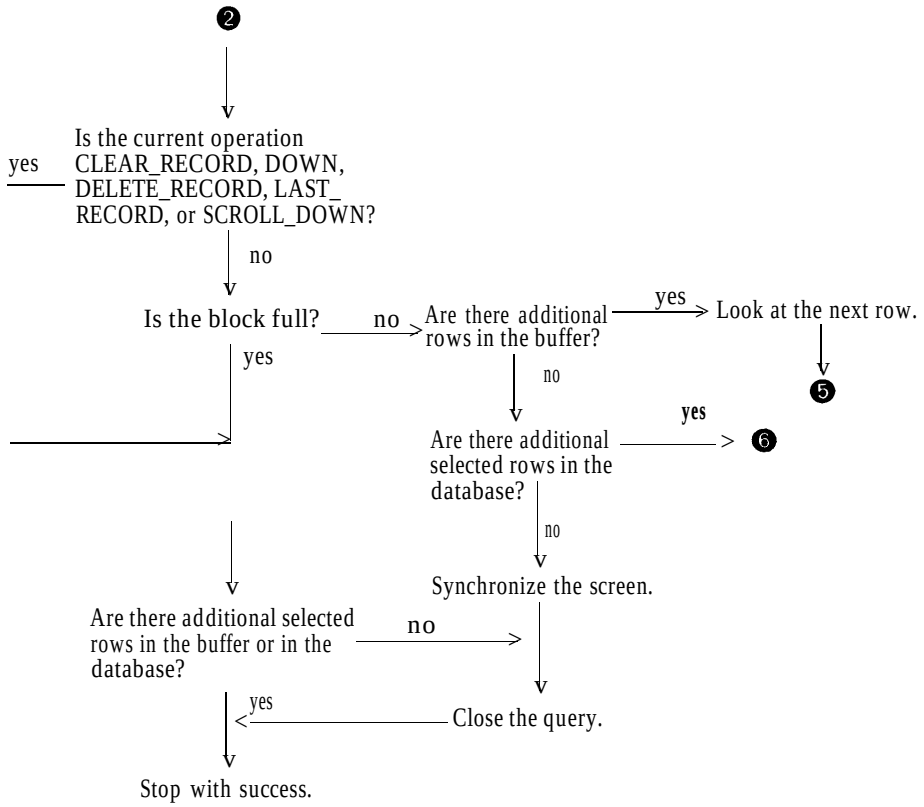


Related Topics: EXIT_FORM, packaged procedure 16-19

Fetch Records

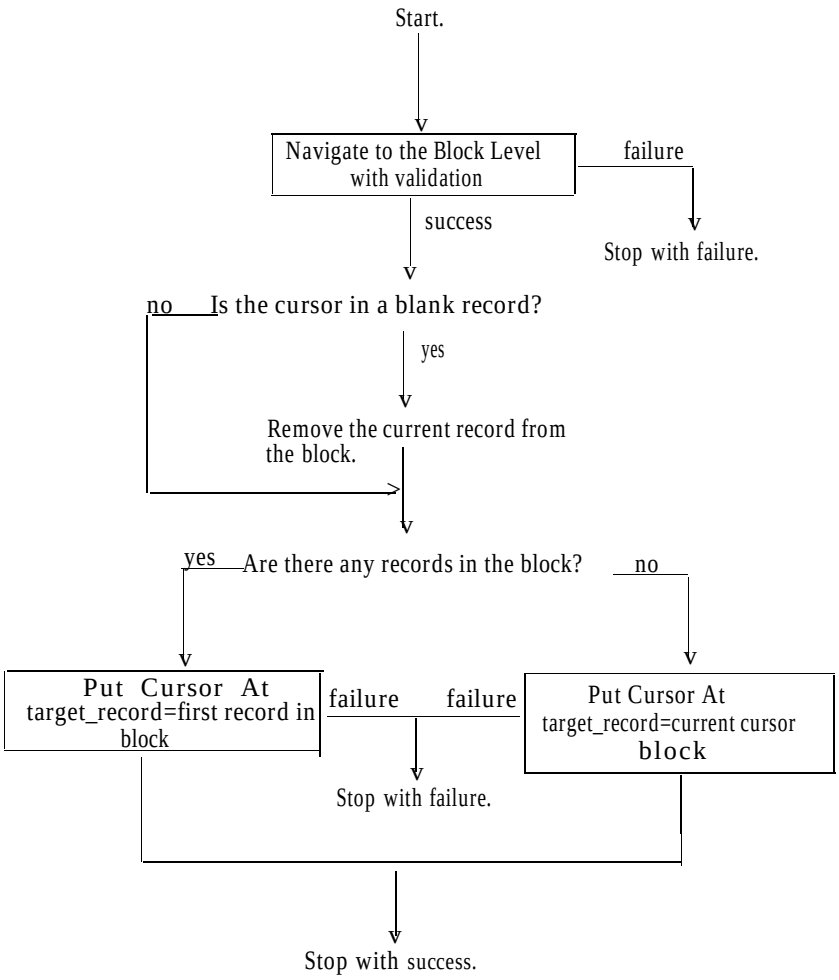



1
Is there a non-NULL field in the record?
no
yes
Look at the first non-NULL field in field sequence.
Fire the Post-change trigger. (Note: See Query Processing Restrictions for details.)
Is there another non-NULL field in the record with a higher sequence number?
yes
Look at the field.
no
Mark the record and all of the fields as Valid.
Fire the Post-Query trigger.
failure
Flush the record form the block.
success
yes
Has the record been modified?
no
Validate the Record
success
failure
Stop with failure.
Is there another row in the buffer?
yes
Look at the next row.
3
Is there an additional selected row in the database?
yes
4
no
Close the query.
Stop with failure.
Is array processing on?
no
Synchronize the screen.
yes
2



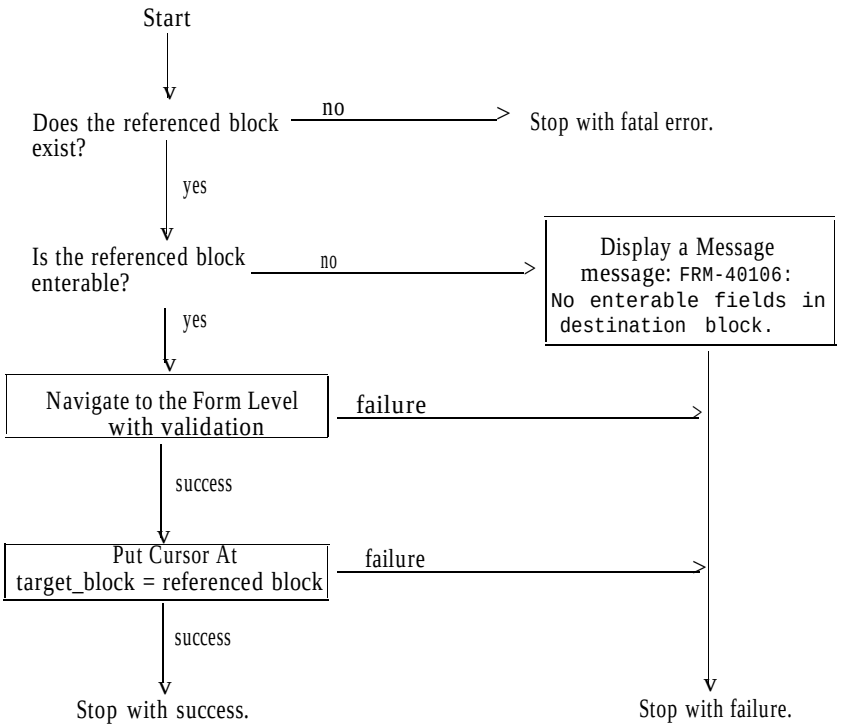
Related Topics: Query processing 3-24

FIRST_RECORD



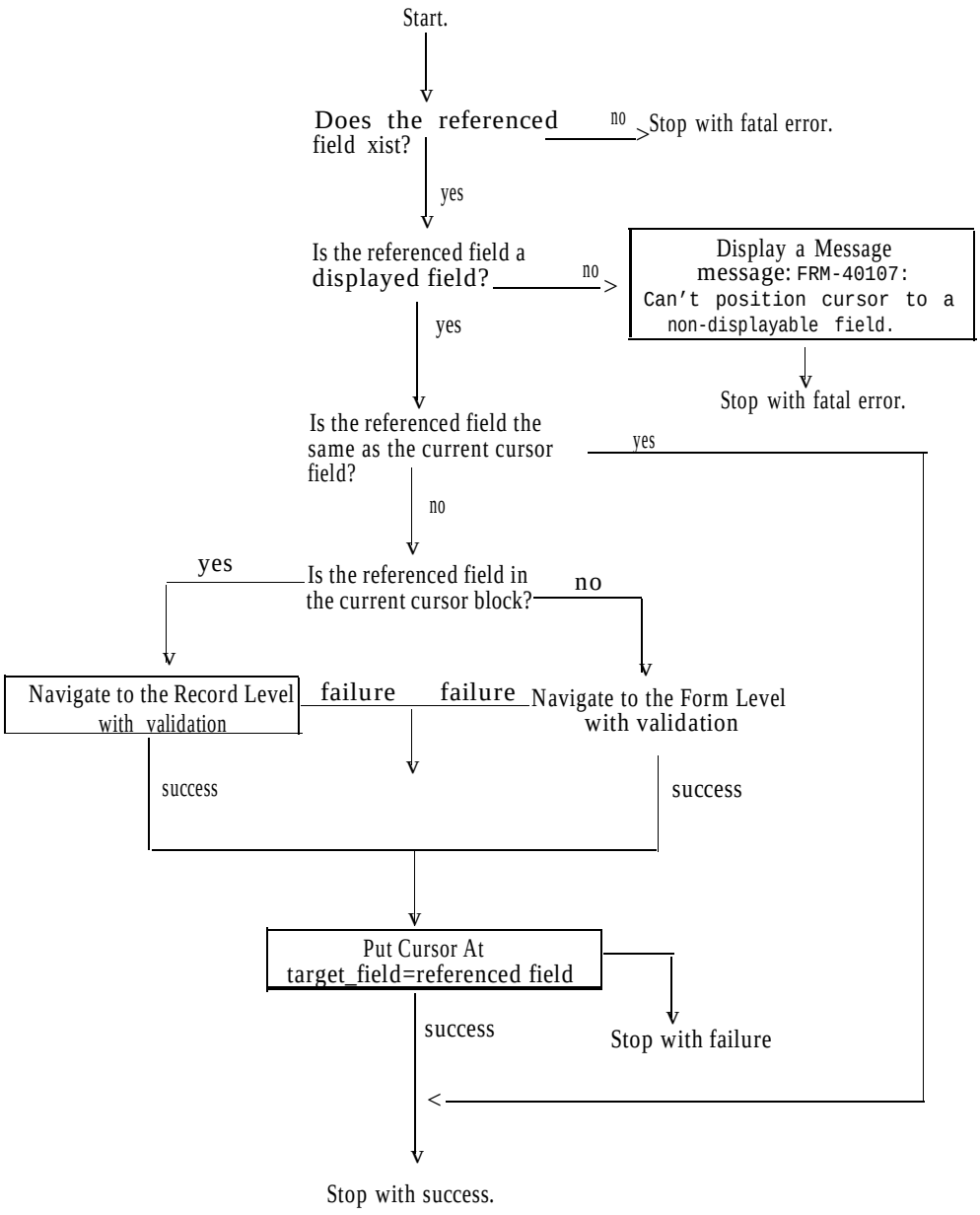
Related Topics: FIRST_RECORD, packaged procedure 16-20

GO_BLOCK



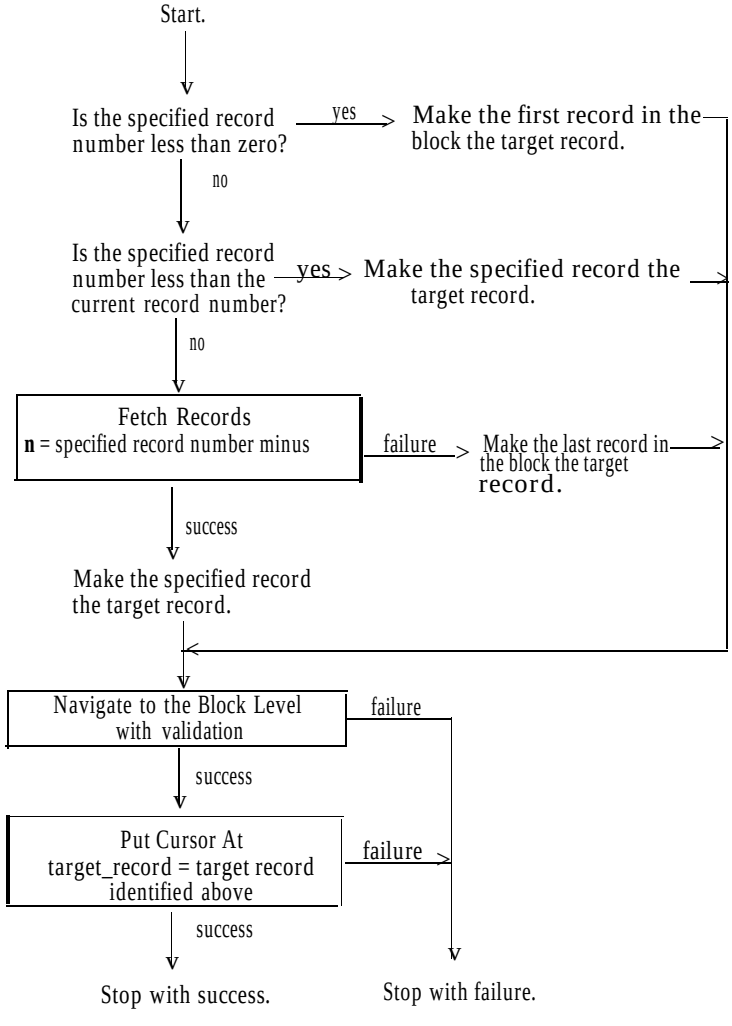
Related Topics: GO_BLOCK, packaged procedure 16-20

GO_FIELD



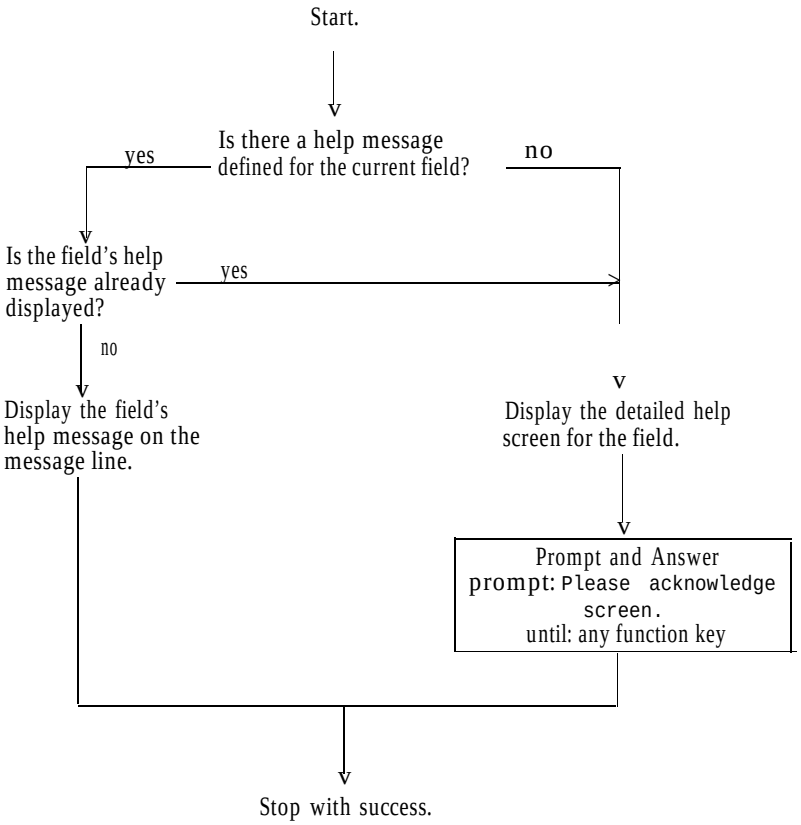
Related Topics: GO_FIELD, packaged procedure 16-20

GO_RECORD



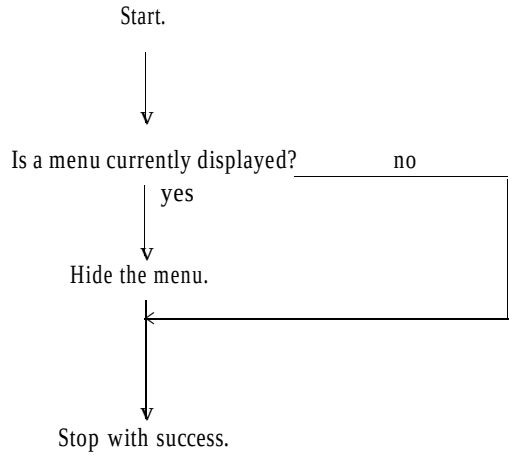
Related Topics: GO_RECORD, packaged procedure 16-21

HELP



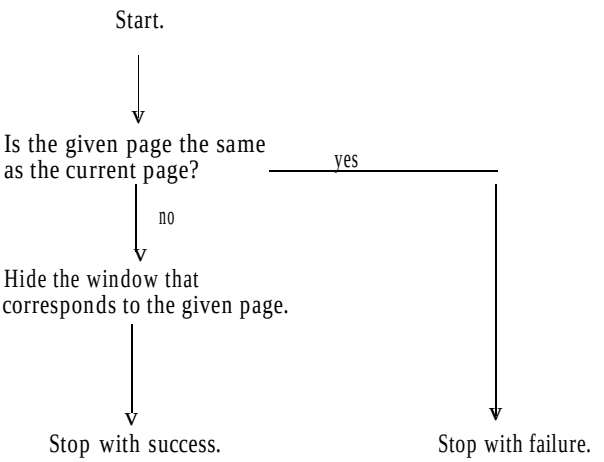
Related Topics: HELP, packaged procedure 16-21

HIDE_MENU



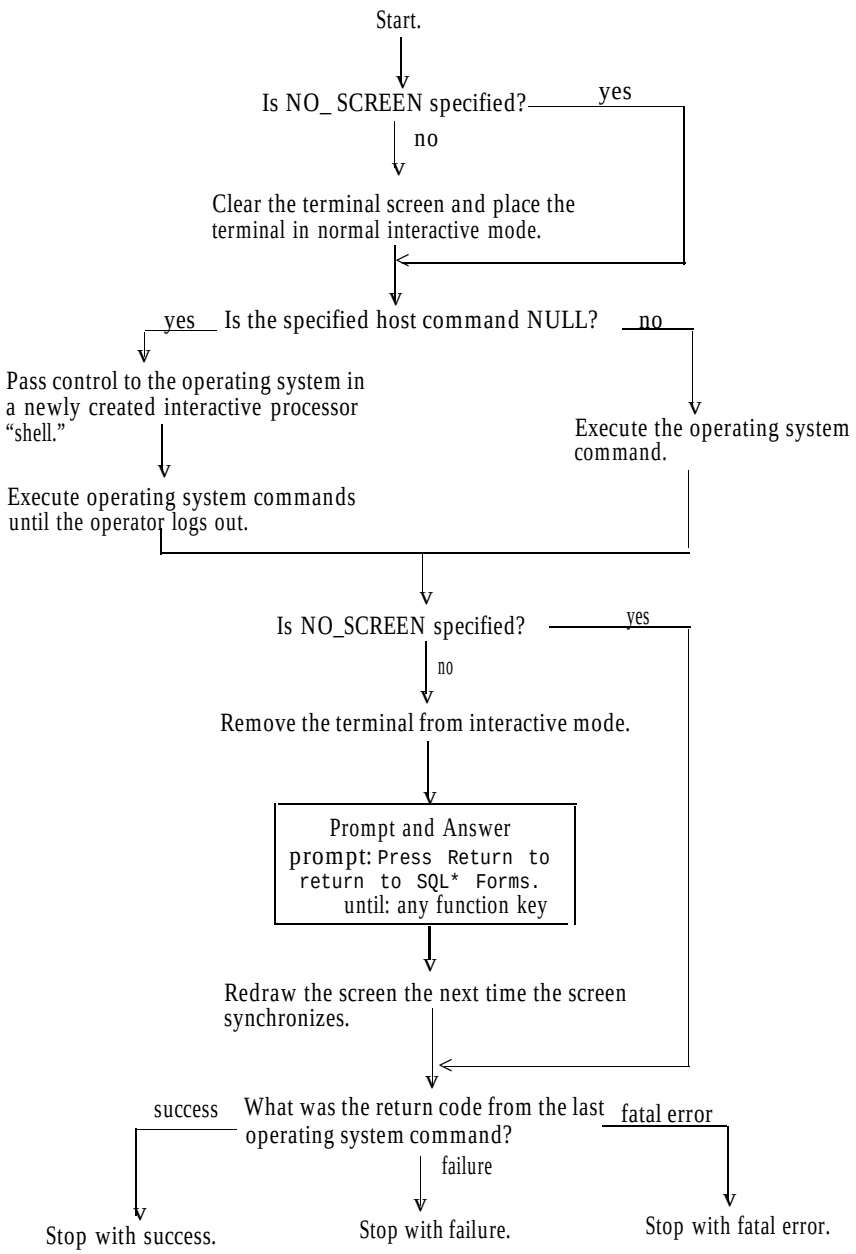
Related Topics: HIDE_MENU, packaged procedure 16-21

HIDE_PAGE



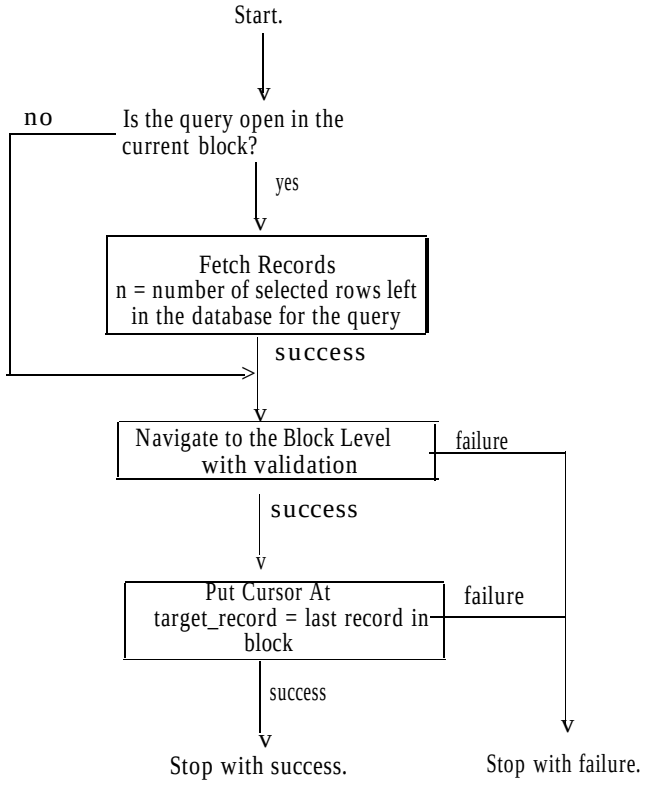
Related Topics: HIDE_PAGE, packaged procedure 16-21

HOST



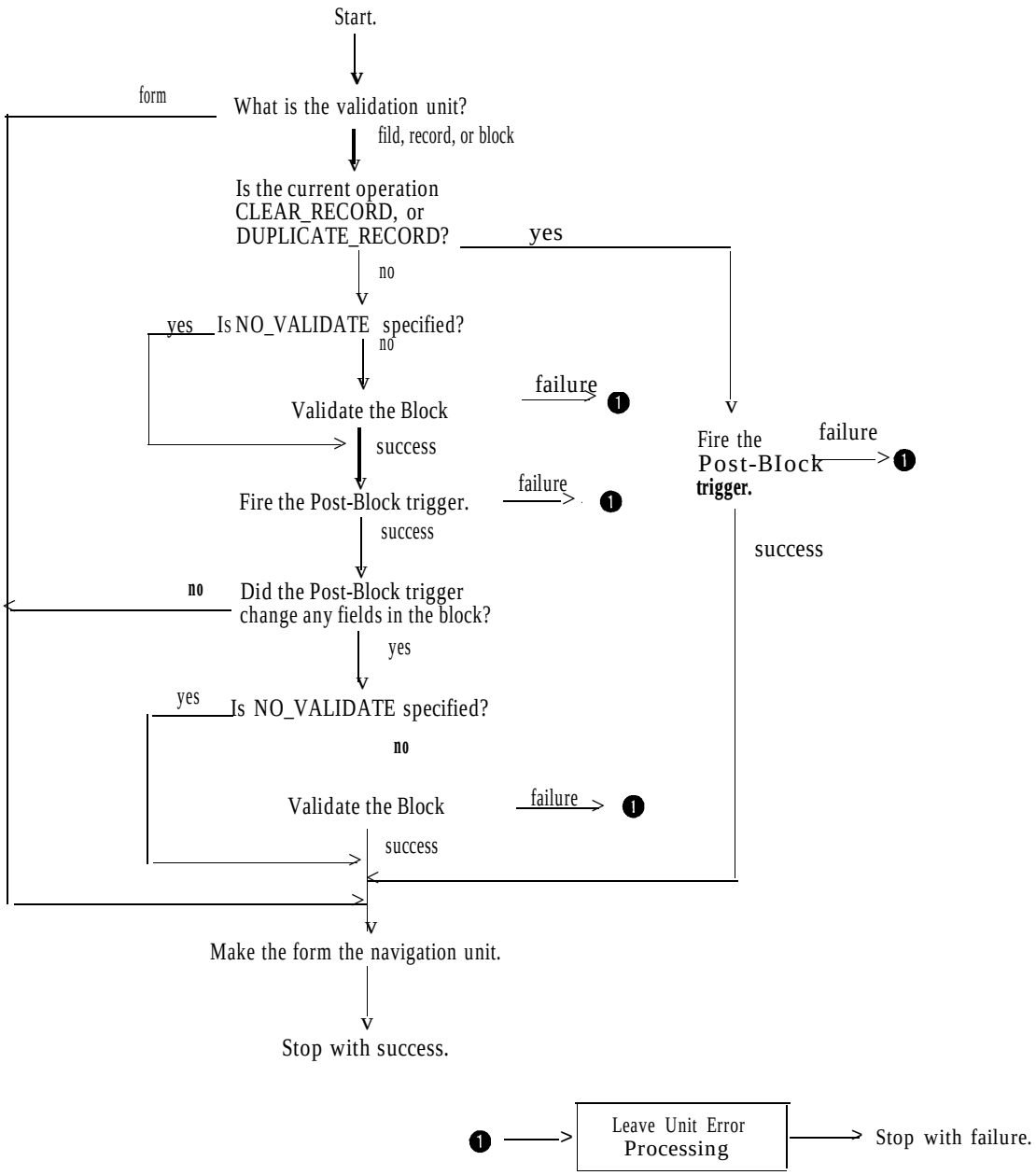
Related Topics: HOST, packaged procedure 16-22

LAST_RECORD



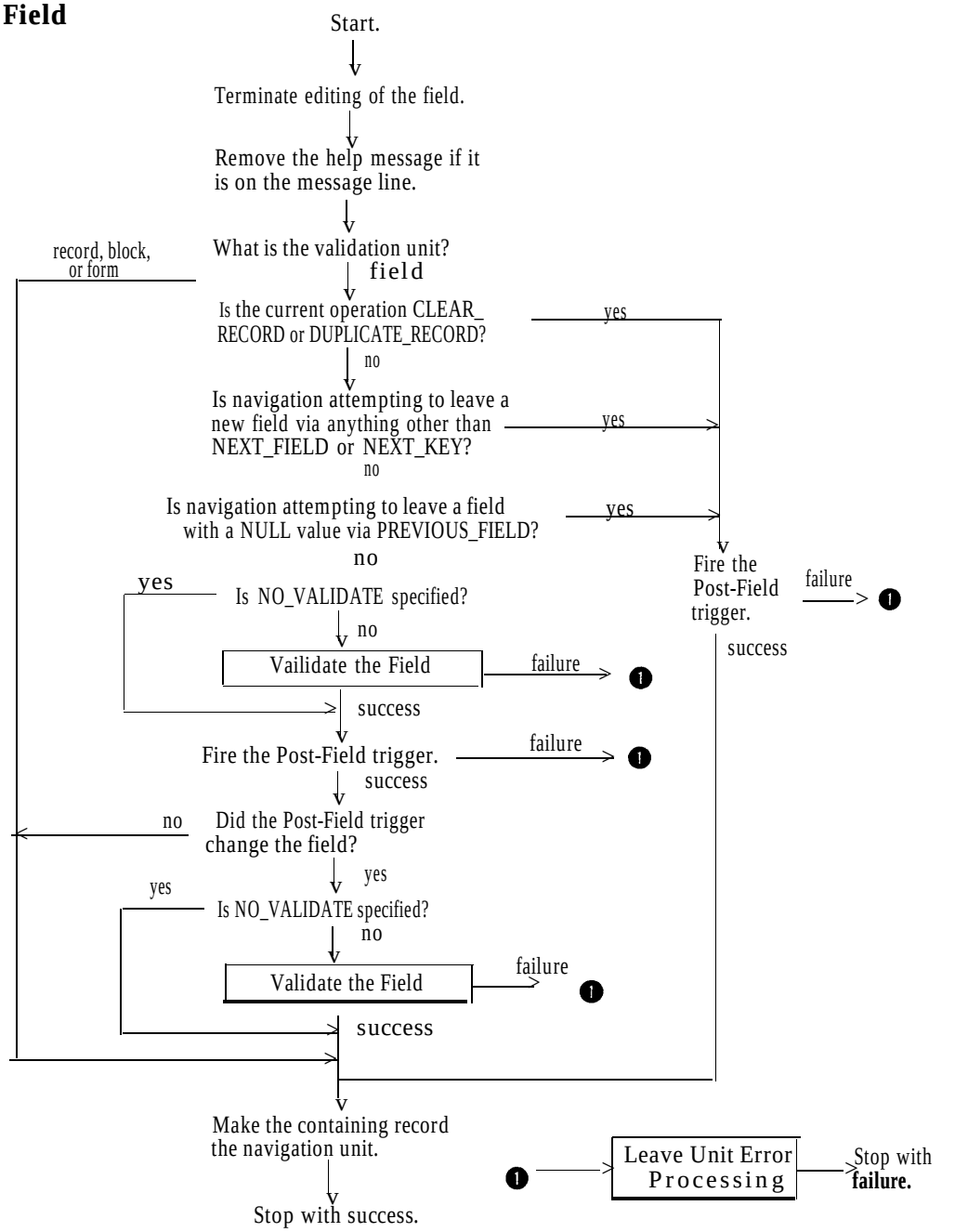
Related Topics: LAST_RECORD, packaged procedure 16-22

Leave the Block



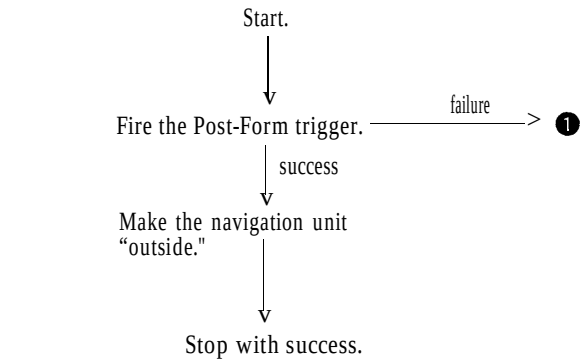
Related Topics: Navigation 3-2

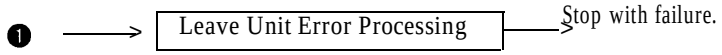
Leave the Field



Related Topics: Navigation 3-2

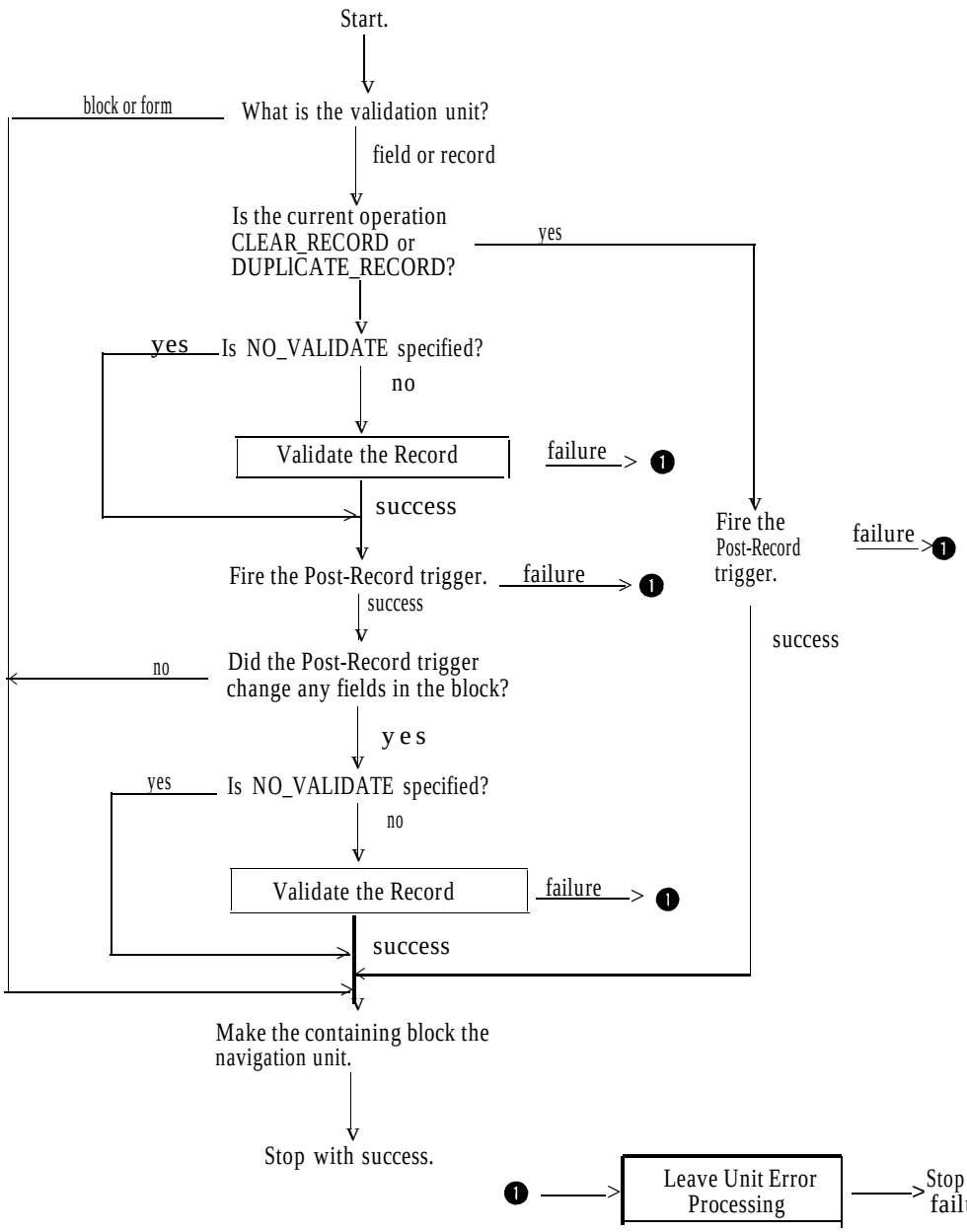
Leave the Form





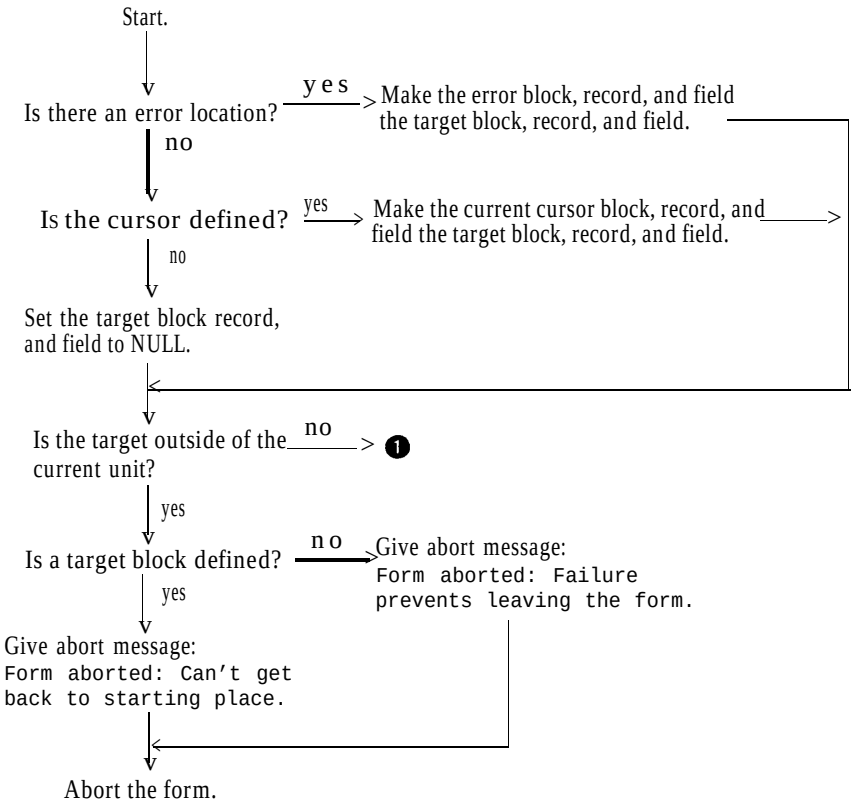
Related Topics: Navigation 3-2

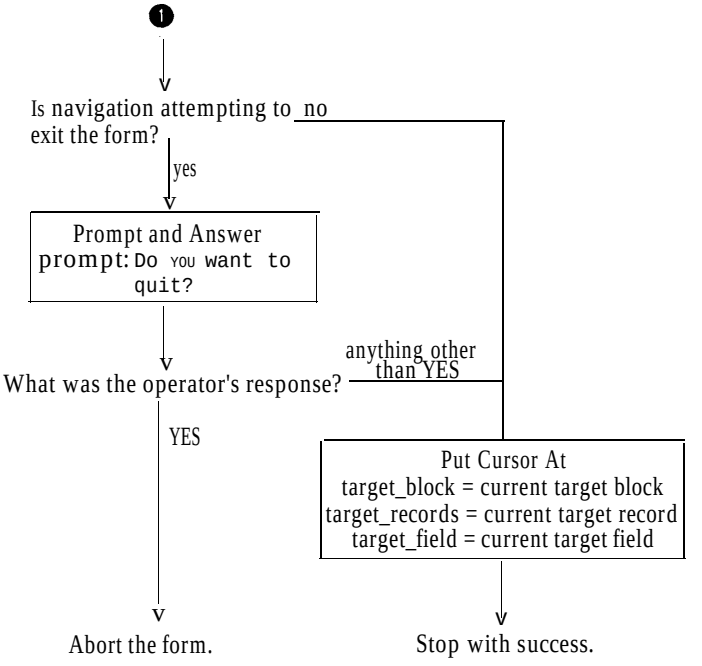
Leave the Record



Related Topics: Navigation 3-2

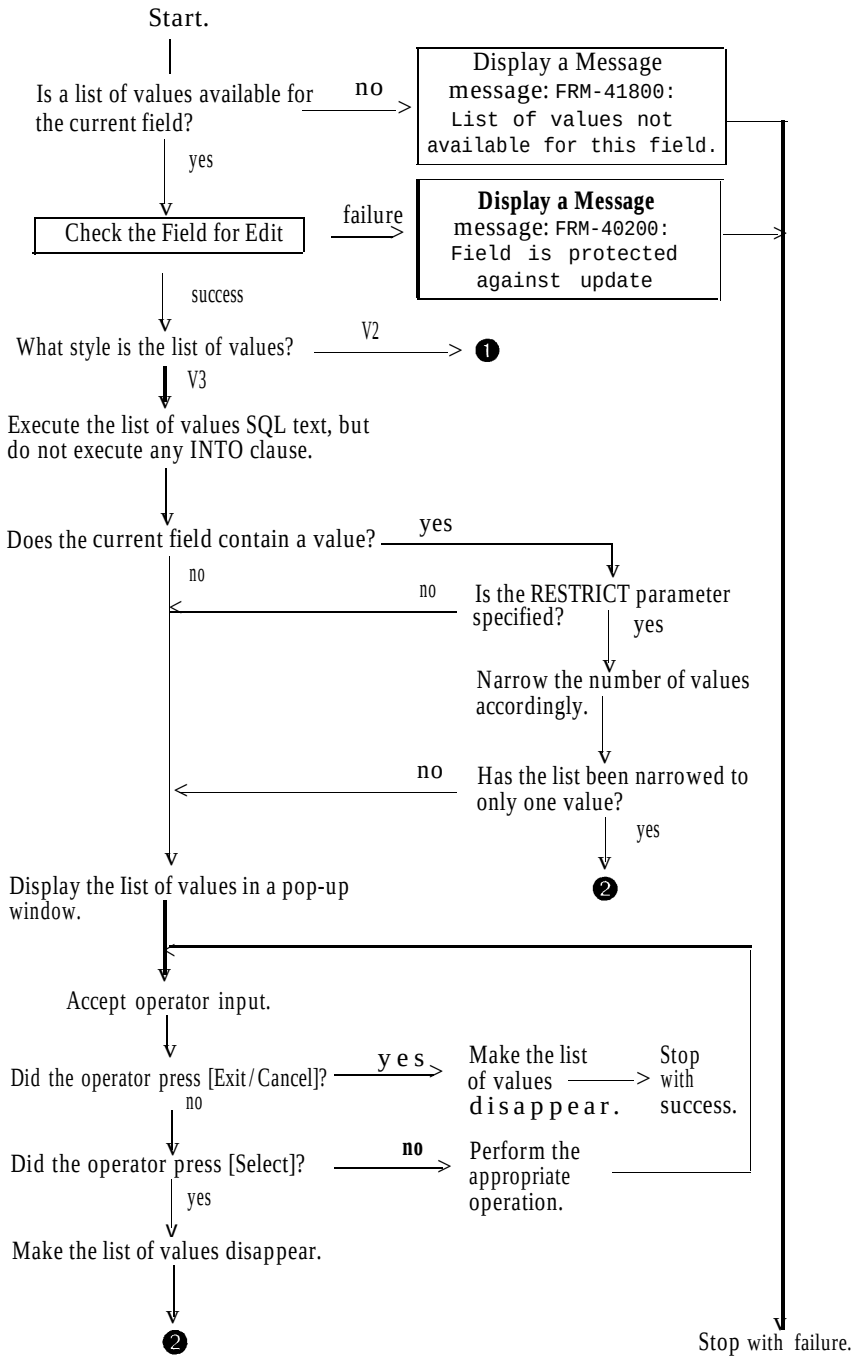
Leave Unit Error Processing

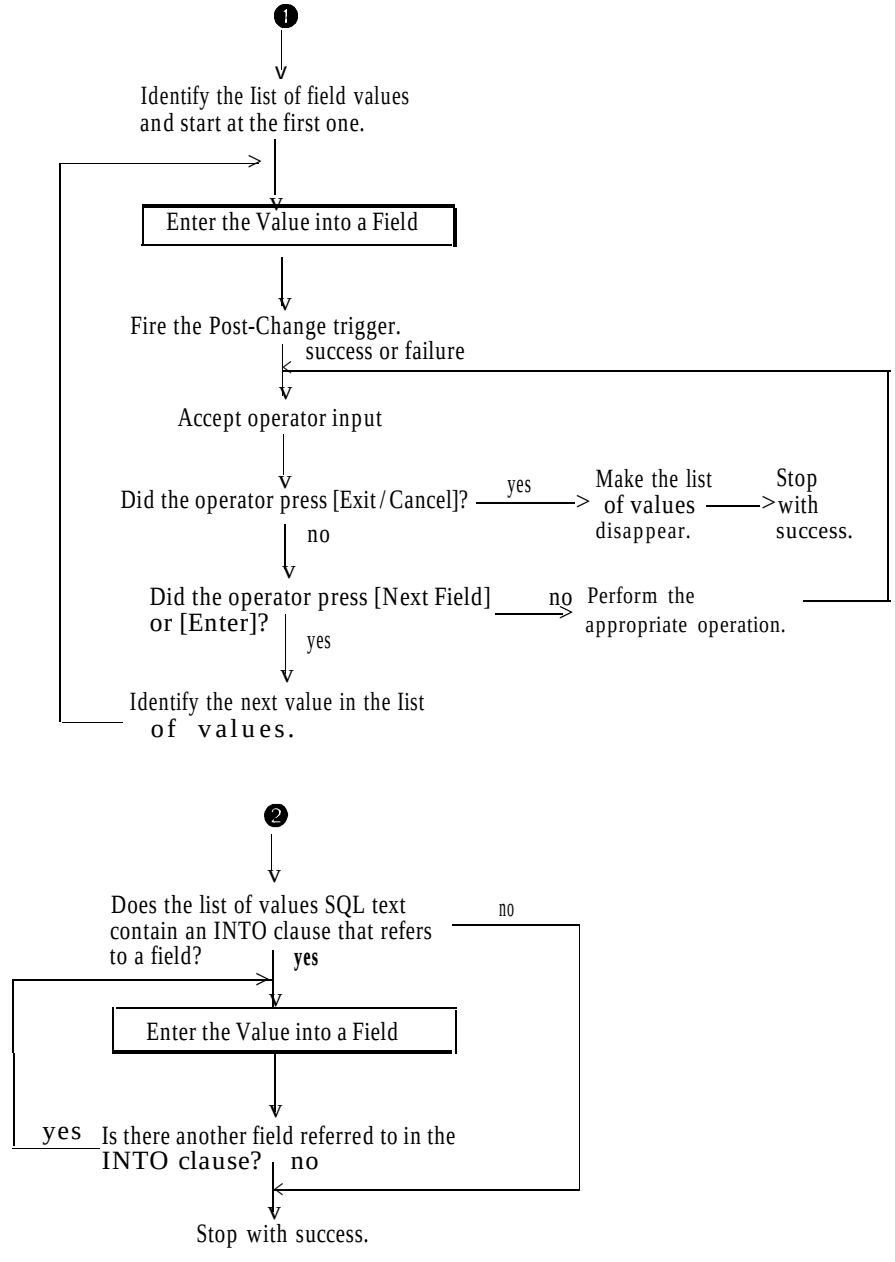




Related Topics: Navigation 3-2

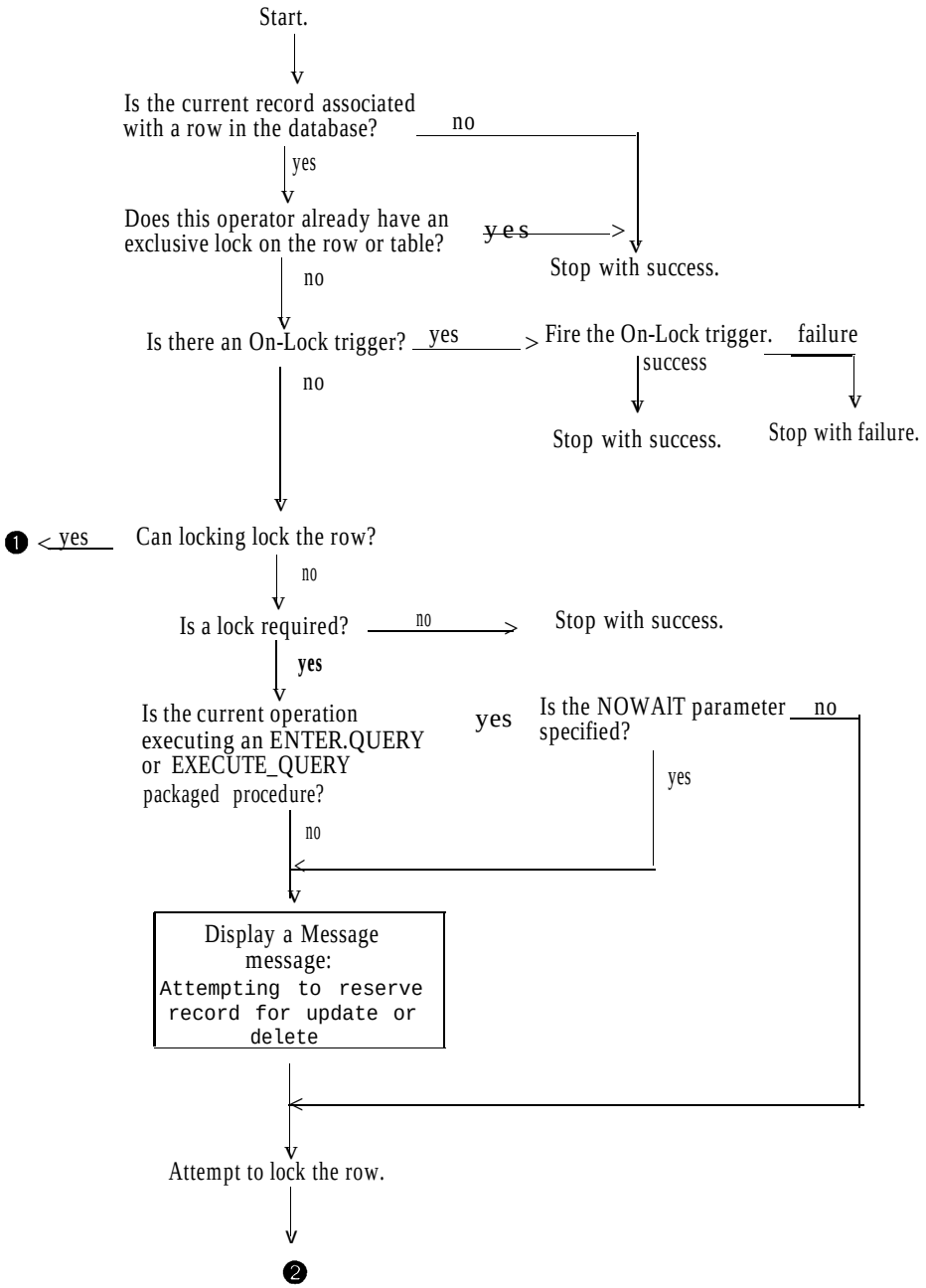
LIST_VALUES

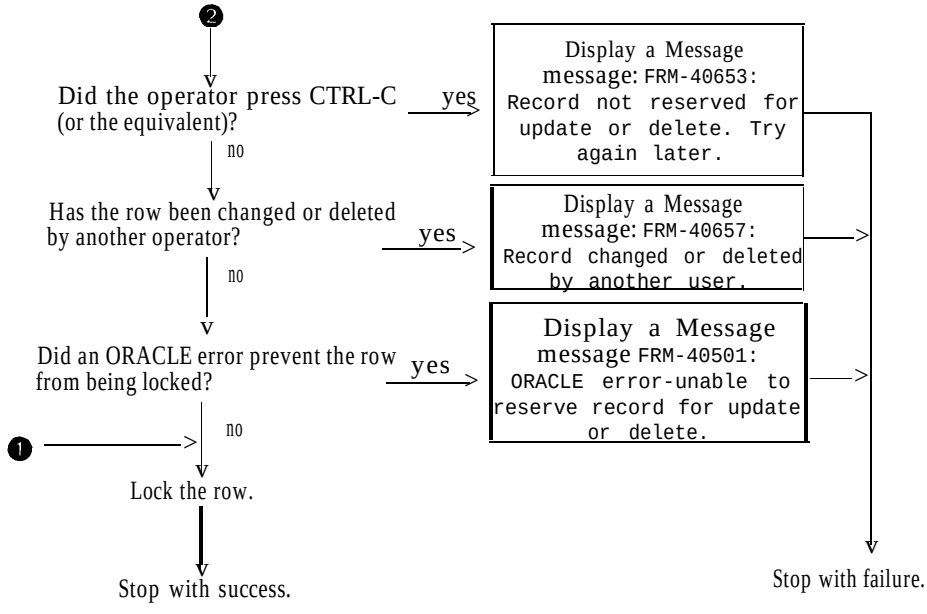

Start.
Is a list of values available for the current field?
no
Display a Message message: FRM-41800: List of values not available for this field.
yes
Check the Field for Edit
failure
Display a Message message: FRM-40200: Field is protected against update
success
What style is the list of values?
V2
1
V3
Execute the list of values SQL text, but do not execute any INTO clause.
Does the current field contain a value?
yes
no
no
Is the RESTRICT parameter specified?
yes
Narrow the number of values accordingly.
no
Has the list been narrowed to only one value?
yes
2
Display the list of values in a pop-up window.
Accept operator input.
Did the operator press [Exit/Cancel]?
yes
Make the list of values disappear.
Stop with success.
no
Did the operator press [Select]?
no
Perform the appropriate operation.
yes
Make the list of values disappear.
2
Stop with failure.



Related Topics: LIST_VALUES, packaged procedure 16-22

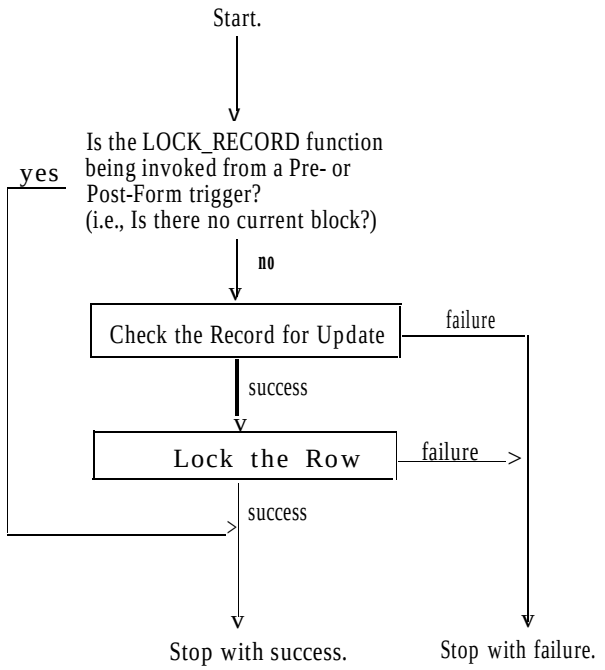
Lock the Row


Start.
Is the current record associated with a row in the database?
no
yes
Does this operator already have an exclusive lock on the row or table?
yes
Stop with success.
no
Is there an On-Lock trigger?
yes
Fire the On-Lock trigger.
failure
success
Stop with success.
Stop with failure.
no
1
yes
Can locking lock the row?
no
Is a lock required?
no
Stop with success.
yes
Is the current operation executing an ENTER.QUERY or EXECUTE_QUERY packaged procedure?
yes
Is the NOWAIT parameter specified?
no
yes
no
Display a Message message:
Attempting to reserve record for update or delete
Attempt to lock the row.
2



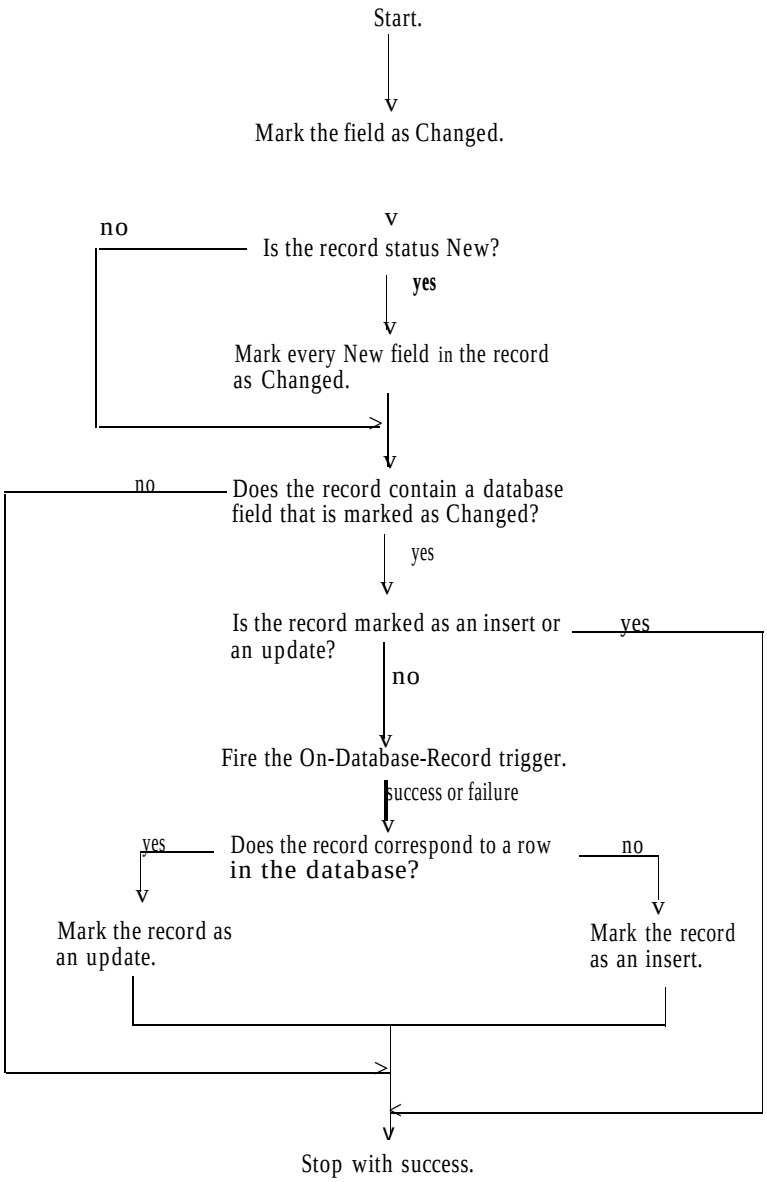
Related Topics: Locking 3-11

LOCK_RECORD

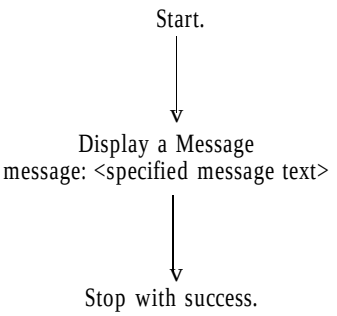


Related Topics: LOCK_RECORD, packaged procedure 16-23

Mark Fields and Records as Changed

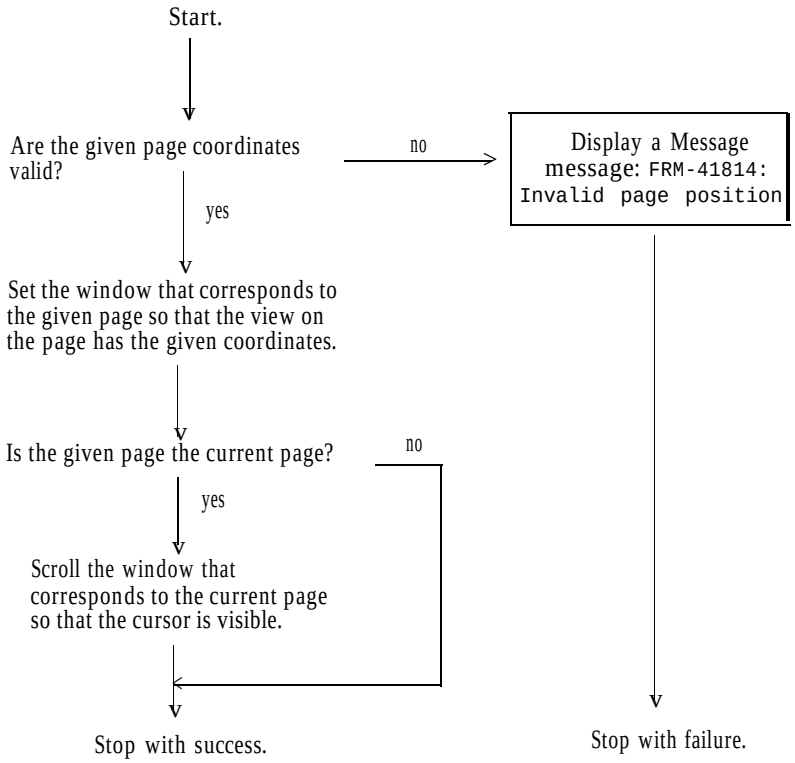


MESSAGE



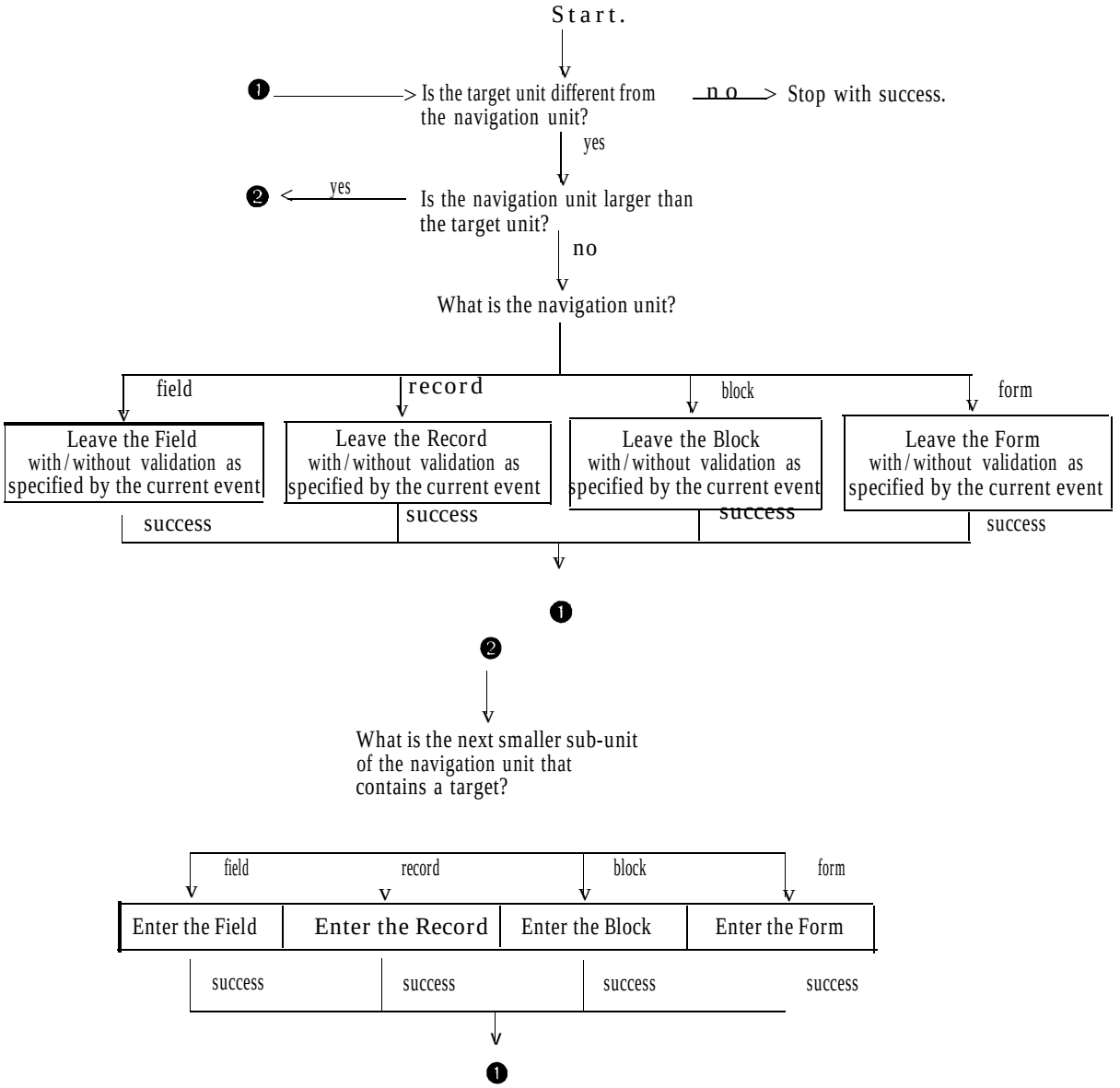
Related Topics: MESSAGE, packaged procedure 16-23

MOVE_VIEW



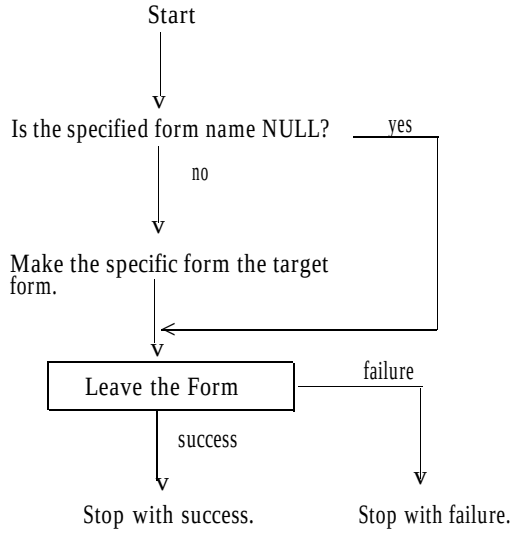
Related Topics: MOVE_VIEW, packaged procedure 16-23

Navigate to the <Navigation Unit> Level



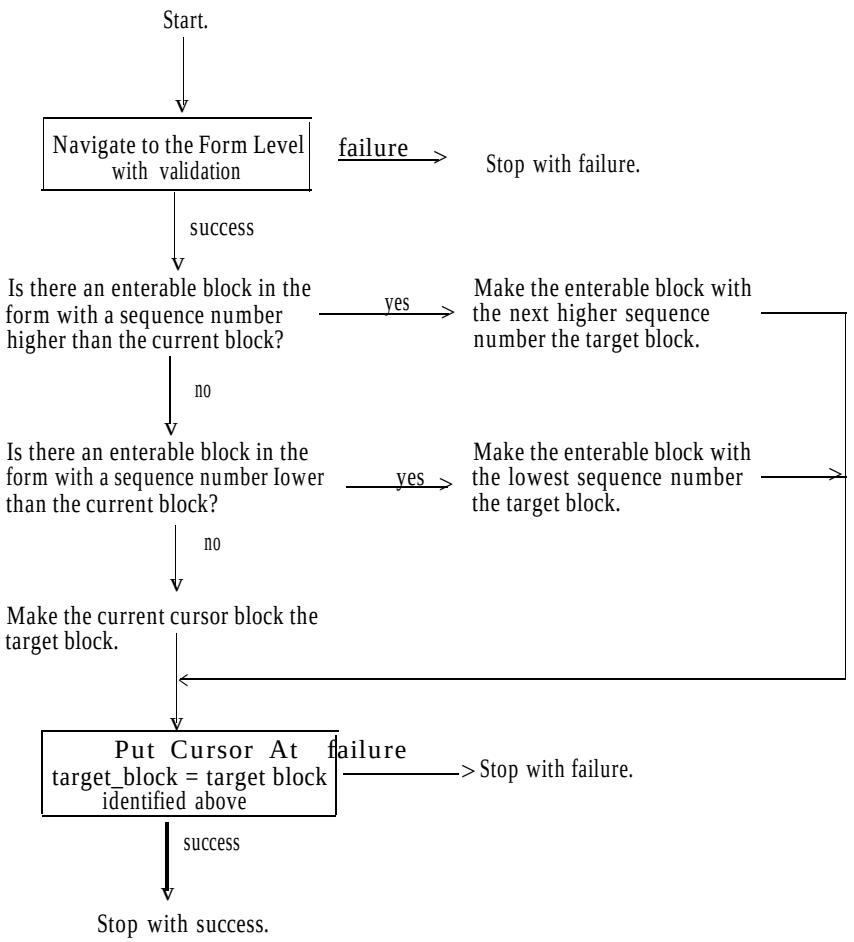
Related Topics: Navigation 3-2

NEW_FORM



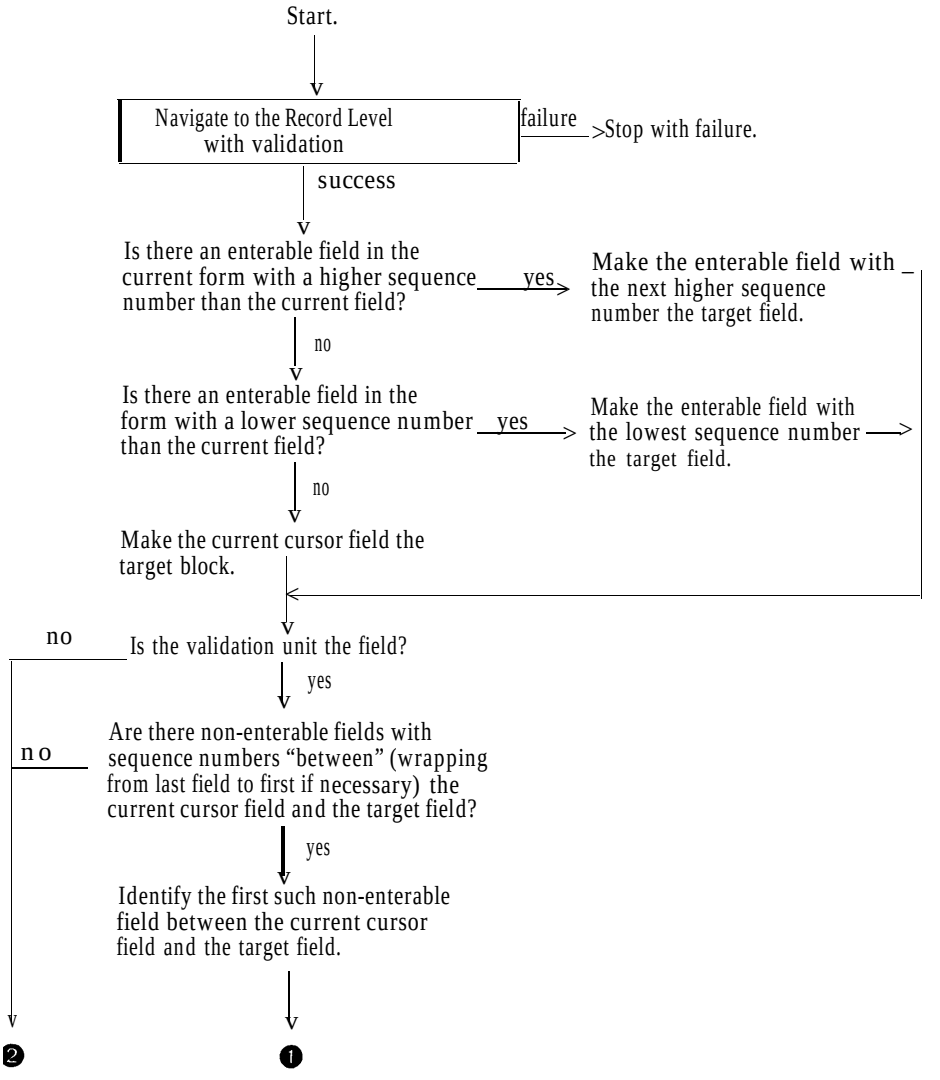
Related Topics: NEW_FORM, packaged procedure 16-24

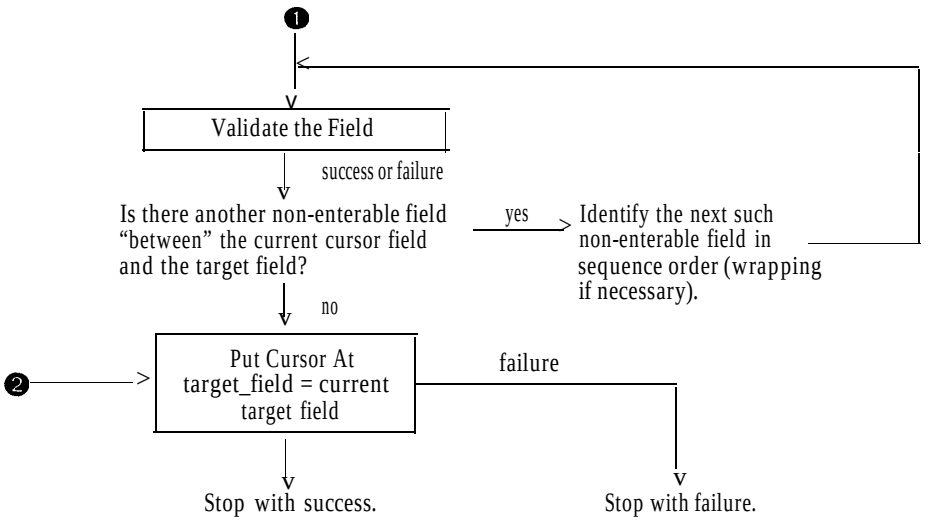
NEXT_BLOCK



Related Topics: NEXT_BLOCK, packaged procedure 16-24

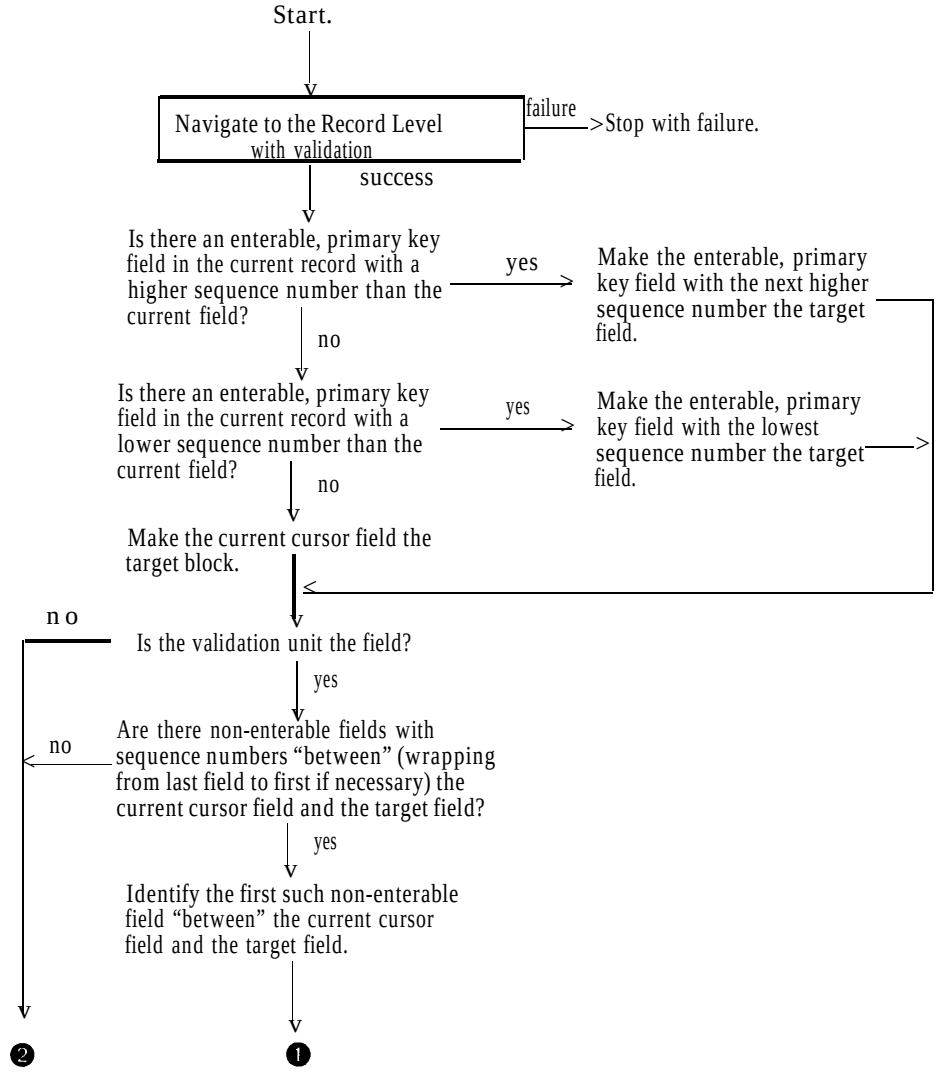
NEXT_FIELD

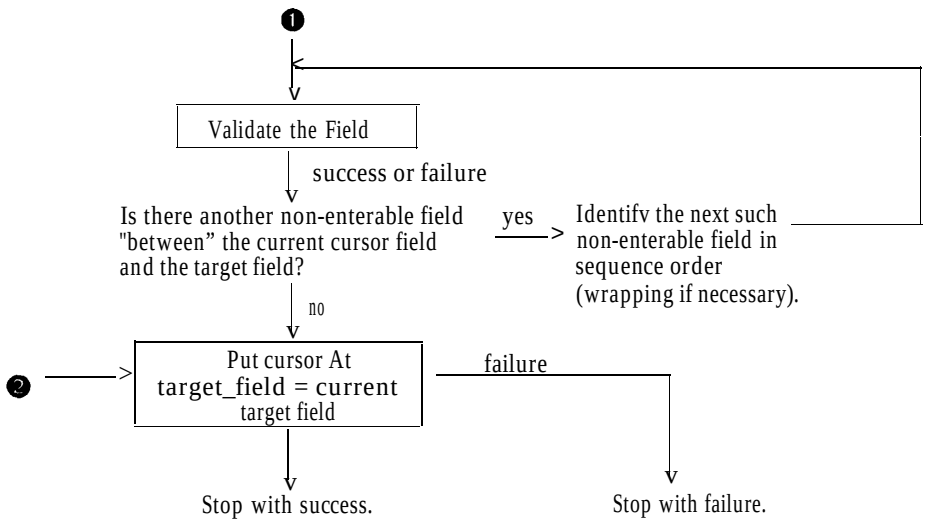

Start.
Navigate to the Record Level with validation
failure
Stop with failure.
success
Is there an enterable field in the current form with a higher sequence number than the current field?
yes
Make the enterable field with the next higher sequence number the target field.
no
Is there an enterable field in the form with a lower sequence number than the current field?
yes
Make the enterable field with the lowest sequence number the target field.
no
Make the current cursor field the target block.
no
Is the validation unit the field?
yes
no
Are there non-enterable fields with sequence numbers "between" (wrapping from last field to first if necessary) the current cursor field and the target field?
yes
Identify the first such non-enterable field between the current cursor field and the target field.
2
1



Related Topics: NEXT_FIELD, packaged procedure 16-24

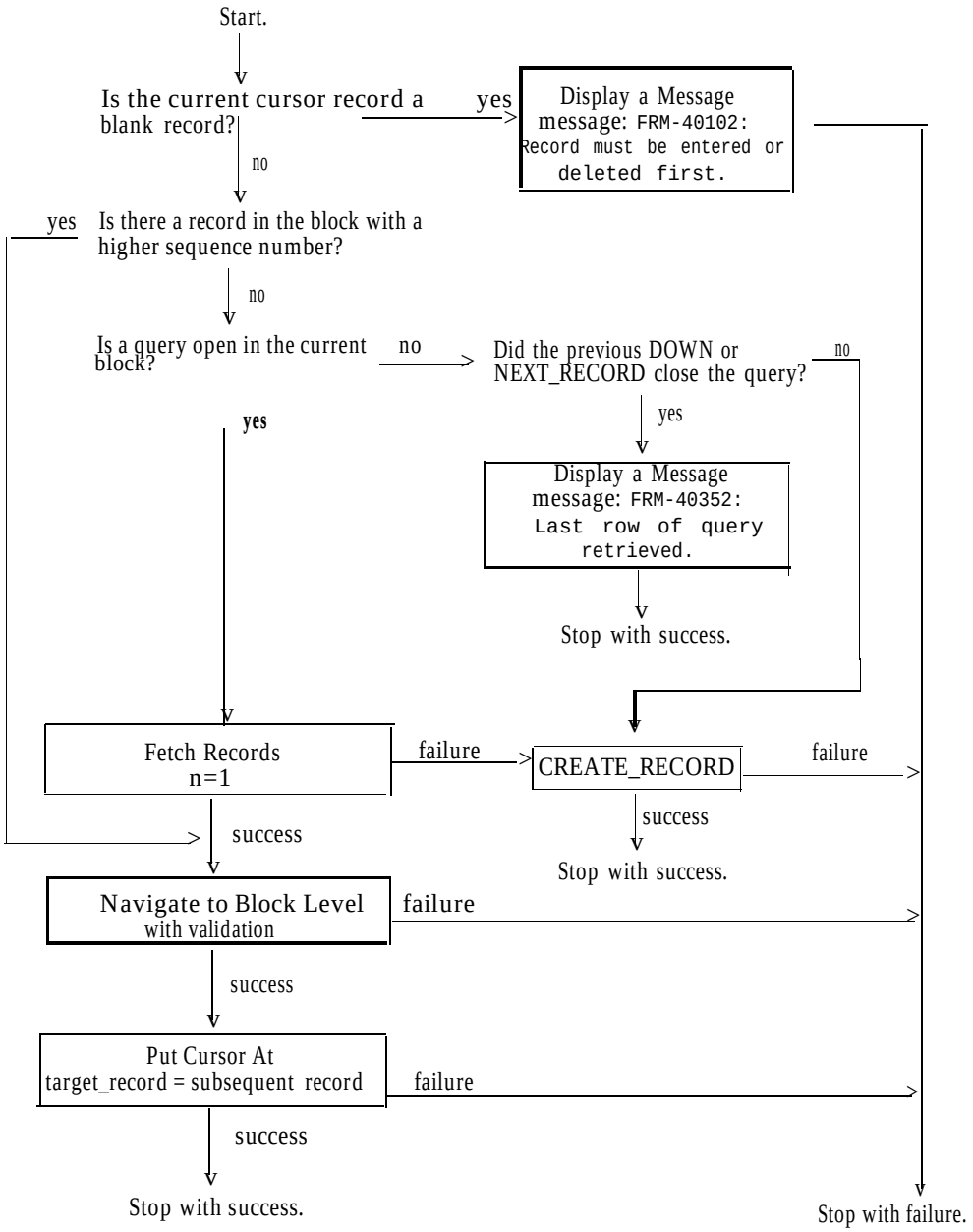
NEXT_KEY


Start.
Navigate to the Record Level with validation
failure
Stop with failure.
success
Is there an enterable, primary key field in the current record with a higher sequence number than the current field?
yes
Make the enterable, primary key field with the next higher sequence number the target field.
no
Is there an enterable, primary key field in the current record with a lower sequence number than the current field?
yes
Make the enterable, primary key field with the lowest sequence number the target field.
no
Make the current cursor field the target block.
no
Is the validation unit the field?
yes
no
Are there non-enterable fields with sequence numbers "between" (wrapping from last field to first if necessary) the current cursor field and the target field?
yes
Identify the first such non-enterable field "between" the current cursor field and the target field.
2
1



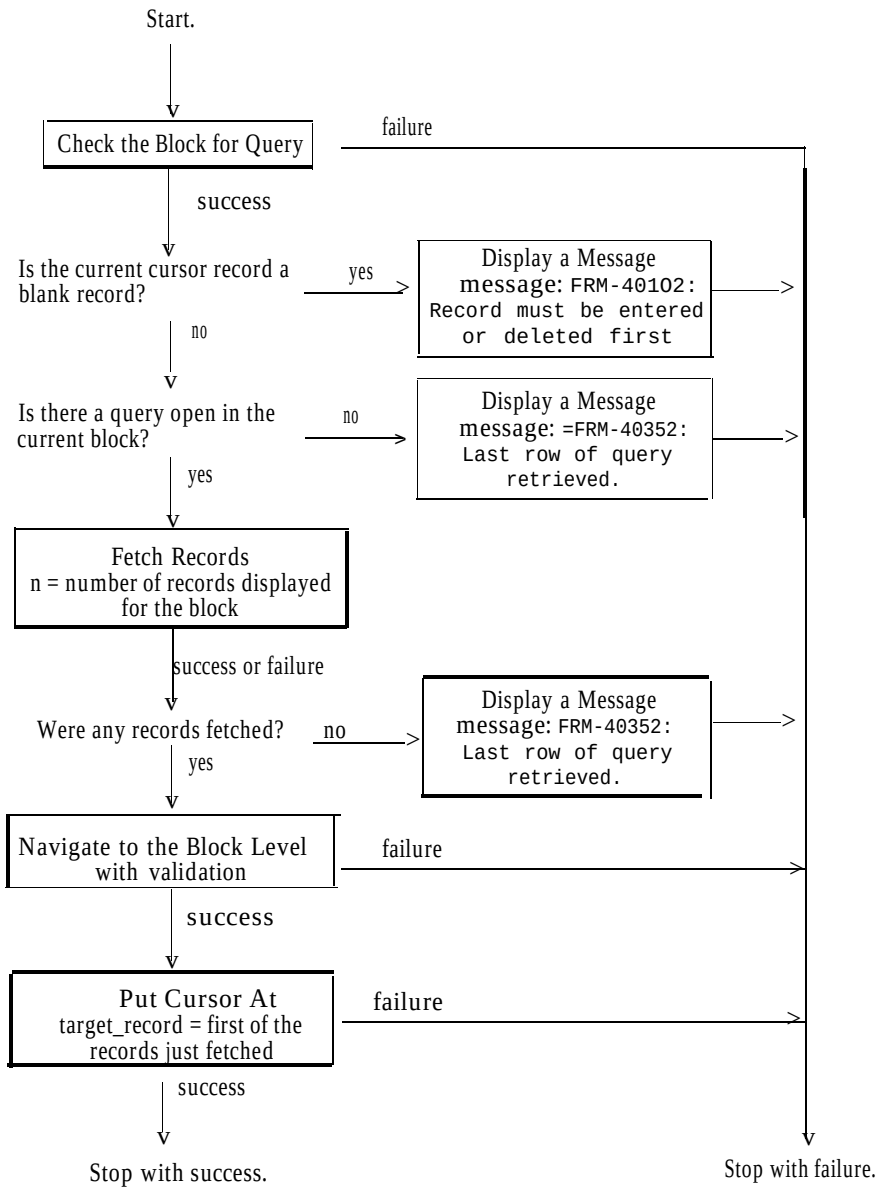
Related Topics: NEXT_KEY, packaged procedure 16-25

NEXT_RECORD



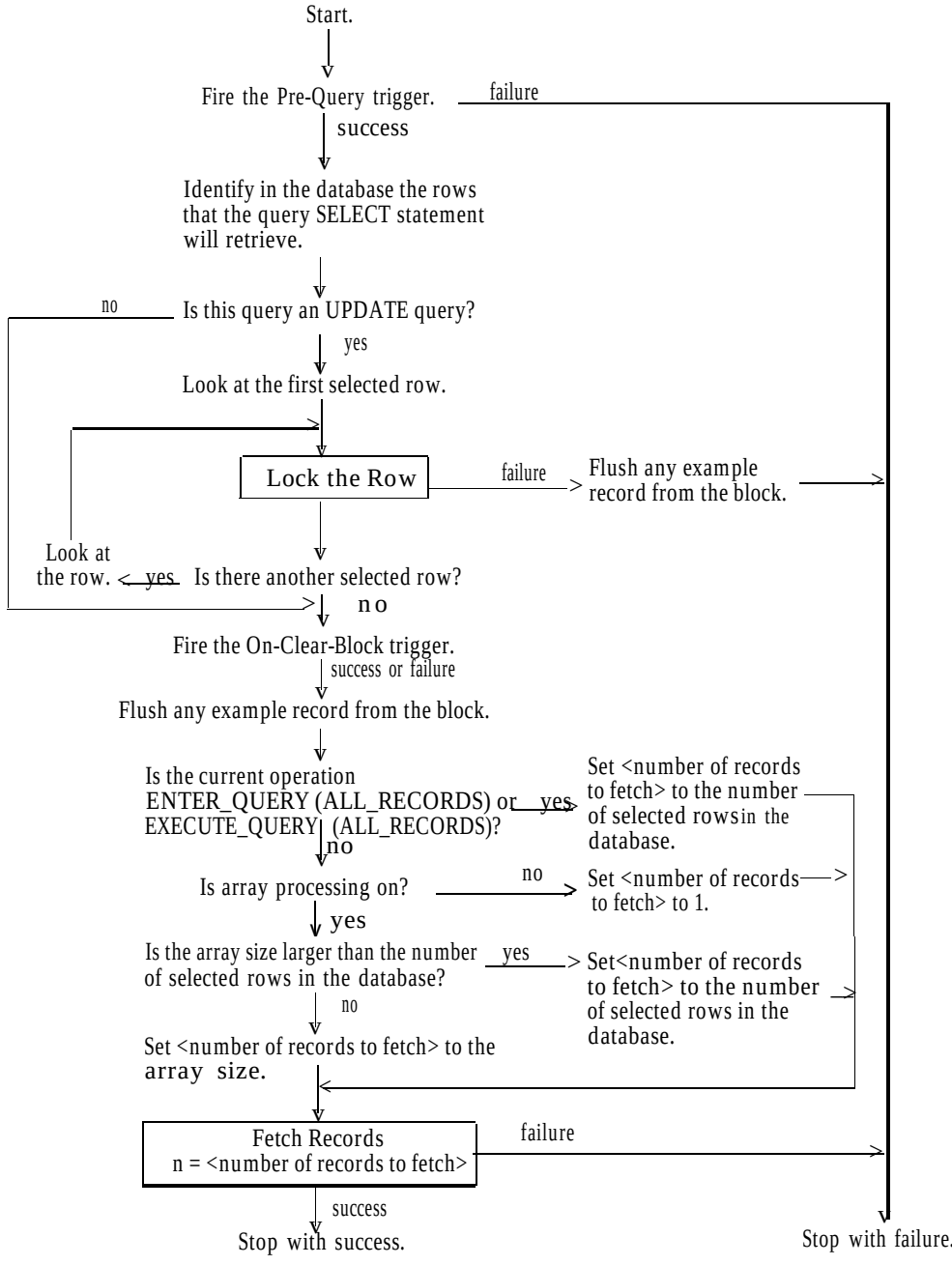
Related Topics: NEXT_RECORD, packaged procedure 16-25

NEXT_SET



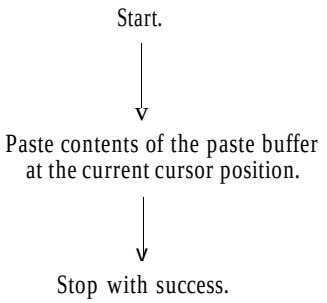
Related Topics: NEXT_SET, packaged procedure 16-25

Open the Query



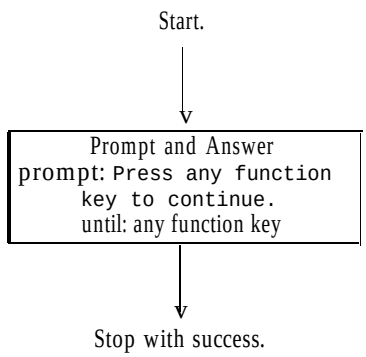
Related Topics: Query processing 3-24

PASTE_REGION



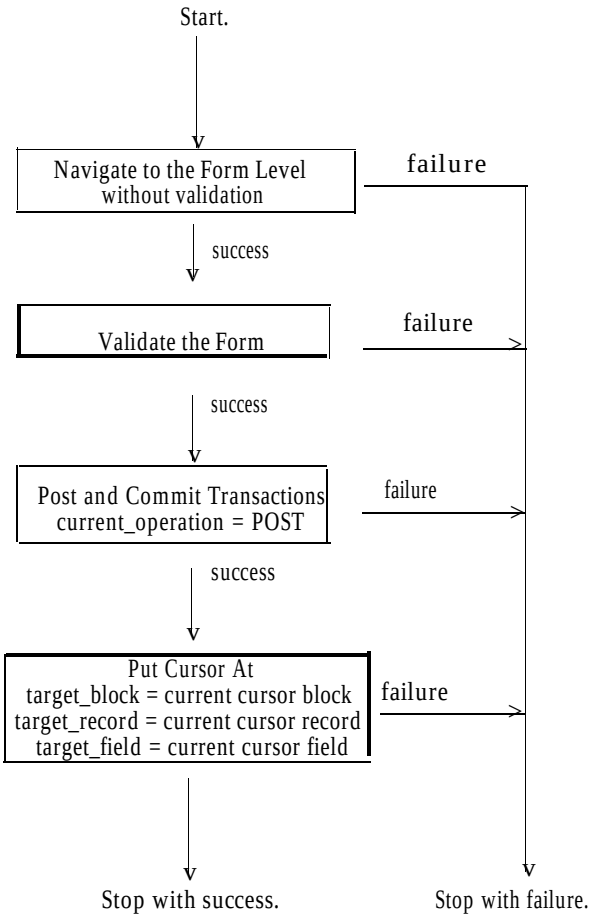
Related Topics: PASTE_REGION, packaged procedure 16-25

PAUSE



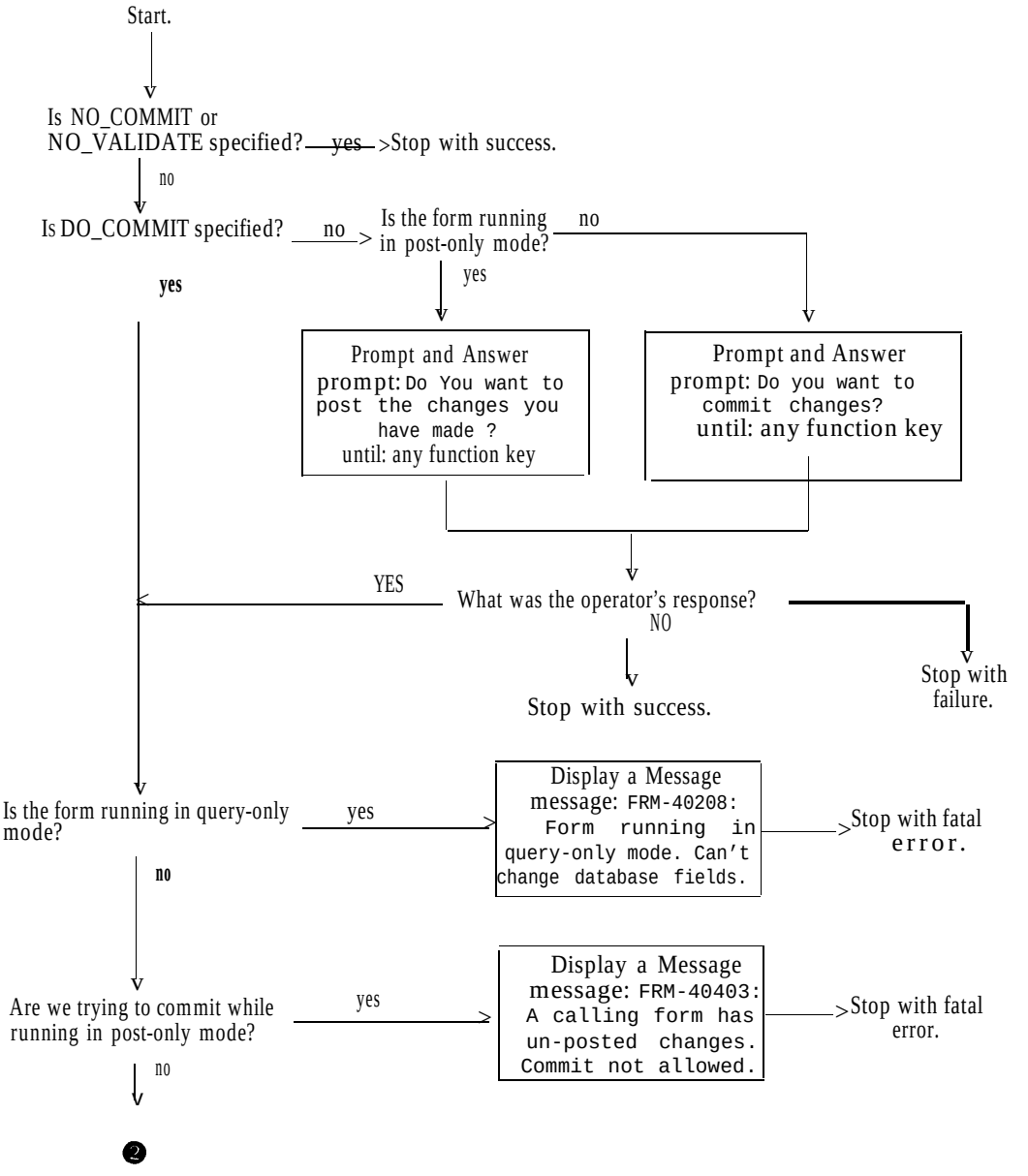
Related Topics: PAUSE, packaged procedure 16-26

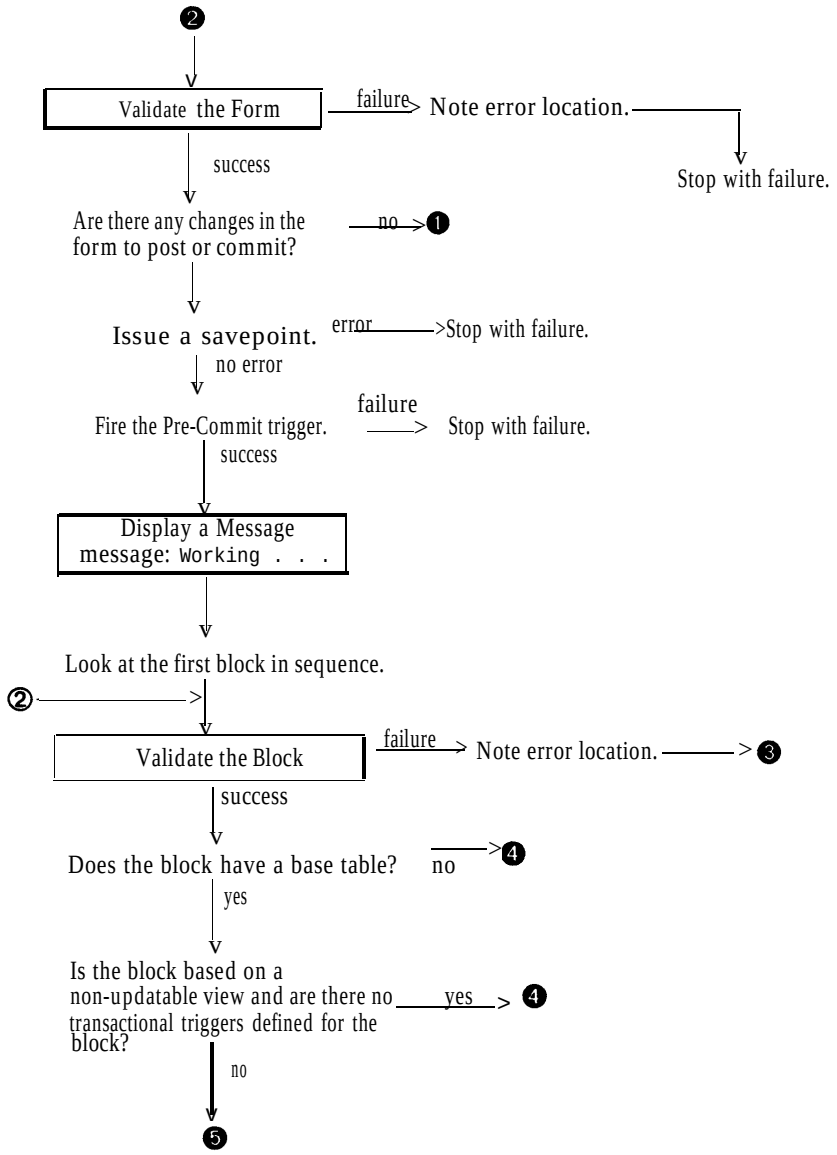
POST

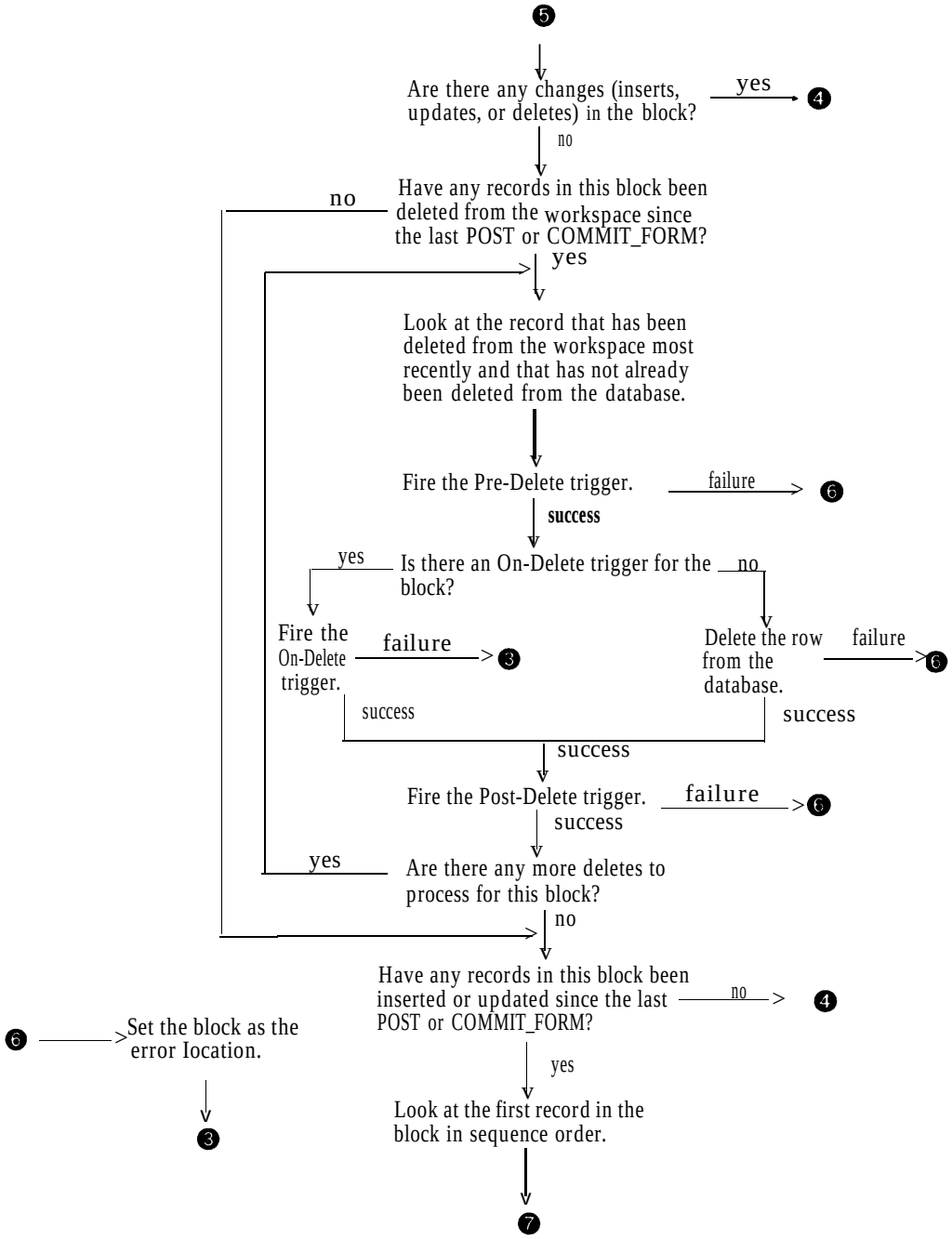


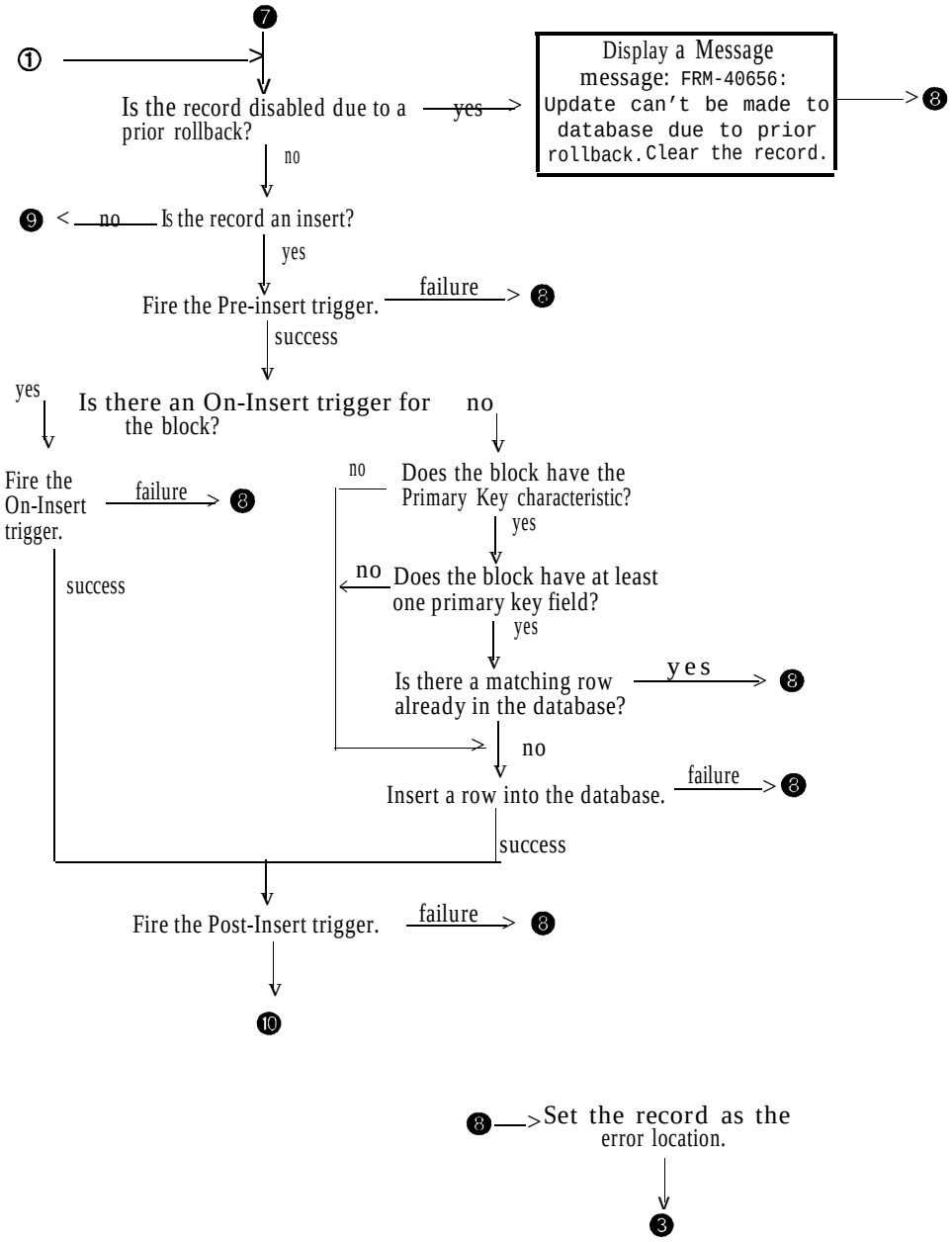
Related Topics: POST, packaged procedure 16-26

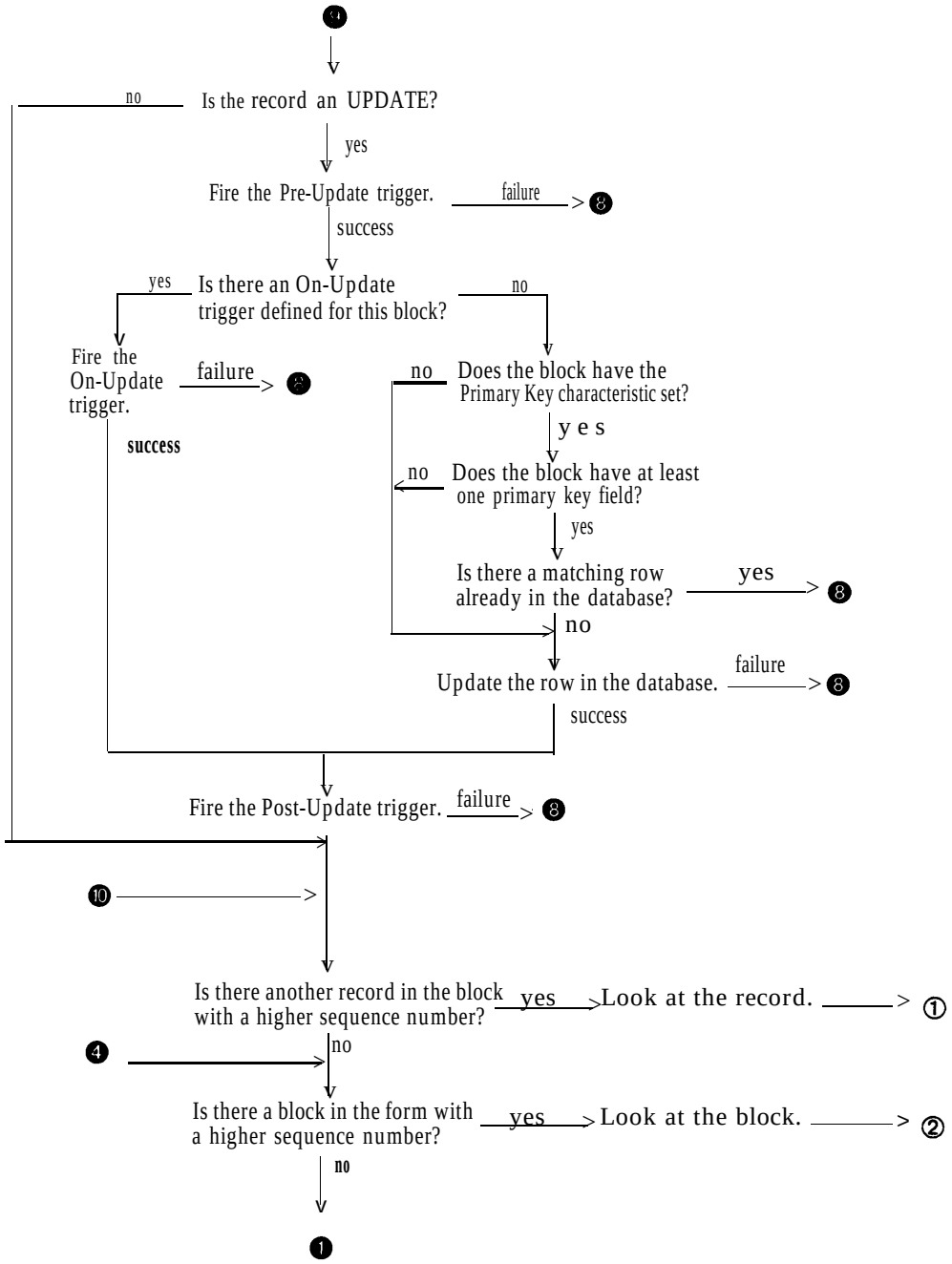
Post and Commit Transactions

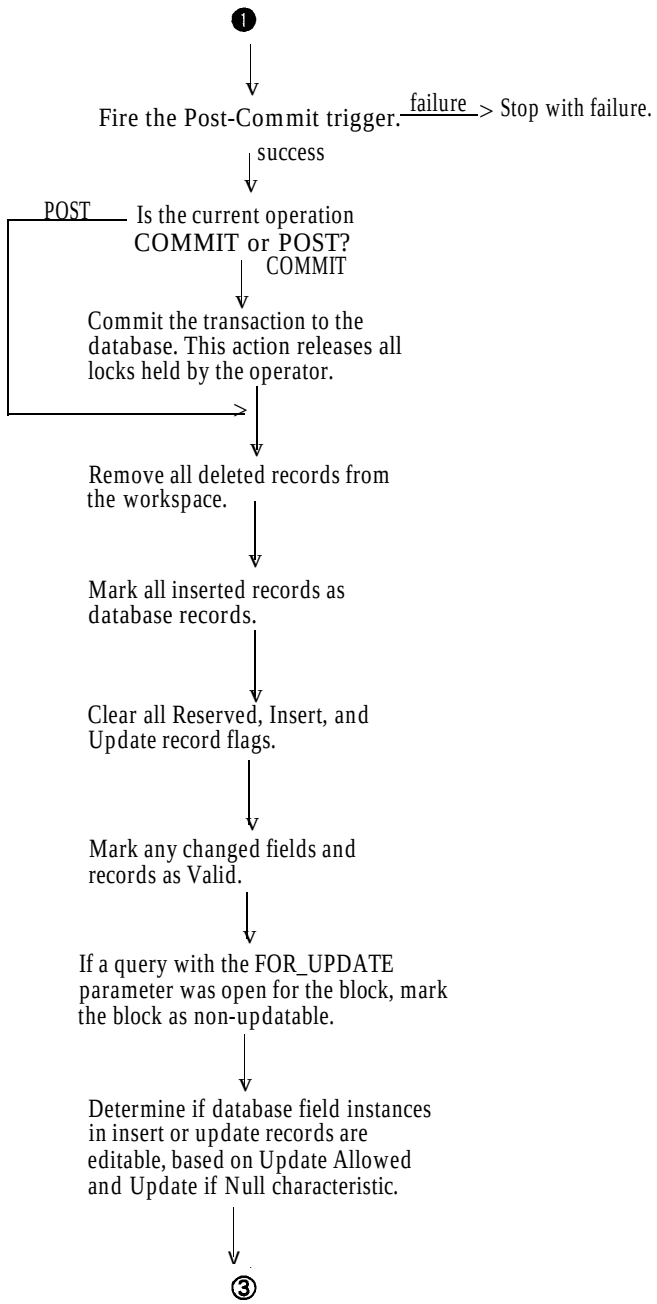


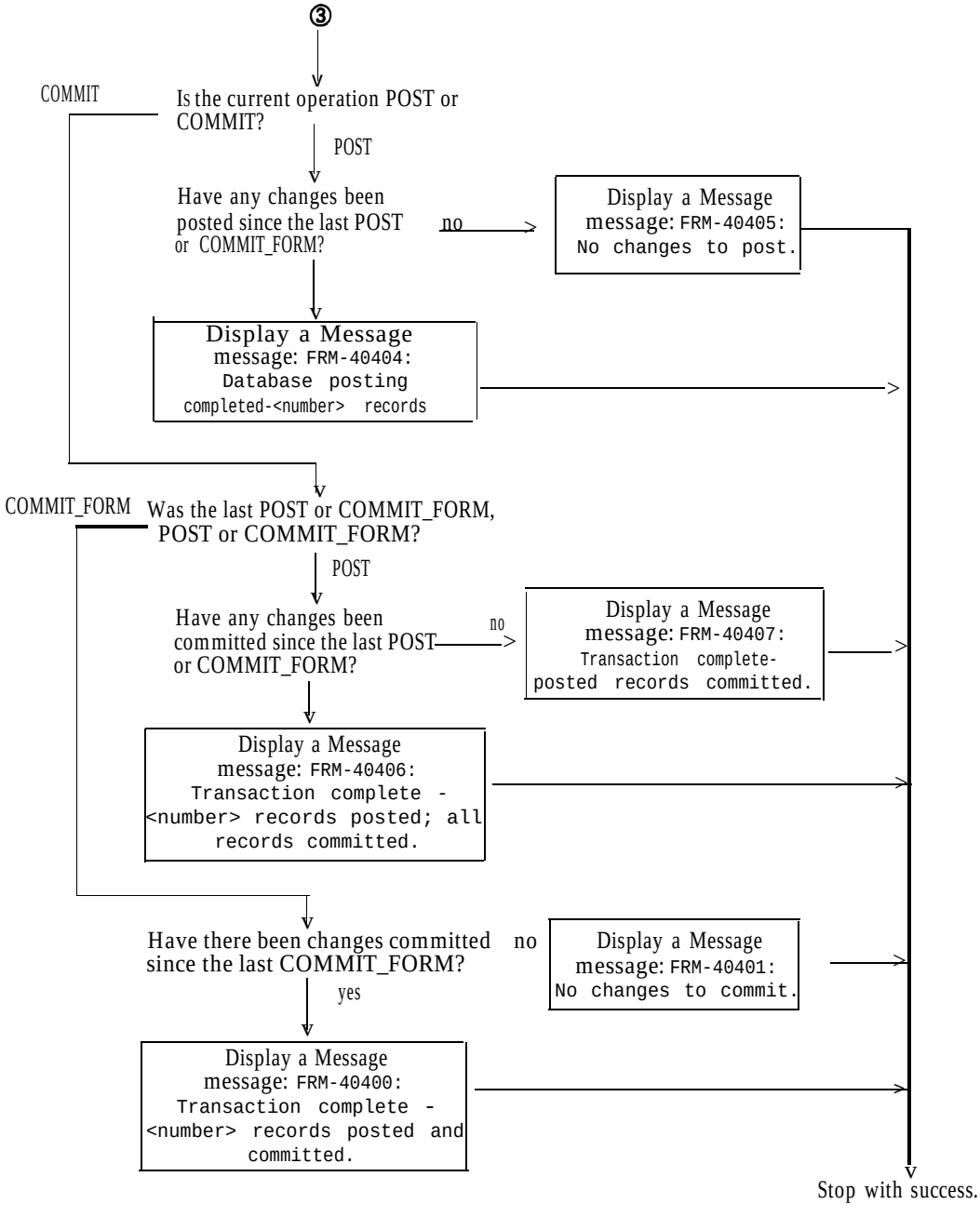

2
Validate the Form
failure
Note error location.
Stop with failure.
success
Are there any changes in the form to post or commit?
no
1
Issue a savepoint.
error
Stop with failure.
no error
Fire the Pre-Commit trigger.
failure
Stop with failure.
success
Display a Message
message: Working . . .
Look at the first block in sequence.
2
Validate the Block
failure
Note error location.
3
success
Does the block have a base table?
no
4
yes
Is the block based on a non-updatable view and are there no transactional triggers defined for the block?
yes
4
no
5

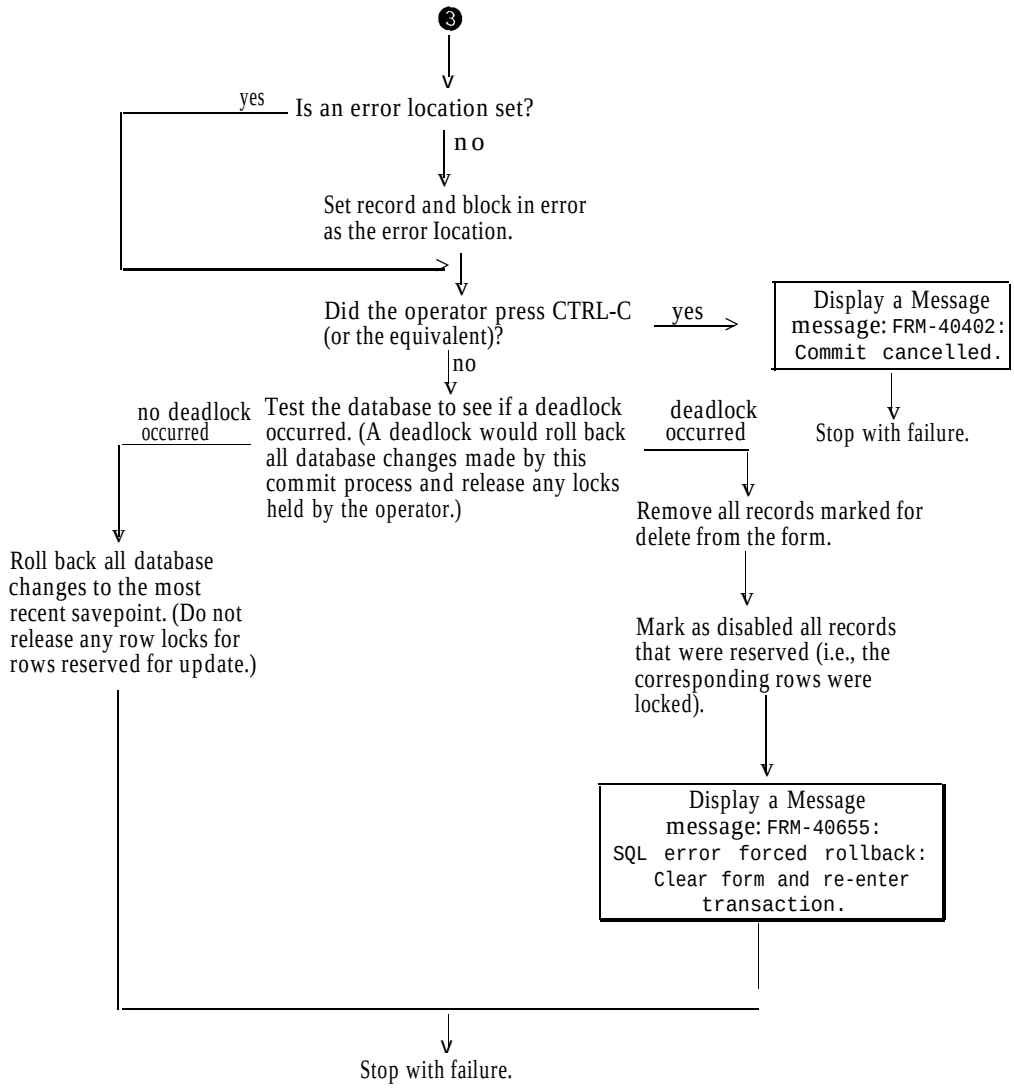

5
Are there any changes (inserts, updates, or deletes) in the block?
yes
4
no
Have any records in this block been deleted from the workspace since the last POST or COMMIT_FORM?
no
yes
Look at the record that has been deleted from the workspace most recently and that has not already been deleted from the database.
Fire the Pre-Delete trigger.
failure
6
success
Is there an On-Delete trigger for the block?
yes
no
Fire the On-Delete trigger.
failure
3
Delete the row from the database.
failure
6
success
success
success
Fire the Post-Delete trigger.
failure
6
success
Are there any more deletes to process for this block?
yes
no
Have any records in this block been inserted or updated since the last POST or COMMIT_FORM?
no
4
yes
Look at the first record in the block in sequence order.
7
6
Set the block as the error location.
3


7
1
Is the record disabled due to a prior rollback?
yes
Display a Message
message: FRM-40656: Update can't be made to database due to prior rollback.Clear the record.
8
no
9
no
Is the record an insert?
yes
Fire the Pre-insert trigger.
failure
8
success
yes
Is there an On-Insert trigger for the block?
no
Fire the On-Insert trigger.
failure
8
success
no
Does the block have the Primary Key characteristic?
yes
no
Does the block have at least one primary key field?
yes
Is there a matching row already in the database?
yes
8
no
Insert a row into the database.
failure
8
success
Fire the Post-Insert trigger.
failure
8
10
8
Set the record as the error location.
3


Is the record an UPDATE?
no
yes
Fire the Pre-Update trigger.
failure
success
Is there an On-Update trigger defined for this block?
yes
no
Fire the On-Update trigger.
failure
success
Does the block have the Primary Key characteristic set?
no
yes
Does the block have at least one primary key field?
no
yes
Is there a matching row already in the database?
yes
no
Update the row in the database.
failure
success
Fire the Post-Update trigger.
failure
Is there another record in the block with a higher sequence number?
yes
Look at the record.
no
Is there a block in the form with a higher sequence number?
yes
Look at the block.
no

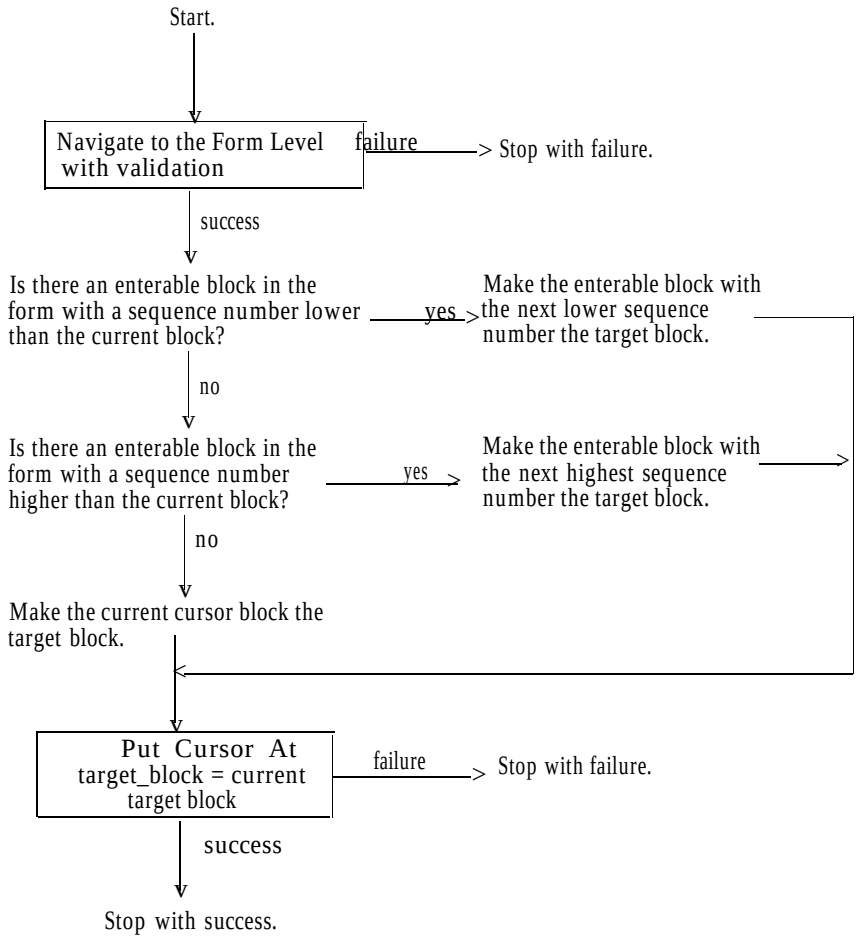

1
Fire the Post-Commit trigger.
failure
Stop with failure.
success
POST
Is the current operation COMMIT or POST?
COMMIT
Commit the transaction to the database. This action releases all locks held by the operator.
Remove all deleted records from the workspace.
Mark all inserted records as database records.
Clear all Reserved, Insert, and Update record flags.
Mark any changed fields and records as Valid.
If a query with the FOR_UPDATE parameter was open for the block, mark the block as non-updatable.
Determine if database field instances in insert or update records are editable, based on Update Allowed and Update if Null characteristic.
3


③
COMMIT
Is the current operation POST or COMMIT?
POST
Have any changes been posted since the last POST or COMMIT_FORM?
no
Display a Message message: FRM-40405: No changes to post.
Display a Message message: FRM-40404: Database posting completed-<number> records
COMMIT_FORM
Was the last POST or COMMIT_FORM, POST or COMMIT_FORM?
POST
Have any changes been committed since the last POST or COMMIT_FORM?
no
Display a Message message: FRM-40407: Transaction complete- posted records committed.
Display a Message message: FRM-40406: Transaction complete - <number> records posted; all records committed.
Have there been changes committed since the last COMMIT_FORM?
no
Display a Message message: FRM-40401: No changes to commit.
yes
Display a Message message: FRM-40400: Transaction complete - <number> records posted and committed.
Stop with success.



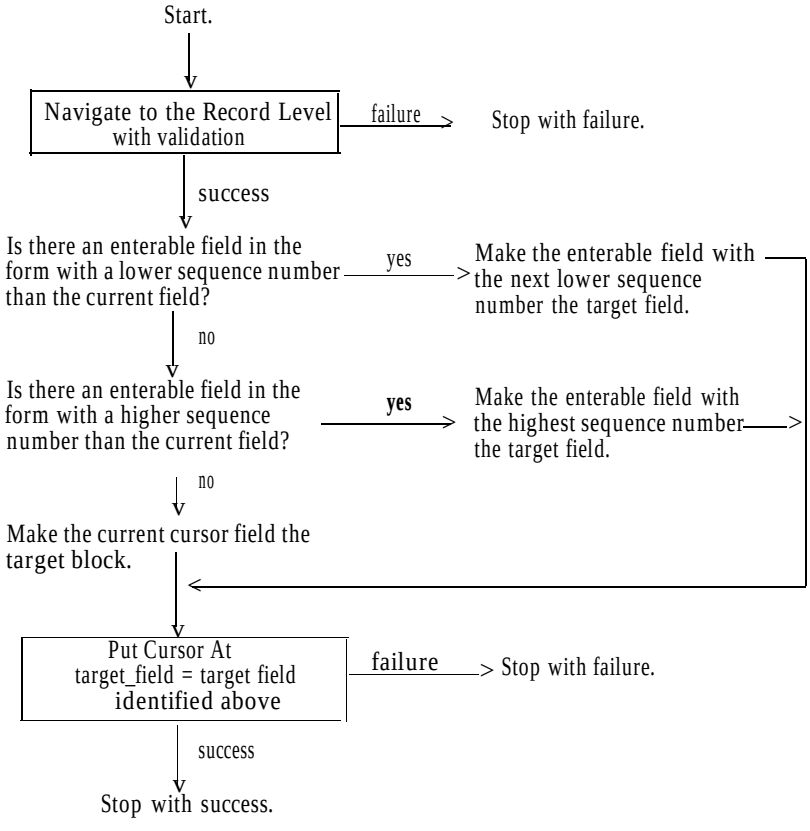
Related Topics: Commit processing 3-16

PREVIOUS_BLOCK



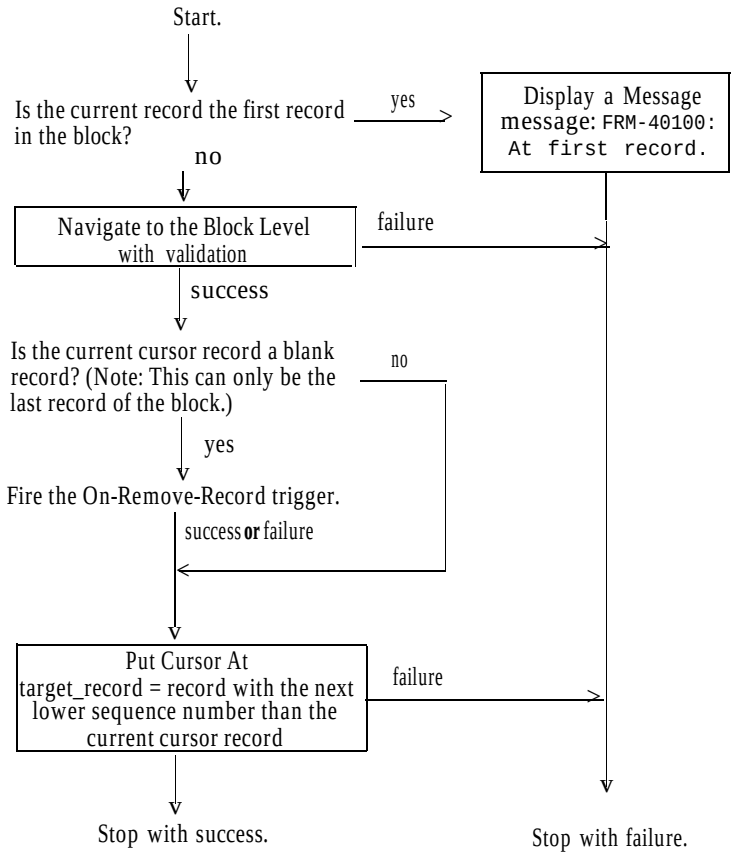
Related Topics: PREVIOUS_BLOCK, packaged procedure 16-26

PREVIOUS_FIELD



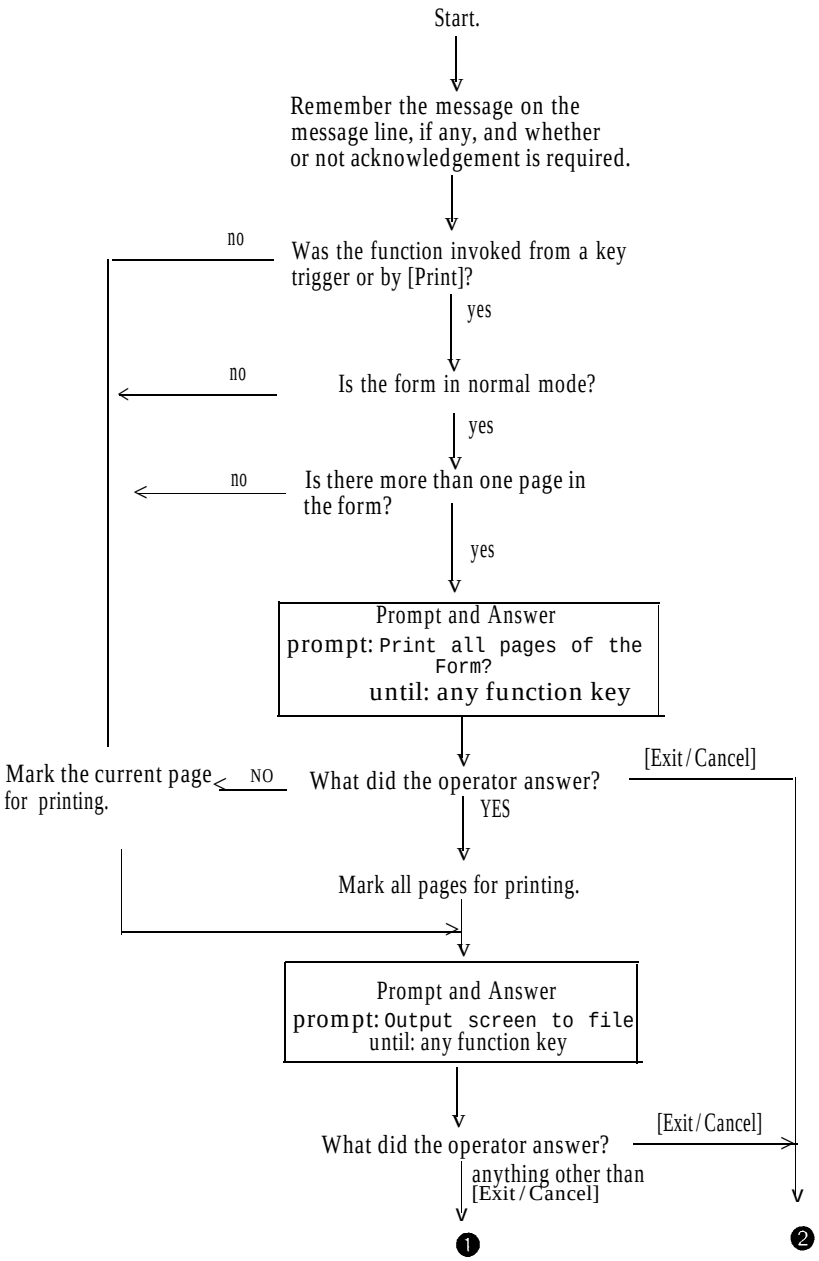
Related Topics: PREVIOUS_FIELD, packaged procedure 16-26

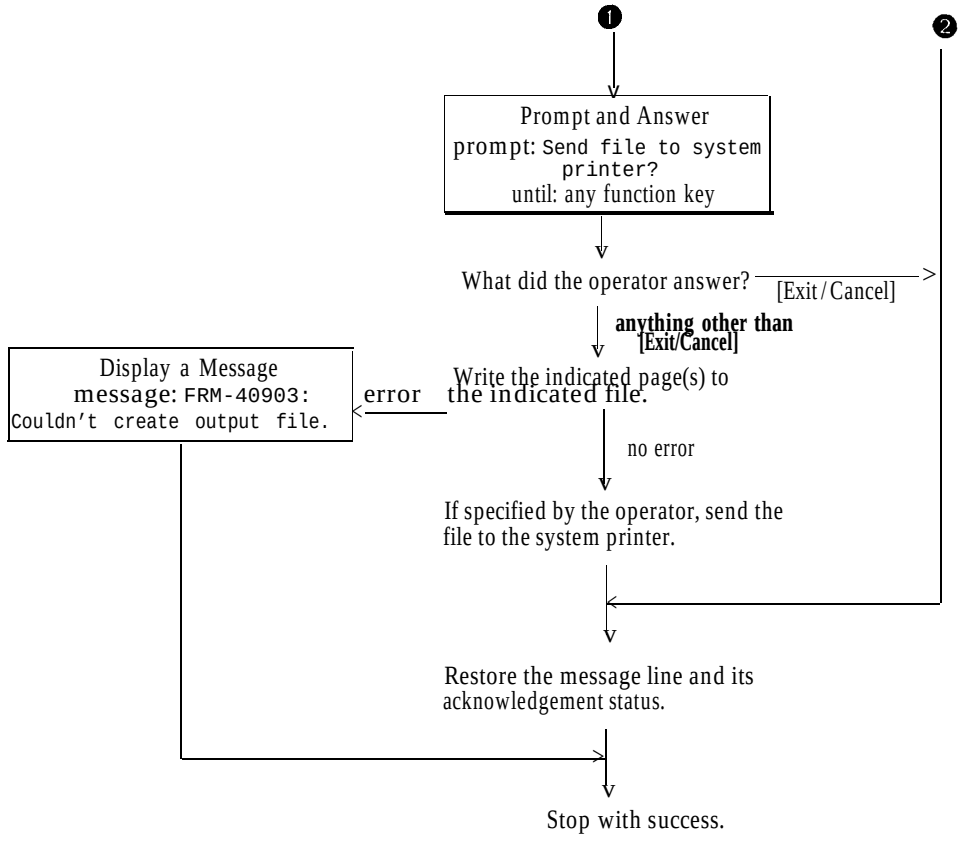
PREVIOUS_RECORD



Related Topics: PREVIOUS_RECORD, packaged procedure 16-27

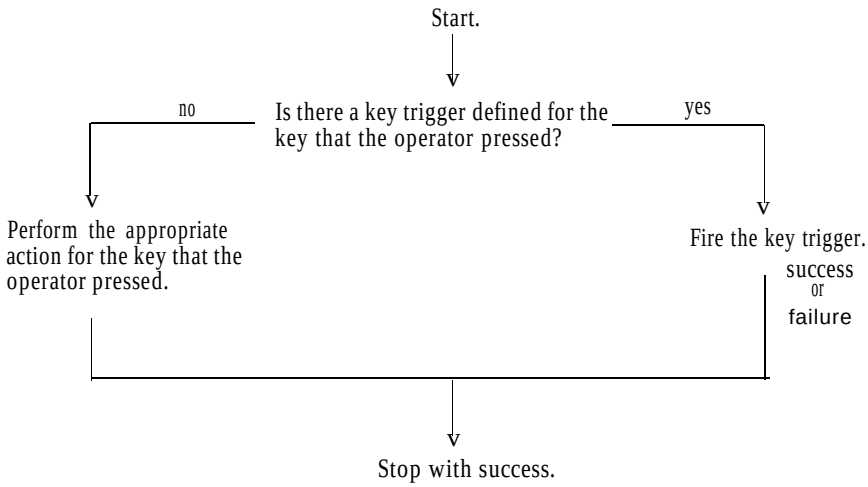
PRINT


Start.
Remember the message on the message line, if any, and whether or not acknowledgement is required.
no
Was the function invoked from a key trigger or by [Print]?
yes
no
Is the form in normal mode?
yes
no
Is there more than one page in the form?
yes
Prompt and Answer
prompt: Print all pages of the Form?
until: any function key
Mark the current page for printing.
NO
What did the operator answer?
[Exit/Cancel]
YES
Mark all pages for printing.
Prompt and Answer
prompt: Output screen to file
until: any function key
What did the operator answer?
[Exit/Cancel]
anything other than [Exit/Cancel]
1
2



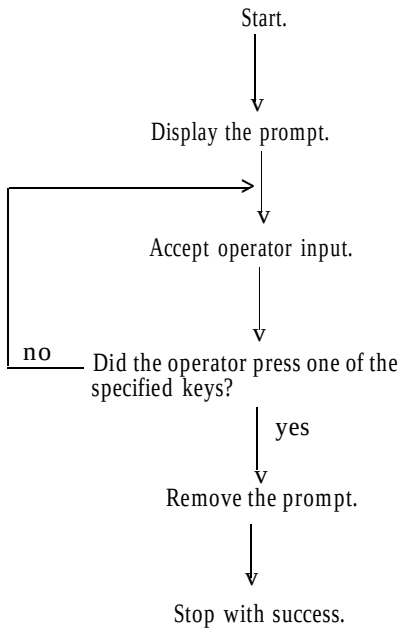
Related Topics: PRINT, packaged procedure 16-27

Process the Function Key

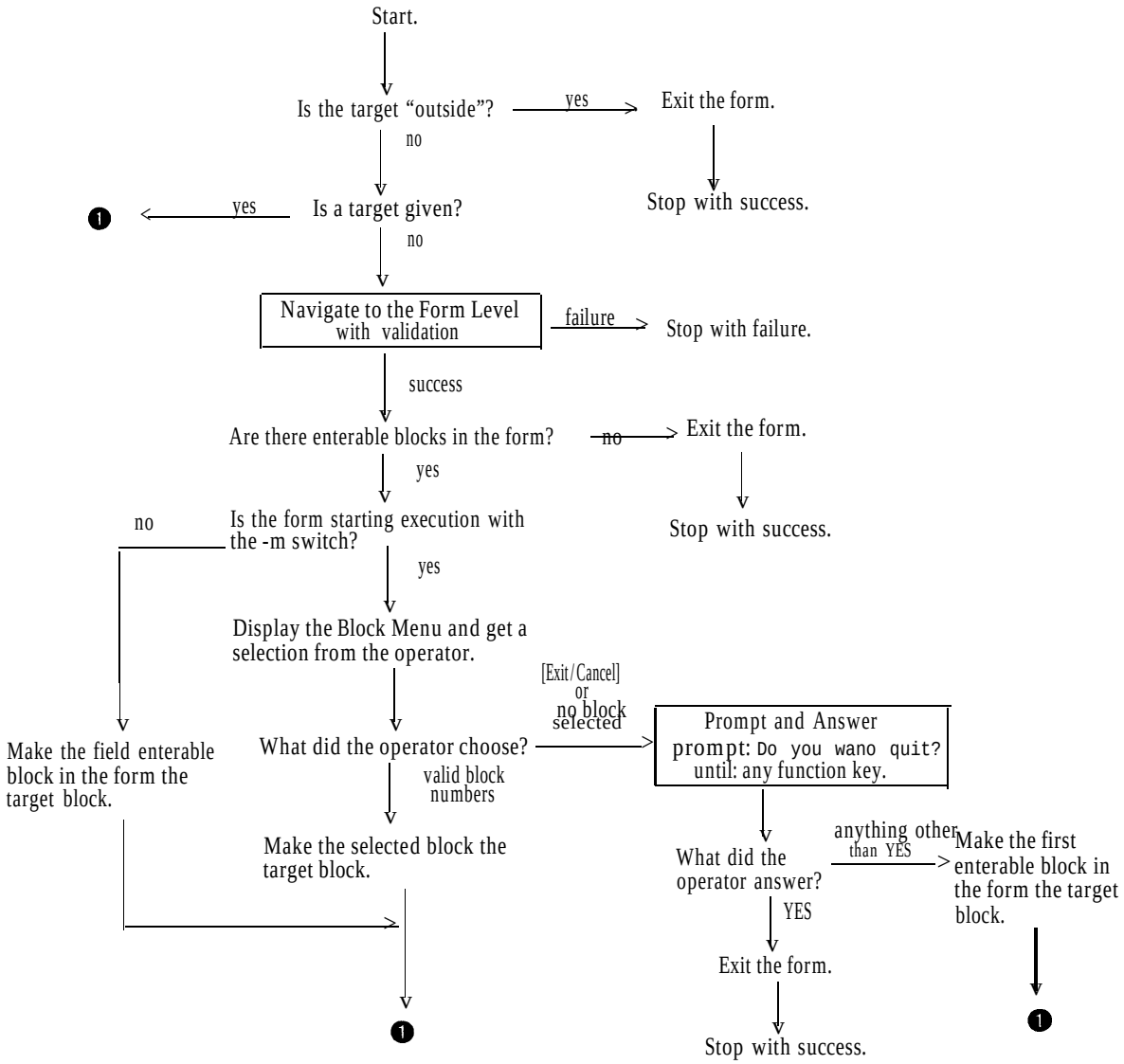


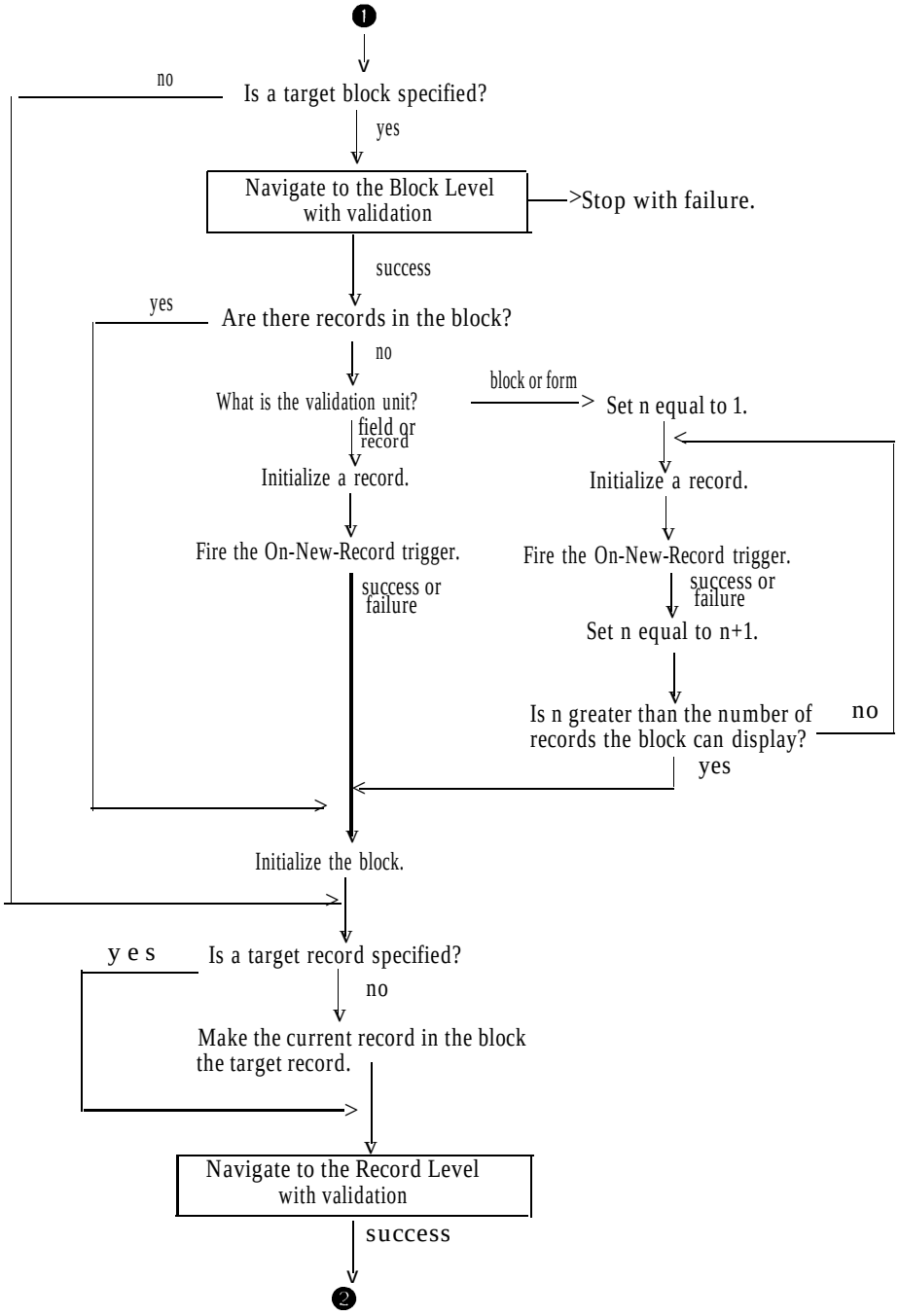
Related Topics: Function keys, processing 3-31

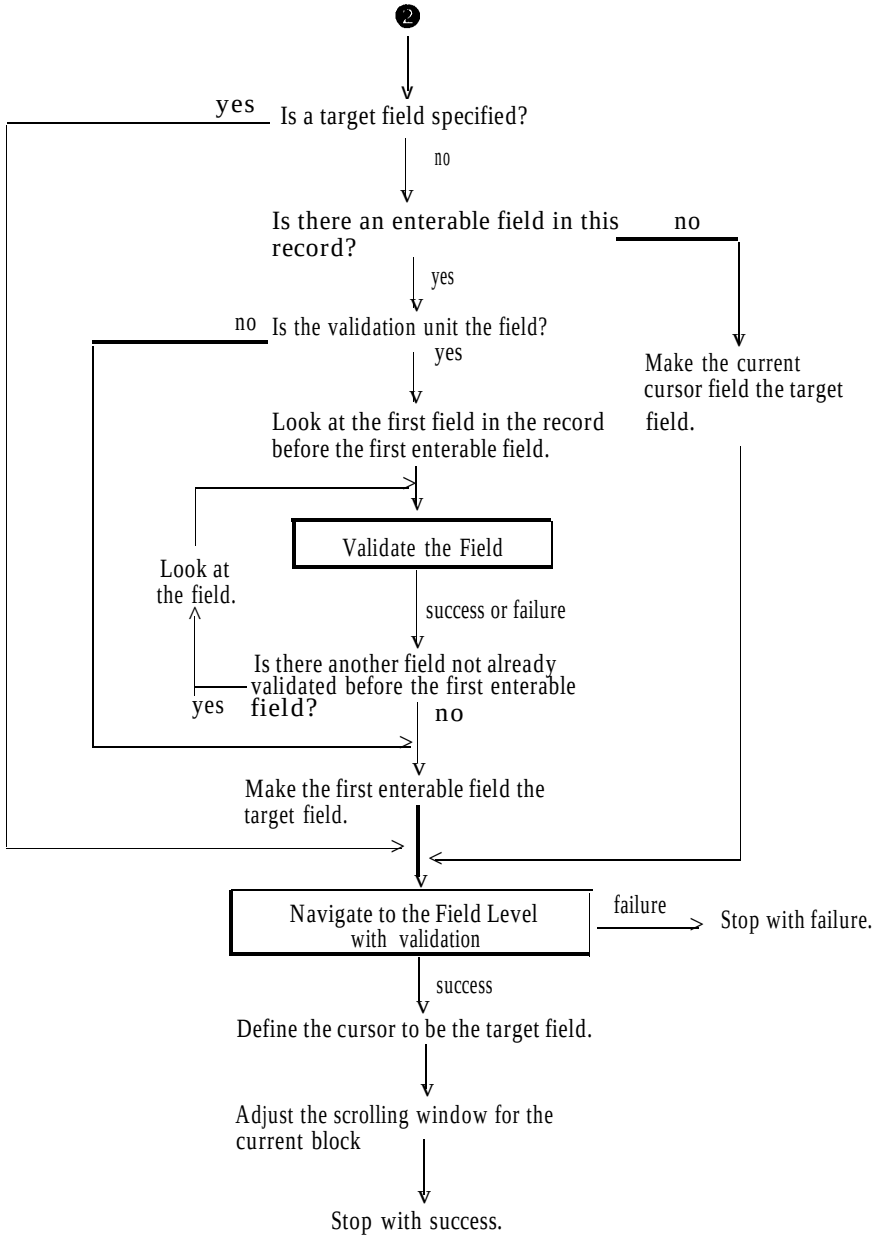
Prompt and Answer



Put Cursor At

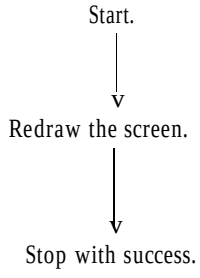



1
no
Is a target block specified?
yes
Navigate to the Block Level with validation
>Stop with failure.
success
yes
Are there records in the block?
no
What is the validation unit?
block or form
Set n equal to 1.
field or record
Initialize a record.
Initialize a record.
Fire the On-New-Record trigger.
Fire the On-New-Record trigger.
success or failure
success or failure
Set n equal to n+1.
Is n greater than the number of records the block can display?
no
yes
Initialize the block.
yes
Is a target record specified?
no
Make the current record in the block the target record.
Navigate to the Record Level with validation
success
2



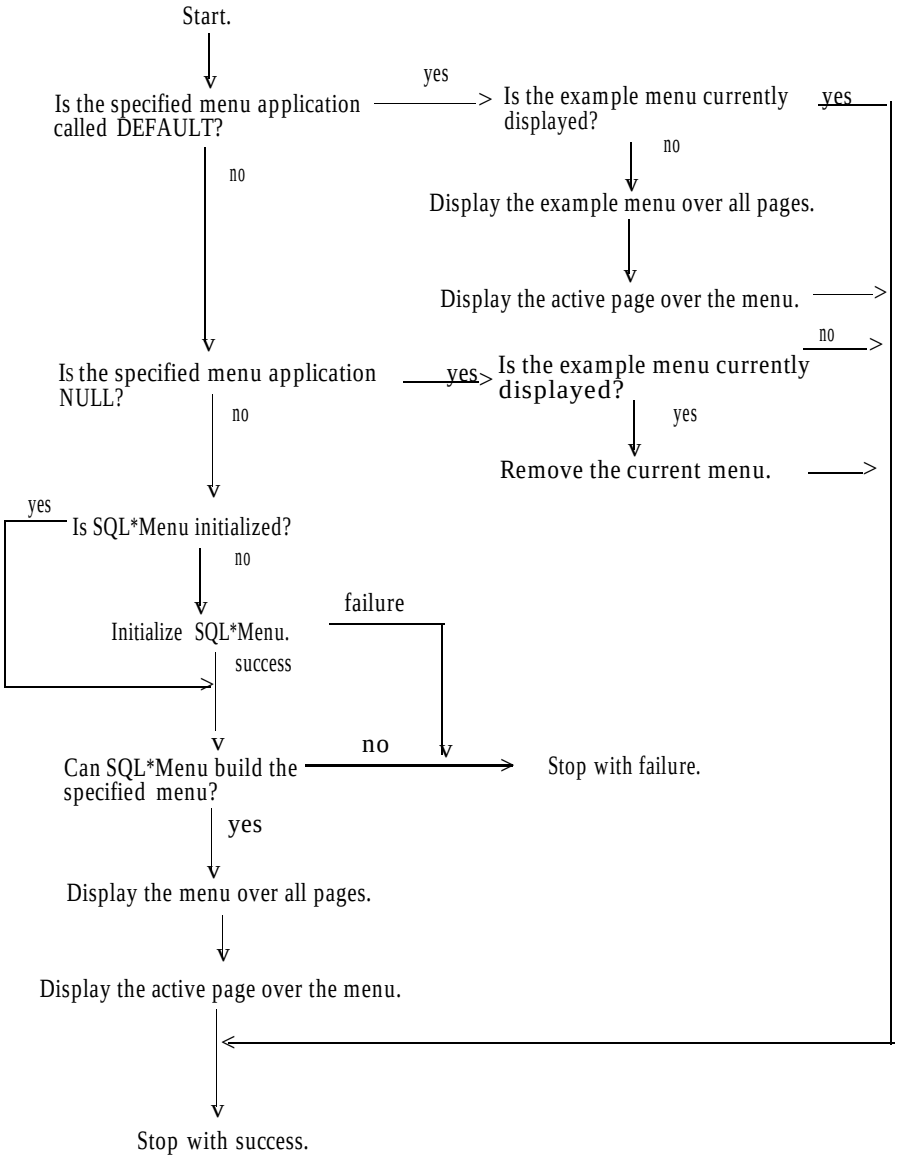
Related Topics: Navigation 3-2

REDISPLAY



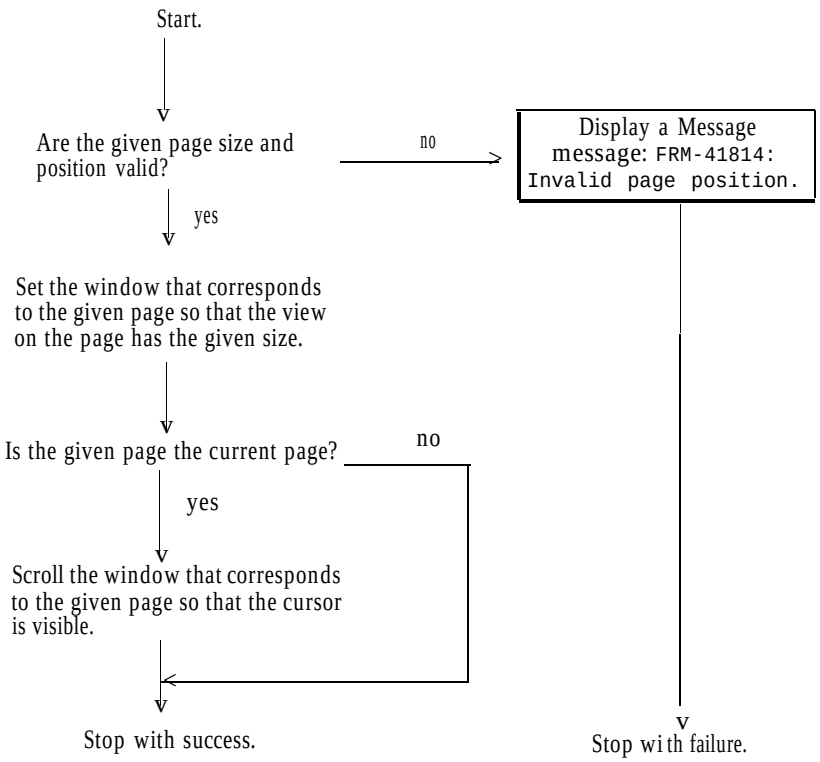
Related Topics: REDISPLAY, packaged procedure 16-27

REPLACE_MENU



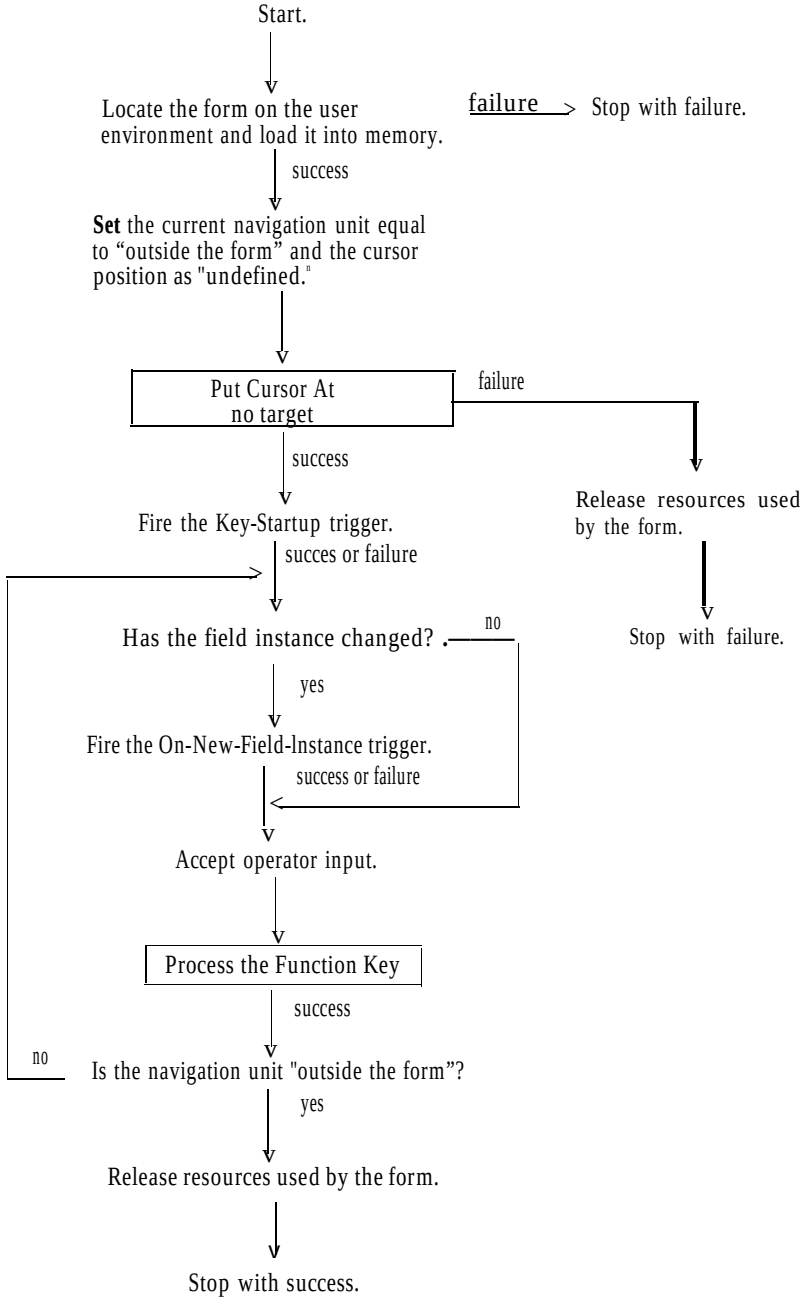
Related Topics: REPLACE_MENU packaged procedure 16-27

RESIZE_VIEW

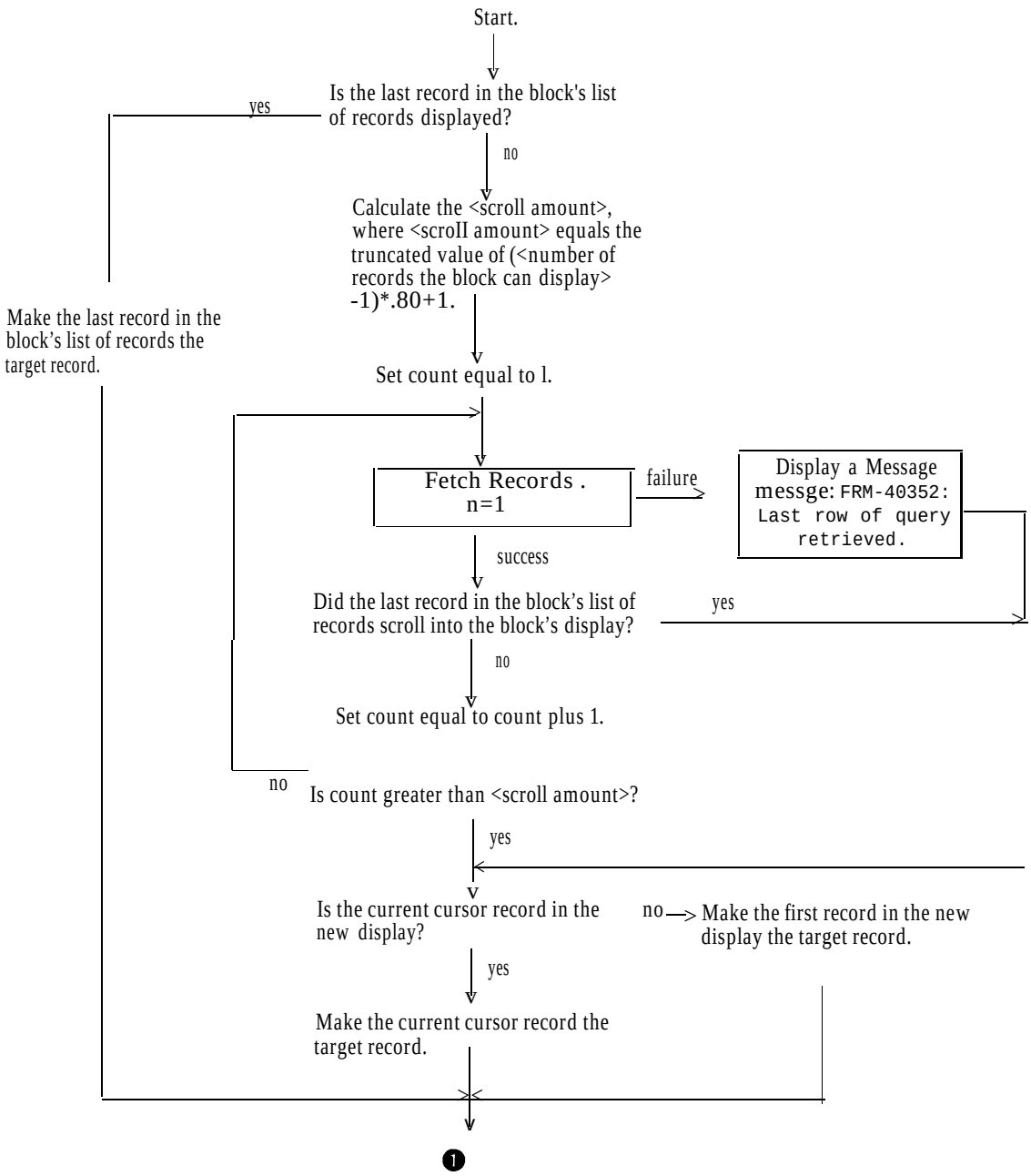


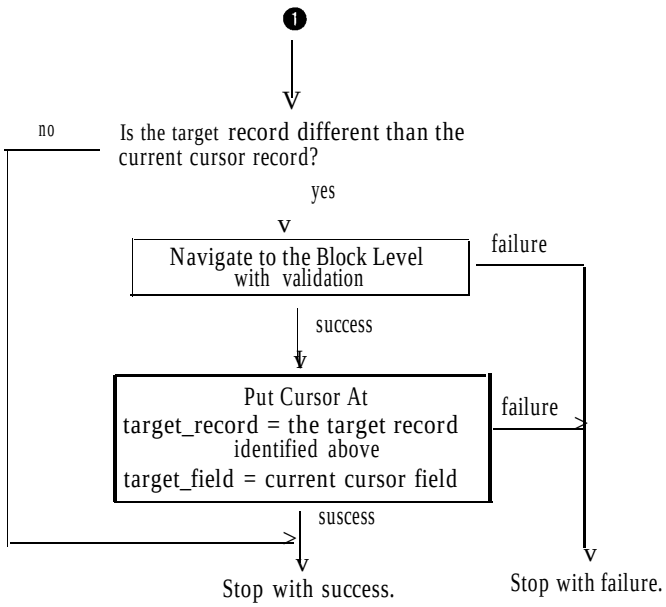
Related Topics: RESIZE_VIEW, packaged procedure 16-28

Run the Form


Start.
Locate the form on the user environment and load it into memory.
failure
Stop with failure.
success
Set the current navigation unit equal to "outside the form" and the cursor position as "undefined."
Put Cursor At no target
failure
success
Release resources used by the form.
Fire the Key-Startup trigger.
succes or failure
Has the field instance changed?
no
Stop with failure.
yes
Fire the On-New-Field-lnstance trigger.
success or failure
Accept operator input.
Process the Function Key
success
no
Is the navigation unit "outside the form"?
yes
Release resources used by the form.
Stop with success.

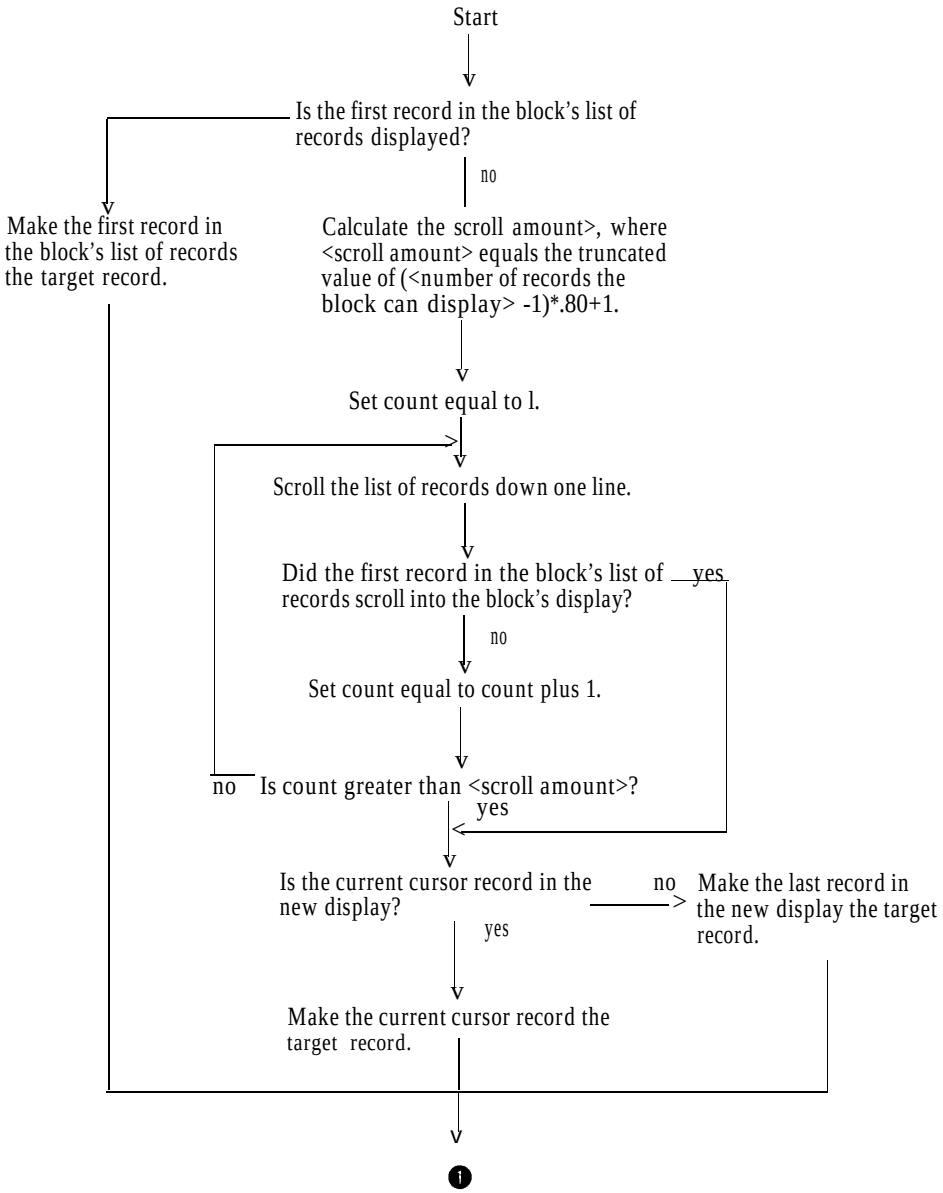
SCROLL_DOWN

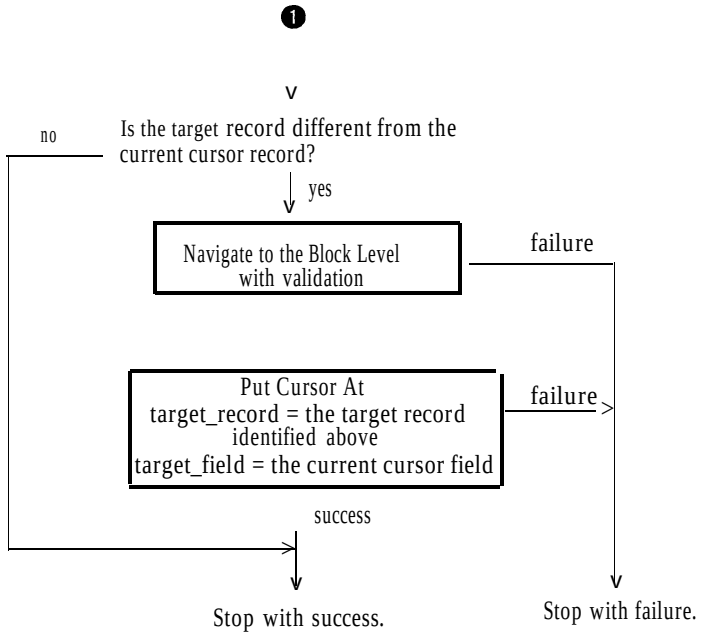

Start.
Is the last record in the block's list of records displayed?
yes
no
Calculate the <scroll amount>, where <scroII amount> equals the truncated value of (<number of records the block can display> -1)*.80+1.
Make the last record in the block's list of records the target record.
Set count equal to l.
Fetch Records . n=1
failure
Display a Message messge: FRM-40352: Last row of query retrieved.
success
Did the last record in the block's list of records scroll into the block's display?
yes
no
Set count equal to count plus 1.
no
Is count greater than <scroll amount>?
yes
Is the current cursor record in the new display?
no
Make the first record in the new display the target record.
yes
Make the current cursor record the target record.
1



Related Topics: SCROLL_DOWN, packaged procedure 16-29

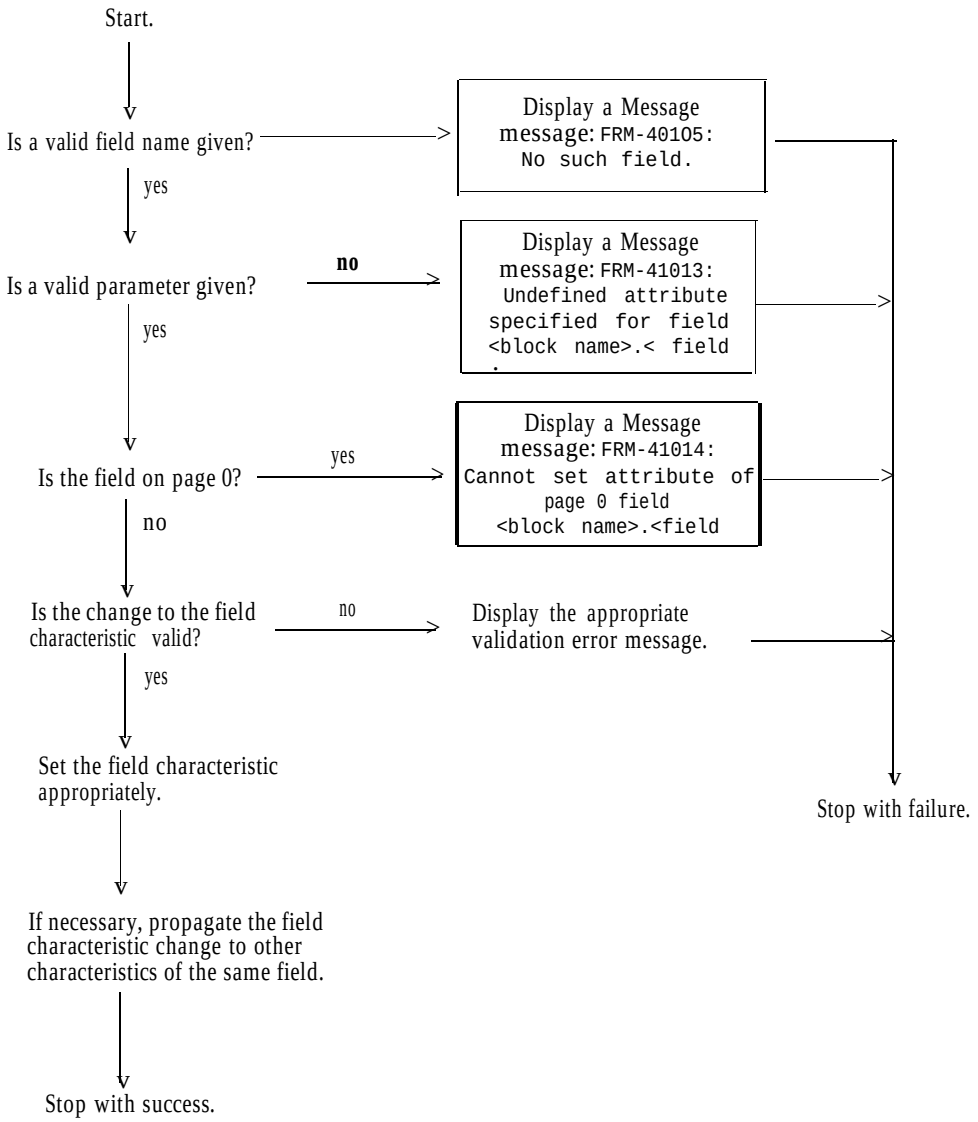
SCROLL_UP


Start
Is the first record in the block's list of records displayed?
no
Make the first record in the block's list of records the target record.
Calculate the scroll amount>, where <scroll amount> equals the truncated value of (<number of records the block can display> -1)*.80+1.
Set count equal to l.
Scroll the list of records down one line.
Did the first record in the block's list of records scroll into the block's display?
yes
no
Set count equal to count plus 1.
no
Is count greater than <scroll amount>?
yes
Is the current cursor record in the new display?
no
Make the last record in the new display the target record.
yes
Make the current cursor record the target record.
1



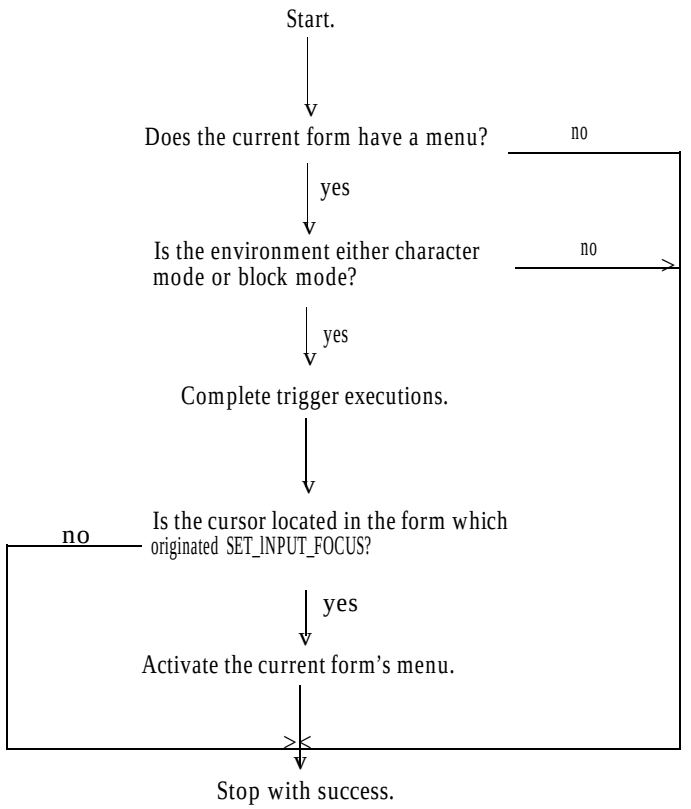
Related Topics: SCROLL_UP, packaged procedure 16-29

SET_FIELD



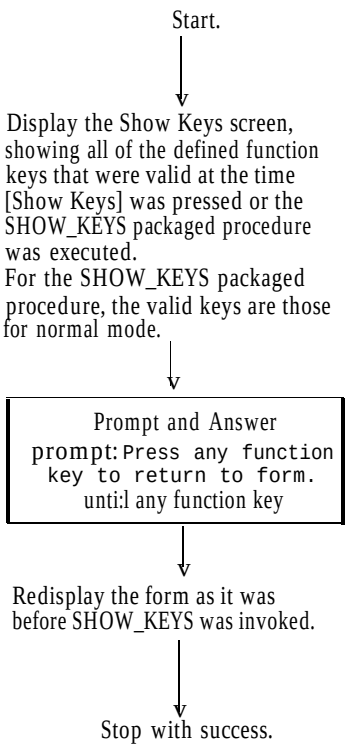
Related Topics: SET_FIELD, packaged procedure 16-30

SET_INPUT_FOCUS



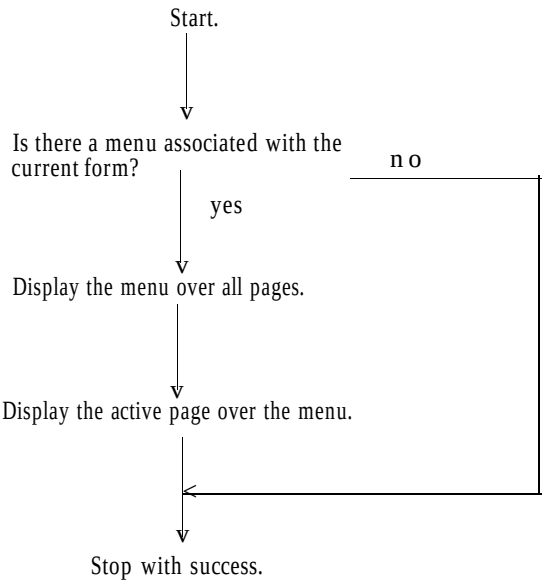
Related Topics: SET_INPUT_FOCUS, packaged procedure 16-33

SHOW_KEYS



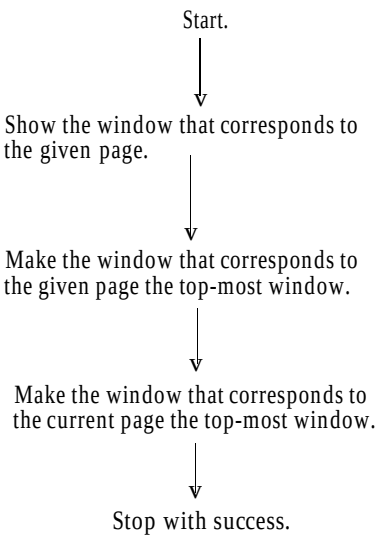
Related Topics: SHOW_KEYS, packaged procedure 16-33

SHOW_MENU



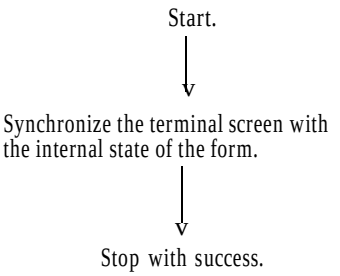
Related Topics: SHOW_MENU, packaged procedure 16-33

SHOW_PAGE



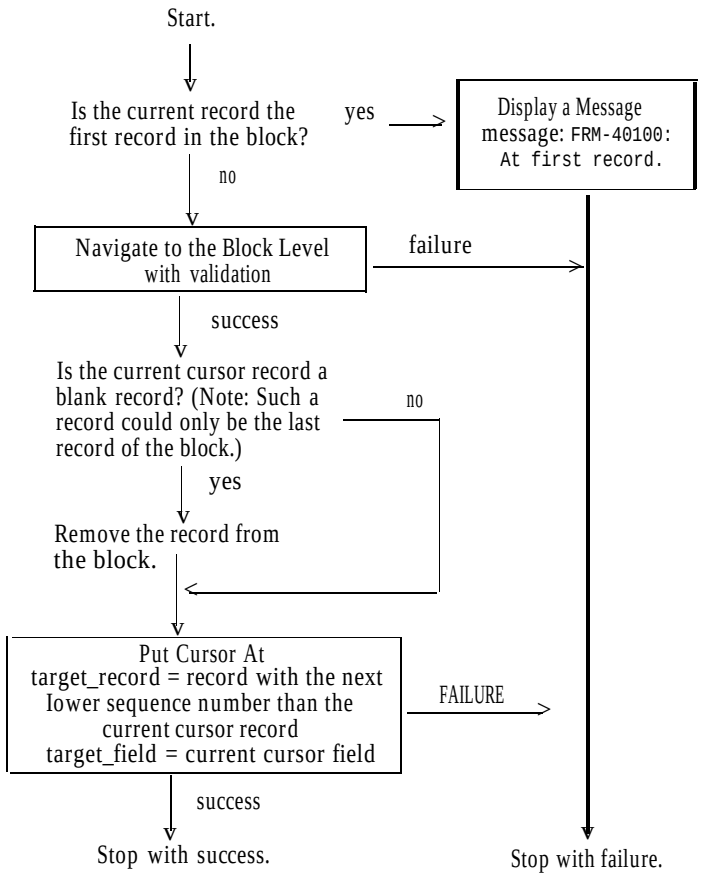
Related Topics: SHOW_PAGE, packaged procedure 16-33

SYNCHRONIZE



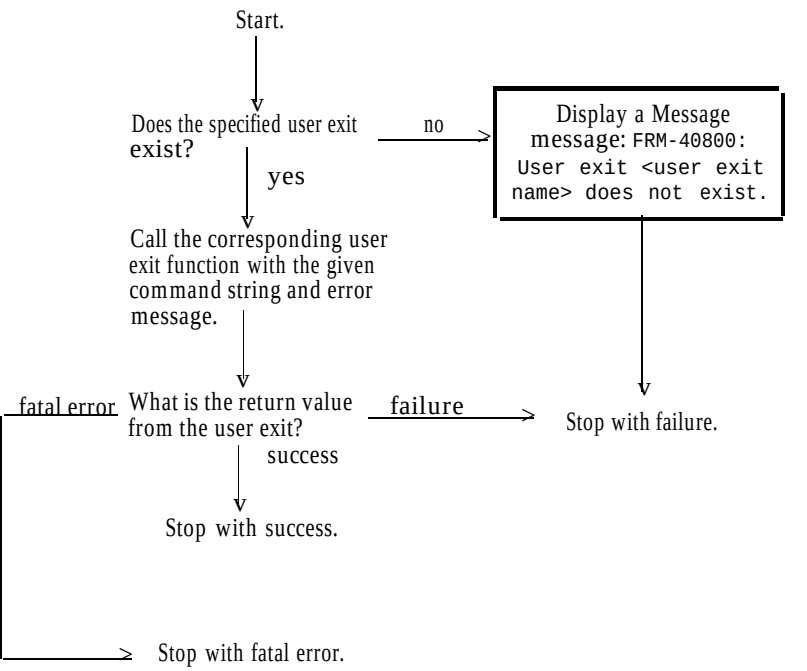
Related Topics: SYNCHRONIZE, packaged procedure 16-34

UP



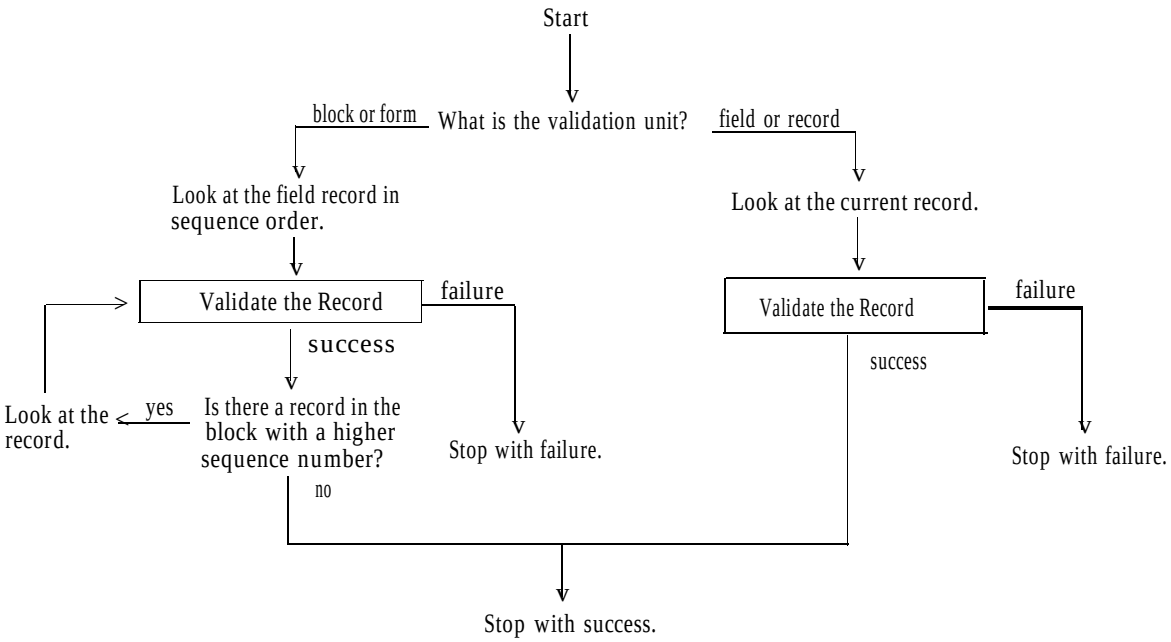
Related Topics: UP, packaged procedure 16-34

USER_EXIT



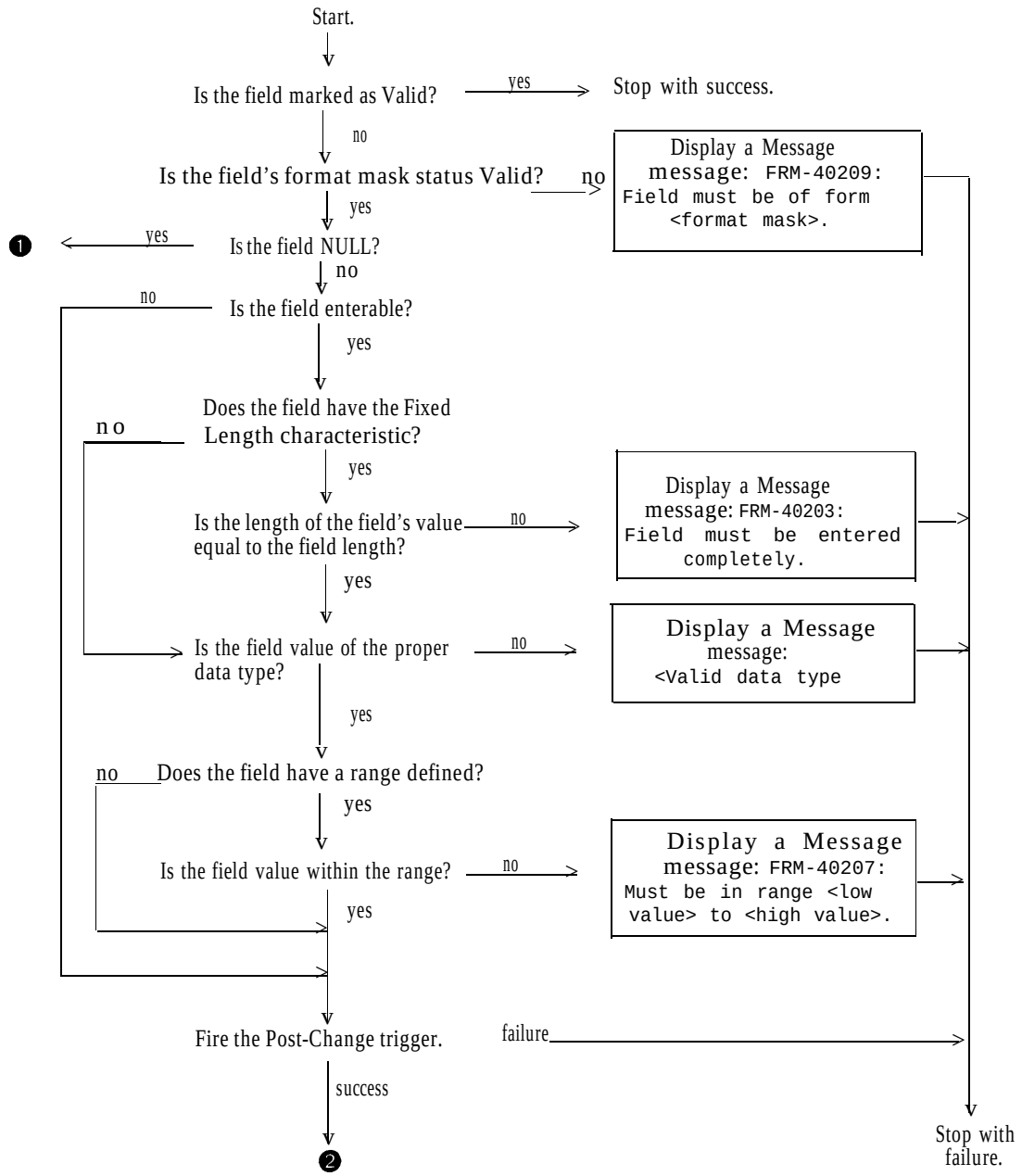
Related Topics: USER_EXIT, packaged procedure 16-34

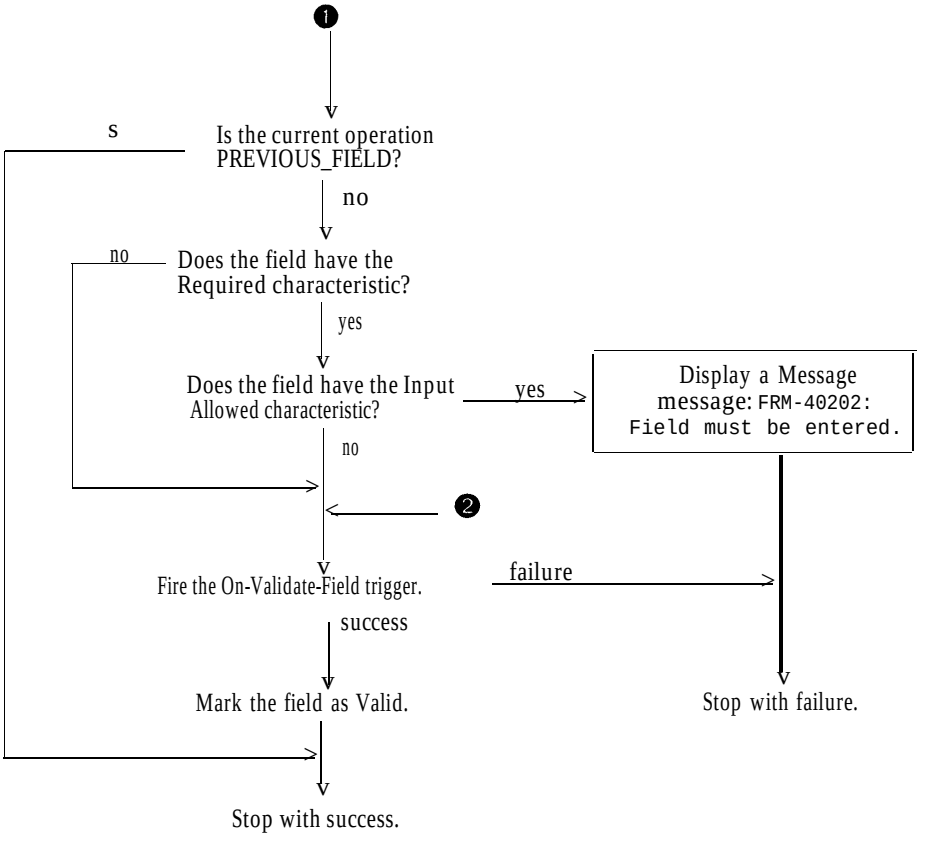
Validate the Block



Related Topics: Validation 3-6

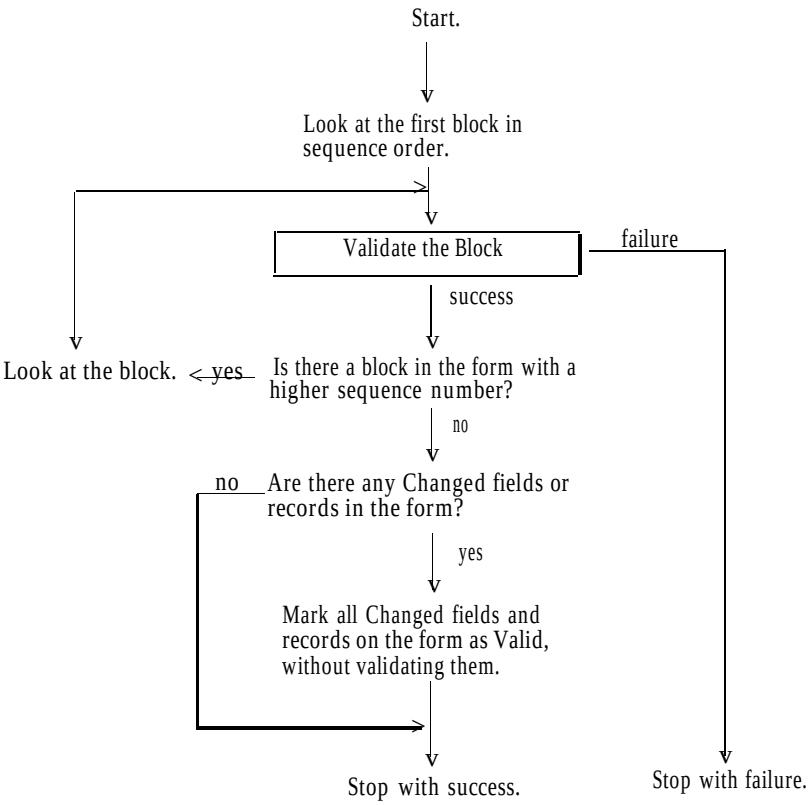
Validate the Field





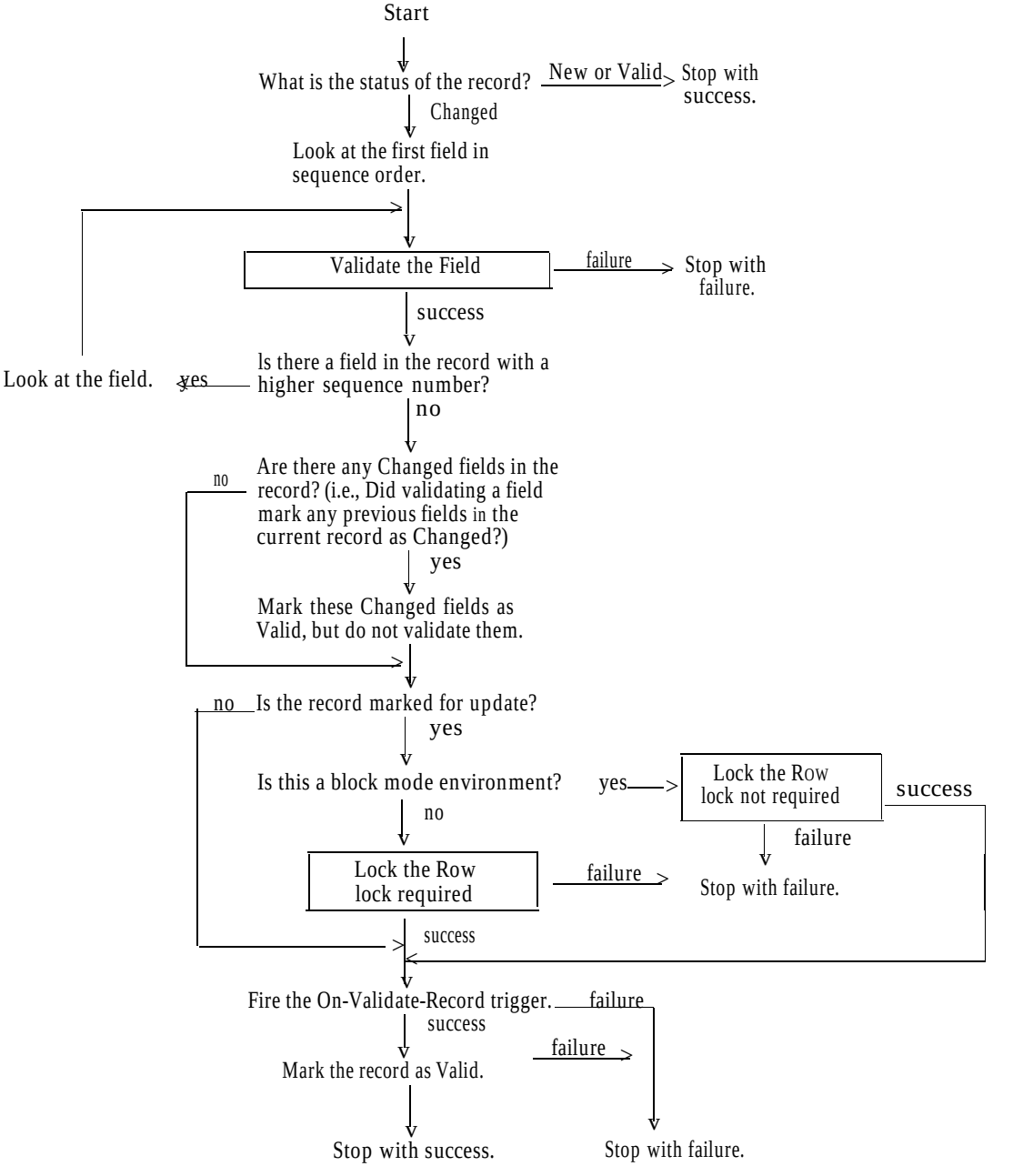
Related Topics: Validation 3-6

Validate the Form



Related Topics: Validation 3-6

Validate the Record



Related Topics: Validation 3-6

Events and Sub-events

The following table lists each event that is a sub-event (i.e., that appears as part of another event) and the events in which it appears. Use this table, for example, to identify all events in which SQL*Forms fetches rows from the database (Fetch Records event).

Sub-event	*Appears in Events*
Break Processing	BREAK
Check the Block for Query	COUNT_QUERY ENTER_QUERY Execute the Query NEXT_SET
Check the Block for Update	Check the Record for Update
Check the Field for Edit	CLEAR_EOL CLEAR_FIELD DUPLICATE_FIELD Enter the Value Into a Field LIST_VALUES
Check the Record for Update	Check the Field for Edit COPY DEFAULT_VALUE DELETE_RECORD LOCK_RECORD
CLEAR_RECORD	DELETE_RECORD
CREATE_ RECORD	DOWN NEXT_RECORD

Table continued on next page.

Sub-event	*Appears in Events*
Display a Message	ABORT_QUERY BLOCK_MENU Break Processing Check the Block for Query Check the Block for Update Check the Field for Edit Check the Record for Update COPY COUNT_QUERY DEFAULT_VALUE DISPLAY_ERROR DISPLAY_FIELD DISPLAY_PAGE DOWN DUPLICATE_FIELD DUPLICATE_RECORD ENTER_QUERY Execute the Query EXECUTE_TRIGGER GO_BLOCK GO_FIELD LIST_VALUES Lock the Row MESSAGE MOVE_VIEW NEXT_RECORD NEXT_SET Post and Commit Transactions PREVIOUS_RECORD PRINT RESIZE_VIEW SCROLL_DOWN UP USER_EXIT Validate the Field
Enter the Block	Navigate to the <Navigation Unit> Level
Enter the Field	Navigate to the <Navigation Unit> Level
Enter the Form	Navigate to the <Navigation Unit> Level

Table continued on next page.

Sub-event	*Appears in Events*
Enter the Record	Navigate to the <Navigation Unit> Level
Enter the Value Into a Field	LIST_VALUES
Execute the Query	EXECUTE_QUERY
Fetch Records	CLEAR_RECORD DELETE_RECORD DOWN GO_RECORD LAST_RECORD NEXT_RECORD NEXT_SET NEXT_SET Open the Query SCROLL_DOWN
Leave the Block	Navigate to the <Navigation Unit> Level
Leave the Field	Navigate to the <Navigation Unit> Level
Leave the Form	EXIT_FORM Navigate to the <Navigation Unit> Level NEW_FORM
Leave the Record	Navigate to the <Navigation Unit> Level
Leave Unit Error Processing	Leave the Block Leave the Field Leave the Form Leave the Record
Lock the Row	Check the Field for Edit LOCK_RECORD Open the Query Validate the Record
Mark Fields and Records as changed	CLEAR_EOL CLEAR_FIELD DUPLICATE_FIELD Enter the Value Into a Field

Table continued on next page.

Sub-event	*Appears in Events*
Navigate to the <Navigation Unit> Level	BLOCK_MENU CLEAR_BLOCK CLEAR_FORM CLEAR_RECORD COMMIT_FORM COUNT_QUERY CREATE_RECORD DELETE_RECORD DOWN ENTER ENTER_QUERY Execute the Query FIRST_RECORD GO_BLOCK GO_FIELD GO_RECORD LAST_RECORD NEXT_BLOCK NEXT_FIELD NEXT_KEY NEXT_RECORD NEXT_SET POST PREVIOUS_BLOCK PREVIOUS_FIELD PREVIOUS_RECORD Put Cursor At SCROLL_DOWN SCROLL_UP UP
Open the Query	Execute the Query
Post and Commit Transactions	CLEAR_BLOCK CLEAR_FORM COMMIT_FORM COUNT_QUERY ENTER_QUERY Execute the Query EXIT_FORM POST

Table continued on next page.

Sub-event	*Appears in Events*
Process the Function Key	CALL_INPUT Run the Form
Prompt and Answer	BLOCK_MENU DISPLAY_ERROR Display a Message HELP HOST Leave Unit Error Processing PAUSE Post and Commit Transactions PRINT Put Cursor At SHOW_KEYS
Put Cursor At	BLOCK_MENU CLEAR_BLOCK CLEAR_FORM CLEAR_RECORD COMMIT_FORM COUNT_QUERY CREATE_RECORD DELETE_RECORD DOWN ENTER Enter the Block Enter the Field Enter the Form Enter the Record Execute the Query FIRST_RECORD GO_BLOCK GO_FIELD GO_RECORD LAST_RECORD Leave Unit Error Processing NEXT_BLOCK NEXT_FIELD NEXT_KEY NEXT_RECORD NEXT_SET

Table ***continued on next page.***

Sub-event	*Appears in Events*
Put Cursor At (continued)	POST PREVIOUS_BLOCK PREVIOUS_FIELD PREVIOUS_RECORD Run the Form SCROLL_DOWN SCROLL_UP UP
Run the Form	CALL CALL_QUERY
Validate the Block	ENTER Leave the Block Post and Commit Transactions Validate the Form
Validate the Field	ENTER Leave the Field NEXT_FIELD NEXT_KEY Put Cursor At Validate the Record
Validate the Form	COMMIT_FORM ENTER POST Post and Commit Transactions
Validate the Record	ENTER Leave the Record Validate the Block

CHAPTER

22 CONSTRAINTS

This chapter discusses processing constraints for SQL*Forms. Specifically, it covers the following topics:

- object constraints
- cursor constraints
- constraints in a 16-bit environment

Note: Memory constraints are machine specific and are not discussed in this book. For information on memory constraints, refer to the ORACLE Installation and User's Guide for your system.

Object Constraints

The following table presents constraints that apply to SQL*Forms objects and any other elements that are related to these objects.

Element	*Constraint/Note*
database field references per form	255 references This constraint applies to references to fields by numeric bind or define variables in a SQL statement.
fields per block	255 fields This constraint applies specifically to SQL*Forms (Design) and SQL*Forms (Convert).
field range checks per form	8190 checks This constraint applies to the number of fields per form for which you specify the Valid Range field characteristic.
field width	255 characters This constraint applies to the length of any field, except a field with the LONG data type. LONG fields can be up to 65,534 characters in length.
blocks per form	255 blocks This constraint applies specifically to SQL*Forms (Design) and SQL*Forms (Convert).
triggers per form	4095 triggers
trigger steps per form	8190 trigger steps
pages	8190 pages
page width	255 characters
global variable width	255 characters
constant text per form	10922 lines
list of values	32767 records

Cursor Constraints

A database cursor is the memory that the ORACLE RDBMS assigns to a SQL command. This applies to the SQL commands that are implicit to posting and querying data and the SQL commands that are explicit in triggers. SQL*Forms controls cursor assignment for all implicit SQL commands and the explicit SQL commands in V2 triggers.

The following table presents constraints that apply to database cursors.

Element	*Constraint/Note*
open cursors (maximum)	value of the OPEN_CURSORS parameter found in the INIT.ORA file
open cursors (minimum)	• 5 per ORACLE connection • 4 per block (1 for SELECT, 1 for UPDATE, 1 for INSERT, and 1 for DELETE) This restriction applies under normal circumstances, but it is possible to change the number of cursors by altering settings on the Set Options form or by using command line switches. • 1 or 0 per trigger step (1 for any SQL statement) This restriction applies under normal circumstances, but it is possible to change the number of cursors by altering settings on the Set Options form or by using command line switches.

Related Topics: New Cursor Area trigger step characteristic F-6
-o switch, SQL*Forms (Run Form) 23-18
Share Cursors for Implicit SQL run option 26-4
Share Cursors for Trigger SQL run option 26-4
-t switch, SQL*Forms (Run Form) 23-20

16-bit Environment Constraints

In a 16-bit environment, the total limit of character string data perform is 65,535 bytes, which is the largest piece of memory that can be allocated. Character string data includes the following items:

- base table names
- bind variable names used in PL/SQL
- block descriptions
- block names
- constraint text
- database table and column names for blocks and fields
- default ORDER BY clauses
- default values
- default WHERE clauses
- field hint message
- field format masks
- field names
- field prompts
- high and low range values
- list of values text
- list of values titles
- page titles
- trigger descriptions
- V2 trigger error messages
- trigger names
- V2 trigger failure labels
- V2 trigger step labels
- V2 trigger success labels
- form-level procedure names

The total limit for trigger step text is 131,070 bytes.

The total limit for the compiled representation of Version 3 style trigger text is 65,535 bytes.

The total limit for the compiled representation of procedure text is 65,535 bytes.

PART

V

COMPONENTS AND PRODUCTS

CHAPTER

23 COMPONENTS

This chapter discusses the components that makeup SQL*Forms. Specifically, it covers the following components:

- SQL*Forms (Convert)
- SQL*Forms (Design)
- SQL*Forms (Generate)
- SQL*Forms (Run Form)

As a designer, you most often use SQL*Forms (Design), from which you can access the other components. However, you will sometimes find it useful to run the other components individually from the command line.

This chapter also discusses the ORACLE automatic login feature, the remote login feature, and SQL*Forms command line switches.

Related Topics: Form storage formats 24-2

Automatic Login

The ORACLE automatic login feature allows you to log in to ORACLE products without explicitly stating a username and password. When you log in in this way, ORACLE uses an OPS$ ID to associate your ORACLE username to your operating system User ID.

You can use the automatic login feature with all SQL*Forms components. Refer to the *ORACLE RDBMS Database Administrator's Guide* for information on how to employ this feature.

Remote Login

Through SQL*Net, you can specify a remote database when you log in to any SQL*Forms component. When you use the remote login feature, the remote database is transparent to you—SQL*Forms behaves the same whether it is accessing a node across the country or accessing the database on your local machine.

To specify a remote database from the command line, append a driver name and a node name to your password using the following syntax:

```
: node_name
```

where:

driver_name	Indicates the driver that you want to use to access the remote database. The default setting is system dependent.
node_name	Indicates the remote network node that contains the target database. You will log into the default database on this node.

For example, you might specify the following command to run the ORDER_ENTRY form on the default database of the LONDON node:

```
RUNFORM ORDER_ ENTRY SCOTT/TIGER@D : LONDON
```

If you want to specify a remote database from a SQL*Forms login screen, append the driver name and node name to your username (e.g., `SCOTT@D:LONDON`).

For information on SQL*Net, refer to the *SQL*Net User's Guide.*

Note: To perform a remote login, you must have an account on the remote computer node where the target database resides.

SQL*Forms (Convert)

SQL*Forms (Convert) creates an INP file from a form in database format or creates a form in database format from an INP file. After a "conversion," both instances of the form exist (unless you specify the-d switch, which deletes the form from the database).

Use SQL*Forms (Convert) in the following ways:

- Create a form in database format from an INP file that was generated with another version or installation of SQL*Forms.
- Create an INP file from a form in database format.
- Create an INP file in Version 3.0 format from an INP file in Version 2.0 or 2.3 format.

Related Topics: Form storage formats 24-2

INP to Database Conversion

You can create a form in database format from an INP file in two ways:

- From the command line, use the CONVFORM command, specifying the -i switch, an INP filename, and a form name. For example, to create a form called SAMPLE2 from the SAMPLE1.INP file, enter the following command:

  ```
  CONVFORM -i SAMPLE1 SAMPLE2 [username/Password]
  ```

- From SQL*Forms (Design), select the Load option from the Action menu and specify the INP file you want to convert in the Load Form dialog box. Then, save the form in the database via the Save option on the Action menu.

Database to INP Conversion

You can create an INP file from a form in database format in two ways:

- From the command line, use the CONVFORM command, specifying an INP filename and a form name. For example, to create an INP file called SAMPLE3.INP from the SAMPLE2 form, enter the following command:

  ```
  CONVFORM SAMPLE3 SAMPLE2 [username/password]
  ```

- From SQL*Forms (Design), select the Generate option from the Action menu and specify the form you want to convert in the Generate Form dialog box. Note that this procedure also creates an FRM file, which is its primary purpose.

INP to INP Conversion

You can create an INP file in Version 3.0 format form an INP file in Version 2.0 or 2.3 format. From the command line, use the CONVFORM command, specifying the -v switch, a version number, an INP filename and its associated form name. For example, to create a new Version 3.0 INP file from a Version 2.3 INP file named OLDFILE.INP (whose form name is OLDFORM), enter the following command:

```
CONVFORM -v23 OLDFILE OLDFORM [username/password]
```

CONVFORM Command

The CONVFORM command invokes SQL*Forms (Convert) from the command line. To use this command, enter the following statement at the system prompt:

```
CONVFORM [option option. . . ] inp_filename formname
[username/password]
```

where:

option	Specifies any command line switch you want to activate for this SQL*Forms (Convert) session. Available command line switches are the -c, -d, -f, -i, -s, and -v switches for SQL*Forms (Convert).
inp_filename	Specifies the name, without the .INP suffix, of the INP file that you want to create or convert.
formname	Specifies the name of the form you want to convert or create.
username / password	Specifies your ORACLE username and password.

If you do not specify your username and password at the system prompt, SQL*Forms uses an OPS$ ID to log in automatically. If ORACLE cannot find an OPS$ ID, it issues an error.

You can type `CONVFORM ?` at the operating system prompt to receive syntax help.

Related Topics: Command line switches,
SQL*Forms (Convert) 23-8

SQL*Forms (Design)

SQL*Forms (Design) creates or modifies a form in the database. SQL*Forms (Design) is the main SQL*Forms component, which can call the other three components of SQL*Forms.

Related Topics: Form storage formats 24-2

SQLFORMS Command

The SQLFORMS command invokes SQL*Forms (Design) from the command line. To use this command, enter the following statement at the system prompt:

```
SQLFORMS [option option...]  [username/password]
```

where:

option	Specifies any command line switch you want to activate for this SQL*Forms (Design) session. Available command line switches are the -c, -e, -i, -l, -r, -s, and -w switches for SQL*Forms (Design).
username / password	Specifies your ORACLE username and password.

If you do not specify your username and password at the system prompt, SQL*Forms uses an OPS$ ID to login automatically. If ORACLE cannot find an OPS$ ID, it displays the Login screen which prompts you for a username and password.

You can type `SQLFORMS ?` at the operating system prompt to receive syntax help.

Related Topics: Command line switches, SQL*Forms (Design) 23-10

SQL*Forms (Generate)

SQL*Forms (Generate) creates an FRM file. It also might create or modify an INP file, depending on whether one already exists and on the command line options you specify:

You can create an FRM file in two ways:

- From the command line, use the GENERATE command, specifying the name of the INP file that you want to convert (or create if it does not already exist). For example, to create an FRM file from the SAMPLE1.INP file, enter the following command:

  ```
  GENERATE SAMPLE1 [username/password]
  ```

- From SQL*Forms (Design), select the Generate option from the Action menu and specify the INP file in the Generate Form dialog box. This INP file can be the file from which you want to generate or the file that you want to create if it does not already exist.

Related Topics: Form storage formats 24-2

GENERATE Command

The GENERATE command invokes SQL*Forms (Generate) from the command line. To use this command, enter the following statement at the system prompt:

```
GENERATE [option option. . . ] inp_filename [username/password]
```

where:

option	Specifies any command line switch you want to activate for this SQL*Forms (Generate) session. Available command line switches are the -b, -n, -o, -s, and -t switches for SQL*Forms (Generate).
inp_filename	Specifies the name, without the .INP suffix, of the INP file that you want to create or convert.
username / password	Specifies your ORACLE username and password.

If you do not specify your username and password at the system prompt, SQL*Forms uses an OPS$ ID to log in automatically. If ORACLE cannot find an OPS$ ID, it issues an error.

You can type `GENERATE ?` at the operating system prompt to receive syntax help.

Related Topics: Command line switches,
SQL*Forms (Generate) 23-13

SQL*Forms (Run Form)

SQL*Forms (Run Form) reads an FRM file and executes the form. This is the component you use to test forms and that operators use to run your applications.

You can execute a form in two ways:

- From the command line, use the RUNFORM command, specifying the name of the form that you want to execute. For example, to execute the SAMPLE4 form, enter the following command:

  ```
  RUNFORM SAMPLE4 [username/password]
  ```

- From SQL*Forms (Design), select the Execute option from the Action menu and specify the name of the form you want to execute in the Execute Form dialog box.

Related Topics: Form storage formats 24-2

RUNFORM Command

The RUNFORM command invokes SQL*Forms (Run Form) from the command line. To use this command, enter the following statement at the system prompt:

```
RUNFORM [option option. . .] formname [username/password]
```

where:

option	Specifies any command line switch you want to activate for this SQL*Forms (Run Form) session. Available command line switches are the -a, -b, -c, -d, -e, -i, -l, -m, -n -o, -q, -r, -s, -t, and -w switches for SQL*Forms (Run Form).
formname	Specifies the name of the form you want to execute.
username / password	Specifies your ORACLE username and password.

If you do not specify your username and password at the system prompt, SQL*Forms uses an OPS$ ID to login automatically. If ORACLE cannot find an OPS$ ID, it displays the Login screen which prompts you for a username and password.

You can type `RUNFORM ?` at the operating system prompt to receive syntax help.

Related Topics: Command line switches, SQL*Forms (Run Form) 23-15

Command Line Switches

Command line switches are options that you apply to a SQL*Forms session from the command line. All switches use the following syntax:

`-switch_designator argument`

where:

switch_designator	Specifies an alphabetic character that identifies the switch to use. Note that while the same switch designation sometimes appears across components, each switch has a unique purpose.
argument	Specifies a required argument for a particular switch. Not all switches take arguments.

Combining Command Line Switches

You can use most switches in combination with other switches. Some combinations are illogical and will cause an error if you try them.

To use more than one switch at a time, simply specify the switches in any order you want If you are using more than one switch and none of the switches requires an argument, you can combine the switches syntactically by using one minus sign in front of the combination and leaving no spaces between the switches. For example, combine the -t switch and the -o switch as -to.

Switches for SQL*Forms (Convert)

The following list provides an alphabetical list of command line switches for SQL*Forms (Convert) and a brief description of the purpose of each switch. The following pages provide more detailed descriptions.

Switch	*Description*
-c	Translate graphic features.
-d	Delete the form from the database.
-f	Preserve original functionality.
-i	Convert an INP file.
-s	Suppress the program banner.
-v	Create a new INP file in Version 3.0 format.

Related Topics: CONVFORM Command 23-4

-c Switch for SQL*Foms (Convert)

The -c switch translates the graphic features defined in an old INP file (created in a version of SQL*Forms prior to Version 2.3) into characters in a new INP file.

You should use the -c switch in conjunction with the -v switch when you are converting an INP file that was created prior to Version 2.3 to Version 3.0 status.

Syntax: `-c crt`

where `crt` is the name of the CRT definition file that you used to create the old INP file.

Note: When you use the -c switch, you (or your DBA) must ensure that the original CRT definition file is in a directory that is accessible to SQL*Forms (Convert).

-d Switch for SQL*Forms (Convert)

The -d switch deletes the specified form from the database. When you use the -d switch, SQL*Forms ignores all other switches for the session.

Syntax: `-d`

-f Switch for SQL*Forms (Convert)

The -f switch inserts the NOFAIL keyword after #EXEMACRO in all trigger steps. Use this switch when you are converting Version 2.0 forms to Version 3.0 status.

For more information on the NOFAIL keyword, refer to the *SQL*Forms Documentation Addendum* Version 2.3.

Syntax: `-f`

Related Topics: Converting existing forms to Version 3.0 status B-2

-i Switch for SQL*Forms (Convert)

The -i switch specifies that SQL*Forms (Convert) convert an INP file to database format. If you do not use the -i switch, SQL*Forms (Convert) defaults to converting a form in database format to an INP file.

Syntax: `-i`

-s Switch for SQL*Forms (Convert)

The -s switch suppresses the program banner and the `Reading. . .` and `Writing. . .` prompts.

Syntax: `-s`

-v Switch for SQL*Forms (Convert)

The -v switch creates an INP file in Version 3.0 format from an INP file in an older INP format.

Syntax: *`-v { version_ number} inp_file_name`*

where:

`version_ number` — Specifies the version number of the INP file that you want to convert. If you are converting an INP file created with Version 2.3 of SQL*Forms, specify `23` for `version_number`. If you are converting an INP file created prior to Version 2.3, specify `20`.

Note: There is no space between the -v and the version number (e.g., `-v23`).

`inp_file_name` — Specifies the name of the INP file that you want to convert to Version 3.0 format.

Note: Use the -c switch in conjunction with the -v switch when you are converting an INP file that was created prior to Version 2.3.

Related Topics: INP file format 24-3
Converting existing forms to Version 3.0
status B-2

Switches for SQL*Forms (Design)

The following list provides an alphabetical list of command line switches for SQL*Forms (Design) and a brief description of the purpose of each switch.

Switch	*Description*
-c	Specify a named mapping.
-e	Create a key script file.
-i	Capture terminal output interactively.
-l	Limit the undo buffer.
-n	Bypass the login screen (i.e., no login).
-r	Read a key script file as input.
-s	Suppress the status line. This switch is obsolete in Version 3.0 of SQL*Forms.
-w	Capture terminal output only in a file.

Related Topics: SQLFORMS command 23-5

-c Switch for SQL*Forms (Design)

The -c switch specifies a mapping other than the default mapping for the current device and product.

Syntax: `-c [filename:]mapping`

where:

filename	Is the name of your Oracle*Terminal resource file. If you do not specify *filename*, SQL*Forms defaults to ORATERM.R.
mapping	Is the name of the mapping that you want to use for this SQL*Forms session.

Note: You or the DBA define mappings with Oracle*Terminal.

Related Topics: Oracle*Terminal 25-10

-e Switch for SQL*Forms (Design)

The -e switch captures in a file the keystrokes involved during the SQL*Forms (Design) session. This file is called a key script file.

The key script file includes the keystrokes involved in navigating within the design interface, invoking functions, and running any form invoked from within SQL*Forms (Design).

Syntax: `-e filename`

where *filename* is the output, or key script file.

Related Topics: Edit keystroke facility 29-1

-i Switch for SQL*Forms (Design)

The -i switch captures the terminal output for a SQL*Forms (Design) session in a display log file. The -i switch also specifies that SQL*Forms display the terminal output on the terminal screen as well (i.e., run interactively).

Syntax: `-i` *`filename`*

where *`filename`* is the output, or display log, file.

Related Topics: Edit keystroke facility 29-1

-l Switch for SQL*Forms (Design)

The -l switch limits the undo buffer to hold only one operation (the most recent one). By default, this buffer has no logical limit. The -1 switch is especially useful with systems that have strict memory limitations.

If you want to use this feature from within the design interface, turn on the Limit Undo Buffer to 1 design option.

Syntax: `-1`

Related Topics: Limit Undo Buffer to 1 design option 26-6

-n Switch for SQL*Forms (Design)

The -n switch bypasses the SQL*Forms (Design) login screen. This feature allows you to connect to SQL*Forms (Design) without logging into a database. When you invoke SQL*Forms this way, you:

- can create and modify forms
- can generate forms that do not contain any of the following items:
  - V3 style triggers or procedures that issue SQL statements
  - references to sequence number objects in the database
  - referenced blocks, fields, triggers, and form-level procedures
  - V3 style list of values
  - blocks with the Column Sec block characteristic turned on
- cannot save changes to the database

- cannot use any database functions, such as creating default blocks or listing available forms

If you do specify a username/password combination, SQL*Forms ignores it.

Syntax: `-n`

-r Switch for SQL*Forms (Design)

The -r switch allows you to read a key script file into SQL*Forms (Design) as input. The key script file starts, executes, and ends the SQL*Forms (Design) session.

SQL*Forms performs all actions on the terminal screen, unless you suppress screen output with the -w switch.

Syntax: `-r filename`

where *filename* is the input, or key script, file.

Related Topics: Edit keystroke facility 29-1

-s Switch for SQL*Forms (Design)

Inversions of SQL*Forms prior to Version 3.0, the -s switch suppresses the status line. This -s switch has no effect in Version 3.0 of SQL*Forms. It exists only for upward compatibility.

Syntax: `-s`

-w Switch for SQL*Forms (Design)

The -w switch captures the terminal output for a SQL*Forms (Design) session in a display log file. The -w switch also specifies that SQL*Forms does not display the terminal output on the terminal screen.

You must use the -r switch whenever you use the -w switch.

Syntax: `-r filename1 -w filename2`

where:

filename1	Is the input, or key script, file.
filename2	Is the output, or display log, file.

Related Topics: Edit keystroke facility 29-1

Switches for SQL*Forms (Generate)

The following list provides an alphabetical list of command line switches for SQL*Forms (Generate) and a brief description of the purpose of each switch.

Switch	*Description*
-b	Suppress the banner and messages.
-n	Bypass the login screen (i.e., no login).
-o	Suppress the creation of a new INP file. This switch is obsolete in Version 3.0 of SQL*Forms.
-s	Suppress the dialog questions. This switch is obsolete in Version 3.0 of SQL*Forms.
-t	Suppress the banner and messages.

Related Topics: GENERATE command 23-6

-b Switch for SQL*Forms (Generate)

In Version 3.0 of SQL*Forms, the -b switch suppresses the SQL*Forms (Generate) banner and messages.

Inversions of SQL*Forms prior to Version 3.0, the -b switch invokes the "batch mode" for the SQL*Forms (Generate) session. In batch mode, SQL*Forms (Generate) does not supply any prompts or warning messages if it encounters an error. Instead, SQL*Forms (Generate) gives an error message and stops processing. This functionality is obsolete in Version 3.0 of SQL*Forms.

Syntax: `-b`

-n Switch for SQL*Forms (Generate)

The -n switch bypasses the need to specify a username/password combination. This feature allows you to connect to SQL*Forms (Generate) without logging in to a database. When you invoke SQL*Forms this way, you can generate any form that was created with a version of SQL*Forms prior to Version 3.0. You can also generate any form that was created with SQL*Forms Version 3.0 that does not contain any of the following items:

- V3 style triggers or procedures that issue SQL statements
- references to sequence number objects in the database
- referenced blocks, fields, triggers, and form-level procedures
- V3 style list of values
- blocks with the Column Sec block characteristic turned on

If you do specify a username/password combination, SQL*Forms ignores it.

Syntax: `-n`

-o Switch for SQL*Forms (Generate)

Inversions of SQL*Forms prior to Version 3.0, the -o switch prevents SQL*Forms from creating a new INP file for the form when it creates the FRM file.

The -o switch has no effect in Version 3.0 of SQL*Forms. It exists only for upward compatibility.

Syntax: `-o`

-s Switch for SQL*Forms (Generate)

Inversions of SQL*Forms prior to Version 3.0, the -s switch suppresses the dialog questions from the new INP file, recording only the answers. This has no effect on the functionality of the INP file. However, it does substantially reduce the size of the file, which might be useful if you transport the file to another operating system.

The -s switch has no effect in Version 3.0 of SQL*Forms. It exists only for upward compatibility.

Syntax: `-s`

-t Switch for SQL*Forms (Generate)

In Version 3.0 of SQL*Forms, the -t switch suppresses the SQL*Forms (Generate) banner and messages.

Inversions of SQL*Forms prior to Version 3.0, the -t switch prevents SQL*Forms from displaying to the screen the dialog by which the INP file was created. This functionality is obsolete in Version 3.0 of SQL*Forms.

Syntax: `-t`

Switches for SQL*Forms (Run Form)

The following list provides an alphabetical list of command line switches for SQL*Forms (Run Form) and a brief description of the purpose of each switch.

Switch	*Description*
-a	Suppress array processing.
-b	Set the number of records buffered.
-c	Specify a named mapping.
-d	Invoke debug mode.
-e	Create a key script file.
-i	Capture terminal output interactively.
-l	Disable automatic login.

-m	Display the block menu at login.
-o	Share cursors for V2 trigger SQL.
-q	Invoke quiet mode.
-r	Read a key script file as input.
-s	Display the number of cursors set and enable the SQL trace facility.
-t	Share cursors for implicit SQL.
-w	Capture terminal output only in a file.

Related Topics: RUNFORM command 23-7

-a Switch for SQL*Forms (Run Form)

The -a switch suppresses array processing during a SQL*Forms (Run Form) session. When you suppress array processing, SQL*Forms fetches one record at a time in response to a query, but waits to display those records all at once. (Use a Post-Query trigger with the SYNCHRONIZE packaged procedure to display each record as it is being fetched.)

Suppressing array processing usually results in the first retrieved record displaying faster than it would if you fetched a number of records with array processing. However, the total time required to fetch and display a number of records is shorter with array processing.

If you are running a form from within SQL*Forms (Design) and you want to use this feature, activate the Disable Array Processing run option.

Syntax: `-a`

Related Topics: Disable Array Processing run option 26-3

-b Switch for SQL*Forms (Run Form)

The -b switch sets the maximum number of records buffered in memory to the number of rows displayed plus some constant (for each block). This constant is equal to at least 3, but can be more depending on the operating system you are using. If a block retrieves any records by a query beyond its maximum, SQL*Forms buffers these additional records to a temporary file on disk.

If you are running a form from within SQL*Forms (Design) and you want to use this feature, activate the Buffer Record in File run option.

Syntax: `-b`

Related Topics: Buffer Record in File run option 26-2
Records Buffered block characteristic 11-15

-c Switch for SQL*Forms (Run Form)

The -c switch specifies a mapping other than the default mapping for the current device and product.

Syntax: `-c [ filename:] mapping`

where:

filename	Is the name of your Oracle*Terminal resource file. If you do not specify *filename,* SQL*Forms defaults to ORATERM.R.
mapping	Is the name of the mapping that you want to use for this SQL*Forms session.

Note: You or the DBA define mappings with Oracle*Terminal.

Related Topics: Oracle*Terminal 25-10

-d Switch for SQL*Forms (Run Form)

The -d switch invokes the debug mode for the SQL*Forms (Run Form) session. Debug mode displays ongoing messages about trigger execution and other states of the current form during execution. Debug mode invokes break processing if the BREAK packaged procedure is in any trigger.

If you are running a form from within SQL*Forms (Design) and you want to use this feature, activate the Debug Mode run option.

Syntax: `-d`

Relted Topics: Debug facility 28-1
Debug Mode run option 26-2

-e Switch for SQL*Forms (Run Form)

The -e switch captures in a key script file the keystrokes involved during the SQL*Forms (Run Form) session. The key script file includes the keystrokes involved in navigating within a form, invoking functions, and Performing transactions.

Syntax: `-e filename`

where *filename* is the output, or key script file.

Related Topics: Edit keystroke facility 29-1

-i Switch for SQL*Forms (Run Form)

The -i switch captures the terminal output for a form in a display log file. The -i switch also specifies that SQL*Forms display the terminal output on the terminal screen as well (i.e., run interactively).

Syntax: `-i filename`

where *filename* is the output, or display log, file.

Related Topics: Edit keystroke facility 29-1

-l Switch for SQL*Forms (Run Form)

The -l switch disables the ORACLE automatic login feature and forces the login screen to display. Do not specify a username and password when you use the -1 switch (SQL*Forms will ignore it if you do).

Use the -l switch when:

- you want to log in to SQL*Forms with a username other than your established OPS$ ID

and

- you do not want to type your password on the command line (where it is visible)

Syntax: `-1`

Related Topics: Automatic login 23-2

-m Switch for SQL*Forms (Run Form)

The -m switch causes SQL*Forms (Run Form) to display the block menu as the first screen (after the login screen, if it displays) instead of a page of the form. Use this feature for complex applications that have more than one potential starting point.

If you are running a form from within SQL*Forms (Design) and you want to use this feature, activate the Display Block Menu run option.

Syntax: `-m`

Related Topics: Block menu 11-13
Display Block Menu run option 26-3

-o Switch for SQL*Forms (Run Form)

By default, SQL*Forms assigns a separate database cursor for each SQL statement that a form executes explicitly in a V2 trigger. This behavior enhances processing because the statements in each cursor only need to be parsed the first time they are executed in a SQL*Forms (Run Form) session-not every time.

When you specify the -o switch, SQL*Forms assigns a single cursor for all SQL statements in V2 triggers. These statements share, or reuse, that cursor. This behavior saves memory, but slows processing because the SQL statements must be parsed every time they are executed.

You can fine tune this behavior through the New Cursor Area trigger step characteristic. If a trigger step that contains a SQL statement has this characteristic turned on, SQL*Forms assigns a separate cursor to the statement, in effect overriding the -o switch for that statement.

If you are running a form from within SQL*Forms (Design) and you want to use this feature, turn on the Share Cursors for Trigger SQL run option.

Note: The -o switch has no effect on statements in V3 triggers.

Syntax: `-o`

Related Topics Cursor, database 22-3
New Cursor Area trigger step characteristic F-6
Share Cursors for Trigger SQL run option 26-4
-t switch, SQL*Forms (Run Form) 23-20

-q Switch for SQL*Forms (Run Form)

The -q switch invokes the quiet mode for the SQL*Forms (Run Form) session. In quiet mode, messages do not ring the bell (i.e., do not produce an audible beep). However, you can still explicitly ring the bell from a trigger via the BELL packaged procedure.

If you are running a form from within SQL*Forms (Design) and you want to use this feature, activate the Quiet Mode run option.

Syntax: `-q`

Related Topics: Quiet Mode run option 26-3

-r Switch for SQL*Forms (Run Form)

The -r switch allows you to read a key script file into a form as input. The key script file starts, executes, and ends the SQL*Forms (Run Form) session.

SQL*Forms performs all actions on the terminal screen, unless you suppress screen output with the -w switch.

Syntax: `-r` *`filename`*

where *`filename`* is the input, or key script, file.

Related Topics: Edit keystroke facility 29-1

-s Switch for SQL*Forms (Run Form)

The -s switch displays a message at the end of the SQL*Forms (Run Form) session that states the maximum number of simultaneous cursors that were used during the session. SQL*Forms writes this message to the terminal screen, not to the message line.

The -s switch also issues the following command to the database:

```
ALTER SESSION SET SQL_TRACE TRUE
```

This command enables the SQL trace facility for the current SQL*Forms (Run Form) session. For more information on this facility-which gathers database performance information-refer to the *ORACLE RDBMS Performance Tuning Guide*.

If you are running a form from within SQL*Forms (Design) and you want to use this feature, activate the Statistics run option.

Syntax: `-s`

Related Topics: Statistics run option 26-5

-t Switch for SQL*Forms (Run Form)

By default, SQL*Forms assigns a separate database cursor for each SQL statement that a form executes implicitly as part of posting or querying data. This behavior enhances processing because the statements in each cursor only need to be parsed the first time they are executed in a SQL*Forms (Run Form) session-not every time.

When you specify the -t switch, SQL*Forms only assigns a separate cursor for each query SELECT statement. All other implicit SQL statements share, or reuse, cursors. This behavior saves memory but slows processing because all INSERT, UPDATE, DELETE, and SELECT FOR UPDATE statements must be parsed every time they are executed.

If you are running a form from within SQL*Forms (Design) and you want to use this feature, turn on the Share Cursors for Implicit SQL run option.

Note: The -t switch has no effect if you replace standard SQL*Forms processing with On-Insert, On-Update, and On-Delete triggers because these triggers replace the implicit issuance of INSERT, UPDATE, and DELETE statements.

Syntax: `-t`

Related Topics: Cursor, database 22-3
-o switch, SQL*Forms (Run Form) 23-18
Share Cursors for Implicit SQL run option 26-4

-w Switch for SQL*Forms (Run Form)

The -w switch captures the terminal output for a form in a display log file. The -w switch also specifies that SQL*Forms does not display the terminal output on the terminal screen.

Note: You must use the -r switch whenever you use the -w switch.

Syntax: *-r filenamel -w filename2*

where:

filenamel	Specifies the input, or key script, file.
filename2	Specifies the output, or display log, file.

Related Topics: Edit keystroke facility 29-1

CHAPTER

24 FORM STORAGE

This chapter discusses how SQL*Forms stores forms. Specifically, it covers the following topics:

- form storage formats
- portability among systems
- SQL*Forms (Design) base tables
- INP file format

Form Storage Formats

SQL*Forms stores forms in three formats:

- database format
- FRM format
- INP format

You can convert a form from one format to another by using the various components of SQL*Forms.

Related Topics: Components 23-1

Database Format

A form in database format consists of the form definition stored in several tables in the ORACLE database. This is the basic storage format for forms. SQL*Forms creates the database representation of a form when you perform either of the following actions:

- save the form definition from SQL*Forms (Design) via the Save option on the Action menu
- convert an INP file from the command line via the CONVFORM command with the -i command line switch

SQL*Forms uses the database representation of the form when you open the form from SQL*Forms (Design) via the Open option on the Action menu.

You can delete the database representation of a form by using the CONVFORM command with the -d command line switch.

Note: Prior to Version 1.0, SQL*Forms did not store forms in the database, but rather used the INP format as the basic storage format.

Related Topics: CONVFORM command 23-4
-d switch, SQL*Forms (Convert) 23-9
-i switch, SQL*Forms (Convert) 23-10

FRM Format

A form in FRM format consists of the form definition stored in a binary file (normally with the .FRM extension). SQL*Forms creates the FRM file for a form when you generate the form from SQL*Forms (Design) via the Generate option on the Action menu or from the command line via the GENERATE command. SQL*Forms uses the FRM file when an operator runs the form.

Note: A file in FRM format is not portable between systems.

Related Topics: GENERATE command 23-6

INP Format

A form in the INP format consists of the form definition stored in a text file (normally with the .INP extension). SQL*Forms creates the INP file when you perform either of the following actions:

- generate a form from SQL*Forms (Design) via the Generate option on the Action menu
- convert the database representation of a form from the command line via the CONVFORM command

SQL*Forms *uses* the INP file when you perform either of the following actions:

- load the form in SQL*Forms (Design) via the Load option on the Action menu
- generate the form from SQL*Forms (Design) via the Generate option on the Action menu or from the command line via the GENERATE command

The INP format allows a form to be portable among systems.

Related Topics: CONVFORM command 23-4
GENERATE command 23-6

Portability among Systems

Database format and INP file format are the same on every type of computer system that supports SQL*Forms; FRM format is not. Therefore, do not try to transport a form from one type of computer to another by transporting the FRM file. Instead, copy the INP file and perform one of the following actions:

- If you want to modify the form, load the INP file into SQL*Forms (Design).
- If you want to save the form in the database, convert the INP file from the command line via the CONVFORM command with the -i command line switch.
- If you want to run the form, generate the form to make a new FRM file. If you are going to generate the form from SQL*Forms (Design), you will have to load the form first.

Also, do not import or export the IAP tables unless the target system has no forms in the database.

Related Topics: CONVFORM Command 23-4
-i switch, SQL*Forms (Convert) 23-10

SQL*Forms (Design) Base Tables

This section describes the database tables used in Version 3.0 of SQL*Forms (Design) that store data on the forms you create and modify. These tables are normally owned by the username SYSTEM.

- FORM_APP
- FORM_BLK
- FORM_COMMENT
- FORM_FLD
- FORM_MAP
- FORM_PAGE
- FORM_PROCEDURE
- FORM_REFERENCE
- FORM_SQLTXT
- FORM_TRG
- FORM_TRIGGER
- FORM_USER

The table descriptions appear for your information. You can write reports based on this information or change the contents of the tables through SQL*Forms. Do *not* manipulate these tables through SQL*Plus.

Note: Oracle Corporation reserves the right to change the names or structures of these tables at any time and without notice.

FORM_APP Table

The FORM_APP table contains information on form identification. The following table provides information about the columns of the table.

Column	*Description*	*Null?*	*Type*
APPOWNER	form owner	NOT NULL	CHAR(30)
APPNAME	form name	NOT NULL	CHAR(30)
APPTITLE	form title		CHAR(80)
APPVAUNTT	validation unit		CHAR(6)
APPMOUSE	mouse navigation limit		CHAR(6)
APPMNUAPL	default menu application		CHAR(30)
APPMNUNAM	starting menu name		CHAR(30)
APPMNUGRP	menu security group name		CHAR(30)

FORM_BLK Table

The FORM_BLK table contains information on block identification and block characteristics. The following table provides information about the columns of the table.

Column	*Description*	*Null?*	Type
BLKAPOWNER	form owner	NOT NULL	CHAR(30)
BLKAPNAME	form name	NOT NULL	CHAR(30)
BLKNAME	block name	NOT NULL	CHAR(30)
BLKDESC	block description		CHAR(60)
BLKHIDE	if the description should appear in the block menu		CHAR(l)
BLKSEQ	block sequence number		NUMBER
BLKUNQKEY	if unique keys should be enforced for the block		CHAR(1)
BLKCTRL	if the block has a base table		CHAR(l)
BLKCOLSEC	if database constraints should be enforced		CHAR(l)
BLKTOWNER	table owner		CHAR(30)
BLKTNAME	table name		CHAR(30)
BLKNOREC	number of records to display		NUMBER
BLKNOBUF	number of records to buffer		NUMBER
BLKBLIN	base line		NUMBER
BLKLNRC	number of lines per record		NUMBER
BLKARSIZ	array size for the block		NUMBER
BLKOBYSQL	link to the FORM_SQLTXT table (SQTNO column) to store the default WHERE / ORDER BY clause for the block		NUMBER

FORM_COMMENT Table

The FORM_COMMENT table contains comment text. The following table provides information about the columns of the table.

Column	*Description*	*Null?*	*Type*
CMTAPOWNER	form owner	NOT NULL	CHAR(30)
CMTAPNAME	form name	NOT NULL	CHAR(30)
CMTPROC	procedure name		CHAR(30)
CMTBLK	block		CHAR(30)
CMTFLD	field		CHAR(30)
CMTTRGTYP	trigger		CHAR(30)
CMTTRGSEQ	trigger step		NUMBER
CMTLINE	line number		NUMBER
CMTTEXT	comment text		CHAR(80)

FORM_FLD Table

The FORM_FLD table contains information on fields and their characteristics. The following table provides information about the columns of the table.

Column	*Description*	*Null?*	*Type*
FLDAPOWNER	form owner	NOT NULL	CHAR(30)
FLDAPNAME	form name	NOT NULL	CHAR(30)
FLDBLK	block that owns the field	NOT NULL	CHAR(30)
FLDNAME	field name	NOT NULL	CHAR(30)
FLDSEQ	sequence number		NUMBER
FLDTYPE	data type		CHAR(8)
FLDLEN	field length		NUMBER
FLDDLEN	display length		NUMBER
FLDQLEN	query length		NUMBER
FLDBTAB	if the field is a base table field		CHAR(1)
FLDKEY	if the field is a primary key field		CHAR(1)
FLDCKBLK	block from which to copy the field's value		CHAR(30)
FLDCKFLD	field from which to copy the field's value		CHAR(30)

Table continued on next page.

Column	*Description*	*Null?*	*Type*
FLDDFLT	default value		CHAR(80)
FLDDISP	if the field is displayed		CHAR(l)
FLDPAGE	page location		NUMBER
FLDLINE	line location		NUMBER
FLDCOL	column location		NUMBER
FLDPROMPT	prompt		CHAR(80)
FLDPRABOV	if the prompt is above the field		CHAR(l)
FLDPRRPT	if the prompt is beside the field		CHAR(1)
FLDENTER	if the field is enterable		CHAR(1)
FLDQUERY	if the field can be queried		CHAR(l)
FLDUPDATE	if the field can be updated		CHAR(l)
FLDUPDNUL	if the field can be updated when NULL		CHAR(1)
FLDMAND	if the field is required		CHAR(l)
FLDFIXED	if the field has a fixed length		CHAR(l)
FLDSKIP	if the field has automatic skip		CHAR(l)
FLDHIDE	if the field is hidden		CHAR(l)
FLDAUTOHLP	if the field has automatic help		CHAR(l)
FLDUPPER	if the field converts to uppercase characters		CHAR(l)
FLDLOVTTT	title for list of values		CHAR(60)
FLDLOVSQL	Link to the FORM_SQLTXT table to store the list of values SQL statement		NUMBER
FLDLOVX	X coordinate for list of values		NUMBER
FLDLOVY	Y coordinate for list of values		NUMBER
FLDLOW	low range value		CHAR(30)
FLDHI	high range value		CHAR(30)
FLDVALPATN	validation pattern		CHAR(30)

Table continued on next page.

Column	*Description*	*Null?*	*Type*
FLDDISMASK	display mask		CHAR(80)
FLDHELP	help text		CHAR(80)
FLDEDTX	X coordinate for field editor		NUMBER
FLDEDTY	Y coordinate for field editor		NUMBER
FLDEDTWRP	word wrap for field editor		CHAR(1)

FORM_MAP Table

The FORM_MAP table contains information on constant text and graphics. The following table provides information about the columns of the table.

Column	*Description*	*Null?*	*Type*
MAPAPOWNER	form owner	NOT NULL	CHAR(30)
MAPAPNAME	form name	NOT NULL	CHAR(30)
MAPPAGE	page	NOT NULL	NUMBER
MAPLINE	line		NUMBER
MAPGRPH	if the values should be interpreted as graphic elements		CHAR(1)
MAPTEXT	constant text		CHAR(255)

FORM_PAGE Table

The FORM_PAGE table contains information on pages and their characteristics. The following table provides information about the columns of the table.

Column	*Description*	*Null?*	*Type*
PAGAPOWNER	form owner	NOT NULL	CHAR(30)
PAGAPNAME	form name	NOT NULL	CHAR(30)
PAGNUM	page number	NOT NULL	NUMBER
PAGTITLE	page title		CHAR(255)
PAGXSIZ	width of page		NUMBER
PAGYSIZ	height of page		NUMBER
PAGVWXS	width of view		NUMBER
PAGVWYS	height of view		NUMBER
PAGVWPX0	X coordinate of view position on page		NUMBER
PAGVWPY0	Y coordinate of view position on page		NUMBER
PAGVWSX0	X coordinate of view position on screen		NUMBER
PAGVWSY0	Y coordinate of view position on screen		NUMBER
PAGPOPUP	if the page is a pop-up page		CHAR(l)
PAGBORDER	if the view has a border		CHAR(l)
PAGDISLV	if the page dissolves when exited		CHAR(l)
PAGHSCRL	if the view has a horizontal scroll bar		CHAR(l)
PAGVSCRL	if the view has a vertical scroll bar		CHAR(l)

FORM_PROCEDURE Table

The FORM_PROCEDURE table contains information on PL/SQL procedures. The following table provides information about the columns of the table.

Column	*Description*	*Null?*	*Type*
PROCAPOWNER	form owner	NOT NULL	CHAR(30)
PROCAPNAME	form name	NOT NULL	CHAR(30)
PROCNAME	procedure name	NOT NULL	CHAR(30)
PROCTEXT	link to the FORM_SQLTXT table (column SQTNO) to store the procedure text		NUMBER

FORM_REFERENCE Table

The FORM_REFERENCE table contains information on all references to SQL*Forms objects. The following table provides information about the columns of the table.

Column	*Description*	*Null?*	*Type*
REFAPOWNER	form owner	NOT NULL	CHAR(30)
REFAPNAME	form name	NOT NULL	CHAR(30)
REFBLK	referencing block		CHAR(30)
REFFELD	referencing field		CHAR(30)
REFTRGTYP	referencing trigger		CHAR(30)
REFPROC	referencing procedure		CHAR(30)
REFFOWNER	owner of referenced object	NOT NULL	CHAR(30)
REFFAPP	owner of the form that contains the referenced object	NOT NULL	CHAR(30)
REFFBLK	referenced block		CHAR(30)
REFFFLD	referenced field		CHAR(30)
REFFTRGTYP	referenced trigger		CHAR(30)
REFFPROC	referenced procedure		CHAR(30)

FORM_SQLTXT Table

The FORM_SQLTXT table contains the text of all SQL statements contained in your forms, including default WHERE/ORDER BY clauses. The following table provides information about the columns of the table.

Column	*Description*	*Null?*	*Type*
SQTAPOWNER	form owner	NOT NULL	CHAR(30)
SQTAPNAME	form name	NOT NULL	CHAR(30)
SQTNO	identifier number for the SQL text	NOT NULL	NUMBER
SQTLINE	line of text	NOT NULL	NUMBER
SQTTEXT	SQL text		CHAR(80)

FORM_TRG Table

The FORM_TRG table contains information on the location and characteristics of trigger steps. The following table provides information about the columns of the table.

Column	*Description*	*Null?*	*Type*
TRGAPOWNER	form owner	NOT NULL	CHAR(30)
TRGAPNAME	form name	NOT NULL	CHAR(30)
TRGBLK	block with which the trigger is associated		CHAR(30)
TRGFLD	field with which the trigger is associated		CHAR(30)
TRGTYPE	type of trigger	NOT NULL	CHAR(30)
TRGSEQ	sequence number		NUMBER
TRGLABEL	label of trigger step		CHAR(30)
TRGSQL	SQL text number the step is tied to (as described in the FORM_SQLTXT table)		NUMBER
TRGCURS	if the trigger step should have a separate cursor		CHAR(l)
TRGMVE	if the trigger should abort when the step fails		CHAR(l)

Table continued on next page.

Column	*Description*	*Null?*	*Type*
TRGINV	if the return code for trigger step should be reversed		CHAR(1)
TRGROLL	if the trigger should return success when aborting the trigger		CHAR(1)
TRGSLAB	success label		CHAR(30)
TRGFLAB	failure label		CHAR(30)
TRGMSG	failure message line		CHAR(80)

FORM_TRIGGER Table

The FORM_TRIGGER table contains information on the location and characteristics of triggers. The following table provides information about the columns of the table.

Column	*Description*	*Null?*	*Type*
TRIGAPOWNER	form owner	NOT NULL	CHAR(30)
TRIGAPNAME	form name	NOT NULL	CHAR(30)
TRIGBLK	block		CHAR(30)
TRIGFLD	field		CHAR(30)
TRIGTYPE	type of trigger	NOT NULL	CHAR(30)
TRIGDESC	description		CHAR(20)
TRIGHIDE	if the description should be included on the Show Keys screen (only for key triggers)		CHAR(1)
TRIGPLSQL	if trigger is a V3 trigger		CHAR(1)

FORM_USER Table

The FORM_USER table contains information on user privileges. The following table provides information about the columns of the table.

Column	*Description*	*Null?*	*Type*
USRAPOWNER	form owner	NOT NULL	CHAR(30)
USRAPNAME	form name	NOT NULL	CHAR(30)
USRUSER	grantee	NOT NULL	CHAR(30)

INP File Format

The complete syntax description of an INP file appears on the following pages. This material appears only as reference information. While you can modify a form by editing its INP file, doing so can cause severe damage to your applications.

Note: Oracle Corporation does not support the editing of INP files.

The following syntax diagram starts at a high level and works down to the lowest level of definition. Any term that appears in angled brackets (e.g., <form_definition>) is defined at a lower level of the diagram.

```
copyright notice
SQL* FORMS_VERSION = forms_version
<form_definition>
```

where <form_definition> has the following syntax:

```
DEFINE FORM
        application-level_comments
        NAME = form_name
        TITLE = form_ title
        DEFAULT_MENU_APPLICATION = default_menu_application
        ROOT_MENU = starting_menu_name
        GROUP_NAME = menu_ security_group_name
        MOUSE_NAVIGATION_LIMIT = mouse_navigation_limit
        VALIDATION_UNIT = {nothing |FIELD |RECORD |BLOCK |FORM}
        for each procedure in form
                <procedure_definition>
        for each form-level trigger in form
                <trigger_definition>
        for each block in form
                <block_definition>
        <screen_definition>
ENDDEFINE FORM
```

where <procedure_definition> has the following syntax:

```
<non-referenced_procedure_definition> |
<referenced_procedure_definition>
```

where <non-referenced_procedure_definition> has the following syntax:

```
DEFINE PROCEDURE
        procedure-level_comments
        NAME = procedure_name
        DEFINITION = procedure_text
ENDDEFINE PROCEDURE
```

where <referenced_procedure_definition> has the following syntax:

```
DEFINE  PROCEDURE
        NAME = procedure_name
        <reference_definition>
ENDDEFINE  PROCEDURE
```

where <trigger_definition> has the following syntax:

```
<non-referenced_trigger_definition>  |
<referenced_trigger_definition>
```

where <non-referenced_trigger_definition> has the following syntax.

```
<V2_style_trigger_definition>  |
<V3_style_trigger_definition>
```

where <V2_style_trigger_definition> has the following syntax:

```
DEFINE  TRIGGER
        trigger-level_comments
        NAME = trigger_name
        TRIGGER_TYPE = V2
        SHOW_KEY = {ON | OFF}
        DESCRIPTION = trigger_description
        for each step in trigger
                <step_definition>
ENDDEFINE  TRIGGER
```

where <step_definition> has the following syntax:

```
DEFINE STEP
        step-level_comments
        LABEL = step_label
        TEXT = step_text
        SUCCESS_STEP = success_label
        FAILURE_STEP = failure_label
        FAIL_MESSAGE = failure_message
        ABORT = { ON | OFF}
        REVERSE = {ON | OFF }
        NEW_CURSOR = {ON | OFF }
        IGNORE = {ON | OFF }
ENDDEFINE  STEP
```

where <V3_Style_trigger_definition> has the following syntax:

```
DEFINE TRIGGER
        trigger -level_ comments
        NAME = trigger_ name
        TRIGGER_TYPE = V3
        SHOW_KEY = { ON|OFF}
        DESCRIPTION = trigger_description
        TEXT = V3_style_trigger_text
ENDDEFINE TRIGGER
```

where <referenced_trigger_definition>has the following syntax:

```
DEFINE TRIGGER
        NAME = trigger_name
        <reference_definition>
ENDDEFINE TRIGGER
```

where <block_definition> has the following syntax:

```
<non-referenced_block_definition> |
<referenced_block_definition>
```

where <non-referenced_block_definition> has the following syntax:

```
DEFINE BLOCK
        block-level_comments
        NAME = block_name
        DESCRIPTION = block_description
        TABLE = user.base_table
        UNIQUE_KEY = {ON |OFF}
        UPDATE_KEY = {ON |OFF }
        IN_MENU = { ON|OFF}
        ROWS_DISPLAYED = num_rows_displayed
        ROWS_BUFFERED = num_rows_buffered
        BASE_LINE = base_line
        LINES_PER_ROW = num_lines_per_row
        ARRAY_SIZE = array_size
        COLUMN_SECURITY = {ON |OFF }
        ORDERING = default_where_orderby_clause
        for each block-level trigger in block
                <trigger_definition>
        for each field in block
                <field_definition>
        ENDDEFINE BLOCK
```

where <referenced_block_definition> has the following syntax:

```
DEFINE BLOCK
        NAME = block_name
        <reference_definition>
ENDDEFINE BLOCK
```

where <field_definition> has the following syntax:

```
<non-referenced_field_definition> |
<referenced_field_definition>
```

where <non-referenced_field_definition> has the following syntax:

```
DEFINE  FIELD
        field-level_comments
        NAME = field_name
        DATATYPE = data_type
        LENGTH = field_length
        DISPLAY_LENGTH = display_length
        QUERY_LENGTH = query_length
        BASE_TABLE =  { ON | OFF}
        PRIMARY_KEY = {ON | OFF }
        MANDATORY = {ON | OFF }
        DISPLAYED = { ON | OFF}
        PAGE = page_number
        LINE = line_number
        COLUMN = column_number
        HELP = help_message
        LOW_VALUE = field_range_low_value
        HIGH_VALUE = field_range_high_value
        LOV_TEXT = list_of_values_select_statement  |
                [user].table.column
        LOV_TITLE = list_of_values_title
        LOV_X = x_coord_for_lov
        LOV_Y = y_coord_for_lov
        INPUT_MASK = input_mask
        OUTPUT_MASK = output_mask
        EDIT_X = location_x_coordinate
        EDIT_Y = location_y_coordinate
        EDIT_WORD_WRAP = { ON | OFF}
        DEFAULT = default_value
        ENFORCE_KEY_FROM = source_block.field_of_primary_key_copy
        ECHO = { ON | OFF}
        INPUT = { ON | OFF}
        UPDATE = { On | OFF}
        UPDATE_NULL = {ON | OFF }
        QUERY = { ON | OFF}
        FIXED_LENGTH = {ON | OFF}
        AUTOSKIP = {ON | OFF }
        AUTOHELP = {ON | OFF }
        UPPERCASE = {ON | OFF }
        for each field-level trigger in field
                <trigger_definition>
ENDDEFINE FIELD
```

where <referenced_field_definition> has the following syntax:

```
DEFINE FIELD
        NAME = field_name
        PAGE = page_number
        LINE = line_number
        COLUMN = column_number
        <reference_definition>
ENDDEFINE field
```

where <reference_definition> has the following syntax:

```
DEFINE REFERENCE
        OWNER = owner_name
        APPLICATION = application_name
        BLOCK = block_name
        FIELD = field_name
        TRIGGER = trigger_name
        PROCEDURE = procedure_name
ENDDEFINE REFERENCE
```

where <screen_definition> has the following syntax:

```
DEFINE SCREEN
        for each page definition
                <page_definition>
ENDDEFINE SCREEN
```

where <page_definition> has the following syntax:

```
DEFINE PAGE
        PAGE = page_number
        POPUP = {ON | OFF }
        BORDER = {ON | OFF }
        HORIZONTAL_SCROLL_BAR = {ON | OFF }
        VERTICAL_SCROLL_BAR = {ON | OFF }
        DISSOLVE = {ON | OFF }
        TITLE = page_title
        PAGE_XS = page_x_size
        PAGE_YS = page_y_size
        PAGE_PX0 = x_coord_of_view_on_page
        PAGE_PY0 = y_coord_of_view_on_page
        PAGE_PXS = x_size_of_view_on_page
        PAGE_PYS = y_size_of_view_on_page
        PAGE_SX0 = x_coord_of_view_on_screen
        PAGE_SY0 = y_coord_of_view_on_screen
        MODE = TEXT
        LINE = line_number_a
        BOILER = <<<
        constant text at line_number
        constant text at line_number + 1
        ...
        >>>
        LINE = line_number_b
        BOILER = <<<
        ...
        >>>
        ...
        MODE = BOX
        LINE = line_number_c
        BOILER = <<<
        constant text at line_number
        constant text at line_number + 1
        ..
        >>>
        LINE = line_number_d
        BOILER = <<<
        ...
        >>>
        ...
ENDDEFINE PAGE
```

Note: SQL*Forms accepts any KEYWORD=VALUE statement in either of the following formats:

```
1. KEYWORD = VALUE

2. KEYWORD = <<<
             VALUE
             >>>
```

Most KEYWORD = VALUE statements are written in format 1. The exceptions, which are written in format 2, are the statements that define the following items:

- list of values text
- procedure definition
- trigger text
- step text
- default WHERE / ORDER BY clause

CHAPTER

25

CONNECTING TO OTHER PRODUCTS

This chapter discusses how to connect to a number of ORACLE products from a SQL*Forms application. Specifically, it covers the following topics:

- connecting to the ORACLE RDBMS
- connecting to application development tools
- using ORACLE Precompilers with SQL*Forms
- using callable forms
- using Oracle*Terminal

Connecting to the ORACLE RDBMS

SQL*Forms is fully integrated with Version 6.0 of the ORACLE RDBMS. All SQL*Forms applications are based on data in the ORACLE RDBMS. This data resides in tables within the data dictionary.

SQL*Forms accesses data in the database to which you log in. By default, SQL*Forms logs in to the local database on your machine. However, you can connect to a remote database through SQL*Net.

When you create a default block, the data dictionary provides SQL*Forms with the data to create that block. By default, SQL*Forms performs the following actions:

- derives field names from column names in the database tables (i.e., specifies the Field Name field characteristic)
- derives field lengths and display lengths from ORACLE data types and column widths (i.e., specifies the Field Length and Display Length field characteristics)
- translates database data types into SQL*Forms data types (i.e., specifies the Data Type field characteristic)
- designates fields that correspond to NOT NULL database columns as required fields (i.e., assigns the Required field characteristic)

In addition, SQL*Forms can perform the following actions:

- define defaults and valid ranges for fields from database table and column constraints (i.e., write On-New-Record, On-Validate-Field, and On-Validate-Record triggers)
- translate referential constraints (i.e., constraints in database tables that reference columns, other tables, etc.) into the master-detail relationship between blocks

Related Topics: Remote login 23-2

Connecting to Application Development Tools

SQL*Forms can connect to the following application tools:

- SQL*Menu
- SQL*Plus
- SQL*ReportWriter

Connecting to SQL*Menu

You can invoke Version 5.0 of SQL*Menu through SQL*Forms to incorporate menus into any SQL*Forms application. Specifically, SQL*Forms allows you to perform the following tasks:

- specify a default relationship between a SQL*Forms application and a SQL*Menu application menu
- show, hide, and replace menus through SQL*Forms packaged procedures
- pass data between a form and a menu

For details on how to set up or run SQL*Menu applications, refer to the *SQL*Menu User's Guide and Reference.*

Specifying Default Menu Information

You specify a default relationship between a SQL*Forms application and a SQL*Menu application through the Form Definition form. The following form characteristics determine the specifics of the relationship:

Default Menu Application	Specifies the menu application that SQL*Forms should invoke whenever the current form is executed.
Starting Menu Name	Specifies the menu that SQL*Forms should display when the current form is executed.
Menu Security Group	Specifies the privilege group that SQL*Menu should use for the menu application, which determines the menu options that will display.

When a form executes, SQL*Menu invokes the default menu application and displays the starting menu across the top of the screen (overlaying part of the form display), showing the options that are allowed for the specified security group.

Related Topics: Default Menu Application form characteristic 9-3
Menu Security Group form characteristic 9-4
Starting Menu Name form characteristic 9-5

Controlling Menus through Packaged Procedures

SQL*Forms provides the following packaged procedures, which allow you to show, hide, and replace the active SQL*Menu menu in a SQL*Forms application.

HIDE_MENU	Makes the current menu disappear.
SHOW_MENU	Displays the current menu.
REPLACE_ MENU	Replaces the current menu, and allows you to change the way the menu displays and the security group that SQL*Menu uses to determine which menu options to display.

With these packaged procedures, you can control the display of menus during a SQL*Forms (Run Form) session through triggers, or even give that control to operators through redefined function keys.

Example The following key trigger executes a query in the ORDERS block and then invokes the DETAIL menu:

```
EXECUTE_QUERY;
REPLACE_MENU ( ' DETAIL ' ) ;
GO_ BLOCK ( ' ORDERS ' ) ;
```

When the trigger invokes the DETAIL menu, one of the options that displays is "Show Associated Item Records." This option specifies the following trigger text:

```
GO_ BLOCK ( ' ITEMS ' ) ;
EXECUTE_QUERY  ;
```

If the operator chooses "Show Associated Item Records," SQL*Menu passes the trigger text to SQL*Forms to execute. The text executes as a part of the original key trigger that invoked the menu. When the passed text has executed, SQL*Forms continues processing the rest of the key trigger. That is, after SQL*Forms queries the ITEMS block it moves the cursor back to the ORDERS block.

Related Topics: HIDE_MENU, packaged procedure 16-21
REPLACE_MENU, packaged procedure 16-27
SHOW_MENU, packaged procedure 16-33

Passing Data between Forms and Menus

A form and a menu can exchange data through SQL*Forms global variables and system variables. Specifically, forms and menus can manipulate the same global variables and system variables during a joint session of SQL*Forms (Run Form) and SQL*Menu.

Default Menu Application

SQL*Forms provides a default menu application. This application is designed to let operators perform most SQL*Forms (Run Form) functions without pressing function keys. Because this default menu application is included with SQL*Forms, you can use it in your SQL*Forms applications even if you do not have access to SQL*Menu.

Application Description: The title of this application is the name of the form that invokes it. The main menu consists of the following options.

Action

Invokes a sub-menu that provides choices that are associated with general function keys:

Choice	*Function Key*
Rollback	[Clear Form/Rollback]
Commit	[Commit]
reFresh	[Refresh]
Print	[Print]
Exit	[Exit/Cancel]

The Commit and Exit choices are not available in Enter Query mode.

Edit

Invokes a sub-menu that provides choices that are associated with editing function keys:

Choice	*Function Key*
Cut	[Cut]
cOpy	[Copy]
Paste	[Paste]
Edit	[Edit]

All choices are only available in Edit mode.

Block

Invokes a sub-menu that provides choices that are associated with block operation function keys:

Choice	*Function Key*
Previous	[Previous Block]
Next	[Next Block]
Clear	[Clear Block]

The Previous and Next choices are not available in Enter Query mode.

Record

Invokes a sub-menu that provides choices that are associated with record operation function keys.

Choice	*Function Key*
Previous	[Previous Record]
Next	[Next Record]
Scroll Up	[Scroll Up]
Scroll down	[Scroll Down]
Clear	[Clear Record]
Remove	[Delete Record]
Insert	[Insert Record]
Duplicate	[Duplicate Record]
Lock	[Lock Record]

Field

Invokes a sub-menu that provides choices that are associated with field operation function keys:

Choice	*Function Key*
Previous	[Previous Field]
Next	[Next Field]
Clear	[Clear Field]
Duplicate	[Duplicate Field]

Query

Invokes a sub-menu that provides choices that are associated with query function keys:

Choice	*Function Key*
Enter	[Enter Query]
eXecute	[Execute Query]
Last criteria	[Enter Query]
Cancel	[Exit/Cancel]
count Hits	[Count Query Hits]
Fetch next set	[Next Set of Records]

The Last criteria and Cancel choices are only available in Enter Query mode.

Help

Invokes a sub-menu that provides choices that are associated with help function keys:

Choice	*Function Key*
Help	[Help]
Keys	[Show Keys]
List	[ListJ
Error	[Display Error]

When an operator selects a choice from the default menu, SQL*Forms performs exactly as if the operator pressed the corresponding function key. This behavior includes the execution of key triggers.

Note that not all SQL*Forms (Run Form) function keys are mapped to menu choices. Most of the omitted function keys are cursor movement keys.

Using the Application: You can use this default menu in your applications by specifying DEFAULT for the Default Menu Application form characteristic. The menu displays by default when you execute the form.

Note: When you create a new form, SQL*Forms specifies DEFAULT as the default value for the Default Menu Application form characteristic.

Connecting to SQL*Plus

You can invoke Version 3.0 of SQL*Plus through SQL*Forms to take advantage of the power of SQL. In addition, you can invoke a SQL*Forms application through SQL*Plus.

Invoking SQL*Plus through SQL*Forms

To invoke SQL*Plus from SQL*Forms, use the HOST packaged procedure as follows

```
HOST  ('SQLPLUS_command');
```

where *SQLPLUS_command* specifies a standard SQL*Plus login statement. For details, refer to the *SQL*Plus Reference Manual.*

Examples

1. The following statement starts SQL*Plus and runs a command script called ORDERRPT:

```
HOST ( ' SQLPLUS scott/tiger @ORDERRPT' ) ;
```

2. In the following statement, SQL*Forms passes values to SQL*Plus for the SQL*Plus parameters &1, &2.

```
HOST( 'SQLPLUS scott/tiger @ORDERRPT 10 20 . . . ' ) ;
```

3. You can invoke SQL*Plus by calling a variable. To do so, you must first define the variable. The following PL/SQL statement defines a SQL*Forms variable called my_var and uses variables to pass dynamic parameter values:

```
my_var : = 'SQLPLUS scott/tiger @ORDERRPT ' ||global. var_l ||
' ' ||global.var_2 . . . ;
```

Once you have defined my_var, your trigger can invoke SQL*Plus with the following statement:

```
HOST (my_var ) ;
```

Related Topics: HOST, packaged procedure 16-22

Invoking SQL*Forms through SQL*Plus

You can invoke SQL*Forms (Run Form) from within SQL*Plus. This means that you can run a SQL*Forms application. To do so, use the RUNFORM command. Note that you can only use this feature if SQL*Plus is installed this way (as covered in the *SQL*Plus Reference Manual).*

When you use the RUNFORM command, use the standard syntax, but omit the username and password. In this situation, RUNFORM only accepts the current login information. Because the operator avoids a separate ORACLE login, invoking a form is quicker from within SQL*Plus than it is from the command line. For details on connecting to a different ORACLE login, refer to the CONNECT command in the *SQL*Plus Reference Manual.*

Related Topics: RUNFORM command 23-7

Connecting to SQL*ReportWriter

You can invoke SQL*ReportWriter from SQL*Forms. This means that you can generate reports from a SQL*Forms application and avoid exiting SQL*Forms.

To invoke SQL*ReportWriter from SQL*Forms, use the HOST packaged procedure as follows:

```
HOST ( ' { RUNREP_command |SQLREP_command} ' ) ;
```

where:

RUNREP_command	Specifies a standard login command to the SQL*ReportWriter run-time engine.
SQLREP_command	Specifies a standard login command to the SQL*ReportWriter define-time engine.

Note that you can invoke either the SQL*ReportWriter run-time engine or the define-time engine. However, you will usually invoke the run-time engine from SQL*Forms. For details on the RUNREP and SQLREP commands, refer to the *SQL*ReportWriter Reference Manual.*

Examples

1. The following statement starts SQL*ReportWriter without parameters:

```
HOST ( ' RUNREP ' ) ;
```

2. The following example starts SQL*ReportWriter and passes values for RUNREP parameters.

```
HOST( 'RUNREP sales scott/tiger deptno = 10');
```

3. You can invoke SQL*ReportWriter by calling a variable. To do so, you must first define the variable. The following PL/SQL statement defines a SQL*Forms variable called my_var and uses variables to pass dynamic parameter values:

```
my_var : = 'RUNREP sales scott/tiger deptno = ' ||
TO_CHAR(:deptno) ;
```

Once you have defined my_var, your trigger can invoke SQL*ReportWriter with the following statement:

```
HOST (my_var) ;
```

Related Topics: HOST, packaged procedure 16-22

Using ORACLE Precompilers with SQL*Forms

SQL*Forms can access any precompiled routine that you create with Version 1.3 of an ORACLE Precompiler or a later version. Specifically, SQL*Forms considers the routine a user exit. You invoke the routine through the USER_EXIT packaged procedure. Depending on the contents of the routine, SQL*Forms can pass values to a user exit and a user exit can pass values back to SQL*Forms.

Related Topics: User exits 19-1
USER_EXIT, packaged procedure 16-34

Calling Forms from Programs

The IAPCAL command allows you to call a SQL*Forms application from any C language program that is linked to SQL*Forms (Run Form). The syntax of the IAPCAL command allows you to specify a standard SQL*Forms (Run Form)command line. This feature means you can run a form from a program with the same options that you can use from the command line.

The prototype and constants for the IAFCAL command are located in the iapcal.h file on the medium that contains your SQL*Forms code.

Syntax: The following syntax describes the prototype for the IAPCAL function:

```
int iapcal (command_line)
    text  *  command_ line;
```

where *command_line* specifies the SQL*Forms (Run Form) command line.

The following IAPCAL statement runs the ORDER_ENTRY form with the -a, -b, and -d command line switches under the SCOTT username:

```
int iapcal ("runform -abd order_ entry scott/tiger") ;
```

Note: Although SQL*Forms might ignore it, the IAP parameter is required.

Note: If the operator is not logged into the ORACLE RDBMS and the command line does not invoke SQL*Foms Login screen via the -1 switch, SQL*Forms uses the userid provided in the IAPCAL call. If the command line does not provide a uerid, SQL*Forms uses any existing OPS$ login. If the command line provides neither a userid nor an OPS$ login, then the Login screen displays.

If the operator is not logged into the ORACLE RDBMS and the command line does invoke SQL*Forms Login screen via the -1 switch, SQL*Forms always displays the Login screen. If the command line provides an OPS$ login, SQL*Forms uses that login and ignores any operator entry. If the command line does provide an OPS$ login, SQL*Forms then uses the userid provided by the operator.

If the operator is logged into the ORACLE RDBMS and the command line does not provide the -1 switch, SQL*Forms uses the existing connection even a userid is specified on the command line.

If the operator is logged into the ORACLE RDBMS and the command line does provide the -1 switch, SQL*Forms uses the existing connection, Although SQL*Forms always displays the Login screen, the program ignores any entry the operator makes.

Restrictions: Do not use IAPCAL in a user exit Doing so will ruin all data structures. Use the CALL packaged procedure instead.

Using Oracle*Terminal

Oracle*Terminal is a utility that allows you to create and modify terminal and key mapping definitions. This utility is used by several ORACLE products. In Version 3.0 of SQL*Forms, it replaces the CRT utility that previous versions used.

SQL*Forms (Design) and SQL*Forms (Run Form) require mappings to determine how they display and how function keys map. Each of these components have default mappings for the specific devices that you install them on. However, through the -c command line switch you can specify alternative named mappings that you, or your DBA, create with Oracle*Terminal.

For information on how to define terminal definitions and key mappings to use with SQL*Forms, refer to the Oracle*Terminal *User's Guide.*

Related Topics: -c switch, SQL*Forms (Design) 23-11
-c switch, SQL*Forms (Run Form) 23-16
Preset mapping 26-8

Defining Key-F0 through Key-F9

SQL*Forms allows you to define 10 SQL*Forms (Run Form) function keys to which you can assign custom functionality. To make use of any of these keys in your applications, use Oracle*Terminal to map a physical key or key sequence to any of the Fn functions (where n is the number 0 through 9) within the appropriate mapping.

Note that functions F0 through F9 perform no actions. You assign the desired functionality to these keys by writing the appropriate Key-Fn triggers.

Related Topics: Key-Fn trigger 14-11

PART

VI TESTING

CHAPTER

26 OPTIONS

This chapter discusses SQL*Forms options. Specifically, it covers the following topics:

- SQL*Forms (Run Form) options
- SQL*Forms (Design) options
- user preference file

SQL*Forms (Run Form) Options

SQL*Forms supplies options that allow you to affect the behavior of forms when you execute them from SQL*Forms (Design). With one exception, run options are identical to command line switches for SQL *Forms (Run Form). The difference between the two features is that you use run options from the design interface and you use command line switches from the command line.

You set run options from the set Options form. All run options are turned off by default. Once you turn a run option on, it remains on-and in effect-until you tum it off or end the SQL*Form (Design) session. As long as the option is turned on, it affects any form you run, including any forms that are called by a form you execute. Note, however, that the run options in no way affect form definitions.

Note: You can also set tune options in a user preference file.

Buffer Record in File

The Buffer Record in File run option saves memory at run time by buffering the minimum number of records-the block size plus one record. This option slows processing.

If you are running a form from the command line and you want to use this option, use the -b switch for SQL*Forms (RunForm).

Related Topics: -b switch SQL*Forms (Run Form) 23-16

Debug Mode

The Debug Mode run option invokes debug mode for the SQL*Forms (Run Form) session. Debug mode:

- displays ongoing messages about trigger execution and other states of the current form during execution
- invokes break processing if the BREAK packaged procedure is in any triggers

If you are running a form from the command line and you want to use this option use the -d switch for SQL*Forms(Run Form).

Related Topics: -d switch, SQL*Forms (Run Form) 23-17

Disable Array Processing

The Disable Array Processing run option suppresses the array processing feature. By default, SQL*Forms sets the array size to the number of rows displayed per block. In Version 3.0 of SQL*Forms, you can set the array size for each blosk independently. When you select this option, the array size is set to 1 for all blocks during the current design session.

If you are running a form from the command line and you want to use this option, use the -a switch for SQL*Forms (Run Form).

Related Topics: -a switch for SQL*Forms (Run Form) 23-16
Array Size block characteristic 11-7

Display Block Menu

The Display Block Menu run option displays the block menu automatically when you run the form using the Execute option on the Action Menu.

If you are running a form from the command line and you want to use this option, use the -m switch for SQL*Forms (Run Form).

Related Topics: -m switch for SQL*Forms (Run Form) 23-18

Quiet Mode

The Quiet Mode run option invokes the quiet mode when you run the form using the Execute option on the Action menu. In quiet mode, messages do not ring the bell (i.e., do not produce an audible beep). However, you can still explicity ring the bell from a trigger via the BELL packaged procedure.

If you are running a form from the command line and you want to use this option, use the -q switch for SQL*Forms (Run Form).

Related Topics: -q switch for SQL*Forms (Run Form) 23-19

Set Block Validation as Default

The Set Block Validation as Default run option defines a block as the validation unit instead of a field. By default, a block is the validation unit ony for block mode terminals.

The default validation unit determines when SQL*Forms starts validating the information entered. The default validation unit also determïnes the maximum size of an object that an operator can navigate through before SQL*Forms starts the validation process.

Selecting this option enables you to more easily create and test a form on a character mode terminal that will be run on a block mode terminal.

Related Topics: Validation unit 3-7

Share Cursors for Implicit SQL

By default, SQL*Forms assigns a separate database cursor for each SQL statement that a form execures implicity as part of posting or querying data. This behavior enhances processing because the statements in each cursor only need to be parsed the first time they are executed in a SQL*Forms (Run Form) session-not every time.

Note that the cursors that are assigned to query SELECT statements must be parsed every time they are executed. This exception exists because queries can vary from execution to execution.

When you specify the Share Cursors for implicit SQL run option, SQL*Forms only assigns a separate cursor for each query SELECT statment. All other implicit SQL statements share, or reuse, cursors. This behavior saves memory but slows processing because all INSERT, UPDATE, DELETE, and SELECT FOR UPDATE statements must be parsed every time they are executed.

If you are running a form from the command line and you want to use this feature, use the -t switch for SQL*Forms (Run Form).

Note: The Share Cursors for Implicit SQL run option has no effect if you replace standard SQL*Forms processing with On-Insert, On-Update and On-Delete triggers because these triggers replace the implicit issuance of INSERT, UPDATE, and DELETE statements.

Related Topics: Cursor, database 22-3
Share Cursors for Trigger SQL run option 26-4
-t switch, SQL*Forms (Run Form) 23-20

Share Cursors for Trigger SQL

By Default, SQL*Forms assigns a spearate database cursor for each SQL statement that a form executes explicitly via V2 trigger. This behavior enhances processing because the statements in each curcor only need to be parsed the first time they are executed in a SQL*Forms (Run Form) session-not every time.

When you specify the Share Cursors for Trigger SQL run ooption, SQL*Forms assigns a single cursor for all SQL staements in V2 triggers. These statements share, or reuse, that cursor. This behavior saves memory, but slows processing because the SQL statements must be parsed every time they are executed.

You can fine tune this behavior through the New Cursor Area trigger step characteristic. If a trigger step that contains a SQL statement has this characteristic turned on, SQL*Forms assigns a separate cursor to the statement, in effect overriding the Share Cursors for Trigger SQL option for that statement.

If you are running a form from the command line and you want to use this feature, use the -o command line switch for SQL*Forms (Run Form).

Note: The Share Cursors for Trigger SQL run option has no effect on statements in V3 triggers. It exists for compatibility with previous versions of SQL*Forms.

Related Topics: Cursor, database 22-3
New Cursor Area trigger step characteristic F-6
-o switch, SQL*Forms (Run Form) 23-18
Share Cursors for Implicit SQL run option 26-4

Statistics

The Statistics run option displays a message at the end of the SQL*Forms (Run Form) session that states the maximum number of simultaneous cursors that were used during the session SQL*Forms writes this message to the terminal screen not to the message line.

The Statistocs option also issues the following command to the database:

```
ALTER SESSION SET SQL_TRACE TRUE
```

This command enables the SQL trace facility for the current SQL*Forms (Run Form) session. For more information on this facility-which gathers database performance information-refer to the ORACLE RDBMS *Performance Tuning Guide.*

If you are running a form from the command line and you want to use this option, use the -s switch for SQL*Forms (Run Form).

Related Topics: -s switch, SQL*Forms (Run Form) 23-19

SQL*Forms (Design) Options

SQL*Forms supplies options that allow you to affect the behavior of SQL*Forms (Design) during the current design session.

You set design options from the Set Options form. All design options are turned off by default. Once you turn a design option on, it remains on-and in effect-until you turn it off or end the SQL*Form (Design) session. Note that the design options in no way affect form definitions.

Note: You can also set design options in a user preference file.

Disable Trigger Compilation

The Disabled Trigger Compilation options tells SQL*Forms not to compile PL/SQL triggers or form-level procedures until you generate the form. With this option turned on, SQL*Forms does not compile PL/SQL text when you accept changed triggers and procedures or when you open and load form defintions.

Generate Before Exceuting Form

The Generate Before Executing Form design option asks SQL*Forms to generate the current context form automatically before it executes the from. Specifically, when you execute a form with this option, SQL*Forms performs the following actions:

- bypasses the Generate Form dialog box and automatically creates an INP file and an FRM file with the same name as the INP file
- bypasses the Execute Form dialog box and automatically executes the current context form

Limit Undo Buffer to 1

The Limit Undo Buffer to 1 design option limits the undo buffer to hold only one operation (the most recent one). By default, this buffer has no logical limit.

This feature is especially useful with systems that have strict memory requirements.

If you want to specify this feature from the command line, use the -l command line switch for SQL*Forms (Design).

Related Topics: -1 switch, SQL*Forms (Design) 23-12

Show list of values

The Show List of Values design option signals SQL*Forms to display a pop-up window with a list of available values when a designer enters a field that has a list of values.

Suppress Hints

The Suppress Hints design option suppresses the display of hints on the SQL*Forms message line.

Use Forms as Default

The Use Forms as Default design option makes the default interface element the form instead of the spread table. For example, when you select the Modify option from the Block menu, the Block Definition spread table displays by default. If you turn on the Use Forms as Default design option, the Block Definition form appears instead. [Change Display Type] allows you to view the corresponding spread table.

User Reference File

You can create a user preference file that automatically enforces SQL*Forms options every time that you log into SQL*Forms (Design). This file can contain keywords and settings that allow you to preset each of the SQL*Forms (Design) options and the SQL*Forms (Run Form) options. For example, to ensure that any form that you run from SQL*Forms (Design) runs in quiet mode, you would include the following line in the user preference file:

```
QUIET_ MODE= ON
```

The user preference file also allows you to preset a mapping for SQL*Forms (Design).

To be in effect during a SQL*Forms (Design) session, the user preference file must be named SQLFORMS.CFG and it must reside in the current directory.

If you log in to SQL*Forms (Design) with a command line switch that corresponds to a preset design option or mapping, that command line switch overrides the setting in the user preference file. Also, if a line in the user preference file contains an error, SQL*Forms ignores that line when it reads the file.

Syntax for Preset Options

To preset a SQL*Forms (Design) option or a SQL*Forms (Run Form) option, include the appropriate keyword and setting in the user preference file, using the following syntax:

```
keyword = {ON |OFF)
```

The following table present the keywords for each of the available options. These keywords correspond directly with the names of the options.

Type of Option	*Keyword*
SQL*Form (Run Form)	BUFFER_RECORD DEBUG_MODE DISABLE_ARRAY_PROCESSING DISPLAY_BLOCK_MENU QUIET_MODE SET_BLOCK_VALIDATION_AS_DEFAULT SHARE_CURSORS_FOR_IMPLICIT_SQL SHARE_CURSORS_FOR_TRIGGER_SQL STATISTICS
SQL*Forms (Design)	DISABLE_TRIGGER_COMPILATION GENERATE_BEFORE_EXECUTING_FORM LIMIT_UNDO_BUFFER_TO_l SHOW_LIST_OF_VALUES SUPPRESS_HINTS USE_FORMS_AS_DEFAULT

You can specify as many options as you want in the file, in any order. Note that although all options are turned off by default, you can explicitly turn an option off in the file.

Syntax for a Preset Mapping

To preset a SQL*Forms (Design) mapping, include the MAPPING keyword and mapping name in the user preference file, using the follwing syntax:

```
MAPPING = mapping
```

CHAPTER

27 APPLICATION DOCUMENTATION

This chapter discusses the types of application documentation that you can create for a form and store in an operating system file. Specifically, it covers the following topics:

- detail documentation
- fields documentation
- object reference documentation
- procedures documentation
- summary documentation
- triggers documentation

This chapter does not discuss how to create this documentation.

Note: You can produce application documentation only for forms that are stored in current memory or in database format.

Related Topics: Creating application documentation 2-5

Detail Documentation

Detail documentation presents a complete description of the objects in the current context form. Specifically, this documentation lists object characteristics for the following SQL*Forms objects:

- the form
- all blocks in the form
- all fields in the form
- all triggers in the form
- all form-level procedures in the form
- all pages in the form

Format: Detail documentation begins with a form section, which presents the following information:

- the form characteristics of the specified form and the username of the person who created the form (i.e., the owner)
- a procedure section that presents form-level procedure characteristics for all form-level procedures that belong to the form
- a trigger section that presents trigger characteristics for all triggers that are attached to the form

The documentation continues with block sections for each block in the form. Each block section presents the following information

- block characteristics
- a trigger section that presents trigger characteristics for all triggers that are attached to the block
- field sections for each field in the block

Each field section presents the following information:

- field characteristics
- a trigger section that presents information on all triggers that are attached to the field

At the end of the detail documentation are page sections for each page in the form. Each page section presents page characteristics.

Detail documentation presents each block and field in the order specified by its Sequence Number block characteristic and Sequence Number field characteristic. Each page is presented in page order as specified by its Page Number page characteristic.

Related Topics: Blocks, characteristics 11-6
Fields, characteristics 12-4
Forms, characteristics 9-3
Form-level procedures 15-4
Pages, characteristics 10-4
Triggers, characteristics 14-4

Fields Documentation

Fields documentation presents field characteristics for every field in the current context form.

Format: Fields documentation groups fields by the blocks that they belong to.

Fields documentation begins with a form section, which presents the name of the specified form and the username of the person who created the form (i.e., the owner).

The remainder of the documentation consists of block sections for each block in the form. Each block section presents the following information:

- the name of a block
- field sections for each field in the block

Each field section presents field characteristics.

Fields documentation presents each block and field in the order specified by its Sequence Number block characteristic and Sequence Number field characteristic.

Related Topics: Fields, characteristics 12-4

Object Reference Documentation

Object reference documentation presents the form name reference information for a specified object in the current context form.

Format: Object reference documentation begins with an object section, which presents the name and characteristics of the specified object.

The remainder of the documentation consists of form sections for each form that references the object. Each form section presents the following information:

- the name of each form
- the username of the person who created the form (i.e., the owner)
- where and how in each the object is referenced

Object reference documentation presents each form in alphabetical order.

Pages Documentation

Pages documentation presents page characteristics for every page in the current context form.

Format: Pages documentation begins with a form section, which presents the name of the specified form and the username of the person who created the form (i.e., the owner).

The remainder of the documentation consists of page sections for each page in the form. Each page section presents all page characteristics for a page.

Pages documentation presents each page in the order specified by its Page Number page characteristic.

Related Topics: Pages, characteristics 10-4

Procedures Documentation

Procedures documentation presents characteristics for every form-level procedure in the current context form.

Format: Procedures documentation begins with a form section, which presents the form characteristics of the specified form and the username of the person who created the form (i.e., the owner).

The remainder of the procedures documentation consists of a section that presents characteristics for all procedures that are attached to the form.

Related Topics: Form-level procedures 15-4

Summary Documentation

Summary documentation presents a brief description of the objects in the current context form. Specifically, this documentation lists the following SQL*Forms objects:

- the form
- all blocks in the form
- all fields in the form
- all triggers in the form
- all form-level procedures in the form
- all pages in the form

Format: Summary documentation first presents a form section that specifies the following information:

- the name of the form, the username of the person who created the form (i.e., the owner), and any text associated with the Comment form characteristic
- the name of each form-level procedure that belongs to the form
- the name of each trigger that is attached to the form and any text associated with the Comment trigger characteristic for each trigger

After the form section, summary documentation presents block sections for each block in the form. Each block section specifies the following information:

- the name of the block, the name of the table on which the block is based, and any text associated with the Comment block characteristic
- the names of any triggers that are attached to the block and any text associated with the Comment trigger characteristic
- field sections for each field in the block

Each field section specifies the following information:

- the name of the field, whether or not the field has the Base Table field characteristic turned on, and any text associated with the Comment field characteristic
- the names of any triggers that are attached to the field and any text associated with the Comment trigger characteristic

At the end of the detail documentation is a section on page information.

Summary documentation presents each block and field in the order specified by its Sequence Number block characteristic and Sequence Number field characteristic.

Triggers Documentation

Triggers documentation presents complete information for all triggers in the current context form.

Format: Triggers documentation groups triggers by the objects to which they are attached.

Triggers documentation begins with a form section, which presents the following information:

- the name of the specified form and the username of the person who created the form (i.e., the owner)
- a trigger section that presents the trigger characteristics for all triggers that are attached to the form

The remainder of the documentation consists of block sections for each block in the form. Each block section presents the following information:

- the name of the block
- a trigger section that presents the trigger characteristics for all triggers that are attached to the block
- field sections for each field in the block

Each field section presents the following information:

- the name of the field
- a trigger section that presents the trigger characteristics for all triggers that are attached to the field

Triggers documentation presents each block and field in the order specified by its Sequence Number block characteristic and Sequence Number field characteristic.

Related Topics: Triggers, characteristics 14-4

CHAPTER

28 DEBUG FACILITY

The debug facility is a feature that allows you to run an application while monitoring the underlying processing. You can use this facility to pinpoint processing errors.

Using the debug facility makes use of the Debug Mode run option and (optinally) the BREAK packaged procedure.

- If you run an application in debug mode without using the BREAK packaged procedure, the message line displays the standard debug messages as processing continues.
- If you run an application in debug mode and you have the BREAK packaged procedure inserted in your triggers, SQL*Forms executes the application with break processing.

Note: The BREAK V2 function code performs the same function in V2 triggers that the BREAK packaged procedure performs in V3 triggers.

This chapter covers the following topics

- debug messages
- break processing

Related Topics: BREAK, packaged procedure 16-6
Debug Mode run option 26-2

Debug Messages

When you run an application in debug mode, SQL*Forms displays debug messages whenever it processes a trigger. These messages, which display on the message line, identify the name of the trigger that is processing and the object on which the trigger is firing. If the trigger is a V2 style trigger, the message also indicates the sequence number of the trigger step that is firing.

The following is an example of a debug message for a V3 style trigger:

```
Executing POST-FIELD trigger on field SHIPDATE.
```

Break Processing

Break processing allows you to view detailed processing information about an application as you run it. SQL*Forms performs break processing when you run an application in debug mode and one or more triggers in the application contains the BREAK packaged procedure.

When SQL*Forms encounters a BREAK packaged procedure in debug mode, it pauses regular processing and displays the Debug Mode Options menu. This menu presents the following options:

- Continue with Debug Messages
- Continue without Debug Messages
- Display Global Variables
- Display System Variables
- Display Page 0 Fields
- Exit SQL*Forms (Run Form)

When the menu appears, you can select one of the options to see more detailed processing information or to direct further debug mode processing. While the application is paused, you can display as many categories of information (one at a time) as you want. The application remains paused until you request to continue or cease debug processing.

Related Topics: BREAK, packaged procedure 16-6

Continue with Debug Messages

The Continue with Debug Messages debug mode option terminates the pause and continues processing the application in debug mode. While processing, SQL*Forms displays any debug messages.

Continue without Debug Messages

The Continue without Debug Messages debug mode option terminates the pause and continues processing the application in debug mode. While processing, SQL*Forms does not display any debug messages.

Display Global Variables

The Display Global Variables debug mode option displays a two column spread table that shows the name and current value of every global variable in the current form.

Display System Variables

The Display System Variables debug mode option displays a two-column spread table that shows the name and current value of every system variable in the current form.

Display Page 0 Fields

The Display Page 0 debug mode option displays a two-column spread table that shows the name and current value of every field located on page 0 of the current form.

Exit SQL*Forms (Run Form)

The Exit SQL*Forms (Run Form) debug mode option terminates the SQL*Foms (Run Form) session:

- If you are executing a form from SQL*Forms (Design), SQL*Forms returns to the SQL*Forms (Design) main menu.
- If you are executing a form from the command line, SQL*Forms returns to the command line.

CHAPTER

29

EDIT KEYSTROKE FACILITY

The edit keystroke facility is a quality assurance feature that allows you to capture an operator's or designer's keystrokes while working in SQL*Forms, modify the record of those keystrokes, and "replay" them. You can use this facility to capture and replay keystrokes in both SQL*Forms (Design) and SQL*Forms (Run Form). However, you will primarily use it in SQL*Forms (Run Form) to test the functionality of forms across software versions and across hardware.

This chapter covers the following topics:

- using the edit keystroke facility
- the key script file

Using the Edit Keystroke Facility

Using the edit keystroke facility means making use of several SQL*Forms command line switches and a text file editor. The process consists of combinations of the following tasks.

Note: These tasks assume that you are using the facility with SQL*Forms (Run Form).

- Capture an operator's keystrokes in a key script file.

  Run SQL*Forms (Run Form) with the -e switch. This switch will cause SQL*Forms to write the operator's keystrokes to a designated key script file during the session.

- Edit a key script file to change the actions that it will perform.

  Use your computer's text file editor to edit the key script file. This action will change the behavior of the application when you use the key script file as input.

- Use a key script file as input to run an application.

  Run SQL*Forms with the -r switch. This switch will cause SQL*Forms to perform all actions called for in the key script file. By default, SQL*Forms routes the results of these actions to the terminal screen. However, you can use the -w switch with the -r switch to route the results to a display log file (and not to the terminal screen).

  Note: The key script file must contain an exit character (<x>) for the form to run.

- Write terminal output to a display log file.

  To log the output of an interactive session, run SQL*Forms with the -i switch. To log the output of a key script session, run SQL*Forms with the -r and -w switches.

By using the -e switch and keeping a history of key script files, you can run forms with these files using the -r and -i switches. Then you can run operating system-specific programs that compare the display log files to see if the behavior of the form has changed across software versions or across hardware.

Related Topics: -e switch, SQL*Forms (Design) 23-11
-e switch, SQL*Forms (Run Form) 23-17
-i switch, SQL*Forms (Design) 23-11
-i switch, SQL*Forms (Run Form) 23-17
-r switch, SQL*Forms (Design) 23-13
-r switch, SQL*Forms (Run Form) 23-19
-w switch, SQL*Forms (Design) 23-13
-w switch, SQL*Forms (Run Form) 23-20

Key Script File

A key script file is a record of an operator's or designer's keystrokes during a SQL*Forms session. SQL*Forms creates a key script file when you run SQL*Forms (Design) or SQL*Forms (Run Form) with the -e command line switch.

The following is an example of the contents of key script file. (The line numbers might not appear in your files; they appear here only for reference.)

```
1  ; Generated by SQL*Forms Version 3.0 on Jan 6 10:23:13 1989
2  ;
3  s c o t t <Next Field>
4  t i g e r <Next Field>
5  <Enter Query>
6  1 0 0 <Execute Query>
7  <Commit Form>
8  <Exit /Cancel>
```

The semicolons (;) in lines 1 and 2 each indicate one line of comments. The comment on line 1 is a banner generated by SQL*Forms.

The function key names in angled brackets (<>) show where the operator pressed function keys. The indicated names correspond directly with the names of the SQL*Forms (Design) and SQL*Forms (Run Form) function keys.

The following table shows how SQL*Forms displays special, typed characters in key script files:

Typed Character	*Displayed As*
;	\;
< *or* >	\< *or* \>
[*or*]	\[*or* \]
\	\\
blank character	\ b

Note: If you are using the edit keystroke facility on a block mode terminal, your key script file appears slightly different. Each block mode input event appears in square brackets. For example:

```
[X 3 23 1 <Show Keys>]
```

Editing a Key Script File

You can use a key script file as input to a SQL*Forms (Design) or SQL*Forms (Run Form) session. You can edit the file in order to change the actions it causes. For example:

change

```
    1 1 5 <Next Record>
```

to

```
    2 0 0 <Commit Form>
    <Next Record>
```

PART

VII APPENDIXES

APPENDIX

A SUMMARY OF FEATURES

This appendix lists the features in SQL*Forms Version 3.0 that distinguish it from previous versions. Specifically, it covers:

- new features
- changed features

New Features

The following features are new in Version 3.0 of SQL*Forms:

- Online help system
- Ability to call SQL*Forms from a 3GL program, such as a C language program
- Default master-detail relationships
- Support for sequence number generation
- Ability to enforce integrity constraints
- Externalized window routines
- Mouse support
- Dynamic field characteristics
- Field display masks
- Pop-up field editor
- Pop-up pages
- Ability to copy and reference objects
- PL/SQL integration
- Ability to invoke a SQL*Menu menu from a SQL*Forms application

New triggers:

On-Database-Record	On-New-Record
On-Delete	On-Remove-Record
On-Error	On-Update
On-Insert	On-Validate-Field
On-Lock	On-Validate-Record
On-Message	Post-Commit
On-New-Field-Instance	Pre-Commit

- New packaged procedures (also known as macros):
  - ANCHOR_VIEW
  - BREAK
  - CLEAR_EOL
  - COPY
  - COPY_REGION
  - CUT_REGION
  - DISPLAY_FIELD
  - DISPLAY_PAGE
  - DO_KEY
  - FIRST_RECORD
  - GO_RECORD
  - HIDE_PAGE
  - LAST_RECORD
  - MOVE_VIEW
  - PASTE_REGION
  - POST
  - RESIZE VIEW
  - SET_FIELD
  - SET_INPUT_FOCUS
  - SHOW_PAGE
  - USER_EXIT
- Packaged functions:
  - APPLICATION_CHARACTERISTIC
  - BLOCK_CHARACTERISTIC
  - ERROR_CODE
  - ERROR_TEXT
  - ERROR_TYPE
  - FIELD_CHARACTERISTIC
  - FORM_CHARACTERISTIC
  - FORM_FAILURE
  - FORM_FATAL
  - FORM_SUCCESS
  - MESSAGE_CODE
  - MESSAGE_TEXT
  - MESSAGE_TYPE
  - NAME_IN
- New system variables:
  - $$DATETIME$$
  - SYSTEM.BLOCK_STATUS
  - SYSTEM.CURSOR_RECORD
  - SYSTEM.FORM_STATUS
  - SYSTEM.LAST_QUERY
  - SYSTEM.LAST_RECORD
  - SYSTEM.RECORD_STATUS
  - SYSTEM.TRIGGER_RECORD
- DATETIME data type
- LONG data type
- Column-level security
- User preference file

Changed Features

A number of features that existed in previous versions of SQL*Forms have been changed in Version 3.0 as follows:

- The documentation set is entirely revised and updated.
- The SQL*Forms (Design) interface is completely revised.
- You can set the array size on a per-block basis.

- Debug mode is enhanced.
- The INP file format is completely revised.
- The screen painter has the following added capabilities:
  - use of [Previous Block], [Next Block], [Previous Page], and [Next Page]
  - indicators for X and Y cursor coordinates
  - scrolling to allow you to create forms larger than the screen
- You can display a list of values as a pop-up window and you can populate the list with a complete SELECT statement.
- Triggers are available in two styles (V2 and V3).
- More functions are available to edit triggers.
- The Show Keys screen can scroll to show more function key descriptions.
- All SQL*Forms (Run Form) prompts now appear in dialog boxes.
- The definitions of the SQL*Forms base tables are changed.
- The status line in SQL*Forms (Run Form) is changed.
- All messages and errors are numbered for easier identification.
- Oracle*Terminal replaces the CRT utility.

APPENDIX

B

UPWARD COMPATIBILITY

This chapter discusses how applications created in previous versions of SQL*Forms are upwardly compatible with Version 3.0 of SQL*Forms. Specifically, it covers the following topics:

- converting existing forms to Version 3.0 status
- translating triggers into PL/SQL
- changes to data types

Note: You should read all of the material in this appendix before you convert your existing forms.

Converting Existing Forms to Version 3.0 Status

SQL*Forms Version 3.0 supports the conversion of all forms seated with previous versions of SQL*Forms. Version 3.0 continues the feature introduced with Version 2.3 of providing success/failure logic with V2 function codes, or macros (see the *SQL*Forms Documentation Addendum* Version 2.3).

Oracle Corporation recommends that you preserve an existing form's functionality when you convert it to Version 3.0 status. If you do attempt to convert a form that does not have the success/failure provision to a Version 3.0 form that does have that provision, the resulting form is likely to exhibit changed behavior.

Converting Version 2.0 Forms

To upgrade a form to Version 3.0 status you must first convert the existing INP file to Version 3.0 format using the CONVFORM command.

When you convert an INP file which you have been running under Version 2.0, Oracle Corporation recommends that you specify the -f switch in order to preserve the form's functionality. If you decide to take advantage of Version 3.0's success/failure logic, you should not specify the -f switch. However, be advised that the resulting form's changed behavior might require you to perform additional modification and testing.

In any case, you must specify both the -v and -c switches with the CONVFORM command when you convert a Version 2.0 INP file.

After you convert the INP file to Version 3.0 format, you must regenerate the form, creating an FRM file. You can use either the Generate option from the Action menu or the GENERATE command from the command line.

Related Topics: SQL*Forms (Convert) 23-3
SQL*Forms (Generate) 23-6

Converting Version 2.3 Forms

To upgrade a form to Version 3.0 status you must first convert the existing INP file to Version 3.0 format using the CONVFORM command.

When you convert an INP file which you have been running under Version 2.3, do not specify the -f switch. The resulting form will preserve the success/failure behavior it exhibited under Version 2.3.

You must specify the -v switch with the CONVFORM command when you convert a Version 2.3 INP file.

After you convert the INP file to Version 3.0 format, you must regenerate the form, creating an FRM file. You can use either the Generate option from the Action menu or the GENERATE command from the command line.

Related Topics: SQL*Forms (Convert) 23-3
SQL*Forms (Generate) 23-6

Translating Triggers into PL/SQL

The following section describes how to manually translate Version 2.0/2.2/2.3 triggers written in the function code language and SQL into PL/SQL. The aim is to provide a direct translation so that the new trigger behaves identically to the old trigger.

Although V2 triggers will work in Version 3.0 of SQL*Forms, Oracle Corporation recommends that you convert all of your V2 triggers to the V3 style. This conversion should improve the performance of your applications and make maintenance of those applications easier.

Related Topics: V3 triggers versus V2 triggers 4-2

General Translation Issues

To manually translate a V2 trigger into a V3 trigger, you must adhere to the following structure, which converts each step of a V2 trigger into a nested PL/SQL block with some surrounding logic for handling success/failure.

A translated trigger takes the following form:

```
DECLARE
    abort_ trigger BOOLEAN;
    return_code BOOLEAN;
    ignore_ failure BOOLEAN;
    ignore_ trigger_ failure EXCEPTION;

BEGIN
    trigger_step_block;
    [trigger_step_block;
    trigger_step_block;
    ...]
EXCEPTION
    WHEN ignore_ trigger-failure THEN NULL;
END ;
```

where *trigger_step_block* is:

```
{<step_n> |<step_label>)
DECLARE
    fail EXCEPTION;

BEGIN
    step_logic;
    abort_trigger := FALSE;
    ignore_failure := FALSE;
    return_code := TRUE;
EXCEPTION
    WHEN fail THEN
           {① abort_trigger := TRUE; ignore_failure := FALSE; |
            ② abort_trigger := TRUE; ignore_failure := TRUE; |
            ③ abort_trigger := FALSE; ignore_failure := FALSE;}
           return_code := FALSE;
END;
IF abort_trigger
    THEN IF ignore_failure
           THEN  RAISE  ignore_trigger_failure;
           ELSE  RAISE  form_trigger_failure;
    END IF;
    ELSIF  return_code
           THEN GOTO {success  label|step-n+1};
   ELSE GOTO  {failure  label|step_n+l};
END IF;
```

where:

①	Is used if only the Abort On Failure trigger step characteristic is turned on.
②	Is used if both the Abort On Failure and Succeed on Abort trigger step characteristics are turned on.
③	Is used if neither the Abort On Failure or Succeed on Abort trigger step characteristics are turned on.
step_n	Specifies the number for the step.
step_label	Specifies the label for the step. The step label takes precedence over the step number *(step_n).*
step_logic	Specifies the actual translation of the step statement.
success_label	Specifies the Success Label trigger step characteristic for the step if it exists.
step_n+l	Specifies the number of the next step in sequence.
failure_label	Specifies the Failure Label trigger step characteristic for the step if it is specified.

Note: Where any of `success_ label, failure-label, step-label,` or `step_n` lead to nonexistent labels, replace the GOTO statement with `RAISE ignore_ trigger_ failure.`

Related Topics: PL/SQL 15-1

Translating SQL

Translate trigger steps that contain SQL statements into PL/SQL statements. There are two types of SQL steps that you might need to translate:

- steps containing SELECT statements
- steps containing DML statements

Translating SELECT Statements

There are two types of SELECT statements that you might need to translate:

- statements without INTO clauses
- statements with INTO clauses

Note that you should give special consideration to statements that compare fields or perform mathematical expressions. You should write these statements as Boolean expressions. This approach uses much less database resource, executes faster, and is clearer to read. For example, consider a trigger that contains the following steps:

STEPl	Success Label = STEP3
	Failure Label = STEP2
	`SELECT ' X ' FROM DUAL`
	`WHERE : SAL > : COMM`

STEP2	`#EXEMACRO MESSAGE ' Commission cannot be greater than salary. ' ; ENDTRIG;`
STEP3	`#EXEMACRO NXTFLD;`

You should convert these statements to the following single PL/SQL Statement:

```
IF : SAL > : COMM
    THEN NEXT_ FIELD ;
    ELSE MESSAGE ( 'Commission cannot be greater than salary. ' ) ;
END IF;
```

Statements without INTO Clauses For SELECT statements without an INTO clause, add the following items to the DECLARE section of the step_block:

```
FAIL EXCEPTION;
dummy_into_buffer    CHAR(240);
CURSOR dummy_ select_ into_ cursor IS
    SELECT select_list
    FROM   table_list
    [where_ clause]
```

The following code is the text of step_logic for SELECT statements without an INTO clause:

```
OPEN dummy_ select_ into_ cursor;
FETCH  dummy_select_into_cursor
    INTO dummy_into_buffer;
IF  dummy_select_into_cursor%NOTFOUND
    THEN RAISE fail;
END IF;
```

Note: You should use the cursor declaration since a poorly written trigger might actually return more than one row. This condition would be an error if it were just written as a SELECT.. . INTO statement.

Statements with INTO Clauses For SELECT statements with an INTO clause add the following items to the DECLARE section of the step_block:

```
CURSOR DUMMY_ SELECT_ INTO_ CURSOR IS
    SELECT select_list
    FROM   table_list
    [where_clause]
```

The following code is the text of step_logic for SELECT statements without an INTO clause:

```
OPEN dummy_ select_ into_ cursor;
FETCH  dummy_select_into_cursor
    INTO : [block.]field_namel,  : [block.]field_name2, . . .;
IF  dummy_select_into_cursor%NOTFOUND
    THEN RAISE fail;
END IF:
```

Translating DML Statements

Use the following code as the text for step_logic for DML statements (except SELECT):

```
DML_statement;
IF SQL%NOTFOUND
    THEN RAISE fail;
END IF;
```

where *DML_statement* is a direct copy of the step.

Note: SQL is the name of the default cursor.

Translating Function Code Statements

Translate function code statements of the following form into their corresponding SQL*Forms packaged procedures:

```
#EXEMACRO function_code; [function_code: function_code; . ..]
```

Be aware that some function codes do not translate directly into V3 packaged procedures.

Related Topics: Matching V2 function codes to V3 equivalents F-24

Translating CASE Statements

Translate CASE statements to PL/SQL by using IF and ELSIF and ELSE statements. The following CASE statement

```
CASE selector IS
    WHEN criteria_l THEN action_list_l
    WHEN criteria_2 THEN action_list_2
    WHEN OTHERS THEN action_list_3
END CASE;
```

is translated as:

```
IF selector = criteria_l
    THEN action_list_l;
    ELSIF selector = criteria_2
    THEN action_list_2;
    ELSE action_list_3;
END IF;
```

Related Topics: CASE statement F-27

Translating SQL*Forms Commands.

This section discusses how to translate the SQL*Forms commands other than #EXEMACRO.

#COPY

Under most circumstances, you translate #COPY into a PL/SQL assignment statement. If the source or target location is not indirected using a variable reference, then translate the #COPY command into an assignment as in the following example. The following V2 command

```
#COPY {source |quoted_string} [INTO ] [ :] target [ ; ]
```

is translated as:

```
: target := { : source |quoted_string} ;
```

#ERASE

Translate the #ERASE command directly to a packaged procedure, as in the following example. The following #ERASE command

```
#ERASE { [&] GLOBAL. variable_name |[&] [block.] field_name} [;]
```

is translated as:

```
ERASE ({[ NAME_IN(]'GLOBAL .variable_name' [)] |
[NAME_ IN(]' [block.] field_name'});
```

Related Topics: ERASE, packaged procedure 16-18

#HOST

Translate the #HOST command to a SQL*Forms packaged procedure, as in the following example. The following #HOST command

```
#HOST {quoted_string |&{[block.] field-name |
GLOBAL. variable_name}} [;]
```

is translated as:

```
HOST ({quoted_string |NAME-IN ('{[block.] field-name |
GLOBAL .variable_name}') }) ;
```

Related Topics: HOST, packaged procedure 16-22

Translating User Exits

Translate calls to user exits as in the following example. A call in the following form

```
# [#] user_exit_name parameter_list
```

is translated as:

```
USER_EXIT ('user_exit_name parameter_list' , 'error-msg' ) ;
```

Related Topics: User exits, calling from a V3 trigger 19-10

Changes to Data Types

SQL*Forms Version 3.0 fully supports the ORACLE RDBMS data types. In previous versions of SQL*Forms, all data types were manipulated as character strings; in Version 3.0 they are manipulated as their true data types. These changes might cause problems in upgrading applications to Version 3.0. For instance, a V2 NUMBER field based on a CHAR column automatically loses its index when its form is converted to V3. To prevent this from happening, change all such fields to datatype CHAR and use a V3 format mask to enforce the entry of numbers only.

Beginning in Version 2.3 of SQL*Forms, you can create DATE fields that accept either a two-digit year or a four-digit year. If an operator is entering a date, and the field has enough room for a four-digit year, the operator can enter all four digits or just the last two. However, if the operator enters only the last two, the first two digits default to double zero.

For example, if the operator enters 31-DEC-88 in a field that has enough room for a four-digit year, SQL*Forms interprets the date as 31-DEC-0088. This behavior is consistent with the SQL language, and should be kept in mind when defining DATE fields.

To modify an existing DATE field to show a four-digit year, use [Resize Field] from within the screen painter to make the date field two spaces longer.

Note: This new feature does not apply to JDATE and EDATE fields.

APPENDIX

C ADVISORY

This appendix presents information that you should be aware of as you build applications. This information, found throughout this manual, is consolidated here as an advisory of issues that could cause you to work inefficiently or to damage your applications.

The information in this appendix is organized by chapter. If a chapter does not contain advisory information, it is not listed here. Each advisory statement is accompanied by a "Related Topics" list that indicates where in the book the topic of the statement is discussed.

Chapter	*Advisory Statements*	*Related Topics*
Fundamental Processing	Do not attempt to alter the navigational path by placing navigational commands in triggers that fire during navigation.	Navigational path 3-4
	Be careful that validation does not change the value of the field being validated.	Valid, field validation state 3-9
	Do not perform a database commit or rollback via a SQL statement in a trigger.	Locks, when locks are released 3-15
	Avoid placing exclusive locks on tables through your applications.	Locking, recommendations 3-16
	Be careful of changing the data in database fields through triggers that fire during commit processing.	Changing data, during commit processing 3-21
	No triggers fire in Enter Query mode.	Enter Query mode, restrictions 3-29
Trigger Concepts	Design your new applications with V3 style triggers, and replace the V2 style triggers in existing applications with V3 triggers.	V3 triggers versus V2 triggers 4-2
	Adhere to the rules on using SQL Data Manipulation Language (DML) statements in triggers.	Processing, SQL statements 4-5
	Do not use SQL Data Definition Language (DDL) statements in any trigger.	Processing, SQL statements 4-5
Pages	Adhere to the rules on view location and size when defining pop-up pages.	View Loc page characteristic 10-8 View Page page characteristic 10-9 View Size page characteristic 10-9
	Stacking order of windows is not guaranteed when returning to the original form.	

Chapter	*Advisory Statements*	*Related Topics*
Blocks	There is a lower limit on the number of records you can buffer for a block.	Records Buffered block characteristic 11-15
	If operators need to insert records through a form, a deselected column must not be defined as NOT NULL in the database.	Select Columns block characteristic 11-16
	If you want to establish a master-detail relationship automatically, you must create the detail block through the Default Block form.	Master-detail relationship 11-20
Fields	Be cautious of illogical combinations when you combine field characteristics.	Fields, characteristics 12-4
	You cannot directly query a LONG field in the WHERE or ORDER BY clause of any SELECT statement.	LONG data type 12-8
	Coordinate the use of the Enforce Foreign Key field characteristic and the Input Allowed field characteristic to prevent operators from de-enforcing a foreign key relationship.	Enforce Foreign Key field characteristic 12-12
	For fields with the NUMBER data type, make sure that the field length is long enough to avoid truncating data.	Field Length field characteristic 12-13
	You cannot create a field that is based on a column whose name contains blank spaces.	Field Name field characteristic 12-14
	When you set a field's length, you must allow room for format mask elements.	Format Mask field characteristic 12-15
	Always make a field's query length greater than or equal to the field length.	Query Length field characteristic 12-21
	When you cut and paste fields in the screen painter, SQL*Forms changes the default value of field sequence numbers.	Sequence Number field characteristic 12-22
	To prevent "wasted" sequence numbers, use a Pre-Insert trigger to fetch the next sequence number just prior to inserting the record in the database.	Default Value field characteristic 12-10

Chapter	*Advisory Statements*	*Related Topics*
Triggers	If you change the name of a user-named trigger in an existing application, be sure to change all references to that trigger in any other trigger that references it.	Trigger Name trigger Characteristic 14-7
	Be aware of what commands are legal in what triggers.	Triggers, types 14-8
	On-Insert, On-Update, and On-Delete triggers *replace* the default SQL*Forms processing for handling inserts, updates, and deletes during transaction posting.	On-Delete trigger 14-14 On-Insert trigger 14-15 On-Update trigger 14-18
	The On-Error and On-Message triggers *replace* the default writing of messages.	On-Error trigger 14-14 On-Message trigger 14-15
	The On-Lock trigger replaces the default SQL*Forms processing for locking rows.	On-Lock trigger 14-15
	Use form-level procedures instead of user-named triggers.	User-named trigger 14-27
PL/SQL	Adhere to the rules which apply to PL/SQL procedures that refer to SQL*Forms objects or functions.	PL/SQL, restrictions within SQL*Forms 15-8
	If your application is supported in multiple national languages, do not have trigger logic depend on the ERROR_TEXT packaged procedure.	Packaged procedures, failure 15-9

Chapter	*Advisory Statements*	*Related Topics*
Packaged Procedures	Adhere to the rules that apply to displaying, moving, and resizing views when working with pop-up pages.	ANCHOR_VIEW, packaged procedure 16-5 HIDE_PAGE, packaged procedure 16-21 MOVE_VIEW, packaged procedure 16-23 RESIZE_VIEW, packaged procedure 16-28 SHOW_PAGE, packaged procedure 16-33
	If you use PL/SQL COMMIT or ROLLBACK statements in anonymous blocks or form-level procedures, SQL*Forms interprets those statements as COMMIT_FORM and CLEAR_FORM packaged procedures respectively.	CLEAR_FORM, packaged procedure 16-9 COMMIT_FORM, packaged procedure 16-11
	If you use the ENTER_QUERY or EXECUTE_QUERY packaged procedures, use caution when employing the ALL_RECORDS and FOR_UPDATE parameters.	ENTER_QUERY, packaged procedure 16-17 EXECUTE_QUERY, packaged procedure 16-18
	Adhere to the rules that apply when you use the SET_FIELD packaged procedure to modify field characteristics.	SET_FIELD, packaged procedure 16-30
	Some packaged procedures are restricted and should only be used in certain triggers.	Restricted packaged procedures 16-35
Packaged Functions	In V3 triggers, use the NAME_IN packaged function instead of variable references.	NAME_IN packaged function 17-16
	For the datatype DATE, the NAME_IN packaged function returns “DD-MON-YYYY”.	

Chapter	*Advisory Statements*	*Related Topics*
Variables	SYSTEM.CURRENT_BLOCK, SYSTEM.CURRENT_FIELD, and SYSTEM.CURRENT_VALUE are included in Version 3.0 of SQL*Forms for compatibility with previous versions. Oracle Corporation recommends that you use the alternative system variables provided in Version 3.0.	SYSTEM.CURRENT_BLOCK 18-5 SYSTEM.CURRENT_FIELD 18-6 SYSTEM.CURRENT_VALUE 18-7
	You cannot use the SYSTEM.MESSAGE_LEVEL system variable to suppress prompts or vital error messages.	SYSTEM.MESSAGE_LEVEL 18-11
User Exits	Do not perform a database commit or rollback in a user exit if there are changes in a form that have not been posted to the database when the user exit is called.	EXEC SQL statements 19-4
	Adhere to the rules that apply to naming user exits and performing I/O from a user exit.	User exits, restrictions 19-8
	Keep test versions of your user exits in a separate IAPXTB table.	IAPXTB table 19-9
Working with objects	SQL*Forms does not support the ORACLE naming convention of double-quoted names.	Objects, naming conventions 20-2
	Do not use ORACLE reserved words to name SQL*Forms objects.	Reserved words 20-2
	Be aware of the restrictions on modifying referenced objects.	Modifying referenced objects 20-4

Chapter	*Advisory Statements*	*Related Topics*
Constraints	Be aware of the general SQL*Forms constraints on objects and cursors, and of the memory constraints of your system.	Constraints 22-1
Components	An illogical combination of command line switches will cause an error when you attempt to execute the command.	Combining command line switches 23-8
	The following switches have no effect in Version 3.0 of SQL*Forms: -n switch for SQL*Forms (Design) -o switch for SQL*Forms (Generate) -s switch for SQL*Forms (Generate)	-n switch for SQL*Forms (Design) 23-12 -o switch, SQL*Forms (Generate) 23-15 -s switch, SQL*Forms (Generate) 23-15
	The -o switch for SQL*Forms (Run Form) has no effect on statements in V3 triggers.	-o switch, SQL*Forms (Run Form) 23-18
	The -t switch for SQL*Forms (Run Form) has no effect if you replace standard SQL*Forms processing with On-insert, On-Update, and On-Delete triggers.	-t switch, SQL*Forms (Run Form) 23-20

Chapter	*Advisory Statements*	*Related Topics*
Form Storage	Do not try to transport a form from one type of computer to another by transporting the FRM file.	Portability among systems 24-3
	Do not import or export the IAP tables unless the target system has no forms in the database.	Portability among systems 24-3
	Oracle Corporation reserves the right to alter the SQL*Forms (Design) base tables at any time and without notice. Such a change could affect any reports or applications that you might have which depend on the names and contents of these tables.	SQL*Forms (Design), base tables 24-4
	Do not manipulate the SQL*Forms (Design) base tables through SQL*Plus.	SQL*Forms (Design), base tables 24-4
	Editing INP files can cause severe damage to your applications. Oracle Corporation does not support the editing of these files.	INP file format 24-13
Connecting to Other Products	Do not use a callable form in a user exit. Rather, use the CALL packaged procedure.	Calling forms from programs 25-10
Options	The Share Cursors for Implicit SQL run option has no effect if you replace standard SQL*Forms processing with On-Insert, On-Update, and On-Delete triggers.	Share Cursors for Implicit SQL run option 26-4
	The Share Cursors for Trigger SQL run option has no effect on statements in V3 triggers.	Share Cursors for Trigger SQL run option 26-4

Chapter	*Advisory Statements*	*Related Topics*
Upward Compatibility	When you convert an old INP file to Version 3.0 status, you should preserve the form's functionality.	Converting existing forms to Version 3.0 status B-2
System-specific Considerations	If you are designing forms in a block mode environment or for a block mode environment, be aware of the special considerations.	Block mode environment D-2
	If you are designing forms in a bit-mapped environment or for a bit-mapped environment, be aware of the special considerations.	Bit-mapped environment D-4

APPENDIX

D

SYSTEM-SPECIFIC CONSIDERATIONS

SQL*Forms works on many system configurations. This book describes how SQL*Forms works in a character mode environment. While the description is largely accurate for all environments, there are some variations that you should be aware of if you develop applications outside of the character mode environment and/or if you develop applications that operators will run outside of the character mode environment.

This chapter discusses system-specific considerations for developing SQL*Forms applications in the following environments

- block mode environment
- bit-mapped environment

Related Topics: 16-bit environment constraints 22-4

Block Mode Environment

If you are developing SQL*Forms applications in a block mode environment, you should be aware of the rules that apply on the following subjects:

- setting the validation unit
- using triggers
- creating fields

In addition, remember the following behavior, which is specific to using block mode terminals:

- A block mode terminal-unlike a character mode terminal—cannot recognize when an operator re-enters the current value of a field into that field.
- First letter menu selection does not work on a block mode terminal.
- In a block mode environment, SQL*Forms always acts as though the Automatic Hint field characteristic is turned off.

Related Topics: Automatic Hint field characteristic 12-5
Changed, field validation state 3-9

Setting the Validation Unit in Block Mode

In a block mode environment, the default validation unit is the block. This default best matches the behavior of the block mode environment. While you can specify any validation unit in this environment, your forms will run slower with the field or record validation unit due to the increased number of host interrupts. And, you would usually want more frequent validation than the form validation unit would provide.

If you are designing an application that will be run in a block mode environment, but you are working in a character mode environment, use the Set Block Validation as Default run option. This option defines a block as the validation unit instead of a field, and it enables you to more easily create and test a form that will be run on a block mode terminal.

Related Topics: Set Block Validation as Default run option 26-3
Validation unit 3-7

Using Triggers in Block Mode

By default, the forms that you design in a block mode environment will use the block validation unit. As a result, when you design your forms you should be especially careful to take into account that SQL*Forms only fires navigational triggers (e.g., Post-Field, Pre-Block) with definition levels at or above the validation unit. If you do use the field validation unit in the block mode environment, note that while SQL*Forms processes fields according to sequence number, the order in which operators move from field to field is a function of the terminal they are using.

Related Topics: Triggers, definition level 4-2

Creating Fields in Block Mode

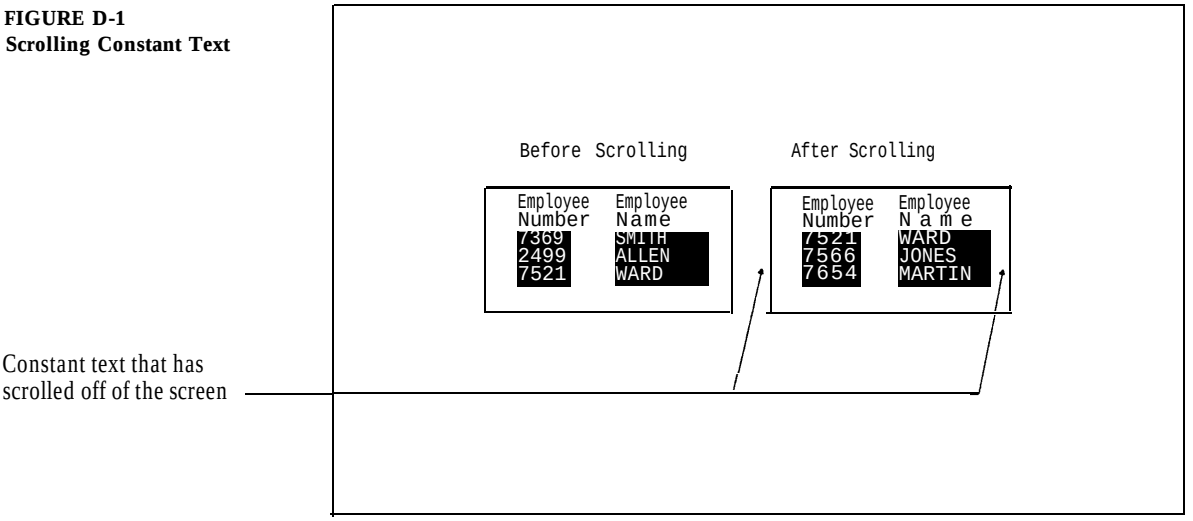
If you are developing forms that will run in a block mode environment, adhere to the following rules when you create fields:

- Insert a blank space before and after every field in the form. This format reserves space for an attribute byte, which identifies qualities about the field. If the space between two fields or between a field and constant text is not blank, the form will not run. Note that when you create a block through the default block facility, the required spaces are automatically inserted.
- Avoid placing part of a field or constant text in the last column of the screen.
- Ensure that no field or constant text in a pop-up page displays at the boundaries of the view-either upon display or upon scrolling. If a field or text does abut a boundary, it is possible for that element to conflict with an element on an underlying page.

Using Constant Text in Block Mode

Some terminals used in block mode environments use split screen scrolling, which scrolls an entire screen line at a time. This can cause undesirable screen behavior if you have constant text on the same line as a field that belongs to a multi-record block. When a record in that block is scrolled off the screen, the constant text on the same line is also scrolled out of view. Figure D-1 illustrates this behavior.

FIGURE D-1
Scrolling Constant Text



Bit-Mapped Environment

If you are developing SQL*Forms applications in a bit-mapped environment, you should be aware of how the user interface differs from character mode operation. Specifically, interface elements appear differently and there are alternatives to using function keys.

If you are developing SQL*Forms applications that will run in a bit-mapped environment, you should be aware of how navigation differs when an operator uses a mouse.

Interface Elements in a Bit-Mapped Environment

In bit-mapped environments, the SQL*Forms (Design) interface might appear differently than described in this book. For example, a check box will appear as an actual box instead of as a bracketed item. The differences exist because SQL*Forrns uses the interface elements native to the bit-mapped system on which it is running.

Function Keys in a Bit-Mapped Environment

In bit-mapped environments, you can use the SQL*Forms function keys to perform standard designer functions. However, you will probably want to use a mouse to perform these functions.

[Select]	Click the mouse on the item you are selecting. You can also use the mouse to turn a check box on and off (i.e., select it and deselect it).
All other function keys	If the function keys are mapped to buttons on your screen, click the mouse on the button that is assigned to the desired function.

Navigation in a Bit-Mapped Environment

If you are developing SQL*Forms applications to run in a bit-mapped environment, pay special attention to the Mouse Navigation Limit form characteristic. This characteristic defines what field(s) an operator can make current at any time by clicking on the field with the mouse.

By default, an operator can click on any displayed field in a form. However, it might be important for your application to restrict an operator's actions. Also, remember that if an operator navigates from one field to another field by clicking on the second field, SQL*Forms navigates directly to the target record and field and does not fire any key trigger defined in the first field (e.g., Key-NXTFLD). This behavior might affect the design of your applications.

Related Topics: Mouse Navigation Limit form characteristic 9-4

APPENDIX

E MESSAGES

This appendix discusses SQL*Forms messages. Specifically, it covers the following topics:

- SQL*Forms (Convert) messages
- SQL*Forms (Design) messages
- SQL*Forms (Generate) messages
- SQL*Forms (Run Form) messages

SQL*Forms (Convert) Messages

FRM-10001: Invalid application name.

Cause: The name does not follow ORACLE naming conventions.

Action: Check the ORACLE naming conventions and type in a different name.

FRM-10002: Can't read terminal definition

Cause: You specified an invalid terminal definition.

Action: Check with your DBA for the correct definition.

FRM-10004: Can't open input filename.

Cause: Either SQL*Forms could not find the input file, or an operating system error occurred while trying to read the file.

Action: Check to make sure that the filename is correct. If it is correct, contact your DBA for assistance.

FRM-10005: Can't open output filename.

Cause: Either SQL*Forms could not find the output file, or an operating system error occurred while trying to write the file.

Action: Check to make sure that the filename is correct. If it is correct, contact your DBA for assistance.

FRM-10006: Form doesn't exist.

Cause: There is no form with this name in the database.

Action: Enter the name of an existing form.

FRM-10007: Error opening cursor for form read.

Cause: SQL*Forms cannot open a cursor to read the form from the database

Action: Contact your DBA or call an Oracle support person.

FRM-10008: Error opening cursor for form store or delete.

Cause: SQL*Forms cannot open a cursor to save the form or to delete it from the database.

Action: Contact your DBA or call an Oracle support person.

FRM-10009: Premature end of file error at line <line number>.

Cause: This error occurs when SQL*Forms is reading a Version 2.3 INP file while converting (using -v23). This error can occur if you edit the INP file, which should be generated by SQL*Forms (Design). If you have not edited the INP file, then this error is caused by an internal ORACLE error.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA or call an Oracle support person if you have not edited the INP file.

FRM-10010: Input file sequence error at line <line number>.

Cause: This error occurs when SQL*Forms is reading a Version 2.3 INP file while converting (using -v23). This error can occur if you edit the INP file, which should be generated by SQL*Forms (Design). If you have not edited the INP file, then this error is caused by an ORACLE internal problem.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA or call an ORACLE support person if you have not edited the INP file.

FRM-10011: File does not contain valid SQL*Forms (Run Form) source.

Cause: This error occurs when SQL*Forms is reading a Version 2.3 INP file while converting (using -v23). This error can occur if you edit the INP file, which should be generated by SQL*Forms (Design). If you have not edited the INP file, then this error is caused by an internal ORACLE error.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA or call an Oracle support person if you have not edited the INP file.

FRM-10013: Block <block name> not found for field <field name>.

Cause: Form definition in database is inconsistent or invalid.

Action: Contact your DBA to correct the form definition in the database.

FRM-10014: Block <block name> not found for trigger.

Cause: Form definition in database is inconsistent or invalid.

Action: Contact your DBA to correct the form definition in the database.

FRM-10015: Field <field name> of block <block name> not found for trigger.

Cause: Form definition in database is inconsistent or invalid.

Action: Contact your DBA to correct the form definition in the database.

FRM-10016:SQL number <internal number> not found for text line number: <line number>:<text line>.

Cause: Form definition in database is inconsistent or invalid.

Action: Contact your DBA to correct the form definition in the database.

FRM-10018: Block <block name> not found for comment.

Cause: Form definition in database is inconsistent or invalid.

Action: Contact your DBA to correct the form definition in the database.

FRM-10019: Field <field name> of block <block name> not found for comment.

Cause: Form definition in database is inconsistent or invalid.

Action: Contact your DBA to correct the form definition in the database.

FRM-10020: Resource is currently locked by another process.

Cause: Another user is working on the same form as you are.

Action: Wait and try again later.

FRM-10035: Illegal field type <field name> in <block name><field name>.

Cause: This error can occur if you edit the INP file, which should be generated by SQL*Forms (Design). If you have not edited the INP file, then this error is caused by an internal ORACLE error.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA or call an Oracle support person if you have not edited the INP file.

FRM-10036: Warning! Prompt above line 1 for <block name>.<field name>.

Cause: This error can occur if you edit the INP file, which should be generated by SQL*Forms (Design). If you have not edited the INP file, then this error is caused by an internal ORACLE error.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA or call an Oracle support person if you have not edited the INP file.

FRM-10037: Warning! Prompt doesn't fit beside field <table name>.<column name>.

Cause: This error can occur if you edit the INP file, which should be generated by SQL*Forms (Design). If you have not edited the INP file, then this error is caused by an internal ORACLE error.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA or call an Oracle support person if you have not edited the INP file.

FRM-10038: Warning! Illegal list of values table.column <table name>.<column name> ignored.

Cause: There is a problem with the list of values table and column.

Action: Correct the list of values table and/or column.

FRM-10041: Expected <proper input line>...Found: <incorrect input line>.

Cause: This error can occur if you edit the INP file, which should be generated by SQL*Forms (Design). If you have not edited the INP file, then this error is caused by an internal ORACLE error.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA or call an Oracle support person if you have not edited the INP file.

FRM-10042: Error at line <line number> - Expected: Yes/No answer...Found: <incorrect value>.

Cause: This error can occur if you edit the INP file, which should be generated by SQL*Forms (Design). If you have not edited the INP file, then this error is caused by an internal ORACLE error.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA or call an Oracle support person if you have not edited the INP file.

FRM-10043: Error at line <line number> - Expected: Integer between <high value> and <low value> ...Found: <number>.

Cause: This error can occur if you edit the INP file, which should be generated by SQL*Forms (Design). If you have not edited the INP file, then this error is caused by an internal ORACLE error.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA or call an Oracle support person if you have not edited the INP file.

FRM-10048 Can't allocate <n> bytes of memory.

Cause: An ORACLE internal error has occurred.

Action: Contact your DBA.

FRM-l0051: Illegal command inform text at line <line number> number.

Cause: An ORACLE internal error has occurred.

Action: Contact your DBA.

FRM-10053: Block <block name> not found for reference.

Cause: You tried to reference or copy a non-existent block.

Action: Check the block name that you entered and try again.

FRM-10054: Field <field name> of block <block name> not found for reference.

Cause: You tried to reference or copy a non-existent field.

Action: Check the field name you typed in and try again.

FRM-10060: Couldn't read input file.

Cause: You tried to convert an old INP file to the new Version 3.0 format using the -v switch. SQL*Forms could not read the old INP file that you specified.

Action: Check your INP filename and retry.

FRM-10061: Error occurred at line <line number> while reading INP file.

Cause: An ORACLE internal error has occurred.

Action: Contact your DBA or call an Oracle support person.

FRM-10062: Internal error occurred at line <line number> while reading INP file.

Cause: An ORACLE internal error has occurred.

Action: Contact your DBA or call an Oracle support person.

FRM-10063: Out of memory error occurred at line <line number> while reading INP file.

Cause: An ORACLE internal error has occurred.

Action: Contact your DBA or call an Oracle support person.

FRM-10064: Invalid input at line <line number> while reading INP file.

Cause: This error occurs when SQL*Forms is reading a Version 2.3 INP file while converting (using -v23). This error can occur if you edit the INP file, which should be generated by SQL*Forms (Design). If you have not edited the INP file, then this error is caused by an internal ORACLE error.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA or call an Oracle support person if you have not edited the INP file.

FRM-10065: Invalid DEFINE at line <line number> number while reading INP file.

Cause: This error occurs when SQL*Forms is reading a Version 2.3 INP file while converting (using -v23). This error can occur if you edit the INP file, which should be generated by SQL*Forms (Design). If you have not edited the INP file, then this error is caused by an internal ORACLE error.

ORACLE Corporation does not support the editing of INP files.

Action: Contact your DBA or call an Oracle support person if you have not edited the INP file.

FRM-10066: Invalid ENDDEFINE at line <line number> while reading INP file.

Cause: This error occurs when SQL*Forms is reading a Version 2.3 INP file while converting (using -v23). This error can occur if you edit the INP file, which should be generated by SQL*Forms (Design). If you have not edited the INP file, then this error is caused by an internal ORACLE error.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA or call an Oracle support person if you have not edited the INP file.

FRM-10067: Invalid KEYWORD= VALUE at line <line number> in INP file.

Cause: This error occurs when SQL*Forms is reading a Version 2.3 INP file while converting (using -v23). This error can occur if you edit the INP file, which should be generated by SQL*Forms (Design). If you have not edited the INP file, then this error is caused by an internal ORACLE error.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA or call an Oracle support person if you have not edited the INP file.

FRM-10068: Need a VALUE at line <line number> in INP file.

Cause: This error occurs when SQL*Forms is reading a Version 2.3 INP file while converting (using -v23). This error can occur if you edit the INP file, which should be generated by SQL*Forms (Design). If you have not edited the INP file, then this error is caused by an internal ORACLE error.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA or call an Oracle support person if you have not edited the INP file.

FRM-10071: Couldn't write output file <file name>.

Cause: You tried to convert an old INP file to the new Version 3.0 format using the -v switch. SQL*Forms could not write the new Version 3.0 INP file.

Action: Contact your DBA.

FRM-10074: User is not logged in.

Cause: You pressed [List] while in the New Form dialog box. SQL*Forms cannot list the names of currently stored forms unless you are logged into the database.

Action: To log in to the database, do not specify the -n switch when you use the SQLFORMS command.

SQL*Forms (Design) Messages

FRM-20001: Field is too large for this form.

Cause: The field or area you selected is too large to copy or move to a new location. The field or area in the paste buffer is too large to paste to a new location.

Action: Either change the new location, or cancel the selection by pressing [Undo].

FRM-20002: This field will overlap another field.

Cause: You have tried to create or resize a field in such a way that it will overlap another field. Or, you have tried to move, copy, or paste a field or area to a new location where it will overlap another field.

Action: Change the size or position of the field to eliminate the overlap.

FRM-20003: Your form is too large for your CRT screen.

Cause: Your form is too large for your current display device.

Action: Resize the form, or use another display device with a larger screen.

FRM-20004: Cursor is outside the current form

Cause: You have moved the cursor outside the form that you are defining in the screen painter.

Action: Move the cursor back inside the form.

FRM-20005: You can press [Select] only twice at a time.

Cause: You pressed [Select] three or more times in succession.

Action: Do not press [Select] more than twice in succession.

FRM-20006: You pressed an undefined function key.

Cause: You pressed a function key that does not perform a function in the screen painter.

Action: None. Refer to the keyboard map from the screen painter for the correct function key.

FRM-20007: You cannot select a partial field.

Cause: You tried to select an area that includes part (but not all) of a field.

Action: Select an area that does not cut through a field.

FRM-20008: Field cannot span line boundary.

Cause: You have tried to create or resize a field that spans lines.

Action: None.

FRM-20009: Nothing has been selected.

Cause: You requested an operation, such as creating a new field with [Define Field], that requires one or more prior selections via [select].

Action: Perform the selections as required for the operation you requested. Then, request the operation again.

FRM-20010: You can resize only one field at a time.

Cause: You tried to resize an area that either does not contain a field or has more than one field.

Action: Cancel the selection by pressing [Undo]. Then, resize the field again.

FRM-20011: You can expand fields only to the right.

Cause: You tried to expand a field by moving its left margin.

Action: Expand the field by moving its right margin to the right. To get the effect of expanding the field to the left, move the entire field to the left and then move the right margin to the right.

FRM-20012: Nothing to undo.

Cause: You pressed [Undo], but there was no operation to undo.

Action: None.

FRM-20013: You cannot modify fields that do not belong to the current block

Cause: You have tried to modify a field that does not belong to the current block.

Action: Cancel the selection, if any.

FRM-20014: Nothing to paste.

Cause: You pressed [Paste] with nothing in the paste buffer.

Action: No action is necessary.

FRM-20015: Cannot open file with this name.

Cause: You have specified an invalid filename, or the file you named no longer exists.

Action: Correct the filename.

FRM-20016: Form name is invalid.

Cause: You have specified an illegal form name.

Action: Correct the form name.

FRM-20017: Form with this name already exists.

Cause: You tried to create a form with a name that is already in use.

Action: Choose a different name for the form.

FRM-20018: Form with this name does not exist.

Cause: You tried to refer to a form that does not exist.

Action: Enter the name of an existing form.

FRM-20019: Form name must be entered

Cause: You tried to perform an operation that requires a form name, but you have not entered or selected a name.

Action: Enter or select a form name.

FRM-20020: Block name is invalid.

Cause: You have specified an illegal block name.

Action: Correct the block name.

FRM-20021: Block with this name already exists.

Cause: You tried to create a block with a name that is already in use in this form.

Action: Choose a different name for the block, or rename the conflicting block.

FRM-20022: Block with this name does not exist.

Cause: You tried to refer to a block that does not exist.

Action: Enter the name of an existing block.

FRM-20023: Page number is invalid.

Cause: You entered a page number that either is not a legal number or is outside the range that SQL*Forms can accept,

Action: Enter a legal page number.

FRM-20024: Block name must be entered.

Cause: You tried to perform an operation that requires a block name, but you have not entered or selected a name.

Action: Enter or select a block name.

FRM-20025: Block sequence number is invalid.

Cause: You entered a block sequence number that either is not a legal number or is outside the range that SQL*Forms can accept.

Action: Enter a legal block sequence number.

FRM-20026: This table name is invalid.

Cause: You have specified an illegal table name. A legal table name must follow ORACLE naming conventions .

Action: Correct the table name.

FRM-20027: Table or view with this name does not exist.

Cause: You tried to refer to a table or view that does not exist.

Action: Enter the name of an existing table or view.

FRM-20028: You deleted a block that is being referred to elsewhere.

Cause: You deleted a block that is referred to elsewhere in the current form, such as in a trigger.

Action: Change the form so that it no longer refers to the deleted block.

FRM-20029: SQL sequence number is invalid.

Cause: In a trigger step, you entered a step sequence number that is not a legal number, or is outside the range that SQL*Forms can accept.

Action: Enter a legal step sequence number.

FRM-20030: Syntax error in SQL statement.

Cause: The SQL command you just entered or edited contains a syntax error.

Action: Correct the command. Refer to the *SQL Language Reference Manual* for information about SQL command syntax.

FRM-20031: Field does not exist in specified block.

Cause: You entered a field reference that refers to a block and field combination that does not exist.

Action: Enter a valid block and field combination.

FRM-20032: Number of displayed rows cannot exceed number of buffered rows.

Cause: In the Set Options form, you have requested that the form display more rows than are buffered.

Action: Increase the number of rows that are buffed, and/or decrease the number of rows that are displayed.

FRM-20033: Field name is invalid.

Cause: You have specified an illegal field name. A legal field name must follow object naming conventions .

Action: Correct the field name.

FRM-20034: Field with this name already exists for the current block.

Cause: You have tried to define a field with a name that is already in use in the current block.

Action: First be sure you are not redefining a field that has already been entered. If you are not, give this field a different name.

FRM-20035: This column does not exist in the specified table.

Cause: You referred to a table column that does not exist. This most often means that you gave a non-control field a name that does not correspond to the name of a column in the table associated with the current block.

Action: Either enter the name of a column in the table, or make this field a control field.

FRM-20036: Data type for this field does not match the column in the database table.

Cause: You have specified a data type for this field that is not compatible with the data type of the corresponding column in the associated table.

Action: Check the data type of the column in the table and specify a compatible data type for the field.

FRM-20037: Control block does not correspond to a table in the database.

Cause: You have selected an option for a control block that is meaningful only for non-control blocks.

Action: None. You cannot specify this option for this block unless you make the block a non-control block.

FRM-20038: Primary key field must be in the database table.

Cause: You selected the Primary Key field characteristic for a field without selecting the Base Table characteristic.

Action: Either turn on the Base Table field characteristic or turn off the Primary Key field characteristic.

FRM-20041: Field length cannot be less than display length

Cause: You specified a field length that is less than the display length as defined on the screen.

Action: Either specify a value for the Field Length field characteristic that is at least as great as the display length, or reduce the display length.

FRM-20042: Query length cannot be less than field length or greater than 255.

Cause: You specified a query length that is less than the field length or greater than 255 characters.

Action: Specify a value for the Query Length field characteristic that is at least as great as the field length but no greater than 255 characters.

FRM-20043: Field is part of primary key; default not applicable.

Cause: You specified a value for the Default Value field characteristic for a field that is part of the block's primary key.

Action: Delete the value for the Default Value field characteristic. This characteristic is not meaningful for a field that is part of the primary key.

FRM-20044: You cannot specify a low value that is greater than the high value.

Cause: You specified a Low setting for the Valid Range field characteristic that is greater than the High setting.

Action: Change either or both items so that the value for the Low setting is no greater than the value for the High setting.

FRM-20045: Displayed field is not allowed on page 0.

Cause: You tried to select the Displayed field characteristic for a field on page 0.

Action: None.

FRM-20046: There is no room on current page.

Cause: You tried to create a default block on a page that is already full.

Action: Create the default block on a new page.

FRM-20047: At least one column should be selected

Cause: You tried to create a default block with no columns selected

Action: Select at least one column.

FRM-20048: Cannot fit rows on current page.

Cause: You tried to create a multi-record block by using the default block facility, but the number of rows you asked to be displayed is more than the maximum number of rows that can be displayed on the current page.

Action: Reduce the number of rows to be displayed to the maximum number of rows than can be displayed (or less), or move the block to a page that has more room.

FRM-20049: At least one row should be displayed.

Cause: You tried to create a default block without specifying the number of rows to be displayed.

Action: Enter a number between 1 and the maximum number of rows that can be displayed.

FRM-20050: Trigger statement must be entered or deleted first.

Cause: You tried to define a new trigger step before defining the current trigger step.

Action: Either enter trigger statements for the current step or delete the step.

FRM-20051: Trigger name is invalid.

Cause: You have specified an illegal trigger name.

Action: Correct the trigger name.

FRM-20052: Trigger with this name already exists.

Cause: You tried to create a trigger with a name that is already in use at this level.

Action: Choose a different name for the trigger.

FRM-20053: Trigger with this name does not exist.

Cause: You tried to refer to a trigger that does not exist at this level.

Action: Enter the name of an existing trigger.

FRM-20054: Can't find the source block.

Cause: You have tried to copy or reference from a block that does not exist in the specified form.

Action: Enter the name of an existing block.

FRM-20055: Can't find the source field.

Cause: You have tried to copy or reference from a field that does not exist in the specified block.

Action: Enter the name of an existing field.

FRM-20056: Can't find the source trigger.

Cause: You have tried to copy or reference from a trigger that does not exist in the specified form.

Action: Enter the name of an existing trigger.

FRM-20057: Can't find the source procedure.

Cause: You have tried to copy or reference from a form-level procedure that does not exist in the specified form.

Action Enter the name of an existing form-level procedure.

FRM-20058: Can't find the source page.

Cause: You have tried to copy from a page that does not exist in the specified form.

Action: Enter the number of an existing page.

FRM-20059: Can't reference a page.

Cause: You can only copy a page.

Action: Change the operation to copy instead of reference.

FRM-20060: Source object must be specified.

Cause: You have tried to copy or reference from an object that is not specified correctly. The source object must be an existing block, field, trigger, procedure, or page.

Action: Enter the name of an existing object.

FRM-20061: Target field position is invalid.

Cause: While copying or referencing a field, the new X and Y coordinates you entered for the field are not legal numbers or are outside the range that SQL*Forms can accept.

Action: Enter legal X and Y coordinates for the field.

FRM-20062: Target block name does not exist.

Cause: While copying or referencing a trigger, you entered a block name that does not exist in the current form.

Action: Enter the name of an existing block.

FRM-20063: Target field name does not exist.

Cause: While copying or referencing a trigger, you entered a field name that does not exist in the specified block.

Action: Enter the name of an existing field.

FRM-20064: Target trigger already exists.

Cause: While copying or referencing a trigger, you entered a target trigger name that is already in use at the specified level of the current form.

Action: Enter a different name for the trigger.

FRM-20065: Target field already exists.

Cause: While copying or referencing a field, you entered a target field name that already exists in the current form.

Action: Enter a different name for the field.

FRM-20066: Target block already exists.

Cause: While copying or referencing a block, you entered a target block name that already exists in the current form.

Action: Enter a different name for the block.

FRM-20067: Target procedure already exists.

Cause: While copying or referencing a form-level procedure, the procedure name you entered is already in use in the current form.

Action: None.

FRM-20068: Can't reference a form.

Cause: You have tried to reference an entire form. You can only reference a block, field, trigger, or procedure. However, you can copy a form.

Action: None.

FRM-20069: Mismatched object type.

Cause: While copying or referencing an object, the target object type that you specified does not match the source object type. For example, you might have tried to copy a block to a trigger.

Action: Enter the correct source or target object

FRM-20070: Can't generate form before executing-no form in context.

Cause: You have chosen the Generate Before Executing Form design option and tried to execute a form, but there is no form in the current context.

Action: Either bring a form into the current context or turn off the Generate Before Executing Form option.

FRM-20071: Number of row buffered is invalid.

Cause: While defining a block, the number of buffered rows that you specified is not a legal number or is outside the range that SQL*Forms can accept.

Action: Enter a legal number for row buffered.

FRM-20072: Number of lines per record is invalid.

Cause: While defining a block, the number of lines per record that you specified is not a legal number or is outside the range that SQL*Forms can accept.

Action: Enter a legal number for lines per record.

FRM-20073: Block array size is invalid.

Cause: While defining a block, the block array size you specified is not a legal number or is outside the range that SQL*Forms can accept.

Action: Enter a legal block array size.

FRM-20074: Table name must be entered.

Cause: While creating a default block, you must enter the base table name.

Action: Enter the name of an existing table.

FRM-20075: Master block doesn't exist.

Cause: You have entered a master block name which does not exist in the current form.

Action: Enter or select the name of an existing block.

FRM-20076: Join condition must be entered.

Cause: You have specified a master block name without entering the join condition.

Action: Either enter the join condition or remove the master block name.

FRM-20077: Invalid join condition

Cause: You have entered a join condition which is syntactically incorrect.

Action: Correct the join condition.

FRM-20078: Can't create default block.

Cause: This is an internal ORACLE error, probably caused by insufficient memory.

Action: Contact your DBA or an Oracle support person.

FRM-20079: Input allowed fields must be displayed fields.

Cause: If you turn on the Input Allowed field characteristic, you must turn on the Displayed field characteristic.

Action: Either turn on the Displayed field characteristic or turn off the Input Allowed field characteristic.

FRM-20080: Query allowed fields must be displayed fields.

Cause: If you turn on the Query Allowed field characteristic, you must turn on the Displayed field characteristic.

Action: Either turn on the Displayed field characteristic or turn off the Query Allowed field characteristic.

FRM-20082: Update is only allowed for enterable fields.

Cause: If you turn on the Update Allowed field characteristic, you must turn on the Input Allowed field characteristic.

Action: Either turn on the Input Allowed field characteristic or turn off the Update Allowed field characteristic.

FRM-20083: Field name must be entered.

Cause: You tried to perform an operation that requires a field name, but you have not entered or selected a field name.

Action: Enter or select a field name.

FRM-20084: Field sequence number is invalid.

Cause: You entered a field sequence number that either is not a legal number or is outside the range that SQL*Forms can accept

Action: Enter a legal field sequence number.

FRM-20085: Field data type is invalid.

Cause: You entered an illegal field data type.

Action: Enter or select a legal data type.

FRM-20086: Field length is invalid.

Cause: You entered a field length that either is not a legal number or is outside the range that SQL*Forms can accept.

Action: Enter a legal field length.

FRM-20087: Field query length is invalid.

Cause: You entered a field query length that either is not a legal number or is outside the range that SQL*Forms can accept.

Action: Enter a legal field query length.

FRM-20088: Field display length is invalid.

Cause: You entered a field display length that either is not a legal number or is outside the range that SQL*Forms can accept.

Action: Enter a legal field display length.

FRM-20089: Field position is invalid.

Cause: You entered X and Y coordinates for a field location that are not legal numbers or are outside the range that SQL*Forms can accept.

Action: Enter legal X and Y coordinates.

FRM-20090: Page with this number already exists.

Cause: You tried to define a page with a page number that is already in use in the current form.

Action: First be sure you are not redefining a page that has already been defined. If you are not, give this page a new number.

FRM-20091: Warning! Page number has been changed.

Cause: You tried to change a page number of an existing page definition in the current form.

Action: First be sure you are not redefining a page that has already been defined. If you are not, give this page a new number.

FRM-20092: Procedure name is invalid.

Cause: You have specified an illegal procedure name. A legal procedure name must follow object naming conventions.

Action: Correct the procedure name.

FRM-20093: Trigger name must be entered.

Cause: You tried to perform an operation that requires a trigger name, but you have not entered a trigger name.

Action: Enter a trigger name.

FRM-20094: Trigger style is invalid.

Cause: You entered an illegal trigger style. A trigger style must be either V2 or V3.

Action: Enter or select a legal trigger style.

FRM-20095: User name must be entered.

Cause: In the Form Security dialog box, you must enter either the grantee username or the PUBLIC keyword.

Action: Enter either a grantee username or PUBLIC keyword.

FRM-20096: Filename must be entered.

Cause: You tried to perform an operation that requires a file name, but you have not entered a filename.

Action: Enter a filename and repeat the operation.

FRM-20097: Report generation failure.

Cause: This is an internal error that was caused while generating a document report on the form.

Action: Contact your DBA or an Oracle support person.

FRM-20098: Operation must be either copy or reference.

Cause: In the Source Object dialog box, you entered something other than COPY or REFERENCE into the Copy or Reference Object field.

Action: Enter either COPY or REFERENCE into the Copy or Reference Object field. You can enter the correct value by selecting it from the field's list of values.

FRM-20099: Owner name must be specified for referenced object.

Cause: In the Source Object dialog box, you did not enter an owner name when you tried to reference an object.

Action: Enter the owner name of the source object in the Owner field.

FRM-20100: Form name must be specified for referenced object.

Cause: In the Source Object dialog box, you did not enter a form name when you tried to reference an object.

Action: Enter the form name of the source object in the Form field.

FRM-20101: Operation must be either grant or revoke.

Cause: In the Form Security dialog box, you entered something other than GRANT or REVOKE into the Privilege field.

Action: Enter either GRANT or REVOKE into the Privilege field. You can enter the correct value by selecting it from the field's list of values.

FRM-20102: You can't modify fields that belong to a referenced block

Cause: In the screen painter, you tried to modify a field that belongs to a referenced block.

Action: Do not attempt to modify any fields that belong to a referenced block.

FRM-20103: You can't resize a referenced field.

Cause: In the screen painter, you tried to resize a referenced field.

Action: Do not attempt to resize a referenced field.

FRM-20104: Target object: bad name.

Cause: In the designer, you specified a nonexistent name when trying to copy an object.

Action: Specify a valid object name.

FRM-20119: Page size is too small to fit all fields on this page.

Cause: You defined a page at a size too small for its block.

Action: Change the parameters of the Page Size characteristic so that all the fields of its block fit on the page.

FRM-20124: Update Allowed and Update if Null are mutually exclusive.

Cause: You turned on both the Update Allowed and Update if Null field characteristics for a field.

Action: Turn off either the Update Allowed or Update if Null field characteristic.

FRM-20125: Echo input fields must be displayed fields.

Cause: If you turn on the Echo Input field charateristic, you must turn on the Displayed field characteristic.

Action: Either turn on the Displayed field characteristic or turn off the Echo Input field characteristic.

FRM-20126: Fixed length is only allowed for enterable fields.

Cause: If you turn on the Fixed Length field characteristic, you must turn on the Displayed and Input Allowed field characteristics.

Action: Either turn on the Displayed and Input Allowed field characteristics or turn off the Fixed Length field characteristic.

FRM-20127: Automatic skip is only allowed for enterable fields.

Cause: If you turn on the Automatic Skip field characteristic, you must turn on the Displayed and Input Allowed field characteristics.

Action: Either turn on the Displayed and Input Allowed field characteristics or turn off the Automatic Skip field characteristic.

FRM-20128: Copy/reference of this block causes field overlap.

Cause: While trying to copy or reference a block, you specified x, y, and page coordinates that would cause one or more of its fields to overlap existing fields.

Action: Either specify different target coordinates or move the fields of the target block to resolve the conflict.

FRM-20129: Search string not found within text region.

Cause: While performing a search in a multi-line text region, you specified a string not located between the current cursor position and the end of the text region.

Action: Either modify the search string or relocate the cursor position.

FRM-20204: Field attributes for field on page 0 have been overridden.

Cause: You tried to move a displayed field to page 0.

Action: No action necessary.

SQL*Forms (Generate) Messages

FRM-30001: V2 compatibility no longer supported.

Cause: You specified a command switch that is no longer supported by SQL*Forms.

Action: Check the documentation for legal command switches.

FRM-30009: No blocks were entered.

Cause: This error probably occurred because you edited the INP file.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA if you have not edited the INP file.

FRM-30011: No primary key fields in block.

Cause: You turned on the Primary Key block characteristic for the current block, but the block does not have any primary key fields.

Action: Either turn the Primary Key block characteristic off or designate a primary key field for the block.

FRM-30012: Block name must have at least one field.

Cause: You did not enter any fields in your block.

Action: Enter fields in your block or recreate the block with the default block facility.

FRM-30013: No such field <field name> in block <block name>.

Cause: You referenced a field that does not exist in the current block.

Action: Check the name of the field in the corresponding spread table. Then, either remove the reference to the field or correct the name of the field.

FRM-30014: No such block <block name>.

Cause: You referenced a block that does not exist in the current form.

Action: Check the name of the block in the corresponding spread table. Then, either remove the reference to the block or correct the name of the block.

FRM-30015: Ambiguous field name <field name>.

Cause: The field name you used exists in more than one block.

Action: Clarify the field name by specifying the name of the block in which it exists. To do this, use the block.field format (e.g., EMP.EMPNO, where EMP is the block name and EMPNO is the field name).

FRM-30016: No such field <field name>.

Cause: You referenced a field that does not exist in the current form.

Action: Check the name of the field in the corresponding spread table. Then, either remove the reference to the field or correct the name of the field.

FRM-30017: Warning! Literal not properly quoted.

Cause: This error probably occurred because you edited the INP file.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA if you have not edited the INP file.

FRM-30023: Workspace size must be between 0 and 255.

Cause: This error probably occurred because you edited the INP file.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA if you have not edited the INP file.

FRM-30028: WHERE clause is too long.

Cause: The SQL statement that you entered as part of a V2 trigger is too long.

Action: Shorten the statement. It can be up to 4096 characters long.

FRM-30032: Number of records displayed must be between 0 and 255.

Cause: The number of records displayed is not in the valid range. This error probably occurred because you edited the INP file.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA if you have not edited the INP file.

FRM-30033: Records buffered must be greater than records displayed.

Cause: The number of records buffered was not greater than the number of records displayed. This error probably occurred because you edited the INP file.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA if you have not edited the INP file.

FRM-30040: Not a valid field type.

Cause: The field type you entered was not a legal field type. This error probably occurred because you edited the INP file.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA if you have not edited the INP file.

FRM-30042: Length of field must be between 1 and 133.

Cause: The field length you entered was not in the valid range. This error probably occurred because you edited the INP file.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA if you have not edited the INP file.

FRM-30043: Length of field must be between 1 and 255.

Cause: The field length you entered was not in the valid range. This error probably occurred because you edited the INP file.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA if you have not edited the INP file.

FRM-30044: Display length must be between 1 and 133 but not more than length.

Cause: The field length you entered was not in the valid range. This error probably occurred because you edited the INP file.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA if you have not edited the INP file.

FRM-30045: Query length of field must be 0 (default) or greater than actual length.

Cause: The query length you entered was incorrect. This error probably occurred because you edited the INP file.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA if you have not edited the INP file.

FRM-30055: Warning! Prompt for field <block name>.<field name> was moved from line 1.

Cause: This error probably occurred because you edited the INP file.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA if you have not edited the INP file.

FRM-30056: Warning! Prompt for field <block name>.<field name> was truncated.

Cause: This error probably occurred because you edited the INP file.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA if you have not edited the INP file.

FRM-30069: Not a valid trigger name.

Cause: This error probably occurred because you edited the INP file.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA if you have not edited the INP file.

FRM-30081: SQL statement is too long.

Cause: The SQL statement that you entered as part of a V2 trigger is too long.

Action: Shorten the statement. It can be up to 4096 characters long.

FRM-30082: Only SELECT statements are allowed in non-commit trigger.

Cause: The trigger in error contains a SQL statement other than a SELECT statement. The statement is illegal in this type of trigger.

Action: Move the statement to a trigger where the statement is legal.

FRM-30085: You must enter Y, *Y, N, or *N.

Cause: This error probably occurred because you edited the INP file.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA if you have not edited the INP file.

FRM-30087: Too many blocks.

Cause: This is an internal ORACLE error.

Action: Contact your DBA or an Oracle support person.

FRM-30088: Too many fields.

Cause: This is an internal ORACLE error.

Action: Contact your DBA or an Oracle support person.

FRM-30089: Too many pages.

Cause: This is an internal ORACLE error.

Action: Contact your DBA or an Oracle support person.

FRM-30090: Too many boilerplate items.

Cause: This is an internal ORACLE error.

Action: Contact your DBA or an Oracle support person.

FRM-30091: Too many trigger steps.

Cause: This is an internal ORACLE error.

Action: Contact your DBA or an Oracle support person.

FRM-30092: Too many ranges.

Cause: This is an internal ORACLE error.

Action: Contact your DBA or an Oracle support person.

FRM-30093: Too many triggers.

Cause: This is an internal ORACLE error.

Action: Contact your DBA or an Oracle support person.

FRM-30094: Too many field pointers.

Cause: This is an internal ORACLE error.

Action: Contact your DBA or an Oracle support person.

FRM-30096: Couldn't create output file <file name>.

Cause: An operating system error prevented SQL*Forms from creating an output file.

Action: Contact your DBA or an Oracle support person.

FRM-30101: Too many strings.

Cause: This is an internal ORACLE error.

Action: Contact your DBA or an Oracle support person.

FRM-30102: Couldn't write all of the output file <file name>.

Cause: An operating system error prevented SQL*Forms from creating an output file.

Action: Contact your DBA or an Oracle support person.

FRM-30103: Invalid trigger name.

Cause: This error probably occurred because you edited the INP file.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA if you have not edited the INP file.

FRM-30109: Warning! Right-adjusted scrolled field is <block name>.<field name> .

Cause: The field is both a scrolled field and a right-adjusted field.

Action: Make the field either a scrolled field or a right-adjusted field.

FRM-30110: You must enter Y, *Y, **Y, N, *N, or **N.

Cause: This error probably occurred because you edited the INP file.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA if you have not edited the INP file.

FRM-30118: Couldn't create warning message file.

Cause: An operating system error prevented SQL*Forms from creating the warning file.

Action: Contact your DBA or an Oracle support person.

FRM-30119: Errors and warnings written to file <file name>.

Cause: A SQL*Forms (Generate) error occurred. Errors and warnings are in the specified file.

Action: Look at the specified file.

FRM-30120: Warning! Version 1 macros are no longer supported.

Cause: You used a SQL*Forms Version 1.0 macro (#MACRO, #MACROIF, or #KEY) that is not supported by SQL*Forms Version 2.3 or 3.0.

Action: Remove the macro and replace it with the appropriate V2 or V3 code.

FRM-30121: Error in the INTO clause or missing FROM clause.

Cause: An improper SQL statement was given in a V2 style trigger.

Action: Correct the SQL statement

FRM-30122: Invalid line number.

Cause: An invalid Y position was given for a field. This error probably occurred because you edited the INP file.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA if you have not edited the INP file.

FRM-30123: Invalid column number.

Cause: An invalid X position was given for a field. This error probably occurred because you edited the INP file.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA if you have not edited the INP file.

FRM-30124: Too many procedures.

Cause: This an internal ORACLE error.

Action: Contact your DBA or an Oracle support person.

FRM-30125: Too much library-unit memory.

Cause: This is an internal ORACLE error.

Action: Contact your DBA or an Oracle support person.

FRM-30129: Compile error on procedure specification for procedure <procedure name>.

Cause: You tried to compile a procedure that had an invalid specification.

Action: Correct the specification.

FRM-30130: Code given for trigger <trigger name> is not an anonymous block.

Cause: You entered trigger text that was not an anonymous block.

Action: Rewrite the trigger to make it an anonymous block

FRM-30131: Code given for trigger <trigger name> takes parameters.

Cause: You entered trigger text that takes parameters. This syntax is illegal.

Action: Rewrite the trigger. If you want to use parameters, write a form-level procedure instead.

FRM-30132: Code given for trigger <trigger name> is not executable.

Cause: You entered trigger text that is not executable.

Action: Rewrite the trigger.

FRM-30133: Compile error on trigger <trigger name>.

Cause: You entered incorrect trigger text.

Action: See the error screen for the compilation errors.

FRM-30134: Mismatched procedure and unit name for procedure <procedure name>.

Cause: The name in the Procedure Name field does not match the procedure defined in the Procedure Text field.

Action: Change one of the names to match the other.

FRM-30135: Code given for procedure <procedure name> is not a named unit.

Cause: You entered an anonymous block when you should have entered a form-level procedure.

Action: Rewrite the form-level procedure.

FRM-30136: Code given for procedure <procedure name> is not executable.

Cause: You entered non-executable code for the specified form-level procedure.

Action: Rewrite the form-level procedure.

FRM-30137: Compile error on procedure <procedure name>.

Cause: You entered an incorrect form-level procedure.

Action: See the error screen for the compilation errors.

FRM-30145: Too many bind variables.

Cause: This is an internal ORACLE error.

Action: Contact your DBA or an Oracle support person.

FRM-30147: Too many arguments on command line.

Cause: You specified too many variables.

Action: Check the syntax and retype the arguments.

FRM-30148: Too much buffer memory.

Cause: This is an internal ORACLE error.

Action: Contact your DBA or an Oracle support person.

FRM-30149: Too many lists of values.

Cause: This is an internal ORACLE error.

Action: Contact your DBA or an Oracle support person.

FRM-30150: Error in list of values text for field <field name>.

Cause: You specified an invalid list of values SQL statement.

Action: Rewrite the list of values SQL statement.

FRM-30156: Couldn't close form output file <file name>.

Cause: An operating system error prevented SQL*Forms from closing an output file.

Action: Contact your DBA or an Oracle support person.

FRM-30157: Length of LONG field must be between 0 and 65536.

Cause: Length of the LONG field was not in the valid range. This error probably occurred because you edited the INP file.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA if you have not edited the INP file.

FRM-30158: Display length must be between 0 and 133.

Cause: Display length was not in the valid range. This error probably occurred because you edited the INP file.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA if you have not edited the INP file.

FRM-30159: No text in trigger statement.

Cause: An empty trigger step was found. This error probably occurred because you edited the INP file.

Oracle Corporation does not support the editing of INP files.

Action: Contact your DBA if you have not edited the INP file.

FRM-30165: Username/password must be specified.

Cause: You did not specify a username and password on the command line.

Action: Re-enter the command line with a valid username and password.

SQL*Forms (Run Form) Messages

Through triggers, you can control the messages that operators see when they use a form. Specifically, you can control SQL*Forms (Run Form) messages in the following ways:

- You can suppress specific "severity levels" of messages with the SYSTEM.MESSAGE_LEVEL system variable.
- You can replace the standard processing of messages with the On-Error and On-Message triggers.

Message Severity Levels

SQL*Form (Run Form) ranks messages by severity. By using the SYSTEM.MESSAGE_LEVEL system variable, you can control the minimum severity level that displays to operators.

There are six levels of message severity that you can affect, listed here in increasing order of severity.

Severity Level	*Message Description*
0	(All types of messages from the other levels of severity.)
5	Reaffirms an obvious condition.
10	Indicates the operator has done something procedurally incorrect.
15	Declares that the operator is attempting to perform a function for which the form is not designed.
20	Indicates a condition where the operator cannot continue an intended action due to a problem with a trigger or another outstanding condition.
25	Indicates a condition that could result in the form performing incorrectly.
>25	Indicates a message severity level that you cannot suppress via the SYSTEM.MESSAGE_LEVEL system variable.

Note: The severity levels of individual SQL*Forms (Run Form) messages are specified on the following pages by the "Level" label.

Related Topics: SYSTEM.MESSAGE_LEVEL 18-11

Message Types

All messages that appear in SQL*Forms (Run Form) are considered "messages." However, if you plan to replace these messages via the On-Message and On-Error triggers, you need to be aware of message types. There are three types of SQL*Forms (Run Form) messages:

Informative	Informative messages inform the operator of the present state of processing (e.g., `Last value retrieved. )` or provide the operator with context-sensitive guidance (e.g., `Press [Accept] to enter answer.)`. You can suppress the appearance of these messages with On-Message triggers.
Error	Error messages inform the operator of error conditions that prevent the operator's actions (e.g., `Function key not allowed. Press [Show Function Keys] for list of valid keys.)`. You can suppress the appearance of these messages with On-Error triggers. However, you cannot suppress error messages that appear on the command line (e.g., `Too many arguments on command line. )`.
Working	Working messages inform the operator that SQL*Forms is currently processing (e.g., `working . . .)`. You cannot suppress these messages through On-Message triggers, On-Error triggers, or the SYSTEM.MESSAGE_LEVEL system variable.

Note: The message types of individual SQL*Forms (Run Form) messages are specified on the following pages by the "Type" label.

Related Topics: On-Error trigger 14-14
On-Message trigger 14-15

Attempting to reserve record for update or delete (CTRL—C to cancel)...

Cause: At the moment, another user has this row reserved. When that user is done, SQL*Forms will process your new update. In the meantime, you must wait.

Action: Eventually, another reserve for update message will appear. If you don't want to wait, press CTRL-C (the key sequence might be different in some environments) to cancel.

Type: Working

Working...

Cause: SQL*Forms has started a potentially time-consuming process or action.

Action: No action is necessary.

Type: Working

FRM-40001: Missing arguments on command line.

Cause: You did not enter all the arguments on the command line necessary to execute the command.

Action: Type `RUNFORM - ?` to see the actual syntax and retype the command properly.

Level: >25

Type: Error

FRM-40002: Undefined switch <switch name> on command line.

Cause: You mistyped the command by using an undefined command line switch.

Action: Type `RUNFORM - ?` to see the actual syntax and retype the command properly.

Level: >25

Type: Error

FRM-40003: Too many arguments on command line.

Cause: You entered too many arguments on the command line or you mistyped the command.

Action: Type `RUNFORM -?` to see the actual syntax and retype the command properly.

Level: >25

Type: Error

FRM-40007: Invalid user identifier or password. Re-enter.

Cause: You entered an incorrect ORACLE username or password.

Action: Retype your username and password properly.

Level: >25

Type: Error

FRM-40008: Name/Password combination too large. Re-enter.

Cause: You typed more than the 61-character limit in a username/password combination. Usernames and passwords are at most 30 characters each.

Action: Retype your username and password properly.

Level: >25

Type: Error

FRM-40010: Can't read form <formname>.

Cause: You entered a nonexistent formname or typed an incomplete path, or you do not have the proper privileges to run the form, or you do not have a generated copy of the form.

Action: Retype the formname correctly or provide the proper path name, or generate the form, or contact the DBA.

Level: 5

Type: Error

FRM-40011: Form was created by an old version of SQL*Forms (Generate).

Cause: The FRM file was created with an old and incompatible version of SQL*Forms (Generate).

Action: Regenerate the form or relink SQL*Forms (Generate).

Level: >25

Type: Error

FRM-40012: Form was created by a new version of SQL*Forms (Generate).

Cause: The FRM file was created by a new and incompatible version of SQL*Forms (Generate).

Action: Relink SQL*Forms (Run Form) and regenerate the form.

Level: >25

Type: Error

FRM-40013: Program Error: Error occurred while reading form.

Cause: An internal error occurred while SQL*Forms was trying to read the FRM file.

Action: Regenerate the form.

Level: >25

Type: Error

FRM-40014: Not enough memory to load the form.

Cause: Your computer does not have enough memory to run your form. This message refers to the computer's main memory, not to free disk space. It might indicate an error in the way the operating system is configured, which prevents SQL*Forms from gaining access to the memory it needs.

Action: The designer might be able to modify the form so that it will run. If that is not feasible, your installation must make more memory available, either by modifying the operating system parameters or by adding more memory to the computer.

Level: >25

Type: Error

FRM-40015: Unexpected end of file reading form.

Cause: The form was fragmented or incomplete.

Action: Regenerate the form.

Level: >25

Type: Error

FRM-40019: Unknown screen number to display.

Cause: Internal error.

Action: Contact your DBA.

Level: >25

Type: Error

FRM-40020: Page <page number> too small for this form.

Cause: Designer error. A field is positioned off of the page.

Action: Ensure that all fields that are associated with the given page fit completely on that page. You can reposition the fields or resize the page.

Level: >25

Type: Error

FRM-40021: Item in form file is too large.

Cause: An internal error occurred while SQL*Forms was reading the form

Action: Contact your DBA.

Level: >25

Type: Error

FRM-40023: Couldn't create record buffer file.

Cause: SQL*Forms could not create a buffer file because there is not enough space on the disk.

Action: Contact your DBA.

Level: >25

Type: Error

FRM-40024: Out of memory.

Cause: Internal error. Your computer does not have enough memory to run your form. This message refers to the computer's main memory, not to free disk space. It may indicate an error in the way the operating system is configured, which prevents SQL*Forms from gaining access to the memory it needs.

Action: The designer might be able to modify the form so that it will run. If that is not feasible, your installation must make more memory available, either by modifying the operating system parameters or by adding more memory to the computer.

Level: >25

Type: Error

FRM-40025: Can't suppress screen output without file input.

Cause: You tried to run a form on the command line using incompatible switches. The -w switch works only in conjunction with the -r switch.

Action: Retype the command to include both the -w and -r switches.

Level: >25

Type: Error

FRM-40026: Error opening key script file.

Cause: SQL*Forms can't open the file you specified with the -r command line switch.

Action: Make sure the file exists and the file protections are set properly. Or create a file with the -r command line switch.

Level: >25

Type: Error

FRM-40027: Error opening display spool file.

Cause: Operating system error. SQL*Forms can't open a file specified with the -w switch because there is insufficient disk space or you have specified an incorrect filename.

Action: Contact your DBA or system administrator.

Level: >25

Type: Error

FRM-40100: At first record.

Cause: You have pressed [Previous Record] while the cursor was at the first record.

Action: No action is necessary.

Level: 5

Type: Error

FRM-40101: Can't position to a key field. None are directly enterable.

Cause: You pressed [Next Primary Key Fld], but there are no enterable primary key fields in this block.

Action: Use [Next Field] for navigation rather than [Next Primary Key Fld].

Level: 10

Type: Error

FRM-40102: Record must be entered or deleted first.

Cause: You have pressed [Next Record] or [Down] in a context where it is meaningless. The last record in a block is the current record, or the block is empty, or you are in a "new" record in the middle of a block that was created using [Insert Record].

Action: No action is necessary.

Level: 5

Type: Error

FRM-40103: Can't position to a key field. None are query-able.

Cause: You tried to use [Next Primary Key Fld], but none of the primary key fields in the block allow you to enter query criteria.

Action: No action is necessary.

Level: 10

Type: Error

FRM-40104: No such block.

Cause: Designer error. A GO_BLOCK statement references a nonexistent block.

Action: Correct the statement.

Level: >25

Type: Error

FRM-40105: No such field.

Cause: Designer error. A GO_FIELD statement references a non-existent field.

Action: Correct the statement.

Level: >25

Type: Error

FRM-40106: No enterable fields in destination block

Cause: Designer error. A GO_BLOCK statement references a block with no enterable fields.

Action: Remove the statement or make at least one field in the block enterable.

Level: >25

Type: Error

FRM-40107: Can't position cursor to a non-displayable field.

Cause: Designer error. A GO_FIELD statement references a non-displayed field.

Action: Remove the statement or turn on the Displayed characteristic for the indicated field.

Level: >25

Type: Error

FRM-40200: Field is protected against update.

Cause: You have tried to update a field that does not allow updates.

Action: No action is necessary. You cannot update this field in this form.

Level: 15

Type: Error

FRM-40201: Field is full. Can't insert character.

Cause: SQL*Forms is in insert mode, and the current field is full.

Action: Delete a character to make room for the new character or press [Insert/Replace] to activate replace mode.

Level: 10

Type: Error

FRM-40202: Field must be entered.

Cause: You have not entered a value (or you have deleted a value) in a field that requires data input

Action: You must enter a value in this field.

Level: 15

Type: Error

FRM-40203: Field must be entered completely.

Cause: You have not entered a complete value (or you have deleted part of a value) in a field that has a fixed length requirement.

Action: Enter a complete value (one that extends to the end of the field).

Level: 15

Type: Error

FRM-40204: Cursor is at beginning of field value.

Cause: You tried to delete a character before the first character position of the field.

Action: Use [Delete Character] to delete the character that the cursor is on.

Level: 10

Type: Error

FRM-40205: Cursor is beyond the current field value.

Cause: On a block mode terminal, you positioned the cursor out of a field.

Action: Move the cursor into the field and try the entry again.

Level: 15

Type: Error

FRM-40206: Previous character is currently hidden.

Cause: You tried to delete a character that is off the screen.

Action: Scroll the character you want to delete into view using the arrow keys or [Scroll Left] and [Scroll Right].

Level: 10

Type: Error

FRM-40207: Must be in range <low value> to <high value>.

Cause: You entered a value not in the valid field range.

Action: Enter a value in the range shown.

Level: 15

Type: Error

FRM-40208: Form running in query-only mode. Can't change database fields.

Cause: You have entered a value on a query-only form.

Action: Do not enter values on this form. You can execute queries and view data, but you cannot alter existing data or enter new data.

Level: 15

Type: Error

FRM-40209: Field must be of form <format mask>.

Cause: The field value that you entered did not match the format mask on the field.

Action: Retry with a field value that matches the format mask.

Level: >25

Type: Error

FRM-40300: Field is not a column of the table being queried.

Cause: You entered query criteria into a non-queryable field.

Action: Do not attempt a query on this field.

Level: 15

Type: Error

FRM-40301: Query caused no records to be retrieved. Re-enter.

Cause: You entered a query while in Enter Query mode, but no records matched the query criteria. SQL*Forms remains in Enter Query mode.

Action: Either adjust the query criteria or press [Exit/Cancel] to leave Enter Query mode.

Level: >25

Type: Informative

FRM-40302: Cannot enter a query. No fields are queryable.

Cause: You pressed [Enter Query] with the cursor in a block that does not have any queryable fields.

Action: No action is necessary.

Level: 15

Type: Error

FRM-40350: Query caused no records to be retrieved.

Cause: The current query has fetched no records from the table. Either the table is empty or it contains no records that meet the query's search criteria.

Action: No action is necessary.

Level: 5

Type: Informative

FRM-40352: Last row of query retrieved.

Cause: You pressed [Down], [Next Record], [Next Set of Records], or [Scroll Down] after all records had been retrieved.

Action: No action is necessary.

Level: 5

Type: Informative

FRM-40353: Query cancelled.

Cause: Either the form executed an ABORT_QUERY function code or you pressed [Exit/Cancel] in Enter Query mode, or you pressed CTRL-C (or its equivalent) while SQL*Forms was fetching rows from the database.

Action: No action is necessary.

Level: The message level is 5 when the message is invoked by CTRL-C. It is 10 when the message is invoked by ABORT_QUERY or [Exit/Cancel] .

Type: Informative

FRM-40355: Query will retrieve <number> records.

Cause: You pressed [Count Query Hits]. If you now press [Execute Query], this number of records will be retrieved.

Action: No action is necessary.

Level: 25

Type: Informative

FRM-40356: Invalid number in example record. Query not issued.

Cause: In Enter Query mode, you entered an invalid number in the example record.

Action: Correct the entry and retry the query.

Level: >25

Type: Error

FRM-40357: Invalid string in example record. Query not issued.

Cause: In query mode, you entered an invalid ALPHA or CHAR value in the example record.

Action: Correct the entry and retry the query.

Level: >25

Type: Error

FRM-40358: Invalid date in example record. Query not issued.

Cause: In Enter Query mode, you entered an invalid DATE in the example record.

Action: Correct the entry and retry the query.

Level: >25

Type: Error

FRM-40359: Invalid date or time in example record. Query not issued.

Cause: In Enter Query mode, you entered an invalid JDATE, EDATE, or TIME value in the example record.

Action: Correct the entry and retry the query.

Level: >25

Type: Error

FRM-40400: Transaction complete - <number> records posted and committed.

Cause: You finished a commit.

Action: No action is necessary.

Level: 5

Type: Informative

FRM-40401: No changes to commit.

Cause: You tried to perform a commit, but no records were added or modified since the last post or commit.

Action: No action is necessary.

Level: 5

Type: Error

FRM-40402: Commit cancelled.

Cause: You pressed CTRL-C (or the equivalent) while waiting for a lock.

Action: No action is necessary.

Level: 10

Type: Informative

FRM-40403: A calling form has unposted changes. Commit not allowed.

Cause: A calling form has unposted changes.

Action: Post the changes or return to the calling form and retry the commit.

Level: 15

Type: Error

FRM-40404: Database posting completed- <number> records posted.

Cause: You finished a post.

Action: No action is necessary.

Level: 5

Type: Informative

FRM-40405: No changes to post.

Cause: You tried to perform a post, but no records were added or modified since the last post or commit.

Action: No action is necessary.

Level: 5

Type: Error

FRM-40406: Transaction complete - <number> records posted; all records committed.

Cause: You finished a commit that recorded your changes and committed previously posted changes.

Action: No action is necessary.

Level: 5

Type: Informative

FRM-40407: Transaction complete - posted records committed.

Cause: You finished a commit that committed previously posted changes.

Action: No action is necessary.

Level: 5

Type: Informative

FRM-40501: ORACLE error-unable to reserve record for update or delete.

Cause: A fatal error occurred while SQL*Forms was trying to select the record for update.

Action: Pressing [Display Error] provides more information, if it is available. You can also try to update or delete this record later. If necessary, contact your DBA.

Level: >25

Type: Error

FRM-40502: ORACLE error - unable to read list of values.

Cause: A fatal error occurred while SQL*Forms was trying to read a list of values.

Action: Pressing [Display Error] provides more information, if it is available. You can also try to update or delete this record later. If necessary, contact your DBA.

Level: >25

Type: Error

FRM-40504: ORACLE error- unable to execute a <trigger type> trigger.

Cause: A fatal error occurred while SQL*Forms was trying to execute a trigger.

Action: Pressing [Display Error] provides more information, if it is available. You can also try to update or delete this record later. If necessary, contact your DBA.

Level: >25

Type: Error

FRM-40505: ORACLE error-unable to perform query.

Cause: SQL*Forms has encountered a processing error. The table associated with the current block of the form might not exist, or your username might not have authority to perform the specified action on the table.

Action: Pressing [Display Error] provides more information, if it is available. You can also try to update or delete this record later. If necessary, contact your DBA.

Level: >25

Type: Error

FRM-40506: ORACLE error-unable to check for record uniqueness.

Cause: SQL*Forms encountered a processing error while checking a record's primary key fields for uniqueness. The table associated with the current block of the form does not exist, or you do not have authority to access the table.

Action: Contact your DBA.

Level: >25

Type: Error

FRM-40507: ORACLE error - unable to fetch next query record.

Cause: A fatal error occurred while SQL*Forms was trying to fetch the next query record.

Action: Contact your DBA.

Level: >25

Type: Error

FRM-40508: ORACLE error-unable to INSERT record.

Cause: A fatal error occurred while SQL*Forms was trying to insert a record. The table associated with the current block of the form might not exist, your username might not have authority to perform the specified action on the table, or some other reason might have caused the fatal error.

Action: Contact your DBA.

Level: >25

Type: Error

FRM-40509: ORACLE error- unable to UPDATE record.

Cause: A fatal error occurred while SQL*Forms was trying to update a record. The table associated with the current block of the form might not exist, your username might not have authority to perform the specified action on the table, or some other reason might have caused the fatal error.

Action: Contact your DBA.

Level: >25

Type: Error

FRM-40510: ORACLE error - unable to DELETE record.

Cause: A fatal error occurred while SQL*Forms was trying to delete a record. The table associated with the current block of the form might not exist, your username might not have authority to perform the specified action on the table, or some other reason might have caused the fatal error.

Action: Contact your DBA.

Level: >25

Type: Error

FRM-40512: ORACLE error -- unable to issue SAVEPOINT command.

Cause: As SQL*Forms was attempting to call a new form or was trying to commit, the issued SAVEPOINT command failed. This generally means that the application has run out of savepoints.

Action: Press [Display Error] to display the specific ORACLE error. You might be able to increase the maximum number of savepoints in the INIT.ORA file.

Level: >25

Type: Error

FRM-40600: Row has already been inserted.

Cause: You have attempted to insert or update a record, but SQL*Forms has enforced uniqueness on the block's primary key fields. The record, as inserted or updated, is not unique.

Action: Change the values in one or more primary key fields of the current record, making them unique. If the requirement of unique primary key fields creates difficulties, consider eliminating the constraint.

Level: 25

Type: Error

FRM-40602: Can't insert into or update data in a view.

Cause: You tried to modify the contents of a view in a manner that is not permitted. (A view is a set of rows extracted from one or more tables. It functions like a table; a form may refer to both views and tables. During a query, there is no difference between a view and a table. There are, however, certain restrictions on a view's insert, update, and delete operations.)

Action: No action is necessary; you cannot perform the operation you have attempted.

Level: 20

Type: Error

FRM-40603: Records no longer reserved for update. Re-query to make changes.

Cause: You committed your modifications in a block where you had previously entered an ENTER_QUERY or EXECUTE_QUERY packaged procedure with the FOR_UPDATE parameter. This action released all locks on the records in this block.

Action: If you want to modify the block, you will need to re-query.

Level: >25

Type: Error

FRM-40651: Record is now reserved for update or delete.

Cause: The lock you were waiting for is now in effect.

Action: No action is necessary.

Level: >25

Type: Informative

FRM-40652: Can't lock table in shared update mode.

Cause: You have received this message for one of two reasons

1. You do not have access to this table.
2. SQL*Forms cannot lock the table in shared update mode.

Action: Contact your DBA.

Level: 15

Type: Error

FRM-40653: Record not reserved for update or delete. Try again later.

Cause: You pressed CTRL-C (or the equivalent) to cancel. The operation that was attempting to update or delete the record terminates with an error.

Action: No action is necessary.

Level: 20

Type: Informative

FRM-40654: Record changed by another user. Re-query to see change.

Cause: Another user has updated this record since you performed a query and has changed at least one field in the record. Your actions have not changed the record in memory.

Action: You can update or delete this record now only if another user has restored the field values back to the way they were when you performed the query. Otherwise, you must re-query to fetch and display the new record into the form before you can update or delete it.

Level: 20

Type: Error

FRM-40655: SQL error forced rollback: Clear form and re-enter transaction

Cause: A deadlock or some other error has caused the current transaction to fail. SQL*Forms has rolled back the changes you have made in the transaction.

Action: Clear the form (or exit and re-enter the form) and re-enter the transaction. You might have to modify the form's design to prevent the error from recurring.

Level: 25

Type: Error

FRM-40656: Update can't be made due to prior rollback. Clear the record.

Cause: This record was already updated, but when you attempted to commit your changes, a serious error prevented this update or any further update or delete from being performed. The error might have occurred due to one of the following reasons

1. A deadlock forced the loss of row locks, or
2. The record had a database cluster key, and the previous attempt to update the record in the database was rolled back due to an error somewhere else in the form.

Action: You must clear this record before you can commit any other transactions in the form.

Level: 25

Type: Error

FRM-40657: Record changed or deleted by another user.

Cause: Another user has deleted the record since the query was executed.

Action: You can clear this record from your screen, but you cannot update or delete it since it no longer exists in the database.

Level: 20

Type: Informative

FRM-40700: No such trigger: <trigger name>.

Cause: Designer error. The form attempted to execute a trigger that doesn't exist, causing a fatal error.

Action: Correct the reference to the trigger.

Level: 20

Type: Error

FRM-40704: Illegal SQL statement in query-only mode.

Cause: Designer error. The form tried to execute a function that is illegal in a query-only form.

Action: You might need to redesign the form.

Level: 20

Type: Error

FRM-40705: Illegal SQL statement in non-commit-time trigger.

Cause: Designer error. The current trigger contains a SQL statement that is illegal for the trigger type.

Action: Rewrite the trigger text or use a different type of trigger.

Level: 20

Type: Error

FRM-40714: Function illegal in this context.

Cause: Designer error. The current trigger contains an illegal. function code.

Action: Rewrite the trigger text or use a different type of trigger.

Level: 20

Type: Error

FRM-40720: No statements found in list of statements.

Cause: Designer error. A step in a V2 trigger uses incorrect syntax.

Action: Correct the trigger step text.

Level: >25

Type: Error

FRM-40721: Missing function code in trigger.

Cause: Designer error. A step in a V2 trigger uses incorrect syntax.

Action: Correct the trigger step text.

Level: >25

Type: Error

FRM-40722: Unrecognized function name.

Cause: Designer error. A trigger references a nonexistent function code.

Action: Correct the reference.

Level: >25

Type: Error

FRM-40723: Missing argument for function code.

Cause: Designer error. An argument is missing in an ERASE, EXECUTE_TRIGGER, GO_BLOCK, GO_FIELD, GO_RECORD, GOSTEP or NEW_FORM statement.

Action: Correct the statement

Level: >25

Type: Error

FRM-40724: Missing selector in CASE statement.

Cause: Designer error. The selector portion is missing in a CASE statement.

Action: Correct the statement.

Level: >25

Type: Error

FRM-40725: Missing keyword in CASE statement.

Cause: Designer error. A keyword is missing in a CASE statement.

Action: Correct the statement.

Level: >25

Type: Error

FRM-40726: Missing choice in CASE statement.

Cause: Designer error. A choice is not specified in a CASE statement

Action: Correct the statement.

Level: >25

Type: Error

FRM-40727: Illegal keyword in CASE statement.

Cause: Designer error. A CASE statement contains a mistyped keyword.

Action: Correct the statement.

Level: >25

Type: Error

FRM-40728: Unrecognized or extra arguments to function.

Cause: Designer error. A function specifies extra arguments.

Action: Correct the statement.

Level: >25

Type: Error

FRM-40729: Missing semicolon at end of statement.

Cause: Designer error. A semicolon is missing at the end of a function code.

Action: Correct the statement

Level: >25

Type: Error

FRM-40730: Invalid message suppress level-unchanged from <current message suppression level>.

Cause: Designer error. A trigger attempted to set the system message level to an invalid number.

Action: Reset the SYSTEM.MESSAGE_LEVEL system variable to a valid number.

Level: >25

Type: Error

FRM-40731: INTO keyword missing.

Cause: Designer error. A COPY statement is missing the INTO clause.

Action: Correct the statement

Level: >25

Type: Error

FRM-40733: PL/SQL error - internal SQL*Forms error occurred.

Cause: A fatal error occurred in SQL*Forms or in PL/SQL during trigger execution.

Action: Notify the DBA.

Level: 25

Type: Error

FRM-40734: PL/SQL error - internal PL/SQL error occurred.

Cause: A fatal error occurred in PL/SQL during trigger execution.

Action: Notify the DBA.

Level: 25

Type: Error

FRM-40735: <trigger name> trigger raised unhandled exception ORA-<exception name>.

Cause: Designer error. The current trigger an exception (other than FORM_TRIGGER_FAILURE), but it did not handle the exception.

Action: Rewrite the trigger text to handle the exception.

Level: 25

Type: Error

FRM-40736: Cannot initialize PL/SQL.

Cause: An internal error occurred while initialing PL/SQL.

Action: Contact your DBA.

Level: >25

Type: Error

FRM-40737: Illegal resticted procedure <procedure> in <trigger> trigger.

Cause: Designer error. A trigger tried to execute a restricted packaged procedure.

Action: Remove the packaged procedure from the trigger text.

Level: >25

Type: Error

FRM-40738: PL/SQL error. NULL argument to procedure not allowed.

Cause: Designer error. A NULL argument was passed to a packaged procedure expecting a non-null parameter.

Action: Check the packaged procedure definition and rewrite the procedure call.

Level: >25

Type: Error

FRM-40739: CLEAR_FORM with FULL.ROLLBACK not allowed in post-only form.

Cause: Designer error. A trigger tried to issue a CLEAR_FORM packaged procedure with the FULL_ROLLBACK parameter in a post-only, called form.

Action: Remove the FULL_ROLLBACK parameter or ensure that the calling form does not have unposted changes when the call occurs.

Level: >25

Type: Error

FRM-40800: User exit <user exit name> does not exist.

Cause: The form tried to invoke a user exit that does not exist. You might be using the wrong version of SQL*Forms (Run Form), or there might be an error in the form.

Action: If you are using the wrong version of SQL*Forms (Run Form), determine which version contains the user exit, and use that version to run the form. If there is an error in the form, the designer should correct the call to the user exit.

Level: 20

Type: Error

FRM-40801: Out of memory processing user exit.

Cause: A fatal error occurred.

Action: Contact your DBA.

Level: 25

Type: Error

FRM-40804: Not enough arguments to COPY command.

Cause: Designer error. A COPY statement does not specify a source or destination

Action: Correct the statement.

Level: >25

Type: Error

FRM-40805: Not enough arguments to ERASE command.

Cause: Designer error. An ERASE statement does not specify a variable.

Action: Correct the statement.

Level: >25

Type: Error

FRM-40807: No command string specified for HOST command.

Cause: Designer error. A HOST statement does not specify a command string.

Action: Correct the statement.

Level: >25

Type: Error

FRM-40808: Can't execute HOST command. Error code= <hex address>.

Cause: SQL*Forms cannot execute a HOST statement because of an operating system error.

Action: Contact your DBA.

Level: 25

Type: Error

FRM-40809: HOST command had error code= <hex address>.

Cause: The operating system command resulted in the above error code.

Action: Verify that you entered the command properly.

Level: 25

Type: Error

FRM-40811: Shell command had error code= <hex address>.

Cause: The operating system command resulted in the above error code.

Action: Verify that you entered the command properly.

Level: 25

Type: Error

FRM-40814: Illegal function code in this context

Cause: Designer error. The NOFAIL keyword is specified for a function code that does not accept that keyword.

Action: Correct the statement

Level: >25

Type: Error

FRM-40815: Variable <global variable name> does not exist.

Cause: Designer error. A trigger references a global variable that does not exist.

Action: Create the global variable or remove the reference.

Level: >25

Type: Error

FRM-40816: Could not allocate memory for new symbol.

Cause: Internal error. SQL*Forms ran out of memory while accessing a new global variable.

Action: Contact your DBA.

Level: >25

Type: Error

FRM-40817: Could not allocate memory for new value.

Cause: Internal error. SQL*Forms ran out of memory while accessing a new global variable.

Action: Contact your DBA.

Level: >25

Type: Error

FRM-40818: System variable name not defined.

Cause: Designer error. A trigger references a system variable that does not exist.

Action: Correct the statement.

Level: >25

Type: Error

FRM-40819: System variable isn't modifiable.

Cause: Designer error. A trigger attempted to put a value into a non-modifiable system variable.

Action: Correct the statement.

Level: >25

Type: Error

FRM-40820: Not enough memory to evaluate system variable.

Cause: Internal error. SQL*Forms ran out of memory while accessing a system variable.

Action: Contact your DBA.

Level: >25

Type: Error

FRM-40822: Not enough arguments to SET function.

Cause: Designer error. A SET function code does not specify enough arguments.

Action: Correct the statement

Level: >25

Type: Error

FRM-40823: Extra argument to SET function.

Cause: Designer error. A SET function code specifies too many arguments.

Action: Correct the statement

Level: >25

Type: Error

FRM-40824: Not enough arguments to PAGE function.

Cause: Designer error. A HIDEPAGE or a SHOWPAGE function code does not specify enough arguments.

Action: Correct the statement.

Level: >25

Type: Error

FRM-40825: Not enough arguments to VIEW function.

Cause: Designer error. A MOVEVIEW, ANCHORVIEW, or RESIZEVIEW function code does not specify enough arguments.

Action: Correct the statement.

Level: >25

Type: Error

FRM-40826: Invalid number of arguments to CALL function:

Cause: Designer error. A CALL or CALLQRY function code does not specify enough arguments.

Action: Correct the statement.

Level: >25

Type: Error

FRM-40827: Not enough arguments to GOREC function.

Cause: Designer error. A GOREC function code does not specify enough arguments.

Action: Correct the statement.

Level: >25

Type: Error

FRM-40831: Truncation occurred: Value too long for field <field name>.

Cause: Designer error. A trigger, query, or user exit read a value into a field that is not long enough to hold the entire value. The field truncated the value.

Action: You might want to increase the target field's field length.

Level: 15

Type: Error

FRM-40901: Note: not enough memory to remember all or part of this query.

Cause: Internal error.

Action: Contact your DBA.

Level: >25

Type: Error

FRM-40902: SQL statement too large.

Cause: Designer error. The form's design includes a SQL command that is more than 2048 characters long.

Action: Shorten the SQL command.

Level: >25

Type: Error

FRM-40903: Couldn't create output file.

Cause: You pressed [Print Screen], but SQL*Forms could not write the contents of the screen to a file. Possible reasons include you have entered an illegal filename; the operating system does not give you authority to create files; or the necessary disk or directory space is not available.

Action: Check the filename you have entered and correct it if necessary. If you need additional help, contact your DBA.

Level: >25

Type: Error

FRM-40904: Program error: Don't know operation to be performed on record.

Cause: Internal error.

Action: Contact your DBA.

Level: >25

Type: Error

FRM-40906: FATAL ERROR: Can't write a buffered record to disk.

Cause: SQL*Forms failed while trying to write a buffered record to the disk.

Action: Contact your DBA.

Level: >25

Type: Error

FRM-40907: FATAL ERROR: Can't read a buffered record from disk.

Cause: SQL*Forms failed while trying to read a buffered record from the disk.

Action: Contact your DBA.

Level: >25

Type: Error

FRM-40908: Internal Error: Unknown screen number to display.

Cause: Internal error.

Action: Contact your DBA.

Level: >25

Type: Error

FRM-40911: Record not created due to sequence number generation error.

Cause: Internal error. Either the sequence number object does not exist, or the designer does not have privileges for the sequence number object, or some other fatal database error occurred.

Action: Contact your DBA.

Level: >25

Type: Error

FRM-40913: List of values maximum exceeded. Some values are not displayed.

Cause: Designer error. SQL*Forms is unable to return all the records in the current list of values; there number exceeds the maximum limit.

Action: Specify no more than 32,767 records to be returned in a list of values.

Level: >25

Type: Error

FRM-41000: You pressed an undefined function key.

Cause: You pressed an undefined function key.

Action: Press [Show Keys] to determine which function key you should have pressed.

Level: 5

Type: Error

FRM-41001: Function not allowed on this device.

Cause: You tried to execute the Insert/Replace function. This function is not allowed in a block mode or bit-mapped environment.

Action: No action is necessary.

Level: 5

Type: Error

FRM-41002: Invalid selection.

Cause: You entered an invalid selection number on the block menu; that block does not exist in this form.

Action: Select an existing block.

Level: 10

Type: Error

FRM-41003: This block does not correspond to a table. Function ignored.

Cause: You tried to perform a function that references a table. The current block is a control block (one that does not correspond to any table), so the function you requested is meaningless.

Action: No action is necessary. You cannot perform the requested function on this block.

Level: 10

Type: Error

FRM-41004: Function key not allowed in this mode.

Cause: You pressed a function key that does not work in this mode.

Action: No action is necessary.

Level: 10

Type: Error

FRM-41005: Function key not implemented (internal error).

Cause: You pressed a disabled function key.

Action: No action is necessary. You cannot use the function key in the current context unless the form's definition is modified.

Level: 25

Type: Error

FRM-41006: Can't find SQL*Forms (Run Form) command.

Cause: You pressed a function key that has not been defined for your keyboard map.

Action: No action is necessary. You can press [Show Keys] to display the keyboard map.

Level: >25

Type: Error

FRM-41007: Cursor not in a valid field. Function key was ignored.

Cause: You were outside of the field when you pressed the function key.

Action: Position the cursor inside the field and press the function key again.

Level: 10

Type: Error

FRM-41008: Undefined function key. Press [Show Keys] for list of valid keys.

Cause: You pressed an undefined function key.

Action: Press [Show Keys] to determime which function key you should have pressed.

Level: >25

Type: Error

FRM-41009: Function key not allowed Press [Show Keys] for list of valid keys.

Cause: You pressed a function key that is not allowed in this environment.

Action: Press [Show Keys] to determine which function key you should have pressed.

Level: >25

Type: Error

FRM-41010: Can't set attribute of the current field.

Cause: Designer error. A SET_FIELD statement tried to turn off the Input Allowed characteristic for the current field.

Action: Eliminate the statement or rewrite the trigger.

Level: >25

Type: Error

FRM-41011: Undefined attribute.

Cause: Designer error. A DISPLAY_FIELD or DISPLAY_PAGE statement tried to set an undefined attribute, or a SET_FIELD statement tried to set an undefined characteristic.

Action: Correct the statement.

Level: >25

Type: Error

FRM-41012: Undefined field or variable reference.

Cause: Designer error. A NAME_IN statement tried to reference a non-existent field or variable.

Action: Correct the statement.

Level: >25

Type: Error

FRM-41013: Undefined attribute specified for field <block name>.<field name>.

Cause: Designer error. A SET_FIELD packaged procedure specifies an undefined parameter.

Action: Correct the statement.

Level: >25

Type: Error

FRM-41014: Cannot set attribute of page 0 field <block name>.<field name>.

Cause: Designer error. A SET_FIELD packaged procedure tried to change some characteristic of a page 0 field.

Action: Move the field to a non-0 page.

Level: >25

Type: Error

FRM-41015: Cannot set ENTERABLE attribute of the current field <block name>.<field name>.

Cause: Designer error. A SET_FIELD packaged procedure tried to change the Input Allowed characteristic of the current field.

Action: Correct the statement.

Level: >25

Type: Error

FRM-41016: Cannot set DISPLAYED attribute of the current field <block name>.<field name>.

Cause: Designer error. A SET_FIELD packaged procedure tried to change the Displayed characteristic of the current field.

Action: Correct the statement.

Level: >25

Type: Error

FRM-41017: Cannot set UPDATEABLE attribute of non-enterable field <block name>.<field name>.

Cause: Designer error. A SET_FIELD packaged procedure tried to turn on the Update Allowed characteristic of a non-enterable field.

Action: To turn on the Update Allowed characteristic of a field you must also turn on the Input Allowed characteristic of the field.

Level: >25

Type: Error

FRM-41018: Cannot set UPDATE_NULL attribute of non-enterable field <block name>.<field name>.

Cause: Designer error. A SET_FIELD packaged procedure tried to turn on the Update if Null characteristic of a non-enterable field.

Action: To turn on the Update if Null characteristic of a field you must also turn on the Input Allowed characteristic of the field.

Level: >25

Type: Error

FRM-41019: Cannot set REQUIRED attribute of non-enterable field <block name>.<field name>.

Cause: Designer error. A SET_FIELD packaged procedure tried to turn on the Required characteristic of a non-enterable field.

Action: To turn on the Required characteristic of a field you must also turn on the Input Allowed characteristic of the field.

Level: >25

Type: Error

FRM-41020: Cannot set ENTERABLE attribute of non-displayed field <block name>.<field name>.

Cause: Designer error. A SET_FIELD packaged procedure tried to turn on the Input Allowed characteristic of a non-displayed field.

Action: To turn on the Input Allowed characteristic of a field you must also turn on the Displayed characteristic of the field.

Level: >25

Type: Error

FRM-41021: Cannot set QUERYABLE attribute of non-displayed field <block name>.<field name>.

Cause: Designer error. A SET_FIELD packaged procedure tried to turn on the Query Allowed characteristic of a non-displayed field.

Action: To turn on the Query Allowed characteristic of a field you must also turn on the Displayed characteristic of the field.

Level: >25

Type: Error

FRM-41022: Cannot set REQUIRED attribute of non-updateable field <block name>.<field name>.

Cause: Designer error. A SET_FIELD packaged procedure tried to turn on the Required characteristic of a non-updateable field.

Action: To turn on the Requried characteristic of a field you must also turn on either the Update Allowed characteristic or the Update if Null characteristic of the field.

Level: >25

Type: Error

FRM-41023: Cannot set UPDATEABLE attribute of secure field <block name>. <field name>.

Cause: Designer error. A SET_FIELD packaged procedure tried to change the Update Allowed characteristic of a database field which the user does not have permission to update.

Action: Either correct the SET_FIELD statement or grant update permission on the column to the user.

Level: >25

Type: Error

FRM-41024: Cannot set UPDATE_NULL attribute of secure field <block name>.<field name>.

Cause: Designer error. A SET_FIELD packaged procedure tried to change the Update if Null characteristic of a database field which the user does not have permission to update.

Action: Either correct the SET_FIELD statement or grant update permission on the column to the user.

Level: >25

Type: Error

FRM-41800: List of values not available for this field.

Cause: You pressed [List], but the form does not provide a list of values for this field.

Action: No action is necessary.

Level: 10

Type: Informative

FRM-41801: Last value retrieved.

Cause: You pressed [List] and then pressed [Next Field] after the last value in the list was displayed.

Action: Enter a field value or press [List] again to redisplay the list of possible values.

Level: 10

Type: Informative

FRM-41802: Duplicate Record function allowed on new records only.

Cause: You pressed [Duplicate Record], but the current record is the one that has been fetched from the database.

Action: No action is necessary. You can use [Duplicate Record] only when creating a new record.

Level: 10

Type: Error

FRM-41803: No previous record to copy value from.

Cause: You pressed [Duplicate Field] or [Duplicate Record], but the current record is the first record in the block.

Action: No action is necessary. [Duplicate Field] and [Duplicate Record] are meaningless in this context.

Level: 10

Type: Error

FRM-41804: Variable wasn't entered: <variable name>.

Cause: You entered a response to the Query Where alert that contained an invalid placeholder (one not used in any of the query fields).

Action: Correct the placeholder in your response, or define it in one of the query fields. Then re-execute the query.

Level: >25

Type: Error

FRM-41805: Ambiguous field name: <field name>.

Cause: Designer error. The current trigger references a field in two places.

Action: Correct the statement by supplying a full field name (i.e., *block.field).*

Level: >25

Type: Error

FRM-41806: Too many variables used.

Cause: You used more than 25 substitution variables in your query.

Action: Reduce the number of substitution variables and re-query.

Level: >25

Type: Error

FRM-41807: This key hasn't yet been entered. Try again.

Cause: You didn't enter data in a primary key field.

Action: Enter data in all primary key fields and re-execute.

Level: 10

Type: Error

FRM-41809: Error initializing SQL*Menu.

Cause: You tried to use SQL*Menu from within SQL*Forms, and an internal SQL*Menu error occurred.

Action: Contact your DBA.

Level: >25

Type: Error

FRM-41810: Error creating menu.

Cause: You tried to use SQL*Menu from within SQL*Forms, and an internal SQL*Menu error occurred.

Action: Contact your DBA.

Level: >25

Type: Error

FRM-41811: Error removing menu.

Cause: You tried to use SQL*Menu from within SQL*Forms, and an internal SQL*Menu error occurred.

Action: Contact you DBA.

Level: >25

Type: Error

FRM-41812: Error resetting menu.

Cause: You tried to use SQL*Menu from within SQL*Forms, and an internal SQL*Menu error occurred.

Action: Contact you DBA.

Level: >25

Type: Error

FRM-41813: Form exited by debug mode.

Cause: You selected the Exit SQL*Forms (Run Form) option on the Break Processing menu.

Action: No action is necessary.

Level: >25

Type: Working

FRM-41814: Invalid page position.

Cause: Designer error. A trigger tried to move or resize a view to a pop-up page that would cause all or part of the view to display off of the screen.

Action: Correct the statement.

Level: >25

Type: Error

FRM-41900: Run aborted by fatal error.

Cause: Fatal error caused form to abort

Action: Contact your DBA.

Level: 20

Type: Error

FRM-41901: Error <number> cursors were not closed.

Cause: Internal error.

Action: Contact your DBA.

Level: >25

Type: Error

FRM-41902: Total cursors used <number>.

Cause: This message appears when you specify the -s switch on the command line.

Action: No action is necessary.

Level: >25

Type: Informative

FRM-42100: No errors encountered recently.

Cause: You pressed [Display Error], but no error has occurred recently.

Action: No action is necessary.

Level: 5

Type: Informative

FRM-42400: Performing event trigger <trigger name>.

Cause: This informative message occurs during a trigger when debug mode is specified.

Action: No action is necessary.

Level: >25

Type: Informative

FRM-42401: Performing program trigger <trigger name>.

Cause: This informative message occurs during a trigger when debug mode is specified.

Action: No action is necessary.

Level: >25

Type: Informative

FRM-42536: Use [Next Field] key to enter answer.

Cause: SQL*Forms is waiting for your response to a prompt.

Action: Enter an appropriate response and press [Next Field].

Level: 10

Type: Informative

FRM-50001: Acceptable characters are a-z, A-Z, and space.

Cause: You entered an unacceptable character into the field.

Action: Enter a character from a-z, A-Z, or a space.

Level: 15

Type: Error

FRM-50002: Month must be between 1 and 12.

Cause: You entered an invalid month value in a date.

Action: Enter a month value from 1 (for January) to 12 (for December).

Level: 15

Type: Error

FRM-50003: Year must be 00-99 or 1000-4712.

Cause: You entered an invalid year.

Action: Enter a valid year. The year maybe a number between 0 and 99 (representing the years 1900 to 1999) or between 1000 and 2100.

Level: 15

Type: Error

FRM-50004: Day must be between 1 and last of month.

Cause: You entered an invalid day.

Action: Enter a valid day. For April, for example, enter a number between 1 and 30.

Level: 15

Type: Error

FRM-50006: Legal characters are 0-9 + and -.

Cause: You entered an unacceptable character in a number field.

Action: Enter a valid number. A valid number has digits 0 through 9. A number maybe preceded by a plus (+) or minus (-) sign. If the message allows it, a number may contain one decimal point at any location, except before the sign.

Level: 15

Type: Error

FRM-50007: Too many decimal digits.

Cause: You entered a number with 3 or more decimal digits after the decimal point in a field with the MONEY or RMONEY data type.

Action: Re-enter a valid number.

Level: 15

Type: Error

FRM-50009: Too many decimal points.

Cause: You entered a number that contains two or more decimal points, or you have entered a number that contains a decimal point in a field that requires a whole (non-decimal) number.

Action: Enter a number with no more than one decimal point. If you have used only one decimal, remove the decimal and the decimal part of the number.

Level: 15

Type: Error

FRM-50010: Format is [+-]9999999.99

Cause: You entered an invalid value in a MONEY or RMONEY field.

Action: Enter a valid value. This value should have zero or dollar digits, followed by a decimal and two cents digits. The entire number can be preceded by a plus (+) or a minus (-) sign.

Level: 15

Type: Error

FRM-50011: Not a valid month name.

Cause: You entered an invalid month name in a date.

Action: Enter a valid month name. SQL*Forms recognizes the first three characters of a month's actual name. For example, JAN stands for January, JUN for June.

Level: 15

Type: Error

FRM-50012: Date format is DD-MON-YY.

Cause: You entered an invalid date.

Action: Re-enter the date in the requested format. In the format shown above, July 7,1989 would be 07-JUL-89.

Level: 15

Type: Error

FRM-50013: Plus or minus must be in first position.

Cause: You entered the plus or minus sign in the wrong position.

Action: Retype with the plus or minus sign in the first position.

Level: 15

Type: Error

FRM-50014: Bad exponent.

Cause: You entered an exponent in a field that does not accept exponents.

Action: Enter a value without an exponent.

Level: 15

Type: Error

FRM-50015: Too many decimal points.

Cause: You entered too many decimal points.

Action: Retype with only one decimal point.

Level: 15

Type: Error

FRM-50016: Legal characters are 0-9 - + E.

Cause: You entered an unacceptable character in a number field.

Action: Enter a valid number. A valid number has digits 0 through 9. A number maybe preceded by a plus (+) or minus (-) sign. If the message allows it, a number may contain one decimal point at any location, except before the sign. You can use an E to specify scientific notation.

Level: 15

Type: Error

FRM-50017: Hour must be between 0 and 23.

Cause: You entered an invalid hour.

Action: Enter a valid hour. SQL*Forms records time on a 24-hour basis.

Level: 15

Type: Error

FRM-50018: Minutes must be between 00 and 59.

Cause: You entered an invalid minute value.

Action: Enter a valid minute value.

Level: 15

Type: Error

FRM-50019: Seconds must be between 00 and 59.

Cause: You entered an invalid second value.

Action: Enter a value between 00 and 59.

Level: 15

Type: Error

FRM-50020: Missing exponent.

Cause: You failed to enter an exponent.

Action: Enter a proper exponent.

Level: 15

Type: Error

FRM-50021: Date format is MM/DD/YY.

Cause: You entered an invalid date or a date in improper format.

Action: Re-enter the date in the requested format. In the format shown above, July 20,1986 would be 07/20/86.

Level: 15

Type: Error

FRM-50022: Time format is HH:MM[:SS].

Cause: You entered an invalid time format.

Action: Re-enter the time in the requested format. For example, 3:00 p.m. is 15:00:00 (or 15:00).

Level: 15

Type: Error

FRM-50023: Date format is DD/MM/YY.

Cause: You entered an invalid date.

Action: Re-enter the date in the requested format. In the format shown above, July 20,1986 would be 20/07/86.

Level: 15

Type: Error

FRM-50024: Spaces allowed in leading positions only.

Cause: You entered spaces intermixed with data.

Action: Re-enter data with no spaces intermixed.

Level: 15

Type: Error

FRM-50025: Datetime format is DD-MON-YYYY HH:MM[:SS].

Cause: You entered an invalid date.

Action: Re-enter the date and time in the requested format. In the format shown above, April 13,1989 at 3:00 pm would be 13-APR-1989 15:00:00.

Level: 15

Type: Error

FRM-50026: Date format is DD-MON-YYYY.

Cause: You entered an invalid date.

Action: Re-enter the date in the requested format. In the format shown above, April 13,1989 would be 13-APR-1989.

Level: 15

Type: Error

APPENDIX

F V2 TRIGGERS

V2 triggers are SQL*Forms triggers that are defined in the V2 trigger style. A V2 trigger consists of one or more trigger steps. Each trigger step consists of a single command plus characteristics that further define what the step does when it executes.

This appendix primarily discusses trigger steps. Specifically, this appendix covers the following topics:

- defining trigger steps
- trigger step characteristics
- commands in trigger steps
- V2 function codes
- the CASE statement
- variables in trigger steps
- V2 trigger processing

Do not use this appendix if your applications use only V3 triggers.

Note: Oracle Corporation strongly recommends that you design your applications with V3 triggers. V2 triggers are available only for compatibility with previous versions of SQL*Forms.

Related Topics: V3 triggers versus V2 triggers 4-2

Defining V2 Triggers

You create and modify V2 triggers the same way you create and modify V3 triggers, using the Trigger Definition form and the Trigger Definition spread table in the SQL*Forms (Design) interface. However, to define a V2 trigger, you must also define trigger steps through the Trigger Step Definition form or Trigger Step Definition spread table.

Related Topics: Defining triggers 14-2

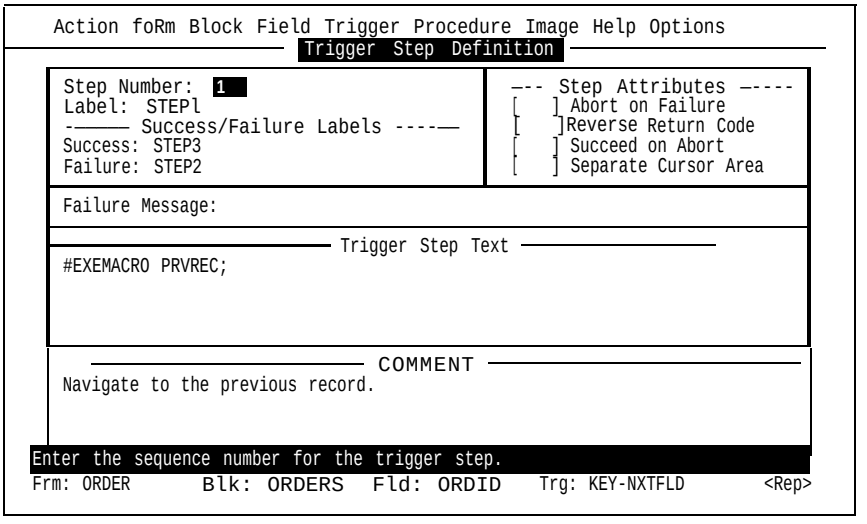
Trigger Step Definition Form

The Trigger Step Definition form is an element in the SQL*Forms (Design) interface where you create, modify, or delete trigger steps. This form allows you to see all information for a single trigger step at one time.

Note: Use the Trigger Step Definition spread table if you want to look at information on more than one trigger step at a time.

Figure F-1 shows the Trigger Step Definition form with sample data.

FIGURE F-1
Trigger Step Definition Form



Location: You can only reach the Trigger Step Definition form from the Trigger Definition form or the Trigger Definition spread table. Also, the current record must correspond to a trigger that has the Trigger Style trigger characteristic set as V2.

To reach the Trigger Step Definition form from the Trigger Definition form or spread table, press [Zoom In]. If the Trigger Step Definition spread table displays, press [Change Display Type].

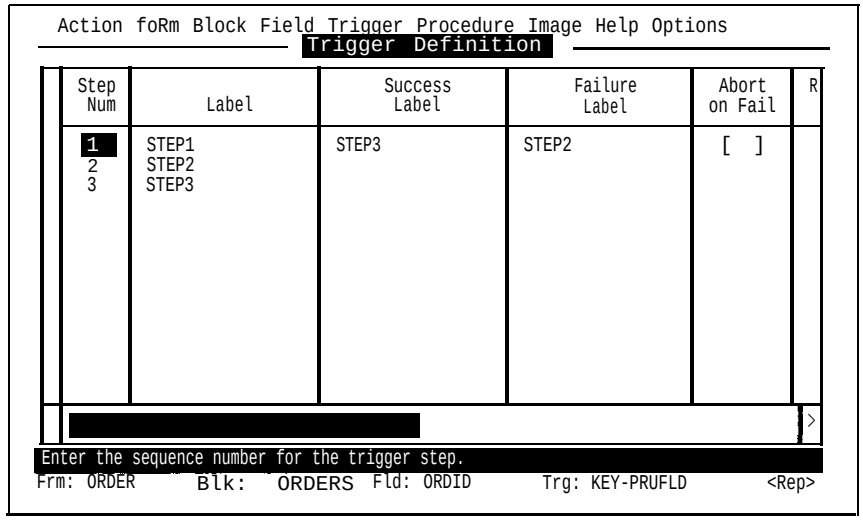
Trigger Step Definition Spread Table

The Trigger Step Definiftion spread table is an element in the SQL*Forms (Design) interface where you create, modify, or delete trigger steps. This spread table allows you to see information on more than one trigger step at a time. Note that all of the elements in this spread table do not appear at the same time.

Note: Use the Trigger Step Definition form if you want to look at all of the information for one field at the same time.

Figure F-2 shows a portion of the Trigger Step Definition spread table with sample data.

FIGURE F-2
Trigger Step Definition Spread Table



Location: You can only reach the Trigger Step Definition spread table from the Trigger Definition form or the Trigger Definition spread table. Also, the current record must correspond to a trigger that has the Trigger Style trigger characteristic set as V2.

To reach the Trigger Step Definition spread table from the Trigger Definition form or spread table, press [Zoom In]. If the Trigger Step Definition form displays, press [Change Display Type].

Trigger Step Characteristics

The process of defining a trigger step consists of setting each of the trigger step's characteristics. These characteristics indicate the following things about a trigger step:

- basic information, including the step's sequence number within the trigger that owns the step
- how the step behaves

This section presents trigger step characteristics in alphabetical order. Note that the names of the trigger step characteristics correspond exactly with the names of the elements in the Trigger Step Definition form and Trigger Step Definition spread table. For example, the Step Number field in the Trigger Step Definition form is the element that defines the Step Number trigger step characteristic.

Note: The full names of the trigger step characteristics are sometimes abbreviated in the SQL*Forms (Design) interface. In this section, these abbreviations follow the full names in parentheses (e.g., Step Number (Step Num)).

You assign trigger step characteristics through a number of different interface elements in a number of different ways:

Interface Element	*To Assign a Field Characteristic*
check box	turn the characteristic on or off
field	type in a value
scroll region	type in a value

You should be aware of these assignment differences because they affect how this section describes the characteristics. For example, a characteristic assigned from a check box has a description for Default Setting (which will be either "on" or "off"). Alternatively, a characteristic assigned from a field might have a Required/Optional description, indicating whether you have to fill out this field element when defining a trigger step.

Related Topics: Interface elements 5-1

Abort On Failure (Abort On Fail)

The Abort On Failure trigger step characteristic indicates that if the trigger step fails, SQL*Forms should abort the trigger that owns the step.

However, if you have specified a failure label for the step (via the Failure trigger step characteristic), the trigger branches to the indicated step and does not abort the trigger. Ultimately, the Reverse Return Code trigger step characteristic determines if the trigger step (and possibly the entire trigger) fails or succeeds.

Note: If you want to use the Abort On Failure characteristic to abort a user-named trigger *and* the trigger that called it, you have to specify the characteristic on both triggers. If you specify this characteristic only on the user-named trigger, the calling trigger will not abort on failure.

Default Setting: on

Related Topics: Failure Label trigger step characteristic F-5
Reverse Return Code trigger step characteristic F-7

Comment

The Comment trigger step characteristic specifies general information about the trigger step. Comment is your tool to comment on the purpose or characteristics of the trigger step. This feature is especially useful to other designers who use your objects or applications and to you when you debug your applications.

Required/Optional: optional

Failure Label (Failure)

The Failure Label trigger step characteristic specifies another trigger step within the trigger to which SQL*Forms branches if the current step fails. The value of the Failure Label characteristic must match the value of the Label characteristic of the target trigger step.

Note: The Reverse Return Code characteristic ultimately determines if the trigger step (and possibly the entire trigger) fails or succeeds.

Required/Optional: optional

Related Topics: Label trigger step characteristic F-6
Reverse Return Code trigger step characteristic F-7
Success/failure logic in V2 triggers F-32

Failure Message

The Failure Message trigger step characteristic specifies the message to display on the message line if this trigger step fails.

Required/Optional: optional

Related Topics: V2 trigger processing F-29

Label

The Label trigger step characteristic indicates the name that SQL*Forms uses to identify the trigger step. This name must contain no blank spaces and must follow object naming conventions.

A label is useful only if the trigger that owns this trigger step uses a success or failure label to branch to the step.

Restrictions: No two trigger steps in the same trigger can have the same Label characteristic.

Required/Optional: optional

Related Topics: Failure Label trigger step characteristic F-5
Success Label trigger step characteristic F-8

New Cursor Area (New Cursor)

The New Cursor Area trigger step characteristic ensures that a trigger step that contains a SQL statement is assigned a separate cursor.

By default, SQL*Forms assigns each trigger step that contains a SQL statement a separate cursor. However, this characteristic ensures a trigger step is assigned a separate cursor even if you run the form with the -o command line switch or the Share Cursors for Trigger SQL run option, which cause SQL*Forms to reuse or share cursors for all SQL statements in V2 triggers.

Default Setting: off

Related Topics: Cursor, database 22-3
-o switch, SQL*Forms (Run Form) 23-18
Share Cursors for Trigger SQL run option 26-4

Reverse Return Code (Reverse Return)

The Reverse Return Code trigger step characteristic indicates that SQL*Forms should reverse the success/failure status of the trigger step. For example, if the trigger step fails and the Reverse Return Code characteristic is on, SQL*Forms treats the step as if it had succeeded.

The Reverse Return Code characteristic has no effect if a trigger step causes a fatal error.

Default Setting: off

Related Topics: Trigger steps, outcomes F-30

Step Number (Step Num)

The Step Number trigger step characteristic indicates where the trigger step is in the sequence of trigger steps within the parent trigger. Provided there is no branching within a trigger step, the Step Number characteristic indicates the order that SQL*Forms processes the trigger steps when executing the parent trigger.

SQL*Forms re-sequences all other trigger steps in a parent trigger if you perform any of the following tasks:

- assign a sequence number to a trigger step that another trigger step already has
- delete a trigger step that does not have the last sequence number in the trigger
- insert a new trigger step in the Trigger Step Definition form or spread table

Restrictions: You cannot have a trigger that has non-contiguous sequence numbering. That is, you cannot have a trigger with four trigger steps that have 1,2,3, and 5 as the step numbers.

Required/Optional: required

Default Value: The default value of the Step Number characteristic is the next highest sequence number that is available in the trigger.

Succeed On Abort

The Succeed On Abort trigger step characteristic indicates that if the execution of the trigger step would cause the trigger that owns it to abort, SQL*Forms should continue processing the trigger.

This characteristic is meaningful only if you also turn on the Abort On Failure trigger step characteristic for the step. When both characteristics are on and the trigger step fails, SQL*Forms halts the execution of the parent trigger and marks the trigger as successful and continues processing.

Default Setting: off

Related Topics: Abort On Failure trigger step characteristic F-5

Success Label (Success)

The Success Label trigger step characteristic specifies another trigger step within the trigger to which SQL*Forms branches if the current step succeeds. The value of the Success Label characteristic must match the value of the Label characteristic of the target trigger step.

Note: The Reverse Return Code characteristic ultimately determines if the trigger step (and possibly the entire trigger) fails or succeeds.

Required/Optional: optional

Related Topics: Label trigger step characteristic F-6
Reverse Return Code trigger step characteristic F-7
Success/failure logic in V2 triggers F-32

Trigger Step Text

The Trigger Step Text trigger step characteristic specifies the SQL or SQL*Forms command that executes when the trigger step fires.

Required/Optional: The Trigger Step Text characteristic is required for V2 triggers.

Commands in Trigger Steps

There are three types of commands that you can enter in a trigger step:

- SQL commands
- SQL*Forms commands
- user exit calls

When you are defining a trigger step, enter these commands directly. into the Trigger Step Text scroll region of the Trigger Step Definition form.

SQL Commands in Triggers Steps

The SQL commands that are legal in trigger steps are identical to the SQL Commands that are legal in V3 triggers. However, some rules are unique to using a SQL statement in trigger steps. These rules allow a SQL statement to act on SQL*Forms objects.

No Semi-colon for SQL

In a trigger step, do not end a SQL statement with a semi-colon.

SQL*Forms Fields in SQL

In a SQL statement, refer to SQL*Forms fields by the names they are defined with in the design interface or by legal variable names. Prefix the field name with a colon (:).

```
:[block.]field
: GLOBAL.variablename
```

Use a fully qualified field name (i.e., block.field) when you refer to a field whose name is not unique in the current form. If you fail to do this, an error occurs when the trigger fires.

Note: For compatibility with forms created with earlier versions of SQL*Forms, you can use an ampersand (&) instead of a colon. However, Oracle Corporation recommends that you use a colon.

INTO Clause in SELECT

Use a SQL SELECT statement with an INTO clause to read a value from a database column into a SQL*Forms field. Such a statement selects one row (at most) from the database, no matter how many rows satisfy the SELECT statement conditions.

The syntax for the INTO clause is:

```
INTO [:] [block.]field[,  [:] [block .] field . ..]
```

This clause always appears between the SELECT keyword and the FROM clause. For example:

```
SELECT NAME
    INTO : ORDERS. NAME
    FROM CLIENT
    WHERE CLIENT. CLIENTID = : ORDERS.ORDERID
```

If a SELECT statement selects values into more than one field, keep the corresponding column names and field names in the same order. For example:

```
SELECT NAME ,  CREDIT_  LIMIT
    INTO  : ORDERS. NAME,  : ORDERS. CREDIT_ LIMIT
    FROM  CLIENT
    WHERE  CLIENT.CLIENTID  =  :ORDERS.ORDERID
```

Note: For compatibility with forms created with earlier versions of SQL*Forms, the colon is optional when you reference SQL*Forms fields in an INTO clause. However, Oracle Corporation recommends that you use a colon.

SQL*Forms commands in Trigger Steps

SQL*Forms commands are commands that you can only use in SQL*Forms V2 triggers. These commands provide trigger steps the means to perform important functionality.

There are four SQL*Forms commands:

- #COPY
- #ERASE
- #EXEMACRO
- #HOST

#COPY Command

Syntax: `#COPY` *`source`* `[INTO]` *`destination`*

where:

`source`	Specifies a *`value`*.
`destination`	Specifies a *`variable`*.

Refer to "V2 Function Code Arguments" for detailed descriptions of *`value`* and *`variable`*.

Description: The #COPY command copies a source value to a destination. The source can be a constant, a field value, a global variable, or a system variable; the destination can be a field, a global variable, or a system variable (if it can be modified).

The #COPY command and the V2 function code, COPY, perform the same functionality. This gives you flexibility in how you use the function.

#ERASE Command

Syntax: `#ERASE GLOBAL .variablename`

Description: The #ERASE command removes a named global variable so that it no longer exists. If you attempt to delete an unassigned global variable, SQL*Forms ignores the #ERASE command.

The #ERASE command and the V2 function code, ERASE, perform the same functionality. This gives you flexibility in how you use the function.

#EXEMACRO Command

Syntax: `#EXEMACRO statement [statement...]`

where *statement* is a SQL*Forms V2 function code or a CASE statement.

Description: The #EXEMACRO statement executes the named function codes, CASE statements, or combination of both.

Related Topics: CASE statement F-27
V2 function codes F-12

#HOST Command

Syntax: `#HOST system.command`

Description: The #HOST command executes an operating system command *(system_ command).*

The #HOST command and the V2 function code, HOST, perform the same functionality. This gives you flexibility in how you use the function.

Refer to "V2 Function Code Arguments" for a detailed description of *system_command.*

User Exit Calls in Trigger Steps

User exits execute in trigger steps identically to the way they execute in V3 triggers. However, the syntax for calling a user exit from a trigger step is different.

Syntax: *# [#] exitname parameters*

where:

exitname	Specifies the name of the user exit you are calling.
parameters	Specifies the parameters you want to pass to the user exit.

Related Topics: User exits 19-1

V2 Function Codes

This section presents the SQL*Forms V2 function codes in alphabetical order. The syntax of the each function code and a description are provided.

Note: If you have used previous versions of SQL*Forms, you might have referred to V2 function codes as "macros" (e.g., the EXEQRY macro).

Related Topics: #EXEMACRO command F-11
Matching V2 function codes to V3 equivalents F-24

V2 Function Code Arguments

For conciseness, some of the arguments in the V2 function codes appear in a shorter form. Refer to the following table for the explicit definitions of these arguments.

Argument	*Definition*
field	*{fieldname | blockname.fieldname}*
message	*value*
system_command	*{'command_ string' | field | variable_reference}*
'text'	Any string up to 78 characters long. If you want to display a single quote character in a text message, use two single quotes in the string (e.g., 'Record isn' 't available').
value	*{'text' | variable}*
variable	*{* [:] *field* | [:] GLOBAL. *variablename* | [:] SYSTEM. *variablename* | *variable_ reference}*
variable_ reference	*&variable*

Concatenating V2 Function Code Arguments

Any V2 function code that takes a value as an argument can express the concatenation of several values. To concatenate values, use the following syntax:

expression = *value[* || value] . . .

For example, the following statement writes a message to the screen that dynamically displays a field value:

```
#EXEMACRO MESSAGE 'The field value ' ||:ordid || ' is too
small.';
```

ABTQRY

Syntax: `ABTQRY;`

Description: ABTQRY aborts a query if one is open in the current block.

BELL

Syntax: `BELL;`

Description: BELL sets the terminal bell to ring the next time the screen synchronizes.

CALL

Syntax: `CALL formname;`

Description: CALL runs an indicated form while keeping the parent form active. SQL*Forms runs the called form with the same options as the parent form. When the called form is exited through the EXIT function or as a result of navigational failure, processing resumes in the parent form at the point the CALL occurred.

CALL causes SQL*Forms to clear the calling form from the screen before drawing the called form CALL also causes SQL*Forms to keep the default menu application of the calling form.

When you call a form, SQL*Forms issues a savepoint for the called form.

Related Topics: Default Menu Application form characteristic 9-3
Savepoints, issued when calling a form 3-18

CALLINPUT

Syntax: `CALLINPUT;`

Description: CALLINPUT accepts and processes function key input from the operator. When CALLINPUT is terminated through the EXIT function or as a result of a navigational failure, processing resumes in the form at the point CALLINPUT occurred.

Note: CALLINPUT is included in Version 3.0 of SQL*Forms for compatibility with previous versions. Oracle Corporation recommends that you do not use this V2 function code in new applications.

CALLQRY

Syntax: `CALLQRY formname;`

Description: CALLQRY runs an indicated form while keeping the parent form active. SQL*Forms runs the called form with the same options as the parent form and as a query-only form. When the called form is exited through the EXIT function or as a result of navigational failure, processing resumes in the parent format the point the CALLQRY occurred.

CALLQRY causes SQL*Forms to clear the calling form from the screen before drawing the called form. CALLQRY also causes SQL*Forms to keep the default menu application of the calling form.

When you call a form, SQL*Forms issues a savepoint for the called form.

Related Topics: Default Menu Application form characteristic 9-3
Savepoints, issued when calling a form 3-18

CHRMODE

Syntax: `CHRMODE;`

Description: CHRMODE switches the typing mode. If the form is currently running in insert mode, CHRMODE puts the form in replace mode. If the form is currently running in replace mode, CHRMODE puts the form in insert mode.

CLRBLK

Syntax: `CLRBLK;`

Description: CLRBLK flushes the current block. If there are changes in the current block that have not been posted or committed, SQL*Forms prompts the operator to commit the changes during CLRBLK processing.

CLRFLD

Syntax: `CLRFLD;`

Description: CLRFLD clears the entire value of the current field-regardless of the current cursor position-changing the field's value to NULL.

CLRFRM

Syntax: `CLRFRM;`

Description: CLRFRM flushes the current form. If there are changes in the current form that have not been posted or committed, SQL*Forms prompts the operator to commit the changes during CLRFRM processing.

CLRFRM rolls back any changes that have not been committed since the current form was first entered during the SQL*Forms (Run Form) session. If the current form has called other forms that contain posted changes, CLRFRM rolls back these changes.

Related Topics: Savepoints, issued when posting 3-17

CLRREC

Syntax: `CLRREC;`

Description: CLRREC flushes the current record from the block. If a query is open in the block, SQL*Forms fetches records to refill the block.

COMMIT

Syntax: `COMMIT;`

Description: COMMIT updates data in the database to match data in the form. SQL*Forms first validates the form. If them are changes to commit to the database, for each block in the form SQL*Forms posts deletes, inserts, and updates to the database and performs a database commit. When the commit completes, SQL*Forms releases all table and row locks held by the operator.

If you have posted data to the database during the current SQL*Forms (Run Form) session, COMMIT commits this data to the database.

Related Topics: V2 function codes, POST F-22

COPY

Syntax: `COPY` *`source`* `INTO` *`destination;`*

Description COPY copies a source value to a destination. The source can be a constant, a field value, a global variable, or a system variable; the destination can be a field or a global variable.

The COPY function code and the #COPY command perform the same functionality. This gives you flexibility in how you use the function.

Related Topics: #COPY command F-10

CQUERY

Syntax: `CQUERY;`

Description: CQUERY identifies the number of rows that a query will retrieve. If there are changes to commit, SQL*Forms prompts the operator to commit them during CQUERY processing.

CREREC

Syntax: `CREREC;`

Description: CREREC creates a new record in the current block after the current record. SQL*Forms then navigates the cursor to the new record.

DEFAULT

Syntax: `DEFAULT value INTO` *`variable;`*

Description DEFAULT copies a value to a variable if the variable's current value is NULL. If the variable's current value is not NULL, DEFAULT does nothing. If the variable is an undefined global variable, SQL*Forms creates the variable.

DELBACK

Syntax: `DELBACK;`

Description: DELBACK deletes the field character to the left of the cursor.

DELCHR

Syntax: `DELCHR;`

Description: DELCHR deletes the field character that the cursor is on.

DELREC

Syntax: `DELREC;`

Description: DELREC removes the current record from the block. If the record corresponds to a row in the database, SQL*Forms locks the record before removing it and marking it as a delete. If a query is open in the block, SQL*Forms fetches records to refill the block.

DERROR

Syntax: `DERROR;`

Description: DERROR displays the Display Error screen if there is a logged error. When the operator presses a function key, SQL*Forms redisplays the form.

DKEYS

Syntax: `DKEYS;`

Description: DKEYS displays the Show Key screen. When the operator presses a function key, SQL*Forms redisplays the form as it was before DKEYS was invoked.

DOWN

Syntax: `DOWN;`

Description: DOWN navigates the cursor to the instance of the current field in the record with the next higher sequence number. If necessary, SQL*Forms creates a new record or fetches a record for the cursor to navigate to.

DUPFLD

Syntax: `DUPFLD;`

Description: DUPFLD assigns the current field the same value as the instance of this field in the previous record.

DUPREC

Syntax: `DUPREC;`

Description: DUPREC copies the value of each field in the previous record to the corresponding fields in the current record. The current record must not correspond already to a row in the database.

ENDSTEP

Syntax: `ENDSTEP [FAIL] [MESSAGE` *`message]`* `;`

Description: ENDSTEP halts the processing of the current trigger step. The trigger step succeeds unless you specify the FAIL keyword. You can use the MESSAGE clause to send a message `(message)` to the message line. Note that trigger step characteristics do affect the behavior of ENDSTEP.

ENDTRIG

Syntax: `ENDTRIG [FAIL] [MESSAGE` *`message]`* `;`

Description: ENDTRIG halts the processing of the current trigger. The trigger succeeds unless you specify the FAIL keyword. You can use the MESSAGE clause to send a message to the message line. Note that ENDTRIG bypasses the success/failure processing of the current trigger step.

ENTER

Syntax: `ENTER;`

Description: ENTER validates data in the current validation unit

ENTQRY

Syntax: `ENTQRY [ALL] [FOR UPDATE];`

Description: ENTQRY flushes the current block and puts the form in enter query mode. If there are changes to commit, SQL*Forms prompts the operator to commit them during ENTQRY processing.

If you use the ALL keyword, SQL*Forms fetches all of the selected records in a single fetch when EXEQRY is invoked. If you use the FOR UPDATE clause SQL*Forms attempts to lock all of the selected records immediately. If SQL*Forms cannot immediately obtain a lock on a record, it keeps trying to obtain the locks without letting the operator cancel the process.

ERASE

Syntax: `ERASE GLOBAL.` *`variablename;`*

Description: ERASE removes an indicated global variable, so that it no longer exists.

The ERASE function code and the #ERASE command perform the same functionality. This gives you flexibility in how you use the function.

Related Topics: #ERASE command F-11

EXEQRY

Syntax: `EXEQRY [ALL] [FOR UPDATE];`

Description: EXEQRY flushes the current block, opens a query, and fetches a number of selected records. If there are changes to commit, SQL*Forms prompts the operator to commit them during EXEQRY processing.

If you use the ALL keyword, SQL*Forms fetches all of the selected records in a single fetch. If you use the FOR UPDATE clause, SQL*Forms attempts to lock all of the selected records immediately. If SQL*Forms cannot immediately obtain a lock on a record, it keeps trying to obtain the locks without letting the operator cancel the process.

EXETRG

Syntax: `EXETRG triggername;`

Description: EXETRG executes an indicated user-named trigger.

EXIT

Syntax: `EXIT;`

Description: During a CALLINPUT, EXIT terminates the CALLINPUT function. In all other contexts, EXIT navigates to "outside" of the form. If there are changes in the current form that have not been posted or committed, SQL*Forms prompts the operator to commit the changes during EXIT processing.

GOBLK

Syntax: `GOBLK blockname;`

Description: GOBLK navigates the cursor to an indicated block.

GOFLD

Syntax: `GOFLD field;`

Description: GOFLD navigates the cursor to an indicated field.

GOSTEP

Syntax: `GOSTEP label;`

Description: GOSTEP branches trigger processing to a named trigger step *(label)* within the current trigger.

HELP

Syntax: `HELP;`

Description HELP displays the field's hint message on the message line. If the hint message is displayed already, HELP displays the detailed help screen for the field.

HOST

Syntax: `HOST system_ command [NOSCREEN];`

Description: HOST executes an indicated operating system command. If you use the NOSCREEN keyword, SQL*Forms does not allow operator input and does not redraw the screen.

The HOST function code and the #HOST command perform the same functionality. This gives you flexibility in how you use the function.

Related Topics: #HOST command F-11

LISTVAL

Syntax: `LISTVAL;`

Description: LISTVAL displays the list of values for the current field. The list of values remains displayed until the operator presses [Exit/Cancel] or selects a value into a field.

MENU

Syntax: `MENU;`

Description: MENU displays the block menu and accepts operator input. If the operator enters a valid selection, MENU navigates the cursor to the appropriate block

MESSAGE

Syntax: `MESSAGE` *`message;`*

Description: MESSAGE displays specified message text on the message line.

MOVLEFT

Syntax: `MOVLEFT;`

Description: MOVLEFT moves the cursor one character space to the left within a field. If necessary, MOVLEFT scrolls the field value in order to place the cursor on the appropriate character space.

MOVRIGHT

Syntax: `MOVRIGHT;`

Description: MOVRIGHT moves the cursor one character space to the right within a field. If necessary, MOVLEFT scrolls the field value in order to place the cursor on the appropriate character space.

NEWFRM

Syntax: `NEWFRM` *`formname;`*

Description: NEWFRM exits the current form and enters an indicated form. The calling form is terminated as the parent form. If the calling form had been called by a higher level form, SQL*Forms keeps the higher level call active and treats it as a call to the new form SQL*Forms releases memory (such as database cursors) that the terminated form was using.

SQL*Forms runs the new form with the same SQL*Forms (Run Form) options as the parent form. If the parent form was a called form, SQL*Forms runs the new form with the same parameters (e.g., HIDE and NO_REPLACE) as the parent form.

Related Topics: V2 function codes, CALL F-13

NOOP

Syntax: `NOOP;`

Description: NOOP displays the message, "Unrecognized command." NOOP performs no function.

NULL

Syntax: `NULL;`

Description: NULL performs no function.

NXTBLK

Syntax: `NXTBLK;`Text

Description: NXTBLK navigates the cursor to the enterable block with the next higher sequence number than the current block. If there is no such block, NXTBLK navigates the cursor to the enterable block with the lowest sequence number. If there is no such block, NXTBLK navigates the cursor to the current block.

NXTFLD

Syntax: `NXTFLD;`

Description: NXTFLD navigates the cursor to the enterable field with the next higher sequence number than the current field. If there is no such field, NXTFLD navigates the cursor to the enterable field with the lowest sequence number. If there is no such field, NXTFLD navigates the cursor to the current field.

If the validation unit is the field, NXTFLD validates any fields with sequence numbers greater than the current field or less than the target field.

NXTKEY

Syntax: `NXTKEY;`

Description: NXTKEY navigates the cursor to the enterable, primary key field with the next higher sequence number than the current field. If there is no such field, NXTKEY navigates the cursor to the enterable, primary key field with the lowest sequence number.

If the validation unit is the field, NXTKEY validates any fields with sequence numbers greater than the current field or less than the target field.

NXTREC

Syntax: `NXTREC;`

Description: NXTREC navigates the cursor to the record with the next higher sequence number than the current record. If there is no such record and a query is open in the block, SQL*Forms fetches records during NXTREC processing.

NXTSET

Syntax: `NXTSET;`

Description: NXTSET fetches another set of records from the database and navigates to the first record that the fetch retrieves. NXTSET succeeds only if a query is open in the current block.

PAUSE

Syntax: `PAUSE;`

Description: PAUSE suspends processing until the operator presses a function key.

PRINT

Syntax: `PRINT;`

Description: PRINT writes one or all pages of a form to a file. SQL*Forms prompts the operator to indicate the pages to write, the filename to which to write the pages, and whether to send the file to the system printer.

PRVBLK

Syntax: `PRVBLK;`

Description: PRVBLK navigates the cursor to the enterable block with the next lower sequence number than the current block. If there is no such block, PRVBLK navigates the cursor to the enterable block with the highest sequence number. If there is no such block, PRVBLK navigates the cursor to the current block.

PRVFLD

Syntax: `PRVFLD;`

Description: PRVFLD navigates the cursor to the enterable field with the next lower sequence number than the current field. If there is no such field, PRVFLD navigates the cursor to the enterable field with the highest sequence number. If there is no such field, PRVFLD navigates the cursor to the current field.

PRVREC

Syntax: `PRVREC;`

Description: PRVREC navigates the cursor to the record with the next lower sequence number than the current record.

REDISP

Syntax: `REDISP;`

Description: REDISP redraws the screen. This eliminates any extraneous system messages.

SCRDOWN

Syntax: `SCRDOWN;`

Description: SCRDOWN scrolls the current block's list of records up by approximately 80 percent of the block's display. This action displays records that were "below" the block's display. If there are available records and a query is open in the block, SQL*Forms fetches records during SCRDOWN processing.

After scrolling, SCRDOWN navigates the cursor to the instance of the cursor field in the displayed record with the highest sequence number.

SCRLEFT

Syntax: `SCRLEFT;`

Description: SCRLEFT scrolls the value in the current field to the right by approximately 80 percent of the field's display. This action displays data that was to the left of the field's display.

SCRRIGHT

Syntax: `SCRRIGHT;`

Description: SCRRIGHT scrolls the value in the current field to the left by approximately 80 percent of the field's display. This action displays data that was to the right of the field's display.

SCRUP

Syntax: `SCRUP;`

Description: SCRUP scrolls the current block's list of records down by approximately 80 percent of the block's display. This action displays records that were "above" the block's display.

After scrolling, SCRUP navigates the cursor to the instance of the cursor field in the displayed record with the lowest sequence number.

SYNCHRONIZE

Syntax: `SYNCHRONIZE;`

Description: SYNCHRONIZE synchronizes the terminal screen with the internal state of the form.

UP

Syntax: `UP;`

Description: UP navigates the cursor to the instance of the current field in the record with the next lowest sequence number.

UPDREC

Syntax: `UPDREC;`

Description: UPDREC attempts to lock the row in the database that corresponds to the current record.

Failure Conditions for V2 Function Codes

In order to use success/failure logic in a V2 trigger that uses function codes, you should be sure of the conditions under which any specific function code will fail. To predict the proper V2 function code behavior, first identify the corresponding packaged procedure in the following section, "Matching V2 Function Codes to V3 Equivalents," and then track the failure of the packaged procedure in the appropriate flow chart in chapter 21, "Event Processing."

Related Topics: Event flow charts 21-2
Success/failure logic in V2 triggers F-32

Matching V2 Function Codes to V3 Equivalents

If you want to convert individual V2 triggers to V3 triggers, use the following table. This table matches each V2 function code name to the name of the V3 packaged procedure that shares the identical functionality.

In the case where there is no corresponding V3 packaged procedure, the table shows the PL/SQL code that performs the equivalent function or indicates that there is no equivalent.

V2 Function Code Name	*V3 Equivalent*
ABTQRY	ABORT_QUERY
BELL	BELL
CALL	CALL
CALLINPUT	CALL_INPUT
CALLQRY	CALL_QUERY
COPY	*source : = destination* or COPY
CHRMODE	no equivalent
CLRBLK	CLEAR_BLOCK
CLRFLD	CLEAR_FIELD
CLRFRM	CLEAR_FORM
CLRREC	CLEAR_RECORD
COMMIT	COMMIT
CQUERY	COUNT_QUERY
CREREC	CREATE_RECORD
DEFAULT	DEFAULT_VALUE
DELBACK	no equivalent
DELCHR	no equivalent
DELREC	DELETE_RECORD
DERROR	DISPLAY_ERROR
DKEYS	SHOW_KEYS
DOWN	DOWN
DUPFLD	DUPUCATE_FIELD
DUPREC	DUPLICATE_RECORD
ENDSTEP	no equivalent
ENDTRIG	no equivalent
ENTER	ENTER
ENTQRY	ENTER_QUERY
ERASE	ERASE
EXEQRY	EXECUTE_QUERY

Table continued on next page.

V2 Function Code Name	*V3 Equivalent*
EXETRG	EXECUTE_TRIGGER
EXIT	EXIT_FORM
GOBLK	GO_BLOCK
GOFLD	GO_FIELD
GOSTEP	no equivalent
HELP	HELP
HOST	HOST
LISTVAL	LIST_VALUES
MENU	BLOCK_MENU
MESSAGE	MESSAGE
MOVLEFT	no equivalent
MOVRIGHT	no equivalent
NEWFRM	NEW_FORM
NOOP	`MESSAGE('FRM-41000: You pressed an undefined function key. ' )`
NULL	`NULL`
NXTBLK	NEXT_BLOCK
NXTFLD	NEXT_FIELD
NXTKEY	NEXT_KEY
NXTREC	NEXT-RECORD
NXTSET	NEXT_SET
PAUSE	PAUSE
PRVBLK	PREVIOUS_BLOCK
PRVFLD	PREVIOUS_FIELD
PRVREC	PREVIOUS_RECORD
PRINT	PRINT
REDISP	REDISPLAY
SCRDOWN	SCROLL_DOWN
SCRLEFT	no equivalent
SCRRIGHT	no equivalent
SCRUP	SCROLL_UP
SYNCHRONIZE	SYNCHRONIZE
UP	UP
UPDREC	LOCK_RECORD

Related Topics: Packaged procedures 16-1
Upward compatibility B-1

CASE Statement

The CASE statement is a SQL*Forms statement that can only be used within the #EXEMACRO command. You can use a CASE statement to perform function codes conditionally.

You can place a CASE statement before or after other function codes within a #EXEMACRO command. Any function codes that appear in the #EXEMACRO but after the CASE statement are executed regardless of which alternative processing executes.

Note that a CASE statement fails if it contains invalid syntax or a statement within it fails.

Syntax: `CASE selector IS` *`alternative [alternative]`* `. . . END CASE;`

where:

`selector`	Specifies a `value`.

`alternative`	Specifies the following constructio:
	`WHEN choice [ |,` *`choice]`* `. . .`
	`THEN` *`statement`* `[statement. ..]`

`choice`	Specifies a `value` or the keyword `OTHERS`.
`statement`	Specifies a V2 function code or a CASE statement.

Refer to "V2 Function Code Arguments" for a detailed description of *`value.`*

Example: Assume you want to perform different actions within the ORDERS block depending on the value of the ORDERID field. The following trigger performs such an operation:

```
#EXEMACRO
    GOBLK ORDERS;
    CASE ORDERS.ORDERID IS
        WHEN ' ' THEN CLRBLK ;
        WHEN OTHERS THEN EXEQRY ;
        END CASE ;
    GOBLK ITEMS;
```

Related Topics: #EXEMACRO command F-11

Variables in Trigger Steps

Trigger steps can use the following types of variables:

- global variables
- system variables
- variable references

Global Variables in Trigger Steps

You use global variables in trigger steps the same way that you use them in V3 triggers, except you cannot reference global variables in SQL statements. In V2 triggers, assign global variables with the COPY V2 function code or the #COPY command.

Related Topics: #COPY command F-10
Global variables 18-2
V2 function codes, COPY F-15

System Variables in Trigger Steps

You use system variables in trigger steps the same way that you use them in V3 triggers. You can use the same system variables that are available in V3 triggers, including the following ones that are new in Version 3.0:

SYSTEM.BLOCK_STATUS
SYSTEM.CURSOR_RECORD
SYSTEM.FORM_STATUS
SYSTEM.LAST_QUERY
SYSTEM.LAST_RECORD
SYSTEM.RECORD_STATUS
SYSTEM.TRIGGER_RECORD

Related Topics: System variables 18-3

Variable References

A variable reference is an indirect reference to the value of a field, global variable, or system variable.

SQL*Forms evaluates a variable reference whenever it executes the SQL*Forms command or V2 function code that contains the reference.

Syntax: *`&variable`*

Example: Consider the following example of a SQL*Forms command that uses a variable reference:

```
#COPY &ORDERS.LOCATION GLOBAL LOCATION
```

When SQL*Forms evaluates the variable reference, it "looks" in the ORDERS.LOCATION field for the name of a variable that contains the value that it will copy into GLOBAL.LOCATION.

- If ORDERS.LOCATION contains SYSTEM.CURSOR_VALUE, SQL*Forms assigns the value in the current field-not the character string 'SYSTEM.CURSOR_VALUE'.
- If ORDERS.LOCATION contains the value &GLOBAL.STATE, SQL*Forms looks in GLOBAL.STATE for the name of another variable that contains the value to copy. Note that if you use variable references recursively, you need to guarantee that you do not create an infinite reference loop.
- If ORDERS.LOCATION contains the value &&GLOBAL.STATE, SQL*Forms evaluates the ampersands "inside out." That is, SQL*Forms looks in GLOBAL.STATE for the name of a variable that contains the name of another variable that contains the value to copy.

Related Topics: NAME_IN packaged function 17-16

V2 Trigger Processing

SQL*Forms processes V2 triggers within events the same way it processes V3 triggers within events. That is, each type of trigger fires during certain events and in certain orders. However, within individual triggers SQL*Forms processes V2 triggers differently than it processes V3 triggers.

When a V2 trigger fires, SQL*Forms executes the trigger steps within the trigger in order, one at a time. Note that you can change the order in which SQL*Forms executes trigger steps by branching with the GOSTEP function code and with the Success Label and Failure Label trigger step characteristics.

Related Topics: Processing, triggers within events 4-5
Success/failure logic in V2 triggers F-32

V2 Trigger Outcomes

A trigger can have one of two outcomes: success or failure.

success	A trigger succeeds if all of its steps succeed *or* if any failing step does not abort the trigger or returns success if it does.
failure	A trigger fails if any step causes a fatal error *or* if any failing step aborts the trigger without returning success.

The results of a V2 trigger failure or success are identical to the results of a V3 trigger failure or success.

Related Topics: Abort On Failure trigger step characteristic F-5
Results of trigger failures 4-7
Succeed On Abort trigger step characteristic F-8

Trigger Step Outcomes

A trigger step can have one of three outcomes: success, failure, or fatal error.

Since a trigger step can contain only one command, the outcome of the step is identical to the outcome of the command it contains. For example, if a trigger step contains a SQL statement and that SQL statement fails, the trigger step fails also.

The outcome of an individual trigger step depends on whether the trigger step consists of a user exit call, a SQL statement, or a SQL*Forms statement.

- SQL*Forms processes user exit calls in V2 triggers identically to the way it processes them in V3 triggers.
- SQL*Forms processes SQL statements in V2 triggers identically to the way it processes them in V3 triggers. However, since SQL*Forms does not check the syntax of V2 triggers, a SQL statement can result in a fatal error if it contains a syntax error.
- How SQL*Forms processes SQL*Forms commands is covered in the following section, "Processing SQL*Forms Commands."

Note: You can reverse the outcome of trigger step by using the Reverse Return Code trigger step characteristic.

Related Topics: Processing, SQL statements 4-5
Processing, user exit calls 4-6
Reverse Return Code trigger step characteristic F-7

Processing SQL*Forms Commands

A SQL*Forms command can have one of three outcomes: success, failure, or fatal error.

success	A SQL*Forms command succeeds if it executes properly.
failure	A SQL*Forms command fails if it cannot execute for any reason or it contains a syntax error.
fatal error	A SQL*Forms command causes a fatal error if the command causes the operating system to run out of memory.

Processing V2 Function Codes

When SQL*Forms processes an #EXEMACRO command, it processes each individual function code in order, one at a time. Any changes that a function code causes are visible to the function codes that follow it.

Each V2 function code can have one of three outcomes: success, failure, or fatal error.

success	A V2 function code succeeds if it executes properly.
failure	A V2 function code fails if it cannot execute for any reason.
fatal error	A V2 function code causes a fatal error if the statement causes the operating system to run out of memory.

In a V2 trigger, the outcome of an #EXEMACRO command (and thus the trigger step containing it) depends on the outcome of each individual function code the command contains.

- If a function code succeeds, SQL*Forms continues processing the #EXEMACRO statement.
- If a function code fails, SQL*Forms performs the following actions:
  - generates an error message
  - continues processing the #EXEMACRO statement
  - fails the trigger step
- If a function code causes a fatal error, SQL*Forms performs the following actions:
  - generates an error message
  - halts processing of the #EXEMACRO statement
  - stops the trigger step with a fatal error

Related Topics: #EXEMACRO command F-11

Success/Failure Logic in V2 Triggers

You can use success/failure logic in V2 triggers in order to conditionally change the order in which SQL*Forms processes trigger steps within an individual trigger.

To use success/failure logic, specify the Label trigger step characteristic for each trigger step in a trigger. Then specify the Success Label and Failure Label trigger step characteristics for each trigger step. For an individual trigger step, the assignment of the Success Label and the Failure Label determines which trigger step SQL*Forms branches to if the current trigger step succeeds or fails.

Example: Assume you want a trigger that redefines [Previous Field] when the cursor is in the ITEMS.ORDERID field (the first field of a record). The desired behavior is to have the cursor move to the ORDERS.TOTAL field (the last field in the previous block) under the specified condition. Attach a key trigger to the ITEMS.ORDERID field with the following characteristics:

- The first step has the following characteristics:

Step Number:	1
Label:	STEPl
Success Label:	STEP3
Failure Label:	STEP2
Trigger Step Text:	`#EXEMACRO PRVREC;`

- The second step has the following characteristics:

Step Number:	2
Label:	STEP2
Trigger Step Text:	`#EXEMACRO PRVBLK; GOFLD ORDERS.TOTAL; ENDTRIG;`

- The third step has the following characteristics:

Step Number:	3
Label:	STEP3
Trigger Step Text:	`#EXEMACRO PRVFLD;`

In this example, if the cursor is in the first record when the trigger executes, STEPl fails, the trigger branches to STEP2, and the cursor moves to the last field of the previous block (ORDERS.TOTAL). If the cursor is not in the first record when the trigger executes, STEP1 succeeds, the trigger branches to STEP3, and the cursor moves to the previous field.

INDEX

!

#COPY command F-10
  translating into PL/SQL B-8
#ERASE command F-11
  translating into PL/SQL B-8
#EXEMACRO command F-11
#HOST command F-11
  translating into PL/SQL B-9
$$DATE$$ 18-4
$$DATETIMES$$ 18-4
$$TIMES$$ 18-5
16-bit environment constraints 22-4

A

-a switch for SQL*Forms (Run Form) 23-16
Abort On Fail trigger step characteristic F-5
Abort On Failure trigger step characteristic F-5
ABORT_QUERY
  flow chart 21-4
  packaged procedure 16-5
[Accept] 7-4
Active page 13-6
Alert 5-3
ALL_RECORDS
  ENTER_QUERY parameter 16-17
  EXECUTE_QUERY parameter 16-18
ALPHA data type 12-7
Alphabetic data type 12-7
ALTER SESSION command 23-20, 26-5
ANCHOR_VIEW
  flow chart 21-5
  packaged procedure 16-5
Anonymous block 1-17, 15-3
Application 1-17
Application development 1-3
Application development tools
  connecting to 25-2
Application documentation 27-1
  creating 2-5
  detail documentation 27-2
  fields documentation 27-3
  object reference documentation 27-4
  pages documentation 27-4
  procedures documentation 27-5
  summary documentation 27-5
  triggers documentation 27-6
APPLICATION_CHARACTERISTIC
packaged function 17-3
Arguments
  packaged functions 17-2
  packaged procedures 16-3
  passing to a user exit 19-7
  V2 function codes F-12
Array processing 23-16
  suppressing 23-16
Array Size block characteristic 11-7
Arrow indicators 5-15

ASK_COMMIT
CLEAR_BLOCK parameter 16-9
CLEAR_FORM parameter 16-10
EXIT_FORM parameter 16-20
Assigning a cursor for all SQL statements
-o switch for SQL*Forms (Run Form) 23-18
Assigning a cursor for each query SELECT
-t switch for SQL*Forms (Run Form) 23-20
ATTR_OFF SET_FIELD parameter 16-30
ATTR_ON SET_FIELD parameter 16-30
Attributes
See Characteristics
Automatic Hint field characteristic 12-5
Automatic login 23-2
disabling 23-18
Automatic search and complete 16-22
Automatic Skip field characteristic 12-5

B

-b switch
SQL*Forms (Generate) 23-14
SQL*Forms (Run Form) 23-16
Banner
suppressing for SQL*Forms (Convert) 23-10
Base Line block characteristic 11-7
Base Table
block characteristic 11-8
field characteristic 12-6
Base table field 1-17,12-26
Base tables 1-17
for SQL*Forms Design) 24-4
Basic tasks 2-2
Basics of SQL*Forms 1-1
[Beginning of Line] 7-4
BELL
flow chart 21-6
packaged procedure 16-6
Bit-mapped environment D-4
function keys D-5
interface elements D-4
navigation D-5
scrolling 5-15
Blank spaces in field values 3-32
Blk: status line item 5-16
Block
block characteristic 11-8
trigger characteristic 14-5
Block Definition
form 11-2
spread table 11-3
Block menu 11-11, 11-13
displaying the menu at login 23-18
Block mode environment D-2
Automatic Hint disabled 12-5
creating fields D-3
key script file 29-4
setting the validation unit D-2
using constant text D-4
using triggers D-3
Block Name
block characteristic 11-8
trigger characteristic 14-5
Block types 11-18
control 11-18
detail 11-18
master 11-19
multi-record 11-19
non-enterable 11-19
See *also* PL/SQL blocks
referenced 20-4
single-record 11-19
BLOCK_CHARACTERISTIC packaged function 17-4
BLOCK_MENU
flow chart 21-7
packaged procedure 16-6
Blocks 1-6, 1-17, 11-1
anonymous (PL/SQL) 15-3
characteristics 11-6
constraints 22-2
context 6-3
defining 11-2
list of records 21-53
master-detail relationship 1-7, 11-20
multi-record 1-18 to 1-19
PL/SQL blocks 15-2
referenced 20-4
sequence 11-16
types 11-18

Boilerplate
See Constant text
[Bookmark] 7-4
Bookmark help feature 8-5
Border page characteristic 10-5
Boxes
drawing 2-11
BREAK
flow chart 21-8
packaged procedure 16-6
Break processing 28-2
Break Processing flow chart 21-9
Buffer
paste 2-10
records 23-16
rows buffered 11-15
undo 26-6
Buffer Record in File run option 26-2
Buffering records 23-16
Button 5-4
Bypassing login
-n switch for SQL*Forms (Design) 23-12, 23-14

C

-c switch
SQL*Forms (Convert) 23-9
SQL*Forms (Design) 23-11
SQL*Forms (Run Form) 23-17
CALL
flow chart 21-10
packaged procedure 16-7
CALL_INPUT
flow chart 21-11
packaged procedure 16-7
CALL_QUERY
flow chart 21-12
packaged procedure 16-8
Callable forms 25-10
Calling a user exit from a trigger 19-10
Calling forms from programs 25-10
[Cancel] 7-4
Capturing terminal output interactively
-i switch for SQL*Forms (Design) 23-12
-i switch for SQL*Forms (Run Form) 23-17
Capturing terminal output only in a file
-w switch for SQL*Forms (Design) 23-13
-w switch for SQL*Forms (Run Form) 23-21
Case insensitivity in object names 20-2
CASE statement F-27
translating to V3 B-8
CAUTION alert label 5-3
[Change Display Type] 7-4
Changed
field validation state 3-9
record validation state 3-10
SYSTEM.BLOCK_STATUS value 18-5
SYSTEM.FORM_STATUS value 18-9
SYSTEM.RECORD_STATUS value 18-12
Changed features A-3
Changing data
after posting 3-22
before posting 3-22
during commit processing 3-21
CHAR data type 12-7
Character data type 12-7
Character mode environment D-1
Characteristics
block 11-6
field 12-4
form 9-3
page 10-4
trigger 14-4
trigger step F-4
characters
embedded 12-15
Check box 5-5
using a mouse D-5
Check the Block for Query 21-13
Check the Block for Update 21-14
Check the Field for Edit 21-15
Check the Record for Update 21-17
[Clear Field] 7-4
CLEAR_BLOCK
flow chart 21-18
packaged procedure 16-8
CLEAR_EOL
flow chart 21-19
packaged procedure 16-9

CLEAR_FIELD
flow chart 21-20
packaged procedure 16-9
CLEAR_FORM
flow chart 21-21
packaged procedure 16-9
CLEAR_RECORD
flow chart 21-22
packaged procedure 16-10
Column Sec block characteristic 11-9
Columns 1-20
naming restrictions 12-15
security 11-9
used in creating a block 11-16
combining command line switches 23-8
command line switches 23-8
combining 23-8
SQL*Forms (Convert) 23-8
SQL*Forms (Design) 23-10
SQL*Forms (Generate) 23-13
SQL*Forms (Run Form) 23-15
Command syntax v
Commands in trigger steps F-8
SQL commands F-9
SQL*Forms commands F-10
user exit calls F-11
comment
block characteristic 11-9
field characteristic 12-6
form characteristic 9-3
trigger characteristic 14-5
trigger step characteristic F-5
Commit processing 3-16
altering the default behavior 3-22
changing data during commit processing 3-21
posting data 3-17
when commit processing occurs 3-20
COMMIT.FORM
flow chart 21-23
packaged procedure 16-11
Compatibility between versions B-1
Compiling PL/SQL
at design time 15-11
at generation time 15-12
error messages 15-13
from SQL*Forms Design) 15-12
from the command line 15-12
Components 23-1
introduction to 1-15
SQL*Forms (Convert) 23-3
SQL*Forms (Design) 23-5
SQL*Forms (Generate) 23-6
SQL*Forms (Run Form) 23-7
Concatenating arguments
packaged functions 17-3
packaged procedures 16-4
V2 function codes F-12
Connecting to other products 25-1
application development tools 25-2
ORACLE RDBMS 25-2
SQL*Menu 25-3
SQL*Plus 25-7
SQL*ReportWriter 25-8
Constant text 1-17
block mode environment D-4
constraints 22-2
copying 2-3
editing 2-10
entering 2-10
ownership 2-11
constraints 1-20,22-1
cursor 22-3
database 11-17
object 22-2
16-bit environment 22-4
Contents help feature 8-4
Context 1-17, 6-1
changing with function keys 6-6
forms, blocks, and fields 6-3
in the screen painter 6-6
relation to main menu choices 6-2
relation to zoom connections 6-2
spread table 5-15
triggers 6-5

Context for forms, blocks, and fields 6-3
default context 6-4
keeping blocks and fields in context order 6-4
lost context 6-4
Context for triggers 6-5
defining triggers out of context 6-5
Continue with Debug Messages debug option 28-3
Continue without Debug Messages debug option 28-3
Control block 11-18
creating 2-3
Control field 12-26
Conventions
naming 20-2
Converting an INP file to database format
-i switch for SQL*Forms (Convert) 23-10
Converting an INP file to the new INP format
-v switch for SQL*Forms (Convert) 23-10
Converting existing forms B-2
Version 2.0 forms B-2
Version 2.3 forms B-2
CONVFORM command 23-4
Coordinates
field editor position 12-13
field position 12-22
list of values position 12-17
page position 10-9
view position 10-8
[copy] 7-5
flow chart 21-24
packaged procedure 16-11
#COPY command F-10
translating into PL/SQL B-8
[copy object] 7-5
COPY_REGION
flow chart 21-25
packaged procedure 16-11
Copying objects 2-3
COUNT_QUERY
flow chart 21-26
packaged procedure 16-11
Counting query hits 3-30
CREATE_RECORD
flow chart 21-27
packaged procedure 16-12
Creating a block
control block 2-3
default block 2-4
Creating a field 2-4
from the screen painter 2-5
in block mode D-3
Creating a form 2-5
Creating a key script file
-e switch for SQL*Forms (Design) 23-11
-e switch for SQL*Forms (Run Form) 23-17
Creating a user exit 19-4
Creating application documentation 2-5
CREATION column in the IAPXTB table 19-9
CRT utility
See Oracle*Terminal
Current cursor position 3-3
Current objects 3-5
Cursor
assigning one for all SQL statements 23-18
assigning one for each query SELECT 23-20
constraints 22-3
current position 3-3
database 22-3
displaying statistics on cursor use 23-19, 26-5
initial position 3-3
movement functions 7-18
navigation 3-3
sharing for implicit SQL 26-4
sharing for trigger SQL 26-4
target position 3-3
[Cut] 7-5
effect on context 6-6
CUT_REGION
flow chart 21-28
packaged procedure 16-12
Cutting an object 2-9

D

-d switch
SQL*Forms (Convert) 23-9
SQL*Forms (Run Form) 23-17
Damage control function 19-16
Data
committing 3-16
posting 3-16

Data Command Language statements
See DCL statements
Data Definition Language statements
See DDL statements
Data dictionary 1-20
Data Manipulation Language statements
See DML statements
Data Type field characteristic 12-6
Data types
ALPHA 12-7
changes B-9
CHAR 12-7
DATE 12-7
DATETIME 12-7
EDATE 12-8
evaluating values 12-25
INT 12-8
JDATE 12-8
LONG 12-8
MONEY 12-9
NUMBER 12-9
RINT 12-9
RMONEY 12-9
RNUMBER 12-10
TIME 12-10
Database 1-20
administrator 1-20
cursors 22-3
remote connect 23-2
terms 1-20
Database field
See Base table field
Database format 24-2
converting to INP format 23-3
$$DATE$$ 18-4
DATE data type 12-7
$$DATETIME$$ l8-4
DATETIME data type 12-7
DBA 1-20
DCL statements
in triggers 4-6
DDL statements
in triggers 4-6
Deadlock
preventing 3-16
Debug facility 28-1
break processing 28-2
debug messages 28-2
Debug mode 26-2
invoking 23-17, 26-2
options 28-2
Debug Mode run option 26-2
Default block
creating 2-4
default block facility 11-6
Default Block form 11-4
Default field characteristic 12-10
Default menu application 25-5
Default Menu Application form characteristic 9-3
Default ORDER BY clause 3-28
Default Value field characteristic 12-10
Default WHERE clause 3-28
Default Where/Order By block characteristic 11-9
syntax 11-10
DEFAULT_VALUE
flow chart 21-29
packaged procedure 16-12
[Define Field] 7-5
effect on context 6-6
Defining blocks 11-2
Defining fields 12-2
Defining form-level procedures 15-4
Defining forms 9-2
Defining pages 10-2
Defining triggers 14-2
out of context 6-5
V2 triggers F-2
Definition level
See Trigger definition level
[Delete Backward] 7-5
[Delete Character] 7-6
Delete Details block characteristic 11-11
[Delete Line] 7-6
[Delete Record] 7-6
DELETE_RECORD
flow chart 21-30
packaged procedure 16-12
Deletes 3-20
Deleting an object 2-9

Deleting the form from the database
-d switch for SQL*Forms (Convert) 23-9
Descrip trigger characteristic 14-5
Description
block characteristic 11-13
trigger characteristic 14-5
Design interface
See Interface elements
Design options 26-5
Designer 1-17
Detail block 11-18
Detail documentation 27-2
Dialog box 5-6
Dis Len field characteristic 12-11
Disable Array Processing run option 26-3
Disable Trigger Compilation
design option 26-6
Disabled records 21-99
Disabling automatic login
-1 switch for SQL*Forms (Run Form) 23-18
Display a Message 21-31
Display Block Menu run option 26-3
Display Global Variables debug option 28-3
Display Length field characteristic 12-11
Display log file
SQL*Forms (Design) 23-13
SQL*Forms (Run Form) 23-21
Display Page O Fields debug option 28-3
Display style
See Use Forms as Default design option
Display System Variables debug option 28-3
DISPLAY_=ERROR
flow chart 21-32
packaged procedure 16-13
DISPLAY_FIELD
flow chart 21-34
packaged procedure 16-13
DISPLAY_PAGE
flow chart 21-35
packaged procedure 16-13
Displayed field characteristic 12-12
Displaying statistics on cursor use
-s switch for SQL*Forms (Run Form) 23-19
Statistics run option 26-5
Displaying the block menu at login
-m switch for SQL*Forms (Run Form) 23-18
DML statements
in triggers 4-5
translating into PL/SQL B-7
DO_COMMIT
CLEAR_BLOCK parameter 16-9
CLEAR_FORM parameter 16-10
EXIT_FORM parameter 16-20
DO_KEY
flow chart 21-33
packaged procedure 16-14
DO_REPLACE
CALL parameter 16-7
CALL_QUERY parameter 16-8
Documentation on forms
See Application documentation
Documentation set 0-ii
your comments 0-v
DOWN
flow chart 21-37
packaged procedure 16-16
[Down] 7-6
effect on context 6-6
[Draw Box/Line] 7-6
Drawing lines and boxes 2-11
DUPLICATE_FIELD
flow chart 21-36
packaged procedure 16-16
DUPLICATE_RECORD
flow chart 21-38
packaged procedure 16-16
Dynamic field characteristics
See SET_FIELD, packaged procedure

E

-e switch
SQL*Forms (Design) 23-11
SQL*Forms (Run Form) 23-17
Echo Input field characteristic 12-12
Echoing keystrokes
SQL*Forms (Design) 23-11
SQL*Forms (Run Form) 23-17
EDATE data type 12-8
Edit keystroke facility 29-1
using 29-2
Edit mode 13-2

EDIT_FIELD
flow chart 21-39
packaged procedure 16-16
Editing
constant text 2-10
key script file 29-4
SQL*Forms (Design) functions 7-18
SQL*Forms (Design) keys 7-18
Editor 13-2
Elements in the design interface
See Interface elements
Elevator 5-15
Embedded characters 12-15
[End of Line] 7-7
Enforce Foreign Key field characteristic 12-12
Enforce Key field characteristic 12-12
ENTER
flow chart 21-40
packaged procedure 16-17
Enter Query mode 3-25
restrictions 3-29
Enter the Block 21-42
Enter the Field 21-43
Enter the Form 21-44
Enter the Record 21-45
Enter the Value Into a Field 21-46
ENTER_QUERY
flow chart 21-47
locking rows 3-13
packaged procedure 16-17
Enterable field 12-27
Entering a query 3-24
Entering constant text 2-10
Environments
bit-mapped D-4
block mode D-2
character mode D-1
portability 24-3
16-bit 22-4
ERASE
flow chart 21-48
packaged procedure 16-18
#ERASE command F-11
translating into PL/SQL B-8
Error help 8-5
Error messages
message type E-43
See also Messages
PL/SQL compilation 15-13
ERROR_CODE packaged function 17-5
ERROR_TEXT packaged function 17-6
ERROR_TYPE packaged function 17-7
European date data type 12-8
Event flow charts 21-2
assumptions 21-3
sub-events 21-2
Event processing 21-1
flow charts 21-2
Events 1-11
in different environments 1-12
involving navigation 3-5
sub-events 21-131
Example record 3-28
Exception handling 15-8 to 15-9
Exclusive lock 3-12
row lock 3-12
table lock 3-12
EXEC IAF GET 19-5
EXEC IAF PUT 19-6
EXEC ORACLE statements 19-7
EXEC SQL statements 19-4
Executable statements
failure 15-8
processing 4-5
Execute the Query 21-49
EXECUTE_QUERY
flow chart 21-50
locking rows 3-13
packaged procedure 16-18
EXECUTE_TRIGGER
flow chart 21-51
packaged procedure 16-19
Executing a form 2-6
Executing a query 3-25
#EXEMACRO command F-11
Exit SQL*Forms (Run Form)
debug option 28-3
EXIT_FORM
flow chart 21-52
packaged procedure 16-19

Exiting SQL*Forms (Design) 2-8
Explicit locking 3-13

F

-f switch
  SQL*Forms (Convert) 23-9
Failing to obtain a lock 3-14
Failure
  failure conditions for V2 function codes F-24
  packaged procedure outcome 4-6
  SQL*Forms command outcome F-31
  trigger outcome 4-7
  user exit outcome 4-6
  V2 function code outcome F-31
  V2 trigger outcome F-30
Failure Label trigger step characteristic F-5
Failure Message trigger step characteristic F-6
Failure trigger step characteristic F-5
FALSE
  SYSTEM.LAST_RECORD value 18-10
Fatal error
  packaged procedure outcome 4-6
  SQL*Forms command outcome F-31
  user exit outcome 4-6
  V2 function code outcome F-31
Features A-1
  changed A-3
  help system 8-3
  new A-2
Fetch processing 3-26
  array size 11-7
  restrictions 3-30
Fetch Records 21-53
Fetching records
  See Fetch processing
Field adjustment 3-32
Field characteristics 12-4
  propagation of changes with SET_FIELD 16-32
  validation of changes with SET_FIELD 16-30
Field Definition
  form 12-2
  spread table 12-3
Field display mask
  See Format mask
Field editor 13-2
Field Editor Location field characteristic 12-13
Field label
  see constant text
Field Length field characteristic 12-13
Field Name
  field characteristic 12-14
  trigger characteristic 14-6
Field trigger characteristic 14-6
Field types 12-26
  base table 12-26
  control 12-26
  enterable 12-27
  hidden 12-27
  look-up 12-27
  non-enterable 12-27
  referenced 20-5
  scrolled 12-28
Field validation
  in the design interface 5-8
  states 3-8
FIELD_CHARACTERISTIC packaged function 17-8
Fields 1-8,1-17, 12-1
  adjustment 3-32
  block mode considerations D-3
  characteristics 12-4
  constraints 22-2
  context 6-3
  creating 2-4
  defining 12-2
  design interface element 5-6
  documentation 27-3
  dynamic changes 16-32
  editor 13-2
  foreign key 11-21
  instances 1-8
  masking 12-15
  modifying 2-7
  referenced 20-5
  referencing in SQL F-9
  selecting 2-9
  sequence 12-22
  types 12-26
  validation states 3-8
First letter selection 5-12
[First Line] 7-7

FIRST_RECORD
flow chart 21-56
packaged procedure 16-20
Fixed Length field characteristic 12-15
Fld Len field characteristic 12-13
Fld status line item 5-16
Flow charts 21-2
assumptions 21-3
sub-events 21-2
Fn functions 25-11
FOR_UPDATE
ENTER_QUERY parameter 16-17
EXECUTE_QUERY parameter 16-18
Foreign key 1-20, 11-20
Form Definition form 9-2
Form processing commands in triggers 1-10
Form storage formats 24-2
database 24-2
FRM 24-2
INP 24-3
Form-level procedures 1-10, 1-18, 15-4
defining 15-4
documentation 27-5
name 15-4
referenced 20-6
spec 15-12
text 15-5
FORM_APP table 24-4
FORM_BLK table 24-5
FORM_CHARACTERISTIC packaged function 17-11
FORM_COMMENT table 24-6
FORM_FAILURE packaged function 17-11
FORM_FATAL packaged function 17-12
FORM_FLD table 24-6
FORM_MAP table 24-8
FORM_PAGE table 24-9
FORM_PROCEDURE table 24-10
FORM_REFERENCE table 24-10
FORM_SQLTXT table 24-11
FORM_SUCCESS packaged function 17-13
FORM_TRG table 24-11
FORM_TRIGGER table 24-12
FORM_TRIGGER_FAILURE exception 4-7
Format mask 12-15
status 3-11
when applied 3-32
Format Mask field characteristic 12-15
Forms 1-18,9-1
characteristics 9-3
creating 2-5
defining 9-2
design interface element 5-8
executing 2-6
generating 2-6
loading 2-7
names 20-2
non-enterable form type 9-6
query-only form 16-8
running 2-6
saving 2-8
storage 24-1
FRM format 24-2
Frm: status line item 5-16
FULL_ROLLBACK
CLEAR_FORM parameter 16-10
Function code statements
translating into PL/SQL B-7
Function codes
See Functions
See V2 function codes
Function keys 0-iv, 7-1
bit-mapped environment D-5
changing context 6-6
cursor movement keys 7-18
descriptions 7-3
editing keys 7-18
function key triggers 14-8
general design keys 7-18
groupings 7-17
Key F0 through Key-F9 25-11
mapping with Oracle*Terminal 25-11
processing 3-31
redefining SQL*Forms (Run Form) keys 14-8
screen painting keys 7-18
static 14-10
triggers 14-8

Functions 1-11
damage control 19-16
Fn functions 25-11
See also Function keys
packaged 17-1
See also Packaged procedures
See also V2 function codes
without packaged procedures 16-2
Fundamental processing 3-1
commit processing 3-16
field adjustment 3-32
function key processing 3-31
locking 3-11
navigation 3-2
query processing 3-24
validation 3-6

G

General design functions 7-18
General translation issues B-3
Generate Before Executing Form design option 26-6
GENERATE command 23-6
Generating a form 2-6
GENXTB
form 19-8
program 19-8
GET 19-5
Getting started 0-i
Global variables 1-18, 18-2
constraints 22-2
GO_BLOCK
flow chart 21-57
packaged procedure 16-20
GO_FIELD
flow chart 21-58
packaged procedure 16-20
GO_RECORD
flow chart 21-59
packaged procedure 16-21
Graphics
See also Drawing lines and boxes
translating between versions 23-9

H

[Help] 7-7
HELP
flow chart 21-60
packaged procedure 16-21
Help on help 8-3
Help System 8-1
error help 8-5
features 8-3
invoking 2-7
organization 8-2
Hidden field 12-27
HIDE
CALL parameter 16-7
CALL_QUERY parameter 16-8
HIDE_MENU
flow chart 21-61
packaged procedure 16-21
HIDE_PAGE
flow chart 21-62
packaged procedure 16-21
High setting for Valid Range 12-24
Hint field characteristic 12-16
Hint Text field characteristic 12-16
History panel 8-4
Horizontal scroll bar 5-15
scrolling 5-15
Horizontal Scroll Bar page characteristic 10-5
HOST
flow chart 21-63
packaged procedure 16-22
#HOST command F-11
translating into PL/SQL B-9
Host languages 19-3

I

-i switch
SQL*Forms (Convert) 23-10
SQL*Forms (Design) 23-12
SQL*Forms (Run Form) 23-17
LAC
See SQL*Forms (Convert)

IAD
See SQL*Forms (Design)
IAG
See SQL*Forms (Generate)
IAP
See SQL*Forms (Run Form)
IAPCAL command 25-10
IAPPAPP table 24-4
IAPXTB table 19-9
Implicit locking 3-13
In Menu block characteristic 11-11
Index
help feature 8-4
unique 1-21, 12-20
using this book's index iii
Informative message type E-43
Inheritance 1-14
Initial cursor position 3-3
INP file format 24-3, 24-13
INP to database conversion 23-3
INP to INP conversion 23-4
Input Allowed field characteristic 12-17
Ins status line item 5-16
INSERT
See also Inserts
SYSTEM.RECORD_STATUS value 18-12
[Insert Line] 7-7
Insert mode 5-16
[Insert Record] 7-7
effect on context 6-7
[Insert/Replace] 7-7
Inserts 3-19
Instance 1-8
INT data type 12-8
Integer data type 12-8
Integrity constraints
See Use Constraints block characteristic
Interactive key capture
SQL*Forms (Design) 23-12
SQL*Forms (Run Form) 23-17
Interface elements 5-1
bit-mapped environment D-4
online descriptions 8-1
INTO clause in SELECT F-9
Invalid
See also Changed, field validation state
See also Changed, record validation state
format mask status 3-11
Invoking debug mode
-d switch for SQL*Forms (Run Form) 23-17
Invoking quiet mode
-q switch for SQL*Forms (Run Form) 23-19

J

JDATE data type 12-8
Join Condition block characteristic 11-11
Julian date data type 12-8

K

Key
See Primary key
Key script file 23-11, 23-17, 29-3
editing 29-4
Key triggers 14-8
failure results 4-8
function key triggers 14-8
Key-Fn trigger 14-11
Key-Others trigger 14-11
Key-Startup trigger 14-12
Key-F0 through Key-F9
defining with Key-Fn triggers 14-11
mapping 25-11
Key-Fn trigger 14-11
Key-Others trigger 14-11
Key-Startup trigger 14-12
Keyboard map 7-2
Keys
See Function keys
Keystroke capture 29-1
Keywords in help screens 8-4

L

-l switch
SQL*Forms (Design) 23-12
SQL*Forms (Run Form) 23-18
Label trigger step characteristic F-6

Languages
See Host languages
[Last Line] 7-8
LAST_RECORD
flow chart 21-64
packaged procedure 16-22
Leading zeros 3-32
Leave the Block 21-65
Leave the Field 21-66
Leave the Form 21-67
Leave the Record 21-68
Leave Unit Error Processing 21-69
[Left] 7-8
Length
display length 12-11
field length 12-13
fixed length 12-15
query length 12-21
Limit Undo Buffer to 1 design option 26-6
Limiting the undo buffer
-l switch for SQL*Forms (Design) 23-12
Limits
See also Constraints
master-detail relationship 11-22
Lines
drawing 2-11
See also Screen line
Lines Per Record block characteristic 11-12
[List] 7-8
List field characteristic 12-17
List of records 21-53
List of values 5-9
automatic search and complete 16-22
constraints 22-2
list area 5-9
pop-up 13-4
search field 5-10
search function 5-10
V2 style 12-17
V3 style 12-17
List of Values Pos field characteristic 12-17
List of Values SQL Text field characteristic 12-17
List of Values Title field characteristic 12-19
List status line item 5-16
LIST_VALUES
flow chart 21-71
packaged procedure 16-22
Loading a form 2-7
Local variables
See PL/SQL local variables
Location
See Field Editor Location field characteristic
Lock the Row 21-73
LOCK_RECORD
flow chart 21-75
packaged procedure 16-23
Locking 3-11
altering the default behavior 3-15
explicit locking 3-13
failing to obtain a lock 3-14
implicit locking 3-13
methods of locking rows 3-13
preventing deadlocks 3-16
recommendations 3-16
types of locks 3-12
when locks are released 3-15
with packaged procedures 3-13
Locks 1-21
caused by operators 3-12
caused by triggers 3-12
exclusive row lock 3-12
exclusive table lock 3-12
when locks are released 3-15
Log file
SQL*Forms (Design) session 23-13
SQL*Forms (Run Form) session 23-21
Login
account 1-18
See also Automatic login
bypassing login 23-12, 23-14
remote login 23-2
LONG data type 12-8
Look-up field 12-27
LOV
See List of values
Low setting for Valid Range 12-24

M

-m switch for SQL*Forms (Run Form) 23-18
Macros
  See V2 function codes
Main menu 5-11
  help system 8-3
Mandatory field
  See Required field characteristic
Mappings
  keyboard map 7-2
  preset mappings 26-8
  SQL*Forms (Design) 23-11
  SQL*Forms (Run Form) 23-17
  terminal and key definitions 25-11
Mark Fields and Records as Changed 21-76
Masking
  See Format Mask field characteristic
Master block 11-19
Master Block block characteristic 11-13
Master-detail relationship 1-7, 11-20
  deletion 11-28
  help on defining 11-21
  limits 11-22
  triggers 11-22
Matching V2 function codes to V3 equivalents F-24
Menu
  See also Block menu
  design interface element 5-11
  See also SQL*Menu
[Menu] 7-8
Menu Description block characteristic 11-13
Menu Security Group Name form characteristic 9-4
MESSAGE
  flow chart 21-77
  packaged procedure 16-23
Message line 5-12
Message severity levels E-42
Message types E-43
  error E-43
  informative E-43
  working E-43
MESSAGE_CODE packaged function 17-13
MESSAGE_TEXT packaged function 17-14
MESSAGE_TYPE packaged function 17-15
Messages E-1
  debug 28-2
  See also Error messages
  severity levels E-42
  SQL*Forms (Convert) E-2
  SQL*Forms (Design) E-n
  SQL*Forms (Generate) E-30
  SQL*Forms (Run Form) E-42
  types E-43
  typography 0-iv
Methods of locking rows 3-13
  explicit locking 3-13
  implicit locking 3-13
Modes
  Edit 13-2
  Enter Query 3-25
  insert 5-16
  normal 3-25
  replace 5-16
MODIFY column in the LAPXTB table 19-9
Modifying a field 2-7
  from the screen painter 2-7
Modifying a form
  See Opening a form
Modifying referenced objects 20-4
MONEY data type 12-9
Mouse
  navigation limits 9-4
  selecting items D-5
  using check boxes D-5
Mouse Navigation Limit form characteristic 9-4
MOVE_VIEW
  flow chart 21-78
  packaged procedure 16-23
Multi-record block 1-18, 11-19

N

-n switch
  SQL*Forms (Design) 23-12
  SQL*Forms (Generate) 23-14
NAME column in the IAPXTB table 19-9
NAME_IN packaged function 17-16

Named mapping
See Mappings
Naming conventions 20-2
National language support 17-6, 17-14
[Navigate] 7-8
effect on context 6-7
Navigate to the Navigation Unit Level 21-79
Navigation 3-2
bit-mapped environment D-5
current objects 3-5
cursor 3-3
events that involve navigation 3-5
failure 3-5
introduction to 1-12
mouse navigation limit 9-4
navigation unit 3-2
navigational path 3-4
triggers 14-8
what navigation is not 3-6
Navigation unit 3-2
Navigational path 3-4
Navigational triggers 14-8
New
field validation state 3-8
record validation state 3-10
SYSTEM.BLOCK.STATUS value 18-5
SYSTEM.FORM_STATUS value 18-9
SYSTEM.RECORD_STATUS value 18-12
New Cursor Area trigger step characteristic F-6
New Cursor trigger step characteristic F-6
New features A-2
NEW_FORM
flow chart 21-80
packaged procedure 16-24
[Next Block] 7-9
effect on context 6-7
[Next Field] 7-9
[Next Page] 7-10
[Next Record] 7-10
NEXT_BLOCK
flow chart 21-81
packaged procedure 16-24
NEXT_FIELD
flow chart 21-82
packaged procedure 16-24
NEXT_KEY
flow chart 21-84
packaged procedure 16-25
NEXT_RECORD
flow chart 21-86
packaged procedure 16-25
NEXT_SET
flow chart 21-87
packaged procedure 16-25
No login
SQL*Forms (Design) 23-12
No semi-colon for SQL F-9
NO_COMMIT
CLEAR_BLOCK parameter 16-9
CLEAR_FORM parameter 16-10
EXIT_FORM parameter 16-20
NO_HIDE
CALL parameter 16-7
CALL_QUERY parameter 16-8
NO_REPLACE
CALL parameter 16-7
CALL_QUERY parameter 16-8
NO_RESTRICT
LIST_VALUES parameter 16-22
NO_ROLLBACK
EXIT_FORM parameter 16-20
NO_SCREEN
HOST parameter 16-22
NO_VALIDATE
CLEAR_BLOCK parameter 16-9
CLEAR_FORM parameter 16-10
EXIT_FORM parameter 16-20
Non-database field
See Control field
Non-enterable block 11-19
Non-enterable field 12-27
Non-enterable form 9-6
Non-ORACLE user exits 19-4
Normal mode 3-25
NOT NULL columns 12-21
NOTE alert label 5-3
NOWAIT
ENTER_QUERY parameter 16-18
EXECUTE_QUERY parameter 16-19
NUMBER data type 12-9

O

-o switch
  SQL*Forms (Generate) 23-15
  SQL*Forms (Run Form) 23-18
Object reference documentation 27-4
Objects 1-18
  a closer look 1-5
  constraints 22-2
  copying 2-3
  current 3-5
  cutting 2-9
  hierarchy 1-6
  introduction to 1-4
  naming conventions 20-2
  pasting 2-10
  referencing 1-5, 2-3, 20-4
  referencing in PL/SQL 15-7
  reusing 1-5
  selecting 2-9
  working with objects 20-1
OCI user exits 19-3
On-Clear-Block trigger 14-13
  failure results 4-8
On-Database-Record trigger 14-13
  failure results 4-8
On-Delete trigger 14-14
  failure results 4-8
On-Error trigger 14-14
  failure results 4-8
On-Insert trigger 14-15
  failure results 4-8
On-Lock trigger 14-15
  failure results 4-8
On-Message trigger 14-15
  failure results 4-8
On-New-Field-Instance trigger 14-16
  failure results 4-8
On-New-Record trigger 14-17
  failure results 4-8
On-Remove-Record trigger 14-17
  failure results 4-8
On-Update trigger 14-18
  failure results 4-8
On-Validate-Field trigger 14-18
  failure results 4-8
On-Validate-Record trigger 14-19
  failure results 4-8
Online help
  See Help system
Open the Query 21-88
Opening a form 2-8
Opening a query 3-25
Operator 1-18
OPS$ ID 23-2
options 26-1
  preset options 26-7
  SQL*Forms (Design) 26-5
  SQL*Forms (Run Form) 26-2
ORACLE Call Interface user exits 19-3
ORACLE Precompilers
  user exits 19-3
  using with SQL*Forms 25-9
ORACLE RDBMS
  connecting to 25-2
Oracle*Terminal 25-11
ORDER BY clause 11-9
  syntax 11-10
Ordering block characteristic
  ORDER BY clause 11-9
  WHERE clause 11-9
Organization
  book 0-iii
  help system 8-2

P

Packaged functions 17-1
  arguments 17-2
  concatenating arguments 17-3
  definitions 17-2

Packaged procedures 1-18, 16-1
arguments 16-3
concatenating arguments 16-4
controlling menus 25-4
definitions 16-2
failure 15-9
functions without packaged procedures 16-2
in PL/SQL 15-8
matching to V2 function codes F-24
outcome 4-6
processing 4-6
restricted 16-35
unrestricted 16-34
Page 0 28-3
Page Definition
form 10-2
spread table 10-3
Page Number
block characteristic 11-13
page characteristic 10-5
Page setting for Screen Position 12-22
Page Size page Characteristic 10-6
Pages 1-8, 1-19, 10-1
characteristics 10-4
constraints 22-2
defining 10-2
display order 13-8
documentation 27-4
making a page active 13-6
pop-up 13-5
view of a page 13-5
Painter
See Screen painter
Passing arguments to a user exit 19-7
Passing data between forms and menus 25-4
Passing return values from a user exit 19-8
[Paste] 7-10
Paste buffer 2-10
PASTE_REGION
flow chart 21-89
packaged procedure 16-25
Pasting an object 2-10
PAUSE
flow chart 21-90
packaged procedure 16-26
Pg Sz page characteristic 10-6
PL/SQL 15-1
blocks 15-2
See also Compiling PL/SQL
exception handling 15-8 to 15-9
executable statement failure 15-8
how SQL*Forms uses PL/SQL 15-2
local variables 18-2
packaged procedures in PL/SQL 15-8
restrictions within SQL*Forms 15-8
SQL*Forms objects in PL/SQL 15-7
SQLCODE function 15-8
SQLERRM function 15-8
PL/SQL blocks 15-2
anonymous block 15-3
form-level procedure 15-4
PL/SQL local variables 18-2
in packaged function arguments 17-2
in packaged procedure arguments 16-4
Pop Up page characteristic 10-6
Pop-up objects
field editor 13-2
list of values 13-4
page 13-5
See also Pop-up windows
Pop-up windows 1-19, 13-1
damage control function 19-16
example program 19-17
user exit routines 19-10
Portability among systems 24-3
Portable development environment 1-4
Position
See List of Values Pos field characteristic
POST
flow chart 21-91
packaged procedure 16-26
Post and Commit Transactions 21-92
Post-Block trigger 14-19
failure results 4-9
Post-Change trigger 14-19
failure results 4-9
Post-Commit trigger 14-20
failure results 4-9
Post-Delete trigger 14-21
failure results 4-9
Post-Field trigger 14-21
failure results 4-9

Post-Form trigger 14-21
failure results 4-9
Post-Insert trigger 14-22
failure results 4-9
Post-Query trigger 14-22
failure results 4-9
Post-Record trigger 14-22
failure results 4-9
Post-Update trigger 14-23
failure results 4-9
Posting data 3-16 to 3-17
Pre-Block trigger 14-23
failure results 4-10
Pre-Commit trigger 14-24
failure results 4-10
Pre-Delete trigger 14-24
failure results 4-10
Pre-Field trigger 14-24
failure results 4-10
Pre-Form trigger 14-25
failure results 4-10
Pre-Insert trigger 14-25
failure results 4-10
Pre-Query trigger 14-25
failure results 4-10
Pre-Record trigger 14-26
failure results 4-10
Pre-Update trigger 14-26
failure results 4-10
Precompilers
See ORACLE Precompilers
Preference file
See User preference file
Preset mapping 26-8
Preset options 26-7
[Previous Block] 7-10
effect on context 6-7
[Previous Field] 7-11
[Previous Page] 7-11
[Previous Record] 7-12
PREVIOUS_BLOCK
flow chart 21-100
packaged procedure 16-26
PREVIOUS_FIELD
flow chart 21-101
packaged procedure 16-26
PREVIOUS_RECORD
flow chart 21-102
packaged procedure 16-27
Prim Key block characteristic 11-14
Primary key 11-20
Primary key 1-21
Primary Key
block characteristic 11-14
field characteristic 12-19
[Print] 7-12
flow chart 21-103
packaged procedure 16-27
Pro*Ada 19-3
Pro*C 19-3
Pro*COBOL 19-3
Pro*FORTRAN 19-3
Pro*Pascal 19-3
Pro*PL/1 19-3
Procedural information 2-1
Procedural logic statements in triggers 1-10
Procedures
documentation 27-5
See also Form-level procedures
name 15-4
spec 15-12
text 15-5
Process the Function Key 21-105
Processing
array processing 23-16
commit processing 3-16
constraints 22-1
event processing 21-1
executable statements 4-5
fundamental processing 3-1
inserts, updates, and deletes 3-18
packaged procedures 4-6
query processing 3-24
SQL statements 4-5
SQL*Forms commands F-31
SQL*Forms model 1-10
triggers within events 4-5
user exits calls 4-6
V2 function codes F-31
V2 triggers F-29
Prompt and Answer 21-106

Propagation of field characteristic changes 16-32
PUT 19-6
Put Cursor At 21-107

Q

-q switch for SQL*Forms (Run Form) 23-19
Qry Len field characteristic 12-21
QUERY
  SYSTEM.BLOCK_STATUS value 18-5
  SYSTEM.FORM_STATUS value 18-9
  SYSTEM.RECORD_STATUS value 18-12
Query Allowed field characteristic 12-20
Query Length field characteristic 12-21
Query processing 3-24
  counting query hits 3-30
  entering a query 3-24
  executing a query 3-25
  fetching records 3-26
  opening a query 3-25
  record selection criteria 3-26
  restrictions 3-29
Query processing restrictions 3-29
  Enter Query mode restrictions 3-29
  fetch processing restrictions 3-30
Query SELECT statement 3-27
Query Where statement 3-29
Query-only form 16-8
Quiet mode 23-19
  invoking 23-19, 26-3
Quiet Mode run option 26-3
Quitting SQL*Forms (Design) 2-8

R

-r switch
  SQL*Forms (Design) 23-13
  SQL*Forms (Run Form) 23-19
Range
  See Valid Range field characteristic
RDBMS
  See ORACLE RDBMS
Read this first i
Reading a key script file as input
  -r switch for SQL*Forms (Design) 23-13
  -r switch for SQL*Forms (Run Form) 23-19
Record selection criteria 3-26
  default ORDER BY clause 3-28
  default WHERE clause 3-28
  example record 3-28
  query SELECT statement 3-27
  Query Where statement 3-29
Record validation states 3-10
Records 1-19
  block's list 21-53
  buffer 23-16
  disabled 21-99
  example record 3-28
  fetching 3-26
  in a spread table 5-14
Records Buff block characteristic 11-15
Records Buffered block characteristic 11-15
Records Disp block characteristic 11-15
Records Displayed block characteristic 11-15
Records Lines block characteristic 11-12
Records Lines Per
  See Lines Per Record block characteristic
Redefining SQL*Forms function keys 14-8
REDISPIAY
  flow chart 21-110
  packaged procedure 16-27
Referencing objects 1-5, 2-3, 20-4
  blocks 20-4
  fields 20-5
  form-level procedures 20-6
  triggers 20-6
[Refresh] 7-12
Releasing locks 3-15
REMARKS column in the IAPXTB table 19-9
Remote login 23-2
Remove on Exit page characteristic 10-7
Rep status line item 5-16
Replace mode 5-16
REPLACE_MENU
  flow chart 21-111
  packaged procedure 16-27
Reports
  See Application documentation
  See SQL*ReportWriter

Required field characteristic 12-21
Reserved words 20-2
[Resize Field] 7-12
RESIZE_VIEW
  flow chart 21-112
  packaged procedure 16-28
RESTRICT
  LIST_VALUES parameter 16-22
Restricted packaged procedures 16-35
Restrictions
  See also Constraints
  PL/SQL 15-8
  query processing 3-29
  user exits 19-8
Results of trigger failures 4-7
[Retrace] 7-12
Retrace help feature 8-4
Reusing objects 1-5
Reverse Return Code trigger step characteristic F-7
Reverse Return trigger step characteristic F-7
[Right] 7-13
Right-integer data type 12-9
Right-justified scrolled field 12-28
Right-money data type 12-9
Right-number data type 12-10
RINT data type 12-9
RMONEY data type 12-9
RNUMBER data type 12-10
Rollbacks 3-17, 16-10
  caused by deadlocks 3-16
Routines for pop-up windows 19-10
Row 1-21
ROWID 3-23
  using in triggers 3-23
Rows
  See also Records
Run options 26-2
Run the Form 21-113
RUNFORM command 23-7
Running a form 2-6

S

-s switch
  SQL*Forms (Convert) 23-10
  SQL*Forms (Design) 23-13
  SQL*Forms (Generate) 23-15
  SQL*Forms (Run Form) 23-19
Savepoints 3-17
  issued when calling a form 3-18, 16-7 to 16-8
  issued when posting 3-17, 21-92
Saving a form 2-8
Scope
  See Trigger scope
Screen line
  selecting 2-9
Screen painter 1-19, 7-13
  cutting and pasting objects 2-9
  functions 7-18
  selecting objects 2-9
  tasks 2-9
  undoing actions 2-10
  using constant text 2-10
Screen Position field characteristic 12-22
screens 1-9
  help screens 8-2
  See also Interface elements
scroll bar
  arrow indicators 5-15
  elevators 5-15
  horizontal scroll bar 5-15
  vertical scroll bar 5-15
[Scroll Down] 7-13
[scroll Left] 7-14
Scroll region 5-13
  scrolling 5-13
  search function 5-13
[Scroll Right] 7-14
[Scroll up] 7-14
SCROLL_DOWN
  flow chart 21-114
  packaged procedure 16-29
SCROLL_UP
  flow chart 21-116
  packaged procedure 16-29
Scrolled field 12-28
  right-justified 12-28

Scrolling
scroll region 5-13
split screen scrolling D-4
spread table 5-15
[search] 7-15
search function
automatic search and complete l6-22
list of values 5-10
scroll region 5-13
Security
See Column Sec block characteristic
[Select] 7-15
Select Columns block characteristic 11-16
SELECT statements
INTO clause in V2 triggers F-9
translating into PL/SQL B-5
Selecting a field 2-9
Selecting a screen line 2-9
Selecting an area 2-9
Selecting items with a mouse D-5
Seq Num
block characteristic 11-16
field characteristic 12-22
Sequence Number
block characteristic 11-16
field characteristic 12-22
Sequence numbers
automatic generation 12-11
effect of pasting a field 2-10
sequencing
blocks 11-16
fields 12-22
Set Block Validation as Default run option 26-3
SET_FIELD
flow chart 21-118
packaged procedure 16-30
SET_INPUT_FOCUS
flow chart 21-119
packaged procedure 16-33
Setting the number of records buffered
-b switch for SQL*Forms (Run Form) 23-16
Setting the validation unit in block mode D-2
Severity levels for messages E-42
Share Cursors for Implicit SQL run option 26-4
Share Cursors for Trigger SQL run option 26-4
Show Keys 7-15
screen in SQL*Forms (Design) 7-2
Show Keys trigger characteristic 14-6
Show List of Values design option 26-6
SHOW_KEYS
flow chart 21-120
packaged procedure 16-33
SHOW_MENU
flow chart 21-121
packaged procedure 16-33
SHOW_PAGE
flow chart 21-122
packaged procedure 16-33
Single-record block 1-19,11-19
Single-user system 14-15
16-bit environment constraints 22-4
Spec 15-12
Specifying a mapping
-c switch for SQL*Forms (Design) 23-11
-c switch for SQL*Forms (Run Form) 23-17
Specifying default menu information 25-3
Split screen scrolling D-4
Spread table 5-14
context 5-15
scrolling 5-15
SQL 1-21
referencing fields F-9
SQL commands
in trigger steps F-9
in user exits 19-4
SQL statements
failure 15-8
in triggers 1-10
in user exits 19-4
list of values 12-17
processing 4-5
translating into PLSQL B-5
SQL trace facility 23-20, 26-5

SQL*Forms
basics 1-1
See also Components
documentation set 0-ii
processing model 1-10
terms 1-16
SQL*Forms (Convert) 1-15, 23-3
command line switches 23-8
CONVFORM command 23-4
database to INP conversion 23-3
INP to database conversion 23-3
INP to INP conversion 23-4
messages E-2
SQL*Forms (Design) 1-15, 23-5
base tables 24-4
command line switches 23-10
messages E-11
options 26-5
SQLFORMS command 23-5
SQL*Forms (Generate) 1-15, 23-6
command line switches 23-13
GENERATE command 23-6
messages E-30
SQL*Forms (Run Form) 1-15, 23-7
command line switches 23-15
messages E-42
options 26-2
RUNFORM command 23-7
SQL*Forms commands in trigger steps F-10
failure F-31
outcome F-31
processing F-31
translating into PL/SQL B-8
SQL*Menu
connecting to 25-3
controlling through packaged procedures 25-4
default application 25-5
passing data between forms and menus 25-4
specifying default information 25-3
SQL*Net 23-2
SQL*Plus
connecting to 25-7
invoking SQL*Forms 25-8
invoking through SQL*Forms 25-7
SQL*ReportWriter
connecting to 25-8
SQLCODE function 15-8
SQLERRM function 15-8
SQLFORMS command 23-5
Standard validation checks 3-11
Starting Menu Name form characteristic 9-5
Starting SQL*Forms (Design) 2-8
Static function keys 14-10
Statistics run option 26-5
Status checking 3-8
Status line 5-16
specifying context 6-1
suppressing in SQL*Forms (Design) 23-13
Step Num trigger step characteristic F-7
Step Number trigger step characteristic F-7
STOP alert label 5-3
Storage formats
See Form storage formats
Structure
book 0-iii
help system 8-2
Structured Query Language
See SQL
Sub-events 21-2, 1-131
Sub-menu 5-11
Succeed On Abort trigger step characteristic F-8
Success
packaged procedure outcome 4-6
SQL*Forms command outcome F-31
trigger outcome 4-7
user exit outcome 4-6
V2 function code outcome F-31
V2 trigger outcome F-30
Success Label trigger step characteristic F-8
Success trigger step characteristic F-8
Success/failure logic in V2 triggers F-32
Summary documentation 27-5
Summary of features A-1
Suppress Hints design option 26-6
Suppressing array processing
-a switch for SQL*Forms (Run Form) 23-16
Suppressing the banner and messages
-b switch for SQL*Forms (Generate) 23-14
-t switch for SQL*Forms (Generate) 23-15
Suppressing the program banner
-s switch for SQL*Forms (Convert) 23-10

Suppressing the status line
-s switch for SQL*Forms (Design) 23-13
Switches
See Command line switches
SYNCHRONIZE
flow chart 21-123
packaged procedure 16-34
Syntax for List of Values SQL Text
V2 12-18
V3 12-18
Syntax for user preference file
preset mapping 26-8
preset options 26-7
System variables 1-19, 18-3
displaying in debug mode 28-3
System-specific considerations D-1
SYSTEM.BLOCK_STATUS 18-5
SYSTEM.CURRENT_BLOCK 18-6
SYSTEM.CURRENT_FIELD 18-6
SYSTEM.CURRENT_FORM 18-7
SYSTEM.CURRENT_VALUE 18-7
SYSTEM.CURSOR_BLOCK 18-7
SYSTEM.CURSOR_FIELD 18-8
SYSTEM.CURSOR_RECORD 18-8
SYSTEM.CURSOR_VALUE 18-9
SYSTEM.FORM_STATUS 18-9
SYSTEM.LAST_QUERY 18-10
SYSTEM.LAST_RECORD 18-10
SYSTEM.MESSAGE_LEVEL 18-11
SYSTEM.RECORD_STATUS 18-12
SYSTEM.TRIGGER_BLOCK 18-12
SYSTEM.TRIGGER_FIELD 18-13
SYSTEM.TRIGGER_RECORD 18-13

T

-t switch
SQL*Forms (Generate) 23-15
SQL*Forms (Run Form) 23-20
Table block characteristic 11-8
Tables 1-21
SQL*Forms base tables 24-4
Target cursor position 3-3
Tasks
basic 2-2
screen painter 2-9
Terminal definitions
See Mappings
Terms 1-16
Testing techniques
application documentation 27-1
debug facility 28-1
edit keystroke facility 29-1
options 26-1
Text
See Constant text
$$TIME$$ 18-5
TIME data type 12-10
Title
form characteristic 9-5
page characteristic 10-7
TO_SAVEPOINT
CLEAR_FORM parameter 16-10
Tools
See Application development tools
Trace facility 23-20, 26-5
Trailing blank spaces 3-32
Transaction 1-21
Transactional triggers 14-8
Translating graphic features
-C switch for SQL*Forms (Convert) 23-9
Translating triggers into PL/SQL B-3
CASE statements B-8
#COPY B-8
DML statements B-7
#ERASE B-8
function code statements B-7
general issues B-3
#HOST B-9
SELECT statements B-5
SQL B-5
SQL*Forms commands B-8
user exits B-9
Trg: status line item 5-16
Trigger Definition
form 14-2
spread table 14-3
Trigger definition level 4-2
Trigger Name trigger characteristic 14-7

Trigger processing 4-5
executable statements 4-5
introduction to 1-13
trigger outcomes 4-7
within events 4-5
Trigger scope 1-14, 1-19, 4-3
Trigger step characteristics F-4
Trigger Step Definition
form F-2
spread table F-3
Trigger Step Text trigger step characteristic F-8
Trigger steps
characteristics F-4
commands F-8
constraints 22-2
outcomes F-30
Trigger Style trigger characteristic 14-7
Trigger Text trigger characteristic 14-7
Trigger trigger characteristic 14-7
Triggers 1-9, 1-19, 4-1, 14-1
block mode considerations D-3
characteristics 14-4
concepts 4-1
constraints 22-2
defining 14-2
definition level 4-2
documentation 27-6
failure results 4-7
master-detail relationship 11-22
navigational 14-8
outcome 4-7
packaged functions 17-1
referenced 20-6
scope 4-3
transactional 14-8
translating into PL/SQL B-3
trigger points 1-11
See also Trigger processing
types 14-8
See also V2 triggers
variables 18-1
TRUE
SYSTEM.LAST_RECORD value 18-10
Tuning applications
See Array processing
See Buffering records
See Cursor
See Statistics run option
TYPE column in the IAPXTB table 19-9
Typographic conventions 0-iv

U

[Undo] 7-15
Undo buffer 26-6
limiting 23-12, 26-6
Undoing screen painter functions 2-10
Unique characteristic 12-19
Unique identifier 11-14
Unique index 12-20
Unrestricted packaged procedures 16-34
[Up] 7-16
effect on context 6-7
UP
flow chart 21-124
packaged procedure 16-34
Update Allowed field characteristic 12-23
Update If Null field characteristic 12-24
Updates 3-19
UPDREC
locking rows 3-13
Upgrading to Version 3.0 B-1
Uppercase field characteristic 12-24
Upward compatibility B-1
Use Constraints block characteristic 11-17
Use Forms as Default design option 26-6
User exit calls in trigger steps F-11
User exits 19-1
call failure 4-6
call outcome 4-6
calling from a V2 trigger F-11
calling from a V3 trigger 19-10
creating 19-4
implementing 19-8
non-ORACLE 19-4
OCI (ORACLE Call Interface) 19-3
ORACLE Precompiler 19-3
passing values from SQL*Forms 19-5
passing values to SQL*Forms 19-6

performing ORACLE commands 19-7
performing SQL commands 19-4
pop-up window routines 19-10
processing calls 46
restrictions 19-8
translating into PL/SQL B-9
types 19-3
writing 19-4
User preference file 26-7
User-named trigger 14-27
failure results 4-10
USER_EXIT
flow chart 21-125
packaged procedure 16-34
UXWACT pop-up window routine 19-11
UXWADD pop-up window routine 19-11
UXWGXY pop-up window routine 19-12
UXWHID pop-up window routine 19-13
UXWMOV pop-up window routine 19-13
UXWPOP pop-up window routine 19-14
UXWPUC pop-up window routine 19-14
UXWPUS pop-up window routine 19-14
UXWREM pop-up window routine 19-15
UXWSHO pop-up window routine 19-15

V

-v switch for SQL*Forms (Convert) 23-10
V2 function codes F-12
ABTQRY F-13
arguments F-12
BELL F-13
CALL F-13
CALLINPUT F-13
CALLQRY F-14
CHRMODE F-14
CLRBLK F-14
CLRFLD F-14
CLRFRM F-15
CLRREC F-15
COMMIT F-15
concatenating arguments F-12
COPY F-15
CQUERY F-16
CREREC F-16
DEFAULT F-16
DELBACK F-16
DELCHR F-16
DELREC F-16
DERROR F-16
DKEYS F-17
DOWN F-17
DUPFLD F-17
DUPREC F-17
ENDSTEP F-17
ENDTRIG F-17
ENTER F-18
ENTQRY F-18
ERASE F-18
EXEQRY F-18
EXETRG F-19
EXIT F-19
failure conditions F-24, F-31
GOBLK F-19
GOFLD F-19
GOSTEP F-19
HELP F-19
HOST F-19
LISTVAL F-20
matching to V3 equivalents F-24
MENU F-20
MESSAGE F-20
MOVLEFT F-20
MOVRIGHT F-20
NEWFRM F-20
NOOP F-21
NULL F-21
NXTBLK F-21
NXTFLD F-21
NXTKEY F-21
NXTREC F-22
NXTSET F-22
outcome F-31
PAUSE F-22
PRINT F-22
processing F-31
PRVBLK F-22
PRVFLD F-23
PRVREC F-23
REDISP F-23
SCRDOWN F-23
SCRLEFT F-23

SCRRIGHT F-23
SCRUP F-24
SYNCHRONIZE F-24
UP F-24
UPDREC F-24
V2 list of values 12-17
syntax 12-18
V2 trigger processing F-29
failure F-30
outcome F-30
SQL*Forms commands F-31
success/failure logic F-32
trigger step outcomes F-30
V2 triggers F-1
CASE statements F-27
commands in trigger steps F-8
defining F-2
function codes F-12
processing F-29
success/failure logic F-32
V3 triggers versus V2 triggers 42
variables F-28
V3 function codes
See Packaged procedures
V3 list of values 12-17
syntax 12-18
V3 triggers
See Triggers
V3 triggers versus V2 triggers 4-2
Valid
field validation state 3-9
format mask status 3-11
record validation state 3-10
Valid Range field characteristic 12-24
Validate the Block 21-126
Validate the Field 21-127
Validate the Form 21-129
Validate the Record 21-130
Validation 3-6
field characteristic changes 16-30
See also Field validation
field validation states 3-8
format mask status 3-11
introduction to 1-13
record validation states 3-10
standard validation checks 3-11
status checking 3-8
validation unit 3-7
when validation occurs 3-8
Validation states
field validation states 3-8
record validation states 3-10
Validation unit 3-7
block mode considerations D-2
Validation Unit form characteristic 9-5
Values
See List of Values
Variable references F-28
See also NAME_IN packaged function
Variables 18-1
global variables 18-2
in trigger steps F-28
PL/SQL local variables 18-2
system variables 18-3
Version 2 triggers
See V2 triggers
Version 2.0/2.2/2.3 compatibility B-1
Version 3 triggers
See Triggers
Vertical scroll bar 5-15
Vertical Scroll Bar page characteristic 10-8
Vertical scrolling
scroll region 5-13
spread table 5-15
View Loc page characteristic 10-8
View of a page 13-5
View Page page characteristic 10-9
View Size page characteristic 10-9
Vw Lc page characteristic 10-8
Vw Pg page characteristic 10-9
Vw Sz page characteristic 10-9

W

-w switch
SQL*Forms (Design) 23-13
SQL*Forms (Run Form) 23-21
WHERE clause 11-9
syntax 11-10
Wildcards 5-10
Windows
See Pop-up windows

Word Wrap field characteristic 12-25
Working message type E-43
Working with objects 20-1
Wrapping text in fields 12-25
Writing a user exit 19-4

X

x setting
  Field Editor Location field characteristic 12-13
  List of Values Pos field characteristic 12-17
  Page Size page characteristic 10-6
  Screen Position field characteristic 12-22
  View Loc page characteristic 10-8
  View Page page characteristic 10-9
  View Size page characteristic 10-9

Y

Y setting
  Field Editor Location field characteristic 12-13
  List of Values Pos field characteristic 12-17
  Page Size page characteristic 10-6
  Screen Position field characteristic 12-22
  View Loc page characteristic 10-8
  View Page page characteristic 10-9
  View Size page characteristic 10-9

Z

Zeros in field values 3-32
Zoom connections 6-2
[Zoom In] 7-16
  effect on context 6-7
  zoom connections 6-2
[Zoom Out] 7-17
  zoom connections 6-2