

# Ingres 10.0

---

## Character-based Querying and Reporting Tools User Guide

INGRES

ING-10-QRT-02

This Documentation is for the end user's informational purposes only and may be subject to change or withdrawal by Ingres Corporation ("Ingres") at any time. This Documentation is the proprietary information of Ingres and is protected by the copyright laws of the United States and international treaties. It is not distributed under a GPL license. You may make printed or electronic copies of this Documentation provided that such copies are for your own internal use and all Ingres copyright notices and legends are affixed to each reproduced copy.

You may publish or distribute this document, in whole or in part, so long as the document remains unchanged and is disseminated with the applicable Ingres software. Any such publication or distribution must be in the same manner and medium as that used by Ingres, e.g., electronic download via website with the software or on a CD-ROM. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of Ingres.

To the extent permitted by applicable law, INGRES PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IN NO EVENT WILL INGRES BE LIABLE TO THE END USER OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USER OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF INGRES IS EXPRESSLY ADVISED OF SUCH LOSS OR DAMAGE.

The manufacturer of this Documentation is Ingres Corporation.

For government users, the Documentation is delivered with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227-7013 or applicable successor provisions.

Copyright © 2010 Ingres Corporation. All Rights Reserved.

Ingres, OpenROAD, and EDBC are registered trademarks of Ingres Corporation. All other trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

# Contents

---

<b>Chapter 1: Introduction</b>	<b>19</b>
In This Guide.....	19
Querying and Reporting Tools at a Glance.....	20
Audience .....	20
What You Need to Know.....	21
Special Considerations .....	21
Query Language Used in this Guide .....	22
System-specific Text in This Guide.....	22
Terminology Used in this Guide .....	22
Syntax Conventions Used in this Guide.....	23
 <b>Chapter 2: Fundamentals of Using Querying and Reporting Tools</b>	 <b>25</b>
Before Using the Querying and Reporting Tools .....	25
Requirements for Enabling Access to the Database .....	26
Setting of System Variables .....	26
Terminal Definition .....	27
Function Key Mapping .....	27
How to Start an Ingres Tool.....	28
Ingres Menu .....	28
Command Syntax to Start Ingres Menu or Tool.....	32
Command Syntax to Access Database on Remote Network Node .....	33
Command Syntax to Access a Non-Ingres Database .....	34
Command Syntax to Access a Distributed Database.....	34
Access to Database Tables .....	35
How to Select a Table You Want to Access.....	35
Synonym Use.....	36
Schemas for Owner Qualification.....	37
Frames and Forms .....	38
Forms .....	38
Fields on Forms .....	39
Find Entry Using First Letter .....	41
Menus .....	42
Menu Environment.....	42
Selecting a Menu Option .....	42
End and Quit Operations—Leave Submenus and Quit Ingres .....	44
Keys and Mouse Support.....	45
Keys for Menu Operations .....	45
Keys for Standard Functions.....	45

---

Cursor Movement and Editing Keys .....	46
On-Screen Help .....	47
Printing and Redrawing the Screen .....	47
Print the Screen .....	48
Refresh the Screen .....	48
Error Messages .....	49
Error Message Viewing .....	49
Naming and Name Use Conventions.....	49
Conventions for Schema and User Names .....	50
Conventions for Regular Identifiers .....	50
Delimited Identifiers .....	51

## **Chapter 3: Using the Tables Utility** **55**

Tables Utility .....	55
Tables, Synonyms, Views, and Indexes .....	56
Start the Tables Utility .....	58
Menu Options for the Table Catalog .....	59
Create a Table .....	60
Table Names .....	60
Column Specification—Column Name .....	61
Column Specification—Data Types.....	61
Column Specification—Key Numbers.....	62
Column Specification—Nulls .....	62
Column Specification—Defaults.....	63
How Unique Keys Are Set .....	65
Move Column Specifications .....	66
Clone Table Specifications with GetTableDef .....	66
Destroy Tables, Synonyms, Views, and Indexes.....	67
Get Information about Tables and Views.....	68
Fields for Examine a Table .....	69
How You Add or Delete Columns in an Existing Table.....	70

## **Chapter 4: Using QBF** **71**

Query-By-Forms (QBF) .....	71
Query Definition and Execution Phases of QBF.....	72
Query Definition Phase .....	72
Query Execution Phase.....	73
Ways to Start QBF .....	73
How You Start QBF from the Operating System.....	73
Start QBF from the Ingres Menu .....	74
Query Target Selection .....	75



---

Catalog Frames for Query Target Selection.....	75
Data Display Forms for Query Targets .....	77
Query Execution .....	79
How Query Results Are Displayed and Saved.....	81

## **Chapter 5: Working with QBF Operations** **83**

QBF Append Operation.....	83
Start the Append Operation.....	83
How to Use the Append Frame.....	84
Exit the Append Operation .....	87
QBF Retrieve Operation.....	87
Start the Retrieve Operation .....	88
Retrieve Frame .....	89
Search Qualifications.....	91
Order Operation—Sort Query Results.....	103
Go Operation—View Retrieved Records .....	105
Transaction Deadlock in Retrieve Mode .....	109
Exit the Retrieve Operation .....	110
QBF Update Operation .....	110
Start the Update Operation .....	110
How Data Is Modified .....	112
How Data Is Deleted .....	115
How Updates Are Saved .....	116
Exit the Update Operation.....	117

## **Chapter 6: Using JoinDefs in QBF** **119**

JoinDefs .....	119
JoinDef Rules .....	121
Join Columns with Coercible Data Types.....	122
Multiple Join Columns.....	123
Join Types.....	124
Master/Master JoinDefs .....	126
Master/Detail JoinDefs.....	127
Automatic Joins .....	128
Fields on a JoinDef Form .....	128
How You Create a JoinDef .....	129
Optional JoinDef Specifications .....	131
JoinDefs Catalog Frame .....	132
JoinDef Definition Frame .....	133
JoinDef Name .....	134
Role Column in JoinDef Definition.....	134

---

Table Name Column in JoinDef Definition .....	135
Owner Column in JoinDef Definition.....	135
Abbreviation Column in JoinDef Definition .....	135
Choosing Table-Field Format .....	135
How Columns Are Joined .....	136
JoinDef Specification Frame—Specify Other Components of the JoinDef .....	136
Single-Table JoinDefs .....	139
How You Build a Single-table JoinDef.....	140
Update and Delete Rules .....	141
Default Update and Delete Rules.....	141
Determine Update Rules .....	142
Determine Delete Rules .....	142
Exit the JoinDef Update and Delete Rules Frame .....	143
JoinDef Change Display.....	143
Delete Fields from JoinDef Displays .....	144
Exit the JoinDef Change Display Frame .....	145
Test JoinDefs.....	145
Save JoinDefs.....	145
Edit JoinDefs .....	146
Delete JoinDefs.....	147

## **Chapter 7: Using RBF 149**

Report-By-Forms (RBF).....	149
RBF Frames and Operations.....	150
Start RBF.....	151
Report Catalog Frame .....	152
Obtain Information About a Report Specification.....	153
MoreInfo about a Report Frame .....	154
RBF PopUp Frames.....	155
Preview Reports.....	156
Report Specifications .....	157
How to Produce a Report .....	157
Sources of Report Data .....	158
Sort Columns and Breaks .....	159
Date, Time, and Page Number .....	160
Report Styles .....	161
Tabular Report Style .....	162
Wrap Report Style .....	163
Block Report Style .....	163
Indented Report Style .....	164
Master/Detail Report Style .....	164
Labels Report Style.....	165

---

Report Structure.....	166
<b>Chapter 8: Working with RBF Report Specifications</b>	<b>169</b>
Create a Default Report Specification .....	169
Choose Columns on which to Base the Report.....	172
Choose a Report Style.....	173
RBF Report Layout Frame.....	175
How You Get to the Report Layout Frame.....	175
Report Components in the Report Layout Frame.....	175
Layout and Create Operations—Create New Report Components .....	181
Create Break Headers, Footers, and Other Report Sections .....	183
Create Trim .....	185
Create a Column .....	186
Aggregates .....	187
Create Additional Heading Lines.....	191
Create Blank Lines.....	192
Layout and Delete Operations—Delete Report Components .....	192
Delete Break Headers, Footers, and Other Report Sections .....	193
Delete Other Report Components.....	194
Edit Operation—Edit Report Components .....	194
Edit Trim and Headings .....	195
Edit Columns .....	195
Column Display Formats .....	196
Representation of Display Formats .....	196
Change Display Formats .....	197
Column Options .....	198
Sort Order .....	198
Runtime Data Selection .....	201
Column Break Options .....	203
Options for Showing a Change of Value.....	204
Use the Break Options Operation .....	205
Move Operation—Move Report Components .....	207
Move Trim, Columns, Aggregates, and Headings .....	208
Move the Report Margins .....	210
Report Options Frame.....	211
Page Length .....	212
Underlining .....	212
Page Header on First Page .....	213
Display of Null Values.....	213
Form Feeds .....	213
Date, Time, and Page Components.....	214
Obtain the Name of a Column .....	216

---

---

Undo Operation—Undo Edits .....	216
Save Operation—Save a Report Specification .....	217
Save Report Frame .....	218
Use the Save Operation .....	220
Save Report Pop-up .....	220
Archive Operation—Archive a Report Definition.....	221
Use the Archive Operation .....	221
Comment Blocks in Archived Reports.....	222
How to Copy Report Specifications.....	223
Delobj Command—Delete a Report Specification.....	224

## **Chapter 9: Producing a RBF Report 225**

How You Produce Reports.....	225
Report Destinations .....	225
Background Mode .....	226
Report Log .....	226
Specify Report Variables.....	227
Produce a Preview Report.....	228
Produce a Report from a Report Specification .....	229
Report Sent to a Screen .....	230
Send Reports from a Screen to a File.....	231
Send Reports from a Screen to a Printer .....	232
Send a Report Directly to a File .....	233
Send a Report Directly to a Printer .....	234
Report Command—Run Report Specification from Command Line.....	234
Passing Parameters on the Command Line .....	235
Send Reports to a Screen using Report Command .....	245
Report Command Examples .....	245

## **Chapter 10: Using Report-Writer 247**

What Is Report-Writer? .....	247
Report-Writer and RBF .....	248
Report Preparation .....	248
How You Obtain Data for a Report.....	249
Sorted Data in a Report.....	249
Breaks in a Report .....	250
Headers and Footers in a Report .....	251
Detail Section .....	251
How You Produce a Report .....	252
Ways to Create a Report Specification .....	252
Sreport Command—Save a Report Specification.....	253

---

Report Command—Execute a Report Specification .....	254
Considerations for the Finished Report .....	255
Types of Report Specification Statements .....	256
Report Setup Statements.....	257
Page Layout and Control Statements .....	258
Report Structure Statements.....	258
Column and Block Statements .....	259
Text Positioning Statements.....	260
Print Statements .....	261
Conditional and Assignment Statements .....	261
Format of Report Specification Statements .....	262
Statement and Parameter Delimiters .....	263
Schemas for Owner Qualification.....	264
Summary of Report-Writer Specifications .....	265
Sample Report.....	266
Report Setup and Format .....	268
How Variables Are Used in a Report.....	270
Reports that Use Multiple Tables .....	272
Sorts and Breaks in Reports .....	272
Pagination in Reports .....	274
Report Margins.....	275
Data Positioning, Formatting, and Printing.....	276
Use of Conditional and Assignment Statements.....	281
Summary Data Calculation and Printing .....	282
Automatic Determination of Default Settings .....	282

## **Chapter 11: Report-Writer Expressions and Formats 285**

Expressions in Report-Writer .....	285
Object Naming Conventions for ANSI/ISO Entry SQL-92 Compliant Databases .....	287
Recognition of Delimited Identifiers .....	288
How to Specify Delimited Identifiers .....	289
Precedence over String Constants .....	291
Reserved Words.....	292
Types of Data in Expressions .....	293
String Constants.....	293
Hexadecimal Strings .....	295
Numeric Constants .....	296
Date Constants .....	297
Columns.....	298
Variables.....	301
Special Report Variables .....	303
Aggregates .....	304

---

Operations .....	309
Arithmetic Operators .....	310
Comparison Operators .....	311
Logical Operators .....	313
Built-in Functions .....	313
Boolean Functions .....	315
Format Specifications .....	316
Default Formats .....	317
Blanking Format B .....	320
Character String Format C .....	321
Date Format D .....	324
Numeric Format E .....	330
Numeric Format F .....	331
Numeric Format G .....	333
Numeric Format I .....	334
Numeric Format N .....	335
Numeric Templates .....	336
Control Character Format Q0 .....	339
Character String Format T .....	341
Expressions and Formats Syntax Summary .....	342
Special Report Variables .....	342
Arithmetic Operators .....	342
SQL Conversion Functions .....	342
QUEL Conversion Functions .....	343
Numeric Functions .....	343
SQL and QUEL String Functions .....	343
Date Functions .....	344
Boolean Function .....	344
Aggregates .....	344
Formats .....	345

## Chapter 12: Report-Writer Statements 347

Report Setup Statements .....	347
.Break Statement—Specify Break Columns .....	347
.Cleanup Statement—Embed SQL Statements that Clean Up .....	349
Comments .....	351
.Data Statement—Specify Table for Report .....	352
.Declare Statement—Declare Variables .....	353
.Delimid Statement—Enable Recognition of Delimited Identifiers .....	357
.Include Statement—Import Files that Contain Report-Writer Code .....	359
.Longremark/.Endremark Statements—Mark a Block of Descriptive Text .....	361
.Name Statement—Assign a Name to a Report .....	362

---

.Output Statement—Specify File Name for a Report .....	363
.Query Statement—Indicate Start of a Query .....	364
.Query Statement for QUEL Users .....	369
.Setup Statement—Embed SQL Statements that Perform Setup.....	374
.Shortremark Statement—Specify Remark.....	377
.Sort Statement—Specify Sort Order of Reported Data .....	378
Page Layout and Control Statements .....	380
.Formfeeds/.Noformfeeds—Force or Suppress Formfeed Characters .....	380
.Leftmargin Statement—Set a Left Margin .....	382
.Need Statement—Keep Lines Together .....	384
.Newpage Statement—Insert Page Break .....	386
.Nofirstff Statement—Suppress Initial Formfeed .....	387
.Pagelength Statement—Set Page Length.....	388
.Pagewidth Statement—Set Page Width .....	390
.Rightmargin Statement—Set the Right Margin .....	392
Report Structure Statements .....	393
.Detail Statement—Begin Data Formatting Statements.....	393
.Footer Statement—Begin Formatting Statements for the Footer .....	395
.Header Statement—Begin Formatting Statements for the Header .....	397
Column and Block Statements .....	398
.Block/.Endblock Statements—Enable/Disable Block Mode .....	399
.Bottom Statement—Make Current Line the Bottom Line .....	401
.Format Statement—Set Default Printing Format .....	402
.Position Statement—Set Position and Width of Column Output .....	404
.Tformat Statement—Change Format of Column Output Temporarily.....	406
.Top Statement—Make Current Line the Top Line .....	408
.Width Statement—Set Output Width of a Column .....	409
.Within/.Endwithin Statements—Enable/Disable Column Formatting Mode.....	411
Text Positioning Statements .....	414
.Center Statement—Center the Text.....	415
.Left Statement—Left Justify the Text.....	418
.Lineend Statement—End a Line .....	420
.Linestart Statement—Print Next Text on New Line .....	421
.Newline Statement—Advance to a New Line .....	422
.Right Statement—Right Justify the Text .....	424
.Tab Statement—Specify Tab Position.....	427
Print Statements .....	428
.Nullstring Statement—Specify String for Null Value.....	429
.Print and .Println Statements—Print Text of Report.....	430
.Ulcharacter Statement—Set Underline Character .....	433
.Underline and .Nounderline Statements—Underline Text .....	435
Conditional and Assignment Statements .....	436

---

---

.If Statement—Specify Alternative Statements .....	436
.Let Statement—Assign Expression Value to a Variable.....	438
Statement Syntax Summary.....	438

## Chapter 13: Using VIFRED 443

What Is Visual Forms Editor (VIFRED)? .....	443
VIFRED Frames and Operations .....	444
Start VIFRED .....	446
Start VIFRED in Expert Mode .....	446
VIFRED Forms Catalog Frame .....	447
Create New and Duplicate Forms .....	449
Duplicate Forms .....	449
Create Blank Forms .....	450
Create Default Forms .....	451
Ways to Create Forms that Use Multiple Tables .....	453
Form Layout Frame .....	454
Alignment Guides .....	455
Layout Frame Menu Options .....	457
Form Attributes .....	458
Form Display Style .....	459
Fullscreen Forms .....	459
Pop-up Forms .....	460
Change the Style of a Form .....	461
Borders for Pop-up Forms .....	461
Form Size and Position.....	462
Set Size and Position Attributes .....	462
Move the Margins of a Form .....	465
VisuallyAdjust Operation—Visually Adjust a Form .....	468
Save a Form.....	472
Save Submenu.....	474
Save Changes Pop-up .....	474
Destroy Forms .....	475
Edit Existing Forms .....	475
Rename Forms .....	475
Compile Forms .....	476
Translate Compiled Form into Object Code .....	477
Print Forms .....	478
QBFFNames Operation .....	479
QBFFNames Catalog Frame.....	480
Assign Additional QBFFNames to Forms .....	482
Run QBF from VIFRED .....	482
Quit Operation—Exit VIFRED .....	483



---

## **Chapter 14: VIFRED Form Components** **485**

Parts of a Form.....	485
Fields on a Form.....	485
Operations on the Form Layout Frame .....	487
Create Operation .....	488
Create and Edit Trim .....	489
Create New Blank Lines on the Form .....	491
Boxes and Lines .....	492
Creation of Simple Fields .....	496
Table Fields .....	506
Create Duplicate Fields.....	515
Delete Form Components .....	516
Change the Tabbing Order of Fields on a Form .....	517
Move Operation—Move Components on a Form .....	518
Move a Single Component.....	519
Move a Group of Components at Once .....	522

## **Chapter 15: VIFRED Field Specifications** **523**

Field Attribute Specifications.....	523
Default Attributes .....	524
Set Attributes for a Field or Column.....	525
Attributes in the Set List .....	528
An Alternative to the BoxField Attribute .....	529
The Invisible Attribute .....	530
Input Masking .....	530
Required and Other Attributes.....	531
Internal Name of a Field .....	532
Data Type of a Field .....	532
Nullable Data Types .....	533
Color of a Field.....	533
Scrollable Field.....	534
Default Values for a Field .....	535
Specify a Validation Check.....	536
Comparison Operator Validation Checks .....	537
Validation Error Message .....	543
Derived Fields.....	544
How Forms are Used with Derived Fields .....	544
Specify a Derived Field .....	545
Guidelines for Specifying Derivation Formulas.....	546
Performance Recommendations for Derived Fields .....	549

---

## **Chapter 16: Interactive Query Language Terminal Monitor** **551**

What Is the Interactive Terminal Monitor? .....	551
Capabilities of the Interactive Terminal Monitor .....	553
Start the Interactive Terminal Monitor.....	554
Interactive Terminal Monitor Frame—Enter Query Language Statements .....	554
Menu Operations on the Terminal Monitor Frame.....	555
Enter Statements from a File.....	556
Write Statements to a File .....	557
Query Language Statement Execution.....	558
Print or File Output.....	561
How Error Messages Are Handled .....	561

## **Chapter 17: Working with Data Types and Data Display Formats** **563**

Data Types .....	563
Character Data Types.....	566
Date Data Type .....	569
Floating Point Data Type.....	570
Decimal Data Type .....	571
Integer Data Types .....	571
Money Data Type.....	571
Nulls.....	572
Data Display and Input Formats .....	572
Data Types and Display Formats.....	572
Display Format Syntax .....	574
Default Data Display Formats .....	577
How Character Data Is Displayed .....	579
How Numeric and Money Data Is Displayed .....	580
How Date Data Is Displayed .....	581
Format Templates.....	581
Numeric Templates .....	583
Input Masking with Numeric Templates.....	583
Special Numeric Template Characters .....	584
Numeric Template Examples .....	586
Date and Time Templates.....	587
Absolute Date and Time Templates .....	587
Time Interval Templates.....	592
String Input Templates .....	594
String Template Creation .....	594
How to Force Mandatory Entry.....	601
Examples of User-Defined Character Sets.....	602
Examples of String Templates .....	602

---

## **Appendix A: Defining Your Terminal** **605**

The Termcap File .....	605
How to Define Your Terminal .....	605
Define Your Terminal (Windows) .....	606
Define Your Terminal (UNIX) .....	607
Define Your Terminal (VMS) .....	608
Terminal Names .....	608

## **Appendix B: Defining Function and Control Keys** **615**

Why Key Mapping? .....	615
Key Mapping Overview (Windows Environment) .....	616
Termcap File (Windows) .....	616
FRS Mapping File (Windows) .....	619
Mapping File Example (Windows) .....	620
Standard FRS Mapping Files (Windows) .....	622
Application Mapping Files (Windows) .....	626
Key Mapping Overview (UNIX and VMS Environments) .....	626
Role of the Termcap File (UNIX and VMS) .....	627
Mapping Files (UNIX and VMS) .....	629
Mapping File Example (UNIX and VMS) .....	630
Levels of Mapping (UNIX and VMS) .....	632
Obtaining Information on Mappings .....	643
FRS Mapping Objects .....	643
FRS Commands .....	644
Menu Items .....	649
FRS Keys .....	651
Mapping File Syntax .....	654
Mapping Statements .....	655
How to Disable Statements .....	659
How to Use Comments .....	660
Mapping File Errors .....	660
Restrictions and Limitations (Windows Environment) .....	661
Troubleshooting Checklist (Windows) .....	662
Restrictions and Limitations (UNIX and VMS Environments) .....	663
Troubleshooting Checklist (UNIX and VMS) .....	665

## **Appendix C: Writing Termcap Descriptions** **667**

Why Modify the Termcap File? .....	667
Supplied Termcap File .....	668
How to Set the II_TERMCAP_FILE Variable .....	669
Format of a Termcap Description .....	670

---

How to Start Writing New Termcap Descriptions .....	674
Eleven Basic Commands .....	676
Cursor Motion Command .....	678
Commands for Advanced Features (Windows Environment) .....	681
Commands Used to Program Video Attributes (Windows) .....	682
Commands Used for Color (Windows) .....	683
Font Specifications (Windows) .....	685
Commands for Advanced Features (UNIX and VMS Environments) .....	686
Commands Used to Program Video Attributes (UNIX and VMS) .....	687
Commands Needed for Boxing Characters (UNIX and VMS) .....	688
Commands Needed for Function Keys (UNIX and VMS) .....	689
Commands Needed for Arrow Keys (UNIX and VMS) .....	691
Commands Used for Color (UNIX and VMS) .....	691
Commands to Specify Display Width (UNIX and VMS) .....	692
Command to Specify FRS Mapping File for Terminal (UNIX and VMS) .....	692
Commands to Optimize Cursor Movement (UNIX and VMS) .....	693
Commands for Special Situations .....	693
Commands from the UNIX Termcap File .....	694
Examples of Termcap Descriptions .....	695
DEC VT100 (All Inclusive) Termcap Description .....	695
DEC VT100 (Simple) Termcap Description .....	695
Envision 230 Termcap Description .....	696
Concept 100 Termcap Description .....	696
Datamedia 3045 Termcap Description .....	696

## **Appendix D: Data Types 697**

Data Types in SQL, OpenSQL, and QUEL .....	697
--	-----

## **Appendix E: Calling Ingres Tools from Embedded SQL and OpenSQL 699**

How Ingres Tools Are Called from Embedded SQL and OpenSQL .....	699
Call Statement—Call an Ingres Tool or Operating System .....	700
Ingres Tool Parameters .....	701
Abf Command Parameters .....	701
Ingmenu Command Parameters .....	701
Isql Command Parameters .....	701
Qbf Command Parameters .....	702
Rbf Command Parameters .....	703
Report Command Parameters .....	704
Sql Command Parameters .....	705
Sreport Command Parameters .....	706
System Command Parameters .....	706

---

Vifred Command Parameters .....	706
<b>Appendix F: Report-Writer Report Examples</b>	<b>707</b>
Overview of Report Examples .....	707
POPULATION Example .....	708
POP2 Example .....	718
QUEL User Notes for POPULATION and POP2 Examples .....	720
ACCOUNT Example.....	721
QUEL User Notes for ACCOUNT Example .....	729
DICTIONARY Example.....	729
DICT2 Example.....	735
QUEL User Notes for DICTIONARY an DICT2 Examples .....	736
LABEL Example.....	737
QUEL User Notes for LABEL Example .....	740
Report Creation Using Several Tables.....	740
How to Join Tables for a Report .....	741
How to Avoid Awkward Page Breaks .....	746
<b>Appendix G: Troubleshooting Report-Writer</b>	<b>753</b>
Parameter Substitution Troubleshooting .....	753
Query Troubleshooting.....	755
Comment Troubleshooting.....	756
Default Print Position Troubleshooting .....	757
Formfeed Troubleshooting .....	757
Performance Problem Troubleshooting .....	759
Query Performance .....	759
Performance and Conversion Function Usage .....	759
Performance and Memory Usage.....	760
<b>Index</b>	<b>761</b>



# Chapter 1: Introduction

---

This section contains the following topics:

[In This Guide](#) (see page 19)  
[Querying and Reporting Tools at a Glance](#) (see page 20)  
[Audience](#) (see page 20)  
[What You Need to Know](#) (see page 21)  
[Special Considerations](#) (see page 21)  
[Query Language Used in this Guide](#) (see page 22)  
[System-specific Text in This Guide](#) (see page 22)  
[Terminology Used in this Guide](#) (see page 22)  
[Syntax Conventions Used in this Guide](#) (see page 23)

## In This Guide

The *Character-based Querying and Reporting Tools User Guide* describes how to use the following forms-based tools:

- Tables Utility
- Query-By-Forms (QBF)
- Report-By-Forms (RBF)
- Visual Forms Editor (VIFRED)
- A terminal monitor (interactive query language facility)
- Report-Writer

The remaining forms-based tools, Application-By-Forms (ABF) and Vision, are discussed in the *Forms-based Application Development Tools User Guide*.

## Querying and Reporting Tools at a Glance

A *forms-based tool* is a user interface to the Ingres® database management system that uses forms and menus to provide database access and manipulation capabilities.

Following are brief descriptions of the forms-based tools discussed in this guide:

### **Tables Utility**

Forms-based utility that allows you to create, modify, or perform other operations on tables in the database.

### **QBF**

Forms-based query tool that allows you to look at, modify, or delete selective data in the database.

### **RBF**

Forms-based reporting tool that allows you to set up report specifications and run reports on the data in the database.

### **VIFRED**

Forms-creation tool that allows you to create and modify forms for use with QBF and customized applications.

### **A terminal monitor (interactive query language facility)**

Query tool that allows you to interact with the database by entering query language instructions on a blank form in the window.

### **Report-Writer**

Report language tool that allows you to create and run reports without writing an application. You can also use it to enhance reports you created with RBF.

## Audience

This guide is designed for anyone who wants to use a forms-based tool to:

- Create tables, views, and JoinDefs to organize data in the database.
- Look at, update, or create reports on the data in the database.
- Create individual components of a database application, such as a query form or report specification.



## What You Need to Know

To use this guide effectively, you must:

- Have a basic understanding of how Ingres or another database management system works, including the interaction between user interfaces and the database servers.
- Be familiar with the structure of Ingres databases and database tables, as well as the concept of local, remote, and distributed databases.
- Understand the differences between and purpose of Ingres tables, views, and JoinDefs.
- Know how to use the database query language (SQL or QUEL) used at your site to query Ingres databases.

If you are a beginner in database management or if you are unfamiliar with the preceding topics, read the *Database Administrator Guide*.

## Special Considerations

Before using this guide or any of the Ingres forms-based tools, you must be aware of the following:

- Enterprise Access Products Compatibility

If you are working through an Enterprise Access product, see your Enterprise Access product documentation for information about command syntax that can differ from that described in this guide. Refer also to the *OpenSQL Reference Guide* for query language details.

- Query Languages

Some of the text and examples in this guide pertain only to the SQL query language. For more information on using the forms-based tools with QUEL, see the appendix "Data Types" or the QUEL User Notes section in some chapters.

## Query Language Used in this Guide

The industry standard query language, SQL, is used as the standard query language throughout the body of this guide.

Ingres is compliant with ANSI/ISO Entry SQL-92. In addition, numerous vendor extensions are included. For details about the settings required to operate in compliance with ANSI/ISO Entry SQL-92, see the *SQL Reference Guide*.

For a description of the QUEL statements available, see the *QUEL Reference Guide*.

## System-specific Text in This Guide

Generally, Ingres operates the same on all systems. When necessary, however, this guide provides information specific to your operating system. For example:

**UNIX:** Information is specific to the UNIX environment.

**VMS:** Information is specific to VMS environment.

**Windows:** Information is specific to the Windows environment.

When necessary for clarity, the symbol ■ is used to indicate the end of system-specific text.

For sections that pertain to one system only, the system is indicated in the section title.

## Terminology Used in this Guide

The documentation observes the following distinction in terminology:

- A *command* is an operation that you execute at the operating system level.
- A *statement* is an operation that you embed within a program or execute interactively from a terminal monitor.

A statement can be written in Ingres 4GL, a host programming language (such as C), a database query language (SQL or QUEL), or in the Report-Writer language (for report specifications).

## Syntax Conventions Used in this Guide

This guide uses the following conventions to describe command and statement syntax:

Convention	Usage
Monospace	Indicates keywords, symbols, or punctuation that you must enter as shown.
<i>Italics</i>	Represent a variable name for which you must supply a value.
[ ] (brackets)	Indicate an optional item.
{ } (braces)	Indicate an optional item that you can repeat as many times as appropriate.
(vertical bar)	Separates items in a list and indicates that you must choose one item.



# Chapter 2: Fundamentals of Using Querying and Reporting Tools

---

This section contains the following topics:

[Before Using the Querying and Reporting Tools](#) (see page 25)

[How to Start an Ingres Tool](#) (see page 28)

[Access to Database Tables](#) (see page 35)

[Frames and Forms](#) (see page 38)

[Find Entry Using First Letter](#) (see page 41)

[Menus](#) (see page 42)

[Keys and Mouse Support](#) (see page 45)

[On-Screen Help](#) (see page 47)

[Printing and Redrawing the Screen](#) (see page 47)

[Error Messages](#) (see page 49)

[Naming and Name Use Conventions](#) (see page 49)

## Before Using the Querying and Reporting Tools

Before you can use the querying and reporting programs, you must do the following:

- Enable access to the database
- Set system variables
- Define your terminal
- Map function keys

## Requirements for Enabling Access to the Database

Ingres provides access to data stored locally on your computer, or if you are connected by a network, to data stored remotely on other nodes as well.

Your database always resides on a remote host computer. Access the database through a server installed on the remote host.

To access a database, you must have been given access to it by the database administrator. Additionally, your system must meet the following requirements to enable access to local and remote databases:

- For access to a local database, a server must be running on your local node.
- For access to a database on a remote node in your network, Ingres Net must be installed on both the local node and the remote node, and a server must be running on the remote node.
- For access to a non-Ingres database, the Enterprise Access product for that database must be installed. See the installation guide specific to that Enterprise Access product.
- For access to a distributed database, if your system supports it, Ingres Star must be installed.

Check with your system administrator for the status of access to local and remote databases on your system from your terminal. Check with the database administrator for a particular database to determine whether you have permission to access it.

## Setting of System Variables

The system administrator or the person installing your system sets environment variables or logicals that affect the operation of the program and the way data is handled. For example, one environment variable/logical sets the time zone; another establishes the type of currency symbol to use. These system variables are discussed in the *System Administrator Guide*.

## Terminal Definition

**UNIX and VMS:** You or the system administrator must define the type of terminal you are using. Normally, the system administrator sets up all such session options and defines your terminal to Ingres. However, if the terminal you are using has not been defined to Ingres, you must do so before you can access your database. ■

The environment variable/logical, TERM\_INGRES, allows you to define to the type of terminal you are using. Depending on your operating system, use one of the following commands:

### UNIX:

For the C shell:

```
setenv TERM_INGRES termname
```

For the Bourne shell:

```
TERM_INGRES=termname  
export TERM_INGRES ■
```

### VMS:

```
define TERM_INGRES termname
```

The *termname* is the name of the terminal description for your terminal in the termcap file, such as VT100i. For details, see the chapter "Defining Your Terminal." ■

## Function Key Mapping

Menu operations, cursor movement, and all other operations can be mapped to a particular function or control keys on your keyboard. Once you have specified a mapping, you can invoke the operation by pressing the specified key. For a discussion of key mapping, see the appendix "Defining Function and Control Keys."

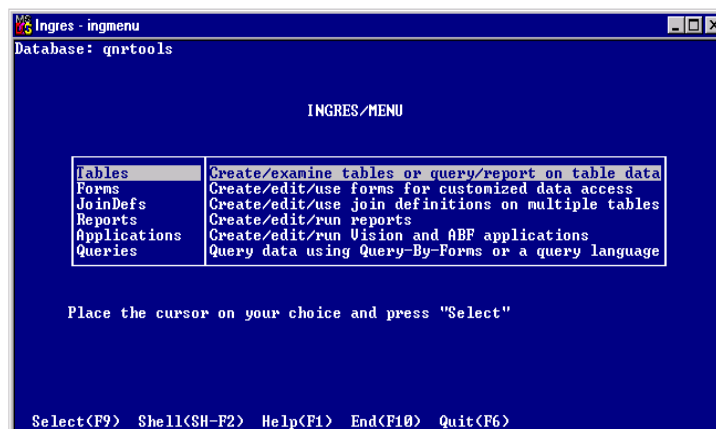
## How to Start an Ingres Tool

You can start any Ingres forms-based tool, such as Ingres QBF or the Visual-Forms-Editor, by doing one of the following:

- Starting Ingres Menu and then choosing an operation to start a particular Ingres tool
- Entering the appropriate startup command at the system prompt (or in a Run dialog in Windows)

### Ingres Menu

Ingres Menu is a visually oriented, forms-driven, menu interface that provides access to all Ingres forms-based tools and data.



Accessing an individual tool through the Ingres Menu gives you the ability to switch quickly from one forms-based tool to another, without having to go back to the window manager or operating system command line.

### Bypass Ingres Menu

You can bypass the Ingres Menu by entering a command to start a specific tool from the operating system command line. In this case, the Ingres Menu is not displayed and you are able to access only the particular tool you specified. For details, see Command Syntax to Start Ingres Menu or Tool (see page 32).

If you choose Ingres from an applications menu or load it from a script file, whether you see the Ingres Menu frame depends on how the menu or script has been configured.



## Access Ingres Menu

### To access the Ingres Menu

Issue the ingmenu command from your operating system command line, as follows:

```
ingmenu dbname|v_node::dbname [/server_type]
```

Details on these parameters can found in Command Syntax to Start Ingres Menu or Tool (see page 32).

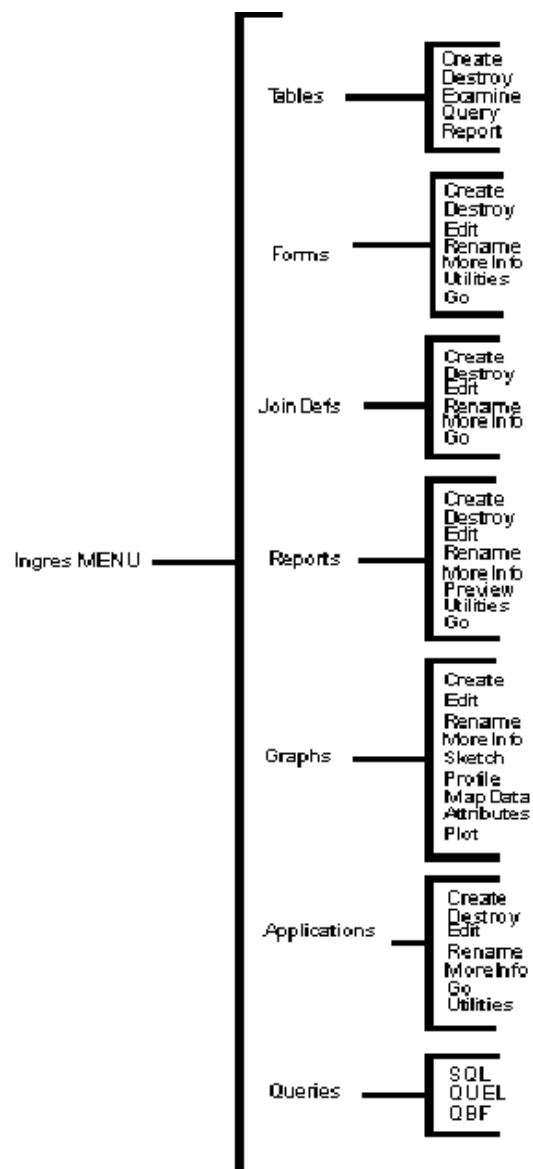
For a complete list of the flags and other parameters associated with the ingmenu command, see the chapter "Using System Commands for the Forms-based Tools."

## Options on the Ingres Menu

The Ingres Menu frame is divided into the following two parts:

- List of the forms-based tools you can access, as described in the table following this discussion
- Menu of operations that enable you to make your selection, get help, end the session, and so forth

The map in the following figure shows each forms-based tool that is accessible from the Ingres Menu main frame. The menu choices you see depend on your authorizations and on your environment.



The following table explains each of the main Ingres Menu choices:

**Tables**

Provides access to the Tables Utility for adding or manipulating tables in your database.

**Forms**

Provides access to tools for creating, editing, and using custom or default forms.

**JoinDefs**

Enables you to run a query on several tables at once using a JoinDef.

**Reports**

Enables you to run a report, or create and edit a report specification designed with RBF.

**Graphs**

Enables you to display or print existing graphs or create new ones. For more information, see the documentation for your graphics software.

**Applications**

Enables you to access the tools for generating applications: Vision and Applications-By-Forms. For additional information about the application generation tools, consult the *Forms-based Application Development Tools User Guide*.

**Queries**

Provides access to the interactive query languages through the Interactive Terminal Monitor and to QBF, a forms-based query system.

**To choose one of the Ingres tools listed on the Ingres Menu frame**

Click on the item with your mouse, or use the cursor control keys to highlight the item you want and choose the Select operation.

## Command Syntax to Start Ingres Menu or Tool

You can start the Ingres Menu or an Ingres forms-based tool by entering a startup command on the operating system command line. You must specify the database in which you want to work by including its name after the command to start the Ingres Menu or the Ingres tool and before any other non-flag parameters, as follows:

*command dbname*

For example, to start the Ingres Menu with access to the personnel database, enter the following command:

`ingmenu personnel`

Depending on the type and location of the database you want to access, you must sometimes specify the server type or location of the database in addition to its name.

The following table shows when you are required to specify the server type or location of the database:

Database Type and Location	Command Syntax	Example
Local Ingres database	<i>command dbname</i>	ingmenu admin
Remote Ingres database	<i>command v_node::dbname</i>	ingmenu ny::admin
Local non-Ingres database	<i>command dbname/server_type</i>	qbf sales/db2
Remote non-Ingres database	<i>command v_node::dbname/server_type</i>	qbf ohio::sales/db2
Local Ingres distributed database	<i>command dbname/d</i>	rbf corpdata/star
	<i>command dbname/star</i>	
Remote Ingres distributed database	<i>command v_node::dbname/d</i>	rbf ny::corpdata/d
	<i>command v_node::dbname/star</i>	

## Access Parameters on Startup Commands

The commands used to start the Ingres Menu or an Ingres forms-based tool have the following access parameters:

### ***command***

Any command used to invoke the Ingres Menu or an Ingres tool.

### ***v\_node***

Virtual node name of the remote node on which the database is located. The *v\_node* name implies the actual network node address and the network protocol. They are identified when the node name is defined with the netu utility. Two colons (::) must follow *v\_node*.

### ***dbname***

Name of the database containing the relevant data. It can be the name of a distributed database.

### ***server\_type***

Name of the type of server being accessed at the local or remote site, (the default is Ingres, which is the *server\_type* designated for the DBMS Server). Other server types are Star or one of the Enterprise Access products to a non-Ingres database. For specific names, see the database documentation.

## Command Syntax to Access Database on Remote Network Node

If the database you want to access is on a remote network node, then you must include the virtual node name (*v\_node*) of the remote node, followed by two colons (::), as follows:

*command v\_node::dbname*

The virtual node name is the name of the computer as registered with Net. For more information about Net, see the *Connectivity Guide*.

For example, to start Ingres Menu for the personnel database on the berkeley node, enter:

```
ingmenu berkeley::personnel
```

## Command Syntax to Access a Non-Ingres Database

If you want to access a non-Ingres database through an Enterprise Access product, you must include the server type for the database, preceded by a slash ( / ).

If the non-Ingres or distributed database is located on a remote node, you must also include the virtual node name.

```
command dbname|v_node::dbname/server_type
```

For example, to use QBF with an IMS database called sales located on a local node, enter:

```
qbf sales/ims
```

If the sales database resides on the remote node, hq, enter:

```
qbf hq::sales/ims
```

For more information about using tools through an Enterprise Access product and for associated server types, see the documentation for the specific Enterprise Access product.

## Command Syntax to Access a Distributed Database

For those systems that support Star Server, using a distributed database requires that you specify the server type. The server type for a distributed database is /d or /star.

To start the Ingres Menu with a distributed database called corpdata, enter either of the following commands:

```
ingmenu corpdata/d  
ingmenu corpdata/star
```

If the database is on the remote oakland node, enter:

```
ingmenu oakland::corpdata/d  
ingmenu oakland::corpdata/star
```

For more information about distributed databases, see the *Database Administrator Guide*.

## Access to Database Tables

You can access a table, view, or synonym in the forms-based tools only if one of the following is true:

- You are the object's owner
- You have been granted the proper permission to access the object

Access to a view automatically provides access to the columns in the underlying table that are defined in the view. The DBA or owner of a table can grant different types of access to a table, called *privileges*. These privileges are:

- Select
- Insert
- Delete
- Update
- Reference

## How to Select a Table You Want to Access

Select a table, view, or synonym you want to access by:

- Typing its name on the command line when entering an appropriate command
- Typing its name in a table name field on the current frame
- Highlighting it in a displayed list of choices and choosing the appropriate operation

Each table, view, or synonym, also has a schema name (see page 37) that associates the object with the user who owns it and distinguishes it from identically named objects in different schemata. Lists of choices in the forms-based tools contain all tables and views, including their schema names, to which you have access, whether owned by you, the DBA, or another user. These lists also contain all synonyms (see page 36) that you own or that are owned by the DBA, to which you have access.

## Synonym Use

A *synonym* is a definable label for a table name. You can use an existing synonym, whether yours or someone else's, to refer to a table when the actual table name is subject to frequent or occasional change. Synonyms also provide a shorter way to reference a long table name. Your entry is interpreted as the table name currently defined by the synonym you entered.

Access to the underlying table implies access to its synonyms. In lists of available choices, the Ingres forms-based tools display only the synonyms that you own and those owned by the DBA to which you have access. Synonyms owned by other users are not displayed.

To view a list of any other synonyms, you can enter the following SQL statement from the Interactive Terminal Monitor:

```
help synonym *
```

If you want to use a synonym that belongs to another user, include its schema name in the table name entry field.

Although you can use existing synonyms in your database, you cannot define them with an Ingres forms-based tool.



## Schemas for Owner Qualification

A *schema* is a collection of database objects, such as tables. Each table, view, and synonym belongs to a schema that is determined when the object is created. The schema name corresponds to the user who owns the object. The schema name helps distinguish between objects with identical names and different owners.

Schema names appear in the Owner column of table selection lists. You can also specify a schema name for a table, view, or synonym on the command line or in a table name entry field to qualify the object name with its owner. You use the following syntax to specify ownership:

*schema.objectname*

For example, to access the table named empinfo having a schema name of dave, you would enter the table name as:

dave.empinfo

The period (.) must immediately follow the schema name and precede the object name, with no intervening spaces. Both the schema name and the object name can be delimited identifiers.

If you do not use a schema name when referencing a table, view, or synonym, Ingres looks first for an object with a schema corresponding to the current user; then it looks for an object owned by the DBA to which you have access. Lastly, if the object name begins with ii, Ingres looks for a system catalog with that name. For more information on schemas, see the *Database Administrator Guide*.

## Frames and Forms

A *frame* is a form with associated program code that defines the available menu items. A frame contains the following parts:

- Window for displaying a form for data entry or data display
- Menu area for appropriate menu options
- Area for displaying program and error messages

Depending on the Ingres tool you are using and the menu operation you have chosen, a form appears in the frame's window with menu choices in the menu area that are appropriate for that particular task.

The menu (see page 42) displays the operations available on the form in the current frame.

The following figure illustrates a typical frame.

The screenshot shows a window titled 'Ingres - ingmenu' containing a form titled 'EMP Table'. The form has four input fields: 'Name: Applegate, Donald', 'Title: Analyst', 'Hourly Rate: \$ 51.00', and 'Manager: Wolfe, Neal'. Below these fields is a section labeled 'TASKS TABLE:' which contains a table with three columns: 'Project', 'Task', and 'Hours'. The table lists four tasks: 'Advertise' (8 hours), 'Graphic Design' (16 hours), 'TextProc Design' (10 hours), and 'TextProc Implement' (10 hours). At the bottom of the window, there is a menu bar with options: 'NextMaster(F9)', 'Query(SH-F2)', 'Help(F1)', and 'End(F10)'.

Project	Task	Hours
Advertise	Test	8
Graphic	Design	16
TextProc	Design	10
TextProc	Implement	10

## Forms

A *form* is the electronic equivalent of a paper form. The Ingres forms-based tools display ready-made *forms* in a frame. These forms can contain data fields in which you view or enter data.

Forms consist of *trim* and *fields*.

- *Trim* is independent of the data input and output functions of the form. It can be text that provides helpful information or box graphics that provide a decorative border.
- A *field* on a form displays data and/or accepts data entry. A field generally corresponds with a value from a table.

## Fields on Forms

Two types of fields appear on forms:

- Simple fields
- Table fields

A *simple field* displays data or accept data input one item at a time. The following figure displays a form with six simple fields.

The screenshot shows a window titled 'Ingres - ingmenu' with a form titled 'PROJECTS Table'. The form contains six simple fields arranged in two rows and three columns:

- Manager: Wolfe, Neal
- Description: Advertising Analysis
- Budget: \$ 9500.00
- Project: Advertise\_
- Dept: Sales
- Due Date: 02-mar-1999

At the bottom of the form, there is a navigation bar with the text: Next(F9) Query(SH-F2) Help(F1) End(F10).

A *table field*, as shown in the following figure, displays several rows and columns of data at a time.

The screenshot shows a window titled 'Ingres - ingmenu' with a form titled 'PROJECTS TABLE:'. The form displays a table field with the following data:

Manager	Project	Description	Dept	Budget
Wolfe, Neal	Advertise	Advertising Analysis	Sales	\$ 9500.00
Beveridge, Fern	Asset	Asset Management	Account	\$ 11700.00
Turner, Russell	EmployBen	Employee Benefits	Admin	\$ 20000.00
Jones, Betty	Expense	Expense Account System	Sales	\$ 12500.00
Parsons, Carol	Graphic	Graphic Design	Commun	\$ 18000.00
Fielding, Wallace	Portfolio	Portfolio Analysis	Account	\$ 11200.00
Jones, Betty	SalesFor	Sales Forecasting	Sales	\$ 9900.00
Wolfe, Neal	TextProc	Text Processing	Admin	\$ 14000.00

At the bottom of the form, there is a navigation bar with the text: Query(SH-F1) Help(F1) End(F10).

You can include simple fields and a table field on the same form. A table field can have more rows than can be displayed at one time. Use the arrow keys or the Scrollup and Scrolldown keys to display more rows.

If the form itself is larger than the window, tabbing to a field or column automatically brings that portion of the form into view. You can also use the Scrollleft and Scrollright keys to move around a large form, or the Scrollup and Scrolldown keys to display a simple field that is beyond the top or bottom of your window.

Each field on a form consists of a data window, attributes, internal name, and an optional title:

- The data window is the area in which data is displayed or entered.
- Attributes affect the way data is displayed in a field and the way a user can work with a form. Some attributes control the *look* of the data. For example, you can use video highlight on the data in the data window. Other attributes control such things as the error message a user sees if the wrong value is entered into a field.
- The *internal name* identifies the field and can be linked to the column in the database table from which the field receives its data, or to which it writes data. An application accesses the field by using the internal name. The internal name is not visible on the form. For more information on internal names, see the chapter "VIFRED Form Components."
- The *title* describes the data that appears in a field. The title is optional and need not correspond to the name of a column in a database table.

The following figure illustrates the title and data window of a simple field.

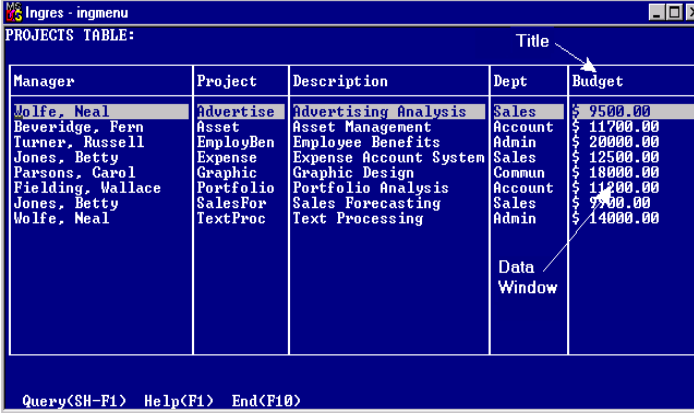
The screenshot shows a window titled 'Ingres - ingmenu' containing a form titled 'PROJECTS Table'. The form has the following fields:

Manager: Wolfe, Neal	Project: Advertise_
Description: Advertising Analysis	Dept: Sales
Budget: \$ 9500.00	Due Date: 02-mar-1999

Below the form, two arrows point upwards to the 'Description' and 'Budget' fields. The arrow pointing to the 'Description' field is labeled 'Title', and the arrow pointing to the 'Budget' field is labeled 'Data Window'.

At the bottom of the window, there is a menu bar with the following options: Next(F9), Query(SH-F2), Help(F1), and End(F10).

The following figure identifies the same components on a table field.



Ingres - ingmenu  
PROJECTS TABLE:

Manager	Project	Description	Dept	Budget
Wolfe, Neal	Advertise	Advertising Analysis	Sales	\$ 9500.00
Beveridge, Fern	Asset	Asset Management	Account	11700.00
Turner, Russell	EmployBen	Employee Benefits	Admin	20000.00
Jones, Betty	Expense	Expense Account System	Sales	12500.00
Parsons, Carol	Graphic	Graphic Design	Commun	18000.00
Fielding, Wallace	Portfolio	Portfolio Analysis	Account	11200.00
Jones, Betty	SalesFor	Sales Forecasting	Sales	9500.00
Wolfe, Neal	TextProc	Text Processing	Admin	14000.00

Query<SH-F1> Help<F1> End<F10>

For more information on forms, see the chapters "Using VIFRED," "VIFRED Form Components" and "VIFRED Field Specifications."

## Find Entry Using First Letter

To find a specific entry in a result list in any character-based tool, you can use the First Letter Find function.

### To locate an entry in the result list

Type the first letter of the item you are searching for.

The cursor jumps to the next item in the list that begins with that letter.

**Note:** To search the entire list using this function, you must scroll to the end of the result set once; otherwise, only a partial list (the buffered rows) is searched.

## Menus

Menu operations are displayed at the bottom of each frame. To access the menu using your keyboard, press the Menu key (the key that is mapped to the menu operation in your key mapping file). The cursor moves to the first menu operation.

The key on your keyboard that acts as the Menu key depends on your keyboard type and the individual key mapping you have chosen. On a VT100i keyboard, it is usually mapped to the PF1 key by default. In some cases, it can be mapped to the Escape or F1 key. On a PC keyboard, it is usually mapped to the Escape or F2 key by default.

Your menu's appearance and the method used to display additional menu options depend on your particular environment and whether you are using a mouse or your keyboard.

### Menu Environment

The menu appears at the bottom of the window and looks similar to this one:

```
Create(1) Destroy(2) Examine(3) Query(4) Report(5) >
```

Some menus are longer than the width of the frame. The presence of additional menu items is indicated by either a greater than sign (>) at the right end of the menu, a less than sign (<) at the left end, or both.

If your system displays classic menus and you are using a mouse, click the right mouse button (or right-click) from anywhere in the frame to access the menu; then right-click the right repeatedly to cycle through the menu options.

If you are using your keyboard, press the Menu key to access the menu; then press the Menu key repeatedly to cycle through the menu options.

To return to the forms window, click on the window or press the Enter key.

### Selecting a Menu Option

Select a menu option using one of these methods:


- Click on the operation with the mouse, if installed.
- Press a function key associated with the operation.
- Access the menu with the Menu key and type the capitalized or first few letters of the operation.

## Selecting a Menu Option with a Mouse

If you are using a mouse, you can select a menu operation by moving the mouse pointer to the operation and clicking on it with the appropriate mouse button.

## Selecting a Menu Option Using a Function Key

Generally, a menu operation has a function key (or key combination) mapped to it. In most environments, the associated key appears in parentheses following the operation on the menu line.

**Windows:** In a Windows environment, the function key associated with the currently highlighted menu item appears on the bottom line of your window. You can also display a definition list of Ingres function keys by pressing F1 or clicking on the Help operation with the mouse and then choosing the Keys operation. 

You can choose the menu operation by pressing its associated function key, regardless of the cursor location. This invokes the operation immediately. You do not have to press Return. For example, if Quit(PF4) appears on the menu line (or if PF4 is associated with Quit in a Help window), you press the PF4 key to invoke the Quit operation. If ^F is shown as the associated key, hold down the Control key and press the F key.


## Selecting a Menu Option by Typing or Highlighting the Operation Name

If your keyboard mapping does not correspond to the key designations on the menu, or if for some other reason the function keys do not work, you can use the following alternative keyboard procedure.

### To select an operation from the menu by typing or highlighting the operation name

1. Move the cursor to the menu by pressing the Menu key.
2. Do one of the following:

Type as many letters as necessary to make the operation name unique. For example, if both the End and Edit operations appear on a menu, you must type **ed** to use the Edit operation or **en** to use the End operation.

**Windows:** In a Windows environment, press the arrow keys or Space Bar to move the Input focus bar to your selection, or type the capitalized letter of the desired operation. For example, type C for the Create operation. 

3. Press Return.

## Typical Menu Operations

### **Blank**

Clears the window of all entries

### **Cancel**

Returns to a previous window without saving changes entered in current window

### **End**

Ends operations in current window and returns to previous window

### **Go**

Follows the specified request

### **Help**

Displays Help windows relevant to the current action

### **Insert**

Puts a blank line at the cursor location

### **ListChoices**

Lists the available choices for the selected field

### **Undo**

Undoes or cancels the previous operation

### **Quit**

Exits module, either to the Ingres Menu or to the operating system

## End and Quit Operations—Leave Submenus and Quit Ingres

When you select an operation from a menu, you are often presented with a submenu of operations on a pop-up form. When you select an operation from that submenu, another pop-up form with a further set of operations can be presented.

When you leave a submenu using the End operation, you always return to the next higher-level menu.

When you leave a submenu using the Quit operation, you return to the Ingres Menu or to the operating system or Windows program (depending on how you entered the application or Ingres forms-based tool). When you quit from the main menu, you quit Ingres and return to the operating system or Windows program.



## Keys and Mouse Support

Through the use of key mapping, you can attain a high degree of flexibility in your keyboard interaction with Ingres. The way the different keys on your keyboard function depends on your keyboard and the specific changes you have made to your key-mapping file. Application developers can also re-map function keys for use in their applications. For more information on key mapping, see the appendix "Defining Function and Control Keys."

You can view the current mappings for your keyboard by choosing the Keys operation on the Help menu.

In addition to using the keyboard keys to move around forms and menus, you can also use a mouse.

### Keys for Menu Operations

Menu choices on frames are generally associated with keyboard keys or key combinations, such as Ctrl-E or F1 that you can press to activate the menu choice. These key assignments assume you are using a keyboard that corresponds to your key-mapping file. The key assignment generally appears in parentheses after each menu option on the menu line. If not displayed on the menu line, you can use the Help operation to display current key assignments.

### Keys for Standard Functions

By default, certain functions—such as access the menu, get help, move cursor to next field, or move cursor to previous field—have been assigned to specific keys that operate consistently on any frame. Your key mapping file determines the exact key you press. The following table lists some of the default function keys that are common to all of the Ingres forms-based tools:

<b>Function or Menu Operation</b>	<b>MS Windows Keyboard</b>	<b>VT100i Keyboard</b>
Go to Menu	Esc or F2	PF1
Get help	F1	PF2 (or sometimes Escape or F1)
Move cursor to next field and clear it of data	Enter	Return
Move cursor to next field and retain its data	Tab	Tab

Function or Menu Operation	MS Windows Keyboard	VT100i Keyboard
Move cursor to previous field for editing	Shift+Tab	Control-P
End operation	F2 or F10	PF3
Quit operation	F6	PF4

The Tab, Return, and Enter keys move the cursor to the next field, but in different ways:

- Tab moves the cursor to the next field or next column in a table field, *without clearing the contents of the field*. If the cursor is in the last field, Tab moves the cursor to the first field of the same form.
- Return (or Enter) moves the cursor to the next field *and clears data to the end of that field at the same time*, unless it is a read-only form or table field. In a table field, this key moves the cursor to the next column. If the cursor is in the last column, it moves the cursor to the first column of the next row.

If you do not want the Return (or Enter) key to clear the data to the end of the field, you can map this key so that it works like the Tab key.

This guide refers to the Enter key as the Return key.

## Cursor Movement and Editing Keys

Keys that facilitate moving the cursor within forms are both system and keyboard-dependent. The keys that normally control cursor movement on your keyboard can usually be used to control scrolling and other movements of the cursor in a form.

You can edit text or work in either overstrike or insert mode. In overstrike mode, which is the default, each character you type replaces the existing character beneath it. In insert mode, characters are moved to the right as you enter new characters. The key that controls your editing mode is called the Mode key, and acts as a toggle that enables you to switch modes.

## On-Screen Help

Context-sensitive help is provided online. This means that the assistance is based on the current task you are attempting to accomplish and the current field you are attempting to complete. Sometimes help is provided on several screens.

You can obtain help by choosing the Help operation from the menu, or you can press the Help key to get help at any time. The Help key is designated in parentheses following the Help operation—if your system displays key assignments on the menu line. The key to which this function is mapped can vary. On a VT100i keyboard, it is usually mapped to the PF2 key by default. On a PC keyboard, it is usually mapped to the F1 key by default. You can also determine your Help key by choosing the Help operation from your menu using another method, and then choosing the Keys operation on the Help submenu.

Help windows for Ingres contain the menu choices shown in the following table:

**Keys**

Describes the function, control, and arrow keys and their current definition.

**SubTopics**

Displays a pop-up with a list of subtopics. For an explanation of a particular subtopic, select it from the pop-up. This option appears only when subtopics are available.

**Help**

Displays the type of Help available.

**End**

Exits from any Help window to the previous window.

To move through the Help windows, use the cursor movement keys specific to your keyboard. If available for your terminal type, you can also use the key mapped to the string-search (FRS Find) operation to search for a specified text string within a Help window. Use the Help Keys operation to determine which key is mapped to the string-search operation.

## Printing and Redrawing the Screen

You can print or redraw the currently displayed screen.

## Print the Screen

Depending on how you have defined your terminal or your keyboard keys, you can print the contents of the currently displayed screen by pressing the designated printscreen key.

You can also set the `II_PRINTSCREEN_FILE` environment variable/logical to a file name that automatically writes the results of the printscreen function, or to printer to send the screen contents to the line printer. For more information on setting environment variables or logicals, see the *System Administrator Guide*.

### To print the current screen

1. Press the key assigned to that function.

On a VT100, press Control-G; on a VT220, press F8; on a PC, press Prtsc. The key assignment can vary from keyboard to keyboard.

If you have set `II_PRINTSCREEN_FILE` to a specific file or to printer, the screen contents are sent to the file or printer. Otherwise, a prompt appears:

Enter file name:

2. Do one of the following:
  - Type the name of a file and press Return to transfer an image of the current form, including all displayed data values, to the specified file. The entire form is included, even though it can be longer and wider than your window.
  - Type printer as the file name and press Return to send the image to the line printer.

## Refresh the Screen

You can also refresh, or *redraw*, the current screen, including any data you have entered into its field. This is useful if you receive messages on the screen or if disruptions in communication with the computer occur. The redrawing function is assigned by default to Control+W, regardless of your terminal or keyboard.

### To refresh the current screen

Press Control+W.

## Error Messages

Context-sensitive error messages are provided in pop-up windows that appear at the bottom of your screen. The error message you receive indicates the error type, the error code, and in some cases offers an explanation.

### Error Message Viewing

For messages with explanations, the first line is displayed, along with a prompt offering you a choice of either End or More if the message contains more than one line.

To exit the message without reading the explanation, press the End key or click on the word End with your mouse.

To read the explanation, press the More key or click on the word More. After reading the explanation, press Return or Enter to return to your work in progress, or click anywhere inside the message box to dismiss the message.

For messages without an explanation, a single-line message is displayed with a prompt that tells you to press Return or Enter after reading the message. This removes the message and returns you to your work in progress. Alternatively, if you are using a mouse, click anywhere inside the message box to remove it and return to your work.

## Naming and Name Use Conventions

You can define a wide range of objects, from databases to reports and join definitions. You must follow certain conventions when naming or referencing objects. *Identifiers* are names for objects, such as table names, JoinDef names, column names, QBFNames, and so forth.

There are two types of identifiers:

- Regular identifiers
- Delimited identifiers

Regular identifiers are user and object names that follow standard naming conventions for that database. Delimited identifiers are delimited by double quotes (") and can contain additional characters and words that are disallowed in regular identifiers.

You can use only regular identifiers to name Ingres tools objects, such as JoinDefs, reports, forms, and QBFNames, which you create with QBF, RBF, and VIFRED. You can use either type of identifier to name Ingres database objects, which you create in the Tables Utility or with query language statements.

## Conventions for Schema and User Names

Schema names for objects can be specified as either regular or delimited identifiers, depending on the schema name's compliance with conventions. If a schema name contains characters unacceptable in a regular identifier, such as in Da Vinci or O'Neil, then it must be specified as a delimited identifier in double quotes.

Default schema names are created based on the user ID associated with the database connection. The user ID can be stored in uppercase, lowercase, or mixed case, depending on choices made by the DBA when the database was created. If a user ID is stored in mixed case, any schema name based on that user ID can also be stored in mixed case and must be specified as a delimited identifier, in double quotes, when qualifying an object with an owner name.

Also, you must specify any mixed-case user ID as a delimited identifier when impersonating that user with the `-u` flag on the command line.

For more information on schema names or on specifying case conventions for user identifiers in a particular database, see the *Database Administrator Guide*.

## Conventions for Regular Identifiers

The following table lists the conventions you must follow when using a regular identifier to specify the name for any object:

Quality	Convention	For ANSI/ISO Entry SQL-92 Compliant Databases
Size	32 bytes maximum; 256 bytes maximum for many objects. For details, see Object Naming Rules in the <i>SQL Reference Guide</i> .	No more than 128 characters
First character	Must be alphabetic (a-z) or the underscore ( <code>_</code> )	Must be alphabetic (a-z)
Other allowable characters	0-9, #, @, and \$ allowed <i>after</i> the first character	Only alphabetic, numeric, or underscore characters allowed
Case sensitivity	Case insensitive	Case insensitive

Examples of valid regular identifiers for object names are:

- new\_york
- march98
- Quinn (equivalent to quinn or QUINN)
- \_geneva\$ (not allowed in databases that comply with ANSI/ISO Entry SQL92 standards)

In addition to the restrictions described above, you must not use reserved words as regular identifiers. For a full list of standard or embedded SQL, OpenSQL, and QUEL reserved words, see your query language reference guide or the *Forms-based Application Development Tools User Guide*.

## Delimited Identifiers

*Delimited identifiers* are database object names that are identical to reserved words or that contain spaces or non-alphanumeric characters that are disallowed in a regular identifier. If the database was created as case sensitive, you can also use delimited identifiers to distinguish among identical names with different case (for example, SALES as distinct from Sales). For more information on allowable characters in delimited identifiers, see the *SQL Reference Guide*.

You can use delimited identifiers for names of all database objects, such as:

- Table names
- View names
- Correlation names
- Column names in tables
- Schema and user names

You are not allowed to use delimited identifiers for Ingres tools objects such as JoinDefs, reports, forms, and QBFNames.

## Use of Wild Card Characters

In some situations, you can use the SQL wild cards, underscore (\_), asterisk (\*), or brackets ([ ]), within delimited identifiers in the forms-based tools. To enter a wild card character as an explicit character, you must dereference it by preceding it with a backslash (\). For more information, see the chapter(s) on the specific forms-based tool or the command descriptions.

## Case Sensitivity in Delimited Identifiers

When specifying delimited identifiers, follow the rules for case as defined for your database. In standard Ingres databases, delimited identifiers can be either case sensitive or case insensitive, as determined by the DBA when creating the database. By default, standard Ingres databases are case insensitive.

In databases compliant with ANSI/ISO Entry SQL-92 standards, delimited identifiers are case sensitive. The following is an example of a case-sensitive delimited identifier:

```
"Sales for March"
```

For more information on setting case for identifiers in Ingres databases, see the *Database Administrator Guide*.

## Use of Delimited Identifier in Forms

You must always enclose delimited identifiers within double quotes ("), as shown in the following examples:

- As a table, view, or synonym:  

```
"dave's table"
```
- As a column name:  

```
"Stocks & Bonds"
```
- In Help statement queries in the query language Interactive Terminal Monitor; for example:  

```
help table "my table"
```
- In VIFRED field validations; for example:  

```
field_name in "table one"."column two"
```
- As a correlation name and column name in *correlation\_name.column\_name* constructs:  

```
select "t-1"."col 1" as col1, "t-2"."col 2" as  
      col2  
from table_one "t-1", table_two "t-2"
```



- When using owner qualification, for either or both the schema name and/or object name, as follows:

```
"schema 1".table2  
dave."Dave's table"  
"schema 1"."view table1 & table2"  
field_name in "schema 1"."table one"."column two"
```

- On the command line, as a username for the -u flag, groupid parameter for the -G flag, or database object in a command parameter.

```
-u"user 2"  
-G"tech sup"
```

Your operating system can require additional delimiting and dereferencing quotes for these parameters and for delimited identifiers on the command line. For details, see the *System Administrator Guide*.

Delimited identifiers are displayed in catalogs and other lists of available choices without their surrounding double quotes, except when displayed for update. You can choose a delimited identifier from one of these lists as you do any other identifier—by moving the cursor to the item and choosing the appropriate operation.

Delimiting quotes and, if present, de-referenced embedded double quotes are shown in pop-up forms and entry fields containing delimited identifiers displayed for update. If you edit a delimited identifier or enter one into a field by manually typing it, you must also include the delimiting double quotes (") and dereference any embedded double quotes by preceding them with another double quote("):

```
""""expert"" witness"
```

Do not dereference single quotes or apostrophes within delimited identifiers:

```
"dave's table"
```

For information about specifying delimiting and dereferencing quotes in delimited identifiers on the command line, see the *System Administrator Guide*.



# Chapter 3: Using the Tables Utility

This section contains the following topics:

[Tables Utility](#) (see page 55)

[Tables, Synonyms, Views, and Indexes](#) (see page 56)

[Start the Tables Utility](#) (see page 58)

[Create a Table](#) (see page 60)

[Destroy Tables, Synonyms, Views, and Indexes](#) (see page 67)

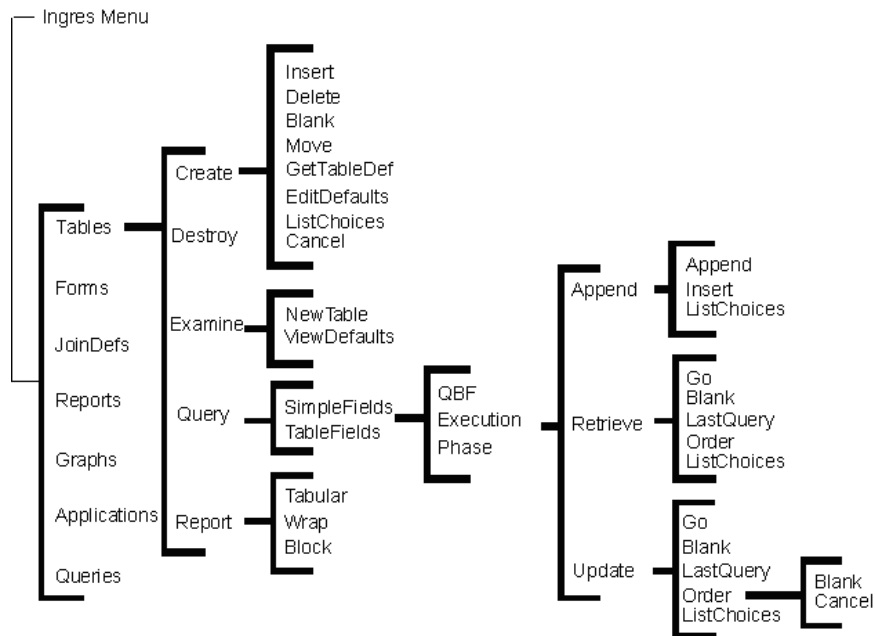
[Get Information about Tables and Views](#) (see page 68)

[How You Add or Delete Columns in an Existing Table](#) (see page 70)

## Tables Utility

The Tables Utility lets you create or destroy tables in a database and examine their structure. It also provides access to Ingres forms-based tools that allow you to run queries and reports on data contained in the tables. You can perform the same operations offered in the Tables Utility by using database query language commands, which are discussed in your query language reference guide.

The tables operation is illustrated in the following figure:



## Tables, Synonyms, Views, and Indexes

Before using the Tables Utility to access tables, synonyms, views, or indexes, you must understand how each is used and what operations you can perform on them in the Tables Utility:

### Tables

Tables are used to store all the data in your database. Each row in a table holds an individual record that contains one or more related data items. Each column contains one particular type of data. You can use the Tables Utility to create and delete tables, to look at the structure of a table, and to retrieve or modify the data contained in a table.

Except for rows containing large objects (long varchar, byte, byte varying, and long byte data types), all rows in a given table are the same width, measured in bytes, with no row exceeding the lesser of the maximum configured row size and 32,000. The number of rows is limited only by disk space.

Each column is assigned a data type (see page 61) to indicate the type of data to be stored and the length of the data (or width of the column). For example, a column with a data type of varchar (25) can hold 25 text (ASCII) characters. A maximum of 1024 columns is allowed in a table. Each column has a name that uniquely identifies the column within the table.

You can access any table for which you have been granted the proper permissions. For more information on permissions, see the *Database Administrator Guide*.

### Synonyms

A synonym is a redefinable label for a table, which you can use as an alternative to the actual table name. You can access a table with an existing synonym, but you cannot create synonyms in the Tables Utility. The list of tables from which you can choose contains only synonyms you or the DBA own, freely intermixed with table names. For more information on synonyms, see Access to Database Tables in the chapter "Fundamentals of Using Querying and Reporting Tools."

### Views

A view is actually a special definition, or virtual table, constructed from one or more tables. A view is a way of looking at or updating data stored in tables and does not contain any data or exist in physical storage.

Views enable you to:

- Limit a user's access to specific rows and columns of a table
- Manipulate data from multiple tables as if all the data were contained in a single table

- Gain access to *aggregates* (sets of data) as if they were individual columns of data

Views simplify retrieval and modify the user's view of data to only certain rows and columns in a table. For example, the table on which you base a view can contain the columns name, title, and hourly\_rate. However, the view based on that table might only show columns for name and title and could be restricted to certain rows. Views also allow you to run queries and reports on specific subsets of data and to specify certain rows that fit particular criteria.

You can use the Tables Utility to look at the structure of a view, and to retrieve, modify, or perform computations on the data in the underlying tables. For example, while in the Tables Utility, you could access a particular view based on the emp and tasks tables to compute the average hourly rate of programmers (emp table) working on a particular project (tasks table).

Views are created with a query language such as SQL. You cannot create or destroy views from within the Ingres Menu or the Tables Utility. For detailed information on views, see the description of the create view command in your query language reference guide.

You can access any view for which you have been granted the proper permissions. For more information on permissions, see the *Database Administrator Guide*.

## Indexes

An index is a table that indicates where data is stored in another table. It contains the locations of specified columns in the base table that are queried frequently. The index can speed up the retrieval of information.

You can examine an index's structure but cannot perform any other operations on it in the Tables Utility. You create an index on one or more columns of a table, using a query language such as SQL. Whenever a user enters a query based on the indexed column in the base table, the index helps locate the information quickly. Use of indexes is recommended to improve performance of the queries in your applications.

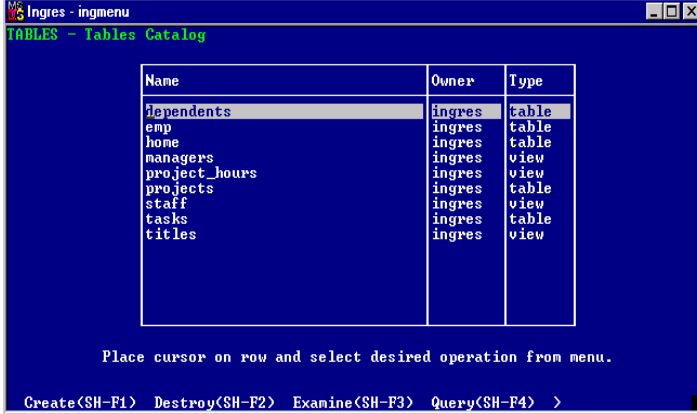
Indexes are created with the create index query language statement. You cannot create, destroy, or directly query or run reports on indexes from within the Ingres Menu or the Tables Utility. For more information on indexes, see the create index statement in your query language reference guide.

## Start the Tables Utility

### To start the Tables Utility from the Ingres Menu

1. Select the Tables operation from the Ingres Menu.

The Tables Catalog frame displays:



Name	Owner	Type
dependents	ingres	table
emp	ingres	table
hone	ingres	table
managers	ingres	view
project_hours	ingres	view
projects	ingres	table
staff	ingres	view
tasks	ingres	table
titles	ingres	view

Place cursor on row and select desired operation from menu.

Create(SH-F1) Destroy(SH-F2) Examine(SH-F3) Query(SH-F4) >

The Tables Catalog frame lists the tables, views, and indexes in the database to which you have access. It also lists the synonyms that you or the DBA own.

2. Locate a table, synonym, view, or index name by scrolling through the list or by using the First Letter Find function (see page 41).

## Menu Options for the Table Catalog

The Tables Catalog frame includes the following menu options:

### **Create**

Create a new table. Displays the Table Create frame. Cannot be used to create synonyms, views, or indexes.

### **Destroy**

Destroys the selected table, synonym, view, or index after asking for verification. Destroying a synonym does not destroy the underlying table.

### **Examine**

Displays information about the names and data types of the columns in the selected table, view, synonym, or index. Invokes the Table Information frame.

### **Query**

Allows you to retrieve data from the selected table (or underlying table for a synonym, view, or index). Also allows update and append operations on tables.

### **Report**

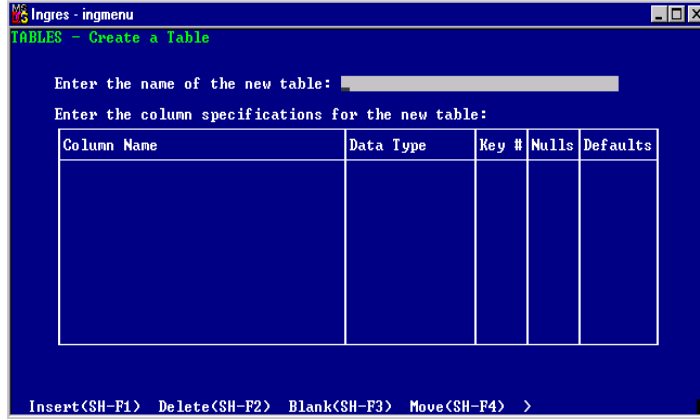
Runs a preview report on the selected table, view, synonym, or index. Displays a submenu with the following report styles: Tabular, Wrap, or Block. For descriptions of preview reports and report styles, see the chapter "Using RBF." After you select the report style, it sends the report to a file or screen. For additional instructions, see the Sending a Report sections in the chapter "Producing a RBF Report."

### **Help, End, Quit**

Standard operations.

## Create a Table

When you choose the Create operation from the Tables Catalog frame, the Create a Table frame displays.



Column Name	Data Type	Key #	Nulls	Defaults
-------------	-----------	-------	-------	----------

### To create a new table in the Tables Utility

1. Enter the name of the table (see page 60) in the first field, and then press Return.

The cursor moves to the first row in the Column Specifications list.

2. Enter a Column Name (see page 61), Data Type, Key # (if any), Nulls (if any), and Defaults (if any), for each of the data columns that you want to have in this table. Each row in the Column Specifications form represents one data column in the table.

You can use the GetTableDef (see page 66) operation to copy in column specifications from other tables.

3. Select End to save the table and return to the Create a Table frame.

You can also use a query language create table statement to create a table, or an SQL create schema statement to create a collection of tables. For details, see your query language reference guide.

## Table Names

Table names must conform to the standard object naming conventions discussed in Naming and Name Use Conventions (see page 49). Table names must be unique by user. That is, one user cannot have two tables with the same name, unless the identical names are delimited identifiers in different case. Different users can each have tables with the same names; we recommend, however, that you do not use the name of a table created by the database administrator.



## Column Specification—Column Name

Column names must conform to the standard object naming conventions discussed in Naming and Name Use Conventions (see page 49). Different tables can have the same column names; however, column names within a single table must be unique, unless the names are delimited identifiers in different case.

## Column Specification—Data Types

Specify data types by entering a type identifier and, in some cases, a number representing the maximum width of the data column. For example, the data type c15 specifies a character data type and a maximum width of 15 characters. Be aware of the maximum possible length of the data type so that you do not exceed it when you are specifying data types.

The following general data types are supported:

- Character
- Numeric
- Money
- Date

For a full description of all data types and their functionality, see the chapter "Working with Data Types and Data Display Formats" and your query language reference guide.

## Column Specification—Key Numbers

A key number is optional. When you specify a key number for a data column, data entered into the table is automatically sorted by ascending data value in the key column. For example, if a column named Lastname had the key number of 1, all data in the table is stored according to the alphabetic order of the names in the Lastname column.

You can specify more than one column as a key. This is known as a *multicolumn* key. The data column is sorted with the lowest key number first, then the column with the next highest number, and so on, until all the key columns are sorted.

For example, in a table containing information about employees, if the Department data column is key 1, and the Lastname column is key 2, all the data rows for each department are stored together, and within each department group the rows are alphabetized by last name.

When you are finished creating the table, you can optionally specify that all key columns in the table be used as unique constraints. These constraints prevent the user from entering multiple rows of data that have identical values in the key columns. For details, see *How Unique Keys Are Set* (see page 65).

You can also use ISQL in the Interactive Terminal Monitor or SQL outside of the Tables Utility to specify unique constraints for the table, using the SQL modify statement. For details, see the *SQL Reference Guide*.

## Column Specification—Nulls

This field allows you to specify whether the data column accepts null values. A *null value* is a special value that represents unknown or unavailable data. On standard installations, a null value appears to be an empty field, but there is a difference between a null value and a blank or a zero. A *nullable* column is one that accepts a null value. The Nulls field works in conjunction with the Defaults field.

To specify that the data column can *accept null values*, enter y in this field. If a user does not enter data into a nullable column, a value is entered for that column as specified by the Defaults field (see page 63).

To specify that the column *cannot accept null values*, enter n. When a user does not enter data into a non-nullable data column, either the value of zero (0) is automatically entered into the column if it is a numeric column, or the column is left blank if it is a character column, or the user is required to enter a value. Use the Defaults column in the Column Specification form to specify this choice.

## Column Specification—Defaults

The Defaults field allows you to control what happens when a user does not enter a value in the specified data column. It works in conjunction with the Nulls field specification.

You can enter standard defaults or user-defined defaults.

### How You Specify Standard Defaults

You can enter certain standard default instructions directly into the Defaults field, as described in the following table:

Nulls Field	Defaults Field	Results if User Does Not Enter Data
Yes	yes	Enters a null value in the data column.
Yes	no	Requires a user to enter data in this column. User can explicitly specify a null value. Any QBFor VIFRED form created from this table can have the Mandatory attribute set for this data column.
No	yes	Enters a default value as follows: Numeric column = 0 Variable-length character column = string of zero length Fixed-length character column = blanks Date column = empty date User cannot enter an explicit null value.
No	no	Requires a user to enter data in this column. User cannot enter a null value. Any QBFor VIFRED form that is created from this table can have the Mandatory attribute set for this data column.
Yes	null	Enters a null value in the data column.
No	null	Not allowed.
Yes	user	Enters the <i>username</i> of the runtime user in the data column if no value is explicitly specified. User can explicitly specify a null value.
No	user	Enters the <i>username</i> of the runtime user in the data column. User cannot enter an explicit null value.
Yes	value	Enters a user-defined default value created with the EditDefaults operation.
No	value	Enters a user-defined default value created with the EditDefaults operation.

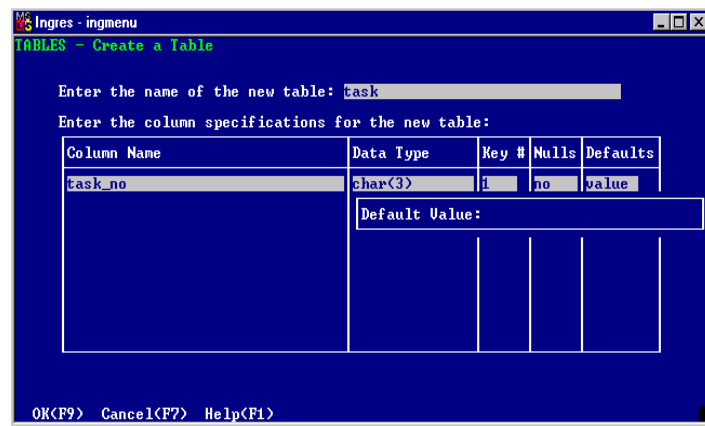
## How You Specify User-Defined Defaults

You can specify a user-defined default entry for a data column, using the EditDefaults menu option on the Create a Table frame. A value is then entered in the specified data column if an end user does not enter a value for that column. You can specify a user-defined default value for a column even if none of the other columns for the row have been filled in on the Create a Table frame.

### To specify a user-defined default value

1. In the Create a Table frame, place the cursor in the Defaults field for the row in the column specifications table field for which you want to define a default value.
2. Type the word, value.
3. Tab out of the field or choose the End operation.

The EditDefaults pop-up window displays:



Column Name	Data Type	Key #	Nulls	Defaults
task_no	Char(3)	1	no	value

The pop-up contains a scrollable default value window that is initially 23 characters wide and scrollable to 1500 characters. The DBMS limit for default values is 1000 characters; however, the extra space in the default window allows for dereferencing characters.

4. Type the default value for the data column. The value can be any valid entry that is compatible with the data type of the column, including today or now for a date column.

You can remove or change the user-defined default value at any time. To remove a user-defined default entirely, change the Defaults field entry to yes, no, user, or null, as appropriate. To change the user-defined default, place the cursor in the Defaults field and choose the EditDefaults menu operation; then enter the new default value in the pop-up frame.

The previous user-defined default value is saved temporarily. To restore it, type value in the Defaults field.

## How Unique Keys Are Set

You can prevent a user from entering duplicate rows in a table by specifying that all key columns for a table shall be *unique keys*. The unique key columns function as a combined unique constraint during data entry, which can improve performance and data integrity. If you use *unique keys* in your table, you require that each *key value* in the table be unique. The *key value* is the concatenation, or combined listing, of all the values in the key columns in a row. If you specify unique keys, the table can only hold one record for each combined key value.

For example, if you have specified that Firstname and Lastname are the key columns in your table, and you specify that your table must use unique keys, then the combined Firstname and Lastname columns are the unique key value for the table. This prevents any entry that has a combined first and last name from being identical to an existing entry in the table.

To make all key columns in your table a combined unique constraint:

1. After creating a table in an Ingres database and entering key columns for the table, select End. The Tables Utility displays a pop-up menu:
2. Move the cursor to the Unique option on the Unique Keys pop-up frame and click the appropriate mouse button or choose Select at the bottom of the frame to make the columns you specified in the Key column your combined unique constraint.

## Move Column Specifications

The order in which you enter the data columns in the Column Specification form determines the way table data is presented in default reports. In default reports, the report data is sorted on the values in the first column. You can use the Move operation to change the order of data columns when creating a table.

The top data column specification in the table field is the first data column in the table and can be the leftmost data column in any default report created from the table.

### **To move a data column specification from one row of the Column Specification form to another**

1. Place the cursor on the data column specification row that you want to move, and then select the Move operation.

The Move submenu appears.

2. Position the cursor on the row to which you want to move the data column specification row, and then select the Place operation from the submenu

The column specification row to be moved is inserted at the new cursor location and the current row is pushed down by the newly moved row.

## Clone Table Specifications with GetTableDef

The GetTableDef operation allows you to copy column specifications from an existing table into your new table. This is useful because tables in the same database often have identical data columns that are frequently used for joins and views. By copying the column specifications from one table to another, you ensure that these repeating columns all have the same data type, column width, and null and default values.

### **To copy column specifications from an existing table**

1. Select the GetTableDef operation.

You are prompted for a table name.

2. Enter the name of the table whose specifications you want to copy.

The column specifications for that table are copied into the Column Specification form. You can copy columns from more than one table into the new table column specifications.

3. Use the Delete and Move operations to eliminate column specifications you do not need and to modify the column order.

## Destroy Tables, Synonyms, Views, and Indexes

To destroy (delete) a table, synonym, view, or index, you must own them or have proper access permissions.

**Note:** Destroying a synonym does not destroy the underlying database table.

### **To destroy a table, synonym, view, or index and all its contents from a database**

1. Place the cursor on the name of the table, synonym, view, or index you want to destroy, and select the Destroy operation.

A pop-up form asks you for confirmation.

2. Place the cursor on yes to confirm that you want to destroy the selected object or no to leave the object as is.
3. Click the appropriate mouse button or select the Select operation.

You can also destroy a table, synonym, view, or index using a query language. For details, see your query language reference guide.

Keep in mind that you cannot later *undo* the destruction of a table, synonym, view, or index, or recover the contents of a destroyed table. To recover a table later, copy it to a file with the copydb command before deleting it from the database. Otherwise, you can recover the object only by restoring the entire database from a backup tape, if you have one. For instructions on using the copydb command, see the *Database Administrator Guide*.

## Get Information about Tables and Views

Use the Examine operation on the Tables Catalog frame to display information about a table, synonym, view, or index.

### To use the Examine operation

Place the cursor over the name of the table or view you wish to learn about, and then select the Examine operation.

The Examine a Table frame is displayed:

Information on Table **emp**

Owner: ingres      Table Type: user table  
Row Width: 70      Storage Structure: btree  
Columns: 4      Pages/Overflow: 6/0  
Rows: 32      Journaling: at next chkpt

Column Name	Data Type	Key #	Nulls	Defaults
name	varchar(20)	1	no	no
title	varchar(15)		no	yes
hourly_rate	money		no	yes
manager	varchar(20)		yes	null

NewTable<SH-F1> ViewDefaults<SH-F2> Help<F1> End<F10>

If you are examining a synonym or view, the Rows, Storage Structure, Pages/Overflow, and Journaling fields do not appear on the frame.

If you are examining a synonym, the base table's name and schema name also display. If you do not have access to the base table, *only* the base table's name and schema name display.

All the fields on the Table Information frame are display-only; you cannot edit or change any information.



## Fields for Examine a Table

The fields on the Examine a Table frame are as follows:

**Owner**

Name of the schema in which the table or view resides.

**Row Width**

Total width, in bytes, of any data row in the table.

**Rows**

Approximate number of data rows (records) currently stored in the table. For a new table that has not yet had data entered into it, this number is zero (0). You can multiply the row width by the number of data rows to calculate how much disk storage space the table now occupies.

**Columns**

Number of columns in the table.

**Table Type**

Table type (table, view, or secondary index).

**Storage Structure**

Type of storage structure that the table uses; for example, heap or compressed B-tree.

**Pages/Overflow**

Number of pages and overflow pages the table now occupies.

**Base Table**

Name of the underlying table (for a synonym only).

**Base Table Owner**

Name of the schema in which the base table resides (for a synonym only).

**Journaling**

Indicates whether journaling is enabled for this table. Journaling is a process of logging all changes to the table over time.

**Column Name**

The name of the column in the table.

**Data Type**

The type of data (and in some cases maximum number of characters) stored in the column.

**Key #**

The key sort priority number of the column, if any.

**Nulls**

Indicates whether null values are accepted in the column.

**Defaults**

Indicates whether default values can be automatically entered in the column if no data is entered by the user.

## How You Add or Delete Columns in an Existing Table

To add or delete columns in an existing table, you must use query language statements outside of the Tables Utility. This allows you to recreate the table with all or some of the columns from the original table, as well as add any new columns. For specific instructions on how to do this, see the *Database Administrator Guide* for the system on which your database resides.

**Note:** When you recreate a table, you also must recreate any synonyms, views, or indexes based on the original table. In addition, you must edit any forms or reports based on the original table, if the form or report references any changed or deleted columns or if it is to access any new column in the table.

# Chapter 4: Using QBF

---

This section contains the following topics:

[Query-By-Forms \(QBF\)](#) (see page 71)

[Query Definition and Execution Phases of QBF](#) (see page 72)

[Ways to Start QBF](#) (see page 73)

[Query Target Selection](#) (see page 75)

[Query Execution](#) (see page 79)

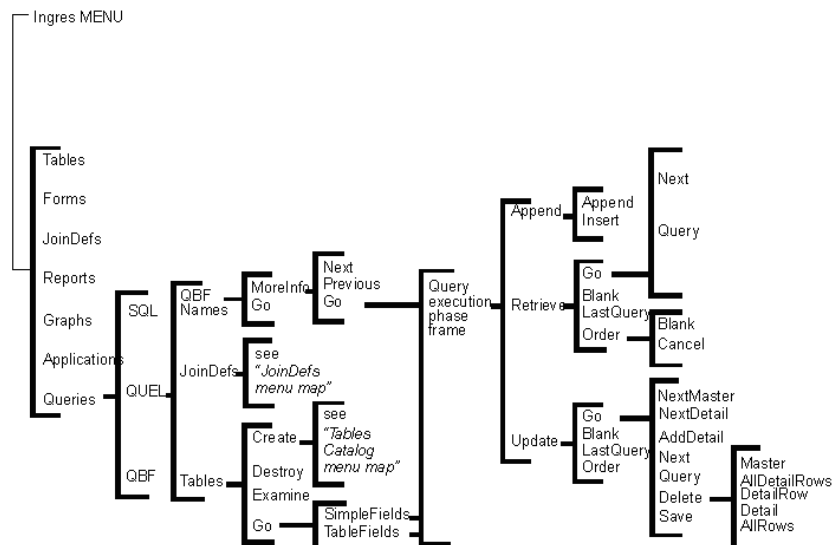
[How Query Results Are Displayed and Saved](#) (see page 81)

## Query-By-Forms (QBF)

QBF is an interactive, visually oriented, forms-based tool for adding, deleting, changing, and viewing data on selected query targets in a database.

A query target can be a table, JoinDef, or QBFName. For ease of use, you can use default forms to access tables or JoinDefs in QBF. You can also define a customized form to suit your particular needs for accessing a table or JoinDef with which the customized form is associated. A table or JoinDef that has been associated with a customized form is called a QBFName. Use the VIFRED to create the customized form for a QBFName, as described in the chapter "Using VIFRED."

You can access QBF frames from the QBF Startup frame. For a map of the options available to you when you choose the JoinDefs operation, see the chapter "Using JoinDefs in QBF."



## Query Definition and Execution Phases of QBF

Before starting QBF, you must understand the difference between the two phases involved in performing a query. They are:

- Query definition phase
- Query execution phase

When starting QBF from the operating system, you can use different commands to enter QBF at the correct phase.

When starting QBF from the Ingres Menu, you can move through each phase in a predetermined order.

### Query Definition Phase

In the *query definition* phase, you choose or create a query target. A *query target* contains the data you want to review, add to, or change. You can use a query target for queries as often as you like.

There are three recognized types of query targets:

#### **Tables**

Two-dimensional arrays of data

#### **JoinDefs**

Two or more tables joined through common values in one or more columns

#### **QBNames**

The association of a table or JoinDef with a form created in the VIFRED

If your query target is a table or JoinDef that does not yet exist, you can create it from the appropriate Catalog frame in the query definition phase. To create a QBName, however, you must use the VIFRED, which is accessed through the Forms menu choice on the main Ingres Menu frame.

You can choose a defined synonym or an existing view as a query target in place of a table name. However, you cannot create a synonym or view from the Tables Catalog frame or any other QBF frame. You can create views only with a query language. For more information on synonyms and views, see the chapter "Using the Tables Utility."

## Query Execution Phase

After choosing or creating the query target, you enter the query execution phase. In this phase, you can manipulate data in these basic ways:

- Append (add) data rows to a table
- Retrieve (view) data from tables
- Update (modify or delete) data in tables

For a discussion of these functions, see the chapter "Working with QBF Operations."

## Ways to Start QBF

You can start QBF from the operating system or from the Ingres Menu.

### How You Start QBF from the Operating System

The command and command line syntax you use to start QBF at the operating system prompt determines the starting QBF phase:

- Using the QBF command without a query target starts QBF in the query definition phase. From the query definition phase, you can then move to the query execution phase.
- Using the QBF command with a specified query target starts QBF in the query execution phase.
- Using the query command, which requires specification of a query target, starts QBF in the query execution phase.

For more information on the QBF and query commands and their parameters, see the chapter "Using System Command for the Forms-based Tools."

## Start QBF from the Ingres Menu

You can start QBF from the operating system or from the Ingres Menu.

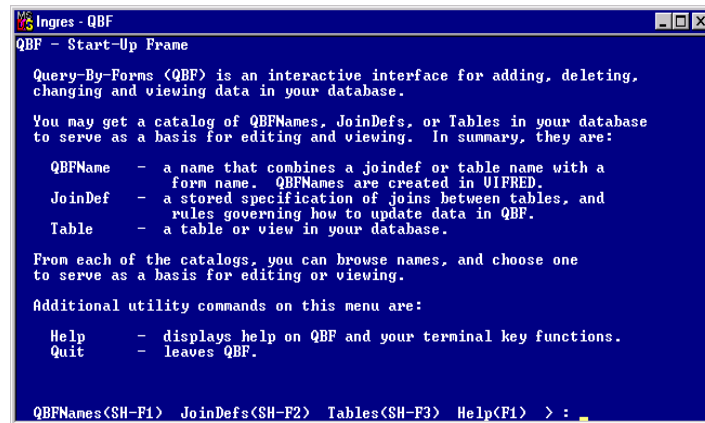
### To start QBF from the Ingres Menu

1. Choose the Queries operation from the Menu.

The Queries frame displays.

Choose the QBF operation from the Queries frame.

The QBF Start-Up frame displays.



2. Select QBFNames, JoinDefs, or Tables.

An appropriate Catalog frame displays, as discussed in Query Target Selection.

## Query Target Selection

Catalogs automatically keep track of query targets. A Catalog frame displays a list of query targets, a menu of available operations, and some explanatory information.

Query targets can be:

- Tables
- QBNames associated with tables or JoinDefs
- Synonyms referring to tables
- Views or JoinDefs based upon tables

You must either own the query target object or have been granted the appropriate permissions to access or perform an operation on the object. For more information on granting or obtaining permissions, see the *Database Administrator Guide*.

To locate the query target you want, scroll through the list in the Catalog frame or use the First Letter Find function (see page 41).

For each type of query target, a different type of form is used for displaying your query results. For more information on data display forms, see Data Display Forms for Query Targets (see page 77).

## Catalog Frames for Query Target Selection

Each type of query target—QBName, JoinDef, and table—has its own Catalog frame. Each Catalog frame offers various operations, such as create, destroy, edit, or examine, that are specific to that type of query target. The Tables Catalog frame operations are discussed in detail in the chapter "Using the Tables Utility." Operations on the JoinDefs and QBNames Catalog frames are discussed in the chapters "Using JoinDefs in QBF" and "Using VIFRED."

All catalog frames offer the following standard operations for use with QBF:

### **Go**

Runs a query on the target you choose

### **Help, Quit**

Standard operations

## QBNames Catalog Frame

The QBNames Catalog frame, shown here, contains a list of the QBNames that you can access. A *QBName* represents a data display form designed or customized with the VIFRED and then linked to a JoinDef or table. For a discussion of QBNames, see the chapter "Using VIFRED."

Name	Owner	Short Remark
emp	ingres	Table field display of emp
projects	ingres	Table field display of projects
tasks	ingres	Table field display of tasks

Place cursor on row and select desired operation from menu.

MoreInfo<SH-F1> Go<F9> Help<F1> End<F10> Quit<F6>

Choosing a QBName query target from the QBNames Catalog frame informs QBF to use the customized data display form and the associated JoinDef or table in the query operation.

## JoinDefs Catalog Frame

The JoinDefs Catalog frame, shown here, contains a list of the JoinDefs that you can access.

Name	Owner	Short Remark
emp_tasks	ingres	Employee Task Assignments
manager_projects	ingres	Manager Project Assignments
projects_tasks	ingres	Projects and tasks
staff	ingres	Project staff
staff_home_addr	ingres	Staff home addresses & phones
task_assignments	ingres	Employee Task Assignments

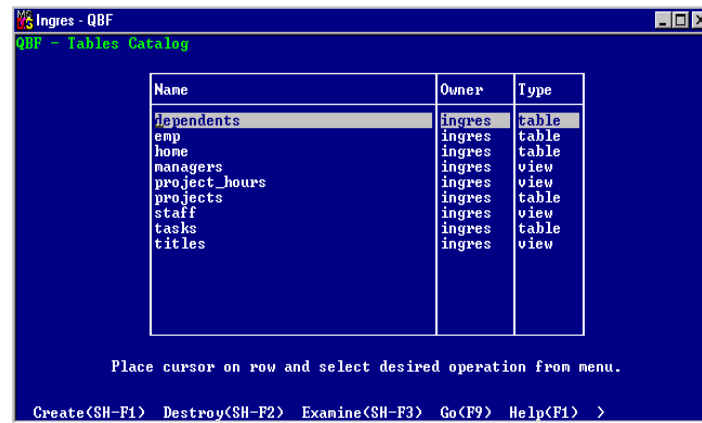
Place cursor on row and select desired operation from menu.

Create<SH-F1> Destroy<SH-F2> Edit<SH-F3> Rename<SH-F4> >



## Tables Catalog Frame

The Tables Catalog frame, shown in the following figure, lists the names and owners of all tables and views to which you have access in the current database.



Choose any table or view in the Tables Catalog frame as a query target for query execution. However, when executing a query on a view, you can only perform a Retrieve operation. You cannot perform the Update or Append operations on a view. For more information on views, see the chapter "Using the Tables Utility."

## Data Display Forms for Query Targets

If you choose an existing QBFFName as your query target, an associated custom form for the query is used. If you choose a query target that is a table or JoinDef, an appropriate default form for the query is used. For a table, you can choose between two additional options, SimpleFields or TableField format, to specify which default form to use.

### Custom Forms

You can use the VIFRED to develop a custom form and associate it with a query target. VIFRED allows you to specify the dimensions and general appearance of the form or form components and to specify different display and behavior attributes for the fields on the form. For information about creating custom forms, see the chapters "Using VIFRED," "VIFRED Form Components," and "VIFRED Field Specifications."

To specify a query target for use with a custom form, you use the QBFFName that associates the form with the table or JoinDef.

## Default Forms

When displaying a default form for a query, QBF uses the column names and data types of the table or tables in the query target. Unless there is a conflict, QBF uses the table column names as the internal name for each field on the form. When conflicts occur, QBF creates unique internal field names by slightly changing one of the column names.

When creating a form, QBF allocates each field enough space on the form for data entry and field titles. If you have specified that a table field must appear on the form, QBF first places all the simple fields on the form and then places the table field at the end of the form.

QBF supports several types of fields, corresponding to the basic data types:

- Integer (whole numbers only)
- Floating-point (numbers including decimal places and scientific notation)
- Character (alphanumeric characters)
- Date
- Money

For more detailed information about data types and formats, see the chapter "Working with RBF Report Specifications."

## Query Execution

You can start query execution from within the Ingres Menu, following the query definition phase, or from the operating system, without traversing the Ingres Menu and QBF definition phase screens. For instructions on starting query execution from the operating system, see the chapter "Using System Commands for the Forms-based Tools."

### To begin the query execution phase from the Ingres Menu

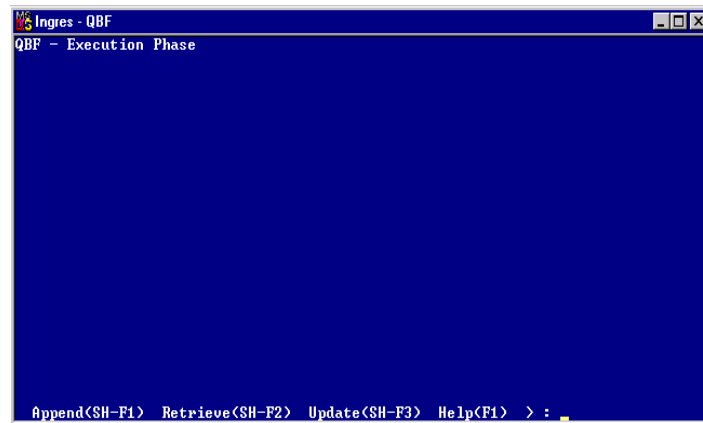
1. Start QBF in the query definition phase and choose a query target, as described in the sections, Starting QBF and Choosing a Query Target:
  - a. Choose Queries.
  - b. Choose QBF.
  - c. From the Start-Up frame, choose Tables, JoinDefs, or QBFNames.

A Catalog frame displays with the available query targets of the type you specified.

2. Place the cursor on the row in the Catalog frame that contains the query target you want and choose the Go operation.

If your query target is a table, a submenu appears. Choose the SimpleFields or TableField operation to specify the format for the default form.

The data is retrieved from the system catalogs and displays an appropriate QBF Execution Phase frame.



3. Choose an appropriate operation, as follows:

**Append**

Adds information to the database.

**Retrieve**

Retrieves information from the database.

**Update**

Changes or deletes information in the database.

**Help, End, Quit**

Standard operations.

When you choose Append, Update, or Retrieve, QBF displays the appropriate frame for the operation and query target you have chosen.

If you access the QBF Execution Phase frame directly from the operating system by specifying a query target in the qbf command, two additional operations on the QBF Execution Phase menu appear:

**NewQueryTarget**

Chooses a new query target to run.

**Start**

Goes to the Start-Up frame.

The NewQueryTarget operation lets you specify a new query target without exiting QBF. You are prompted for the name of a new query target. It searches for a query target with that name first among QBFNames, then, among JoinDefs, and last, among tables. The Start operation returns you to the Start-Up frame.

## How Query Results Are Displayed and Saved

The Retrieve operation allows you to view, but not change, data. Both the Append and Update operations allow you to make changes to the data, which you can save.

A default form for all three operations used with tables and JoinDefs is generated. For example, if your query target is a JoinDef, the default form might look like the following:

The screenshot shows a window titled "Ingres - QBF" with a form titled "EMP Table". The form has four input fields: "Name:" (empty), "Hourly Rate:" (empty), "Title:" (empty), and "Manager:" (empty). Below these fields is a section labeled "TASKS TABLE:" containing a table with three columns: "Project", "Task", and "Hours". The table is currently empty. At the bottom of the window, there is a navigation bar with the following options: "Go<F9>", "Blank<SH-F2>", "LastQuery<SH-F3>", "Order<SH-F4>", and ">".

The data retrieved by this query is shown here:

The screenshot shows the same "Ingres - QBF" window with the "EMP Table" form. The input fields are now populated: "Name:" contains "Alcott, Scott", "Hourly Rate:" contains "\$ 50.00", "Title:" contains "Sr Programmer", and "Manager:" contains "Wolfe, Neal". The "TASKS TABLE:" section now contains a table with three columns: "Project", "Task", and "Hours". The table has two rows of data: the first row has "Advertise" in the Project column, "Implement" in the Task column, and "5" in the Hours column; the second row has "Advertise" in the Project column, "Test" in the Task column, and "8" in the Hours column. At the bottom of the window, the navigation bar now includes "NextMaster<F9>", "Query<SH-F2>", "Delete<SH-F3>", "Save<F3>", and ">".

When you modify data with the Update operation, the changes you make are stored in a temporary location. It does not write them into the actual table until you select the Save operation. If you leave QBF execution without specifically saving your changes, you can lose your modifications and the data in the database tables remains unaltered.

If a join column in a JoinDef has been protected from updates with the Rules operation, you cannot update the join column but you can update the rest of the row. If the table has been protected from updates with the Rules operation, you are not allowed to save changes to the protected table. QBF issues an error message when you try to execute the Save operation. For information on the Rules operation, see the chapter "Using JoinDefs in QBF."

# Chapter 5: Working with QBF Operations

---

This section contains the following topics:

[QBF Append Operation](#) (see page 83)

[QBF Retrieve Operation](#) (see page 87)

[QBF Update Operation](#) (see page 110)

## QBF Append Operation

The Append operation adds new records (rows) to a table. To append data, you fill in a QBF form and choose the appropriate menu operation.

In QBF, you can append data only to tables for which you have been granted the Append privilege to add rows.

### Start the Append Operation

You can start the Append operation from the operating system or from the Ingres Menu.

#### **To start the Append operation from the Ingres Menu**

1. Select Queries.
2. Select QBF to reach the QBF Start-Up frame.
3. Select Tables, QBFNames, or JoinDefs for the appropriate Catalog frame.

When working with a table query target, you must also select SimpleFields or TableField format.

4. Place the cursor on the name of your query target in the Catalog frame and choose the Go operation.

The Execution Phase frame displays.

5. Select Append.

A default query form displays.

The chapter "Using QBF" discusses steps 1 through 5 in more detail.

## How to Use the Append Frame

Two approaches to adding new rows of data to a table with QBF are available:

- Use a table field to append many rows of data at once.
- Use simple fields to append one row at a time.

Each field on the Append frame corresponds to a single column in the query target table(s). On a default form, if the query target is a JoinDef, the join field(s) appears in reverse video, if your terminal has that capability.

Whether you use simple fields or a table field depends on the query target:

- If the query target is a table, the menu choices SimpleFields and TableField appear, following the table you chose in the Tables Catalog frame.
- If the query target is a Master/Detail JoinDef, the default format is a table field for the detail table and simple fields for the master. You can change this default on the JoinDef Definition frame.
- If the query target is a QBFName, the form shown on the frame can have table fields or simple fields or both. The appearance and behavior of a QBFName form can be altered using the VIFRED as described in the chapters "Using VIFRED," "VIFRED Form Components," and "VIFRED Field Specifications."

## How You Add New Rows of Data

To add new rows of data to the database, follow these steps:

1. Start the Append operation from the operating system or from the Ingres Menu, as described in Start the Append Operation.
2. Enter your data in the fields of the form. If you are using a table field, you can tab through the fields in each row. Press Return at the end of the row.

If the query target is a QBFName, the sequence in which the cursor moves from one field to the next is determined by the order specified in the custom form created in VIFRED.

3. When you have typed in all the rows of data that you want to add, choose the Append operation from the Append frame menu to save your additions.

Your data is added to the database tables and displays a message telling you how many rows of data it has appended.

If you do not choose the Append operation in the Append frame, your additions are not saved.

The methods you use to append data differ slightly depending on whether you are using a simple-fields or table-field format.



## SimpleFields Format

If you are appending data using simple-fields format, you can append only one row at a time. When you are finished entering data, choose Append to write your new data into the database. Append is the only operation available.

## TableFields Format

If you are appending data using table-field format, you can enter more than one row of data in the table field before choosing Append. When finished entering data, choose Append to write your new data into the database.

The following menu operations are available on the Append frame if you choose the table-field format:

### **Append**

Appends the data to the table.

### **Insert**

Inserts a new blank row in the table field.

### **ListChoices**

Displays the available choices for the selected field on a QBFName.

### **Help, End**

Standard operations.

The Insert operation on the Append frame allows you to open a new line in the table and insert a new row of data in a specific location. Using Insert does not affect the retrieval or reporting of data in the table(s). The Insert operation is provided as a convenience so that you can visually order your new rows as you enter them on the frame.

### **To use the Insert operation**

Put the cursor where you want the new row to appear, and then select Insert.

QBF places the cursor at the beginning of a new blank row inserted above the previously selected row.

## Data Entry Errors when Appending

When you select Append, each field is checked for errors. If an error is detected, an error message appears and the cursor returns to either the field containing the error or to the first field on the form.

Select Append again to add your corrections to the database.

If your query target is a QBFName, your customized form can contain mandatory fields, validation checks, or special attributes. These requirements must all be met before QBF allows the Append operation to add data from the form to the database.

## Control-A—Duplicate Previous Entry

When entering data in simple-field format, press Control-A to duplicate the previously entered value for that field. For example, pressing Control-A in the Zipcode field enters whatever zip code was entered on the previous form. Control-A does not work in table-field columns.

## Transaction Deadlock in Append Mode

Because transactions are supported, a transaction deadlock can occur in Append mode. This can only happen after you select the Append command to add the data you have entered in the form to the database. If an attempt to add a row to one of your query targets aborts because of a transaction deadlock, QBF informs you that a deadlock has occurred and that your transaction has been aborted. QBF then automatically retries the entire append.

## Confirmation Messages for Append

If an append is successful, QBF displays a confirmation message, for example:

```
Appended 1 master and 2 detail row(s)...
```

QBF displays an error message if you attempt to append information to a table that has update rules preventing this, or if you violate an integrity constraint, which defines a valid range for data.

## Exit the Append Operation

### To exit the append operation

Select End.

You exit the QBF Execution Phase frame.

If you try to exit without committing the data with the Append operation, QBF prompts:

Do you wish to leave APPEND without appending data on the frame?

If you type **y** (yes), QBF displays a message indicating how many rows of data have been appended in this session and returns you to the QBF Execution Phase frame without appending the current data. If you answer **n** (no), QBF returns you to the form where you can select Append.

Each time you exit after appending data, QBF displays a message containing the number of rows successfully appended during the session.

## QBF Retrieve Operation

The Retrieve operation retrieves information from the database and displays it in a window. In Retrieve mode, you can browse through the retrieved data, but you cannot make any changes to it.

When you use Retrieve, QBF displays a blank form representing the JoinDef or tables you want to review. To specify which data you wish to view, enter search qualifications in the fields on a form.

## Start the Retrieve Operation

You can start the Retrieve operation from the operating system or from the Ingres Menu.

### **To start the Retrieve operation from the Ingres Menu**

1. Choose Queries.
2. Choose QBF.  
The QBF Start-Up frame displays.
3. Choose QBFNames, JoinDefs, or Tables for the appropriate Catalog frame.  
When working with a table query target, you must also choose SimpleFields or TableField format.
4. Place the cursor over the name of your query target in the Catalog frame and choose the Go operation.  
QBF displays the Execution Phase frame.
5. On the QBF Execution Phase frame, click Retrieve.  
QBF displays the Retrieve frame with the fields of your query target and a menu of operations. The appearance of the Retrieve frame varies depending on the type of query target. The following section discusses the Retrieve Frame.

Some data columns in a table can be hidden from your view by the JoinDef ChangeDisplay operation.

The chapter "Using QBF" discusses steps 1 through 4 in more detail.

## Retrieve Frame

The following figure shows the Retrieve frame for a JoinDef query target. The Retrieve frame can contain simple fields and/or a table field.

The screenshot shows a window titled "Ingres - QBF". Inside, there's a section for "MANAGERS Table" with fields for "Manager:" and "Title:". Below this is a section for "PROJECTS TABLE:" which contains a table with five columns: "Project", "Description", "Dept", "Budget", and "Due Date". The table is currently empty. At the bottom of the window, there's a menu bar with options: "Go<F9>", "Blank<SH-F2>", "LastQuery<SH-F3>", "Order<SH-F4>", and a right arrow ">".

Join columns are indicated by reverse video on a Retrieve frame if your terminal has that capability.

If the query target contains too many columns to fit in your window, tab through the columns to scroll to those that are outside the displayed window.

## Menu Options for Retrieve Frame

### Go

Executes the query.

### Blank

Clears the current entries from the window.

### Order

Displays current data sort order specification for review or editing.

### LastQuery

Places the contents of the previously run query on the form for review or editing.

### ListChoices

Lists the available choices for the selected field on a QBFName.

### Help, End

Standard operations.

## How to Use the Retrieve Frame

To use the Retrieve frame, follow these steps:

1. Enter search conditions in the columns of the Retrieve frame form to *qualify your search*. Qualifying a search limits the results to records (rows) containing the types of data you specified in the Retrieve frame. For example, to look at only those records pertaining to the advertising department, only records for the month of January, or only records of employees with salaries greater than \$30,000 per year. For more information, see Search Qualifications (see page 91).
2. Use the optional Order operation (see page 103) to *specify the order* in which you want to view the retrieved records. For example, to view records chronologically, or by department, or size of budget.
3. Select the Go operation (see page 105) to start the search.
4. Select the Query operation to begin a new query.

## Search Qualifications

Qualification is the process of specifying which rows in the table(s) you want to retrieve. You indicate which rows to retrieve by entering your qualification criteria, or *search conditions*, in the fields on the Retrieve form. This restricts retrieval to those records (table rows) that match or meet the qualification criteria you entered.

For example, if the Retrieve form has a Lastname field, and you enter Lincoln in that field, the word Lincoln becomes a search condition. QBF retrieves only those records with a value of Lincoln in the Lastname field.

Leaving all the fields on the QBF Retrieve form blank is equivalent to specifying no restrictions on retrieval so that all records of data from the table(s) are displayed.

The following figure gives an example of search conditions that restrict retrieval of records from the Managers table to those data rows with the Project of Advertise.

The screenshot shows a window titled "Ingres - QBF". Inside, there's a section for "MANAGERS Table" with fields for "Manager:" and "Title:". Below this is a section for "PROJECTS TABLE:" which contains a table with five columns: "Project", "Description", "Dept", "Budget", and "Due Date". The "Project" column has a value "Advertise" highlighted. At the bottom of the window, there's a status bar with the text: "Go<F9> Blank<SH-F2> LastQuery<SH-F3> Order<SH-F4> >".

Project	Description	Dept	Budget	Due Date
Advertise				

The following figure shows the results of a query that restricted retrieval of records to data rows with the Projects Advertise and TextProc.

The screenshot shows a window titled "Ingres - QBF" with a sub-header "MANAGERS Table". Below this, there are two input fields: "Manager: Wolfe, Neal" and "Title: Analyst". The main section is titled "PROJECTS TABLE:" and contains a table with the following data:

Project	Description	Dept	Budget	Due Date
Advertise	Advertising Analysis	Sales	\$ 9500.00	02-mar-1999
TextProc	Text Processing	Admin	\$ 14000.00	01-dec-1998

At the bottom of the window, there is a status bar with the text: "NextMaster<F9> Query<SH-F2> Help<F1> End<F10>"

If the query target contains too many columns to fit in your window, tab through the columns to scroll to those that are outside the displayed window.

The search conditions you enter in a field can exceed the field's window width on the form. When you enter a long specification in a field, the window scrolls as you type.

You can duplicate a qualification value from a simple field on a form. For information about using auto-duplication on your terminal, see the appendix "Defining Function and Control Keys."

Trailing blanks are ignored in a qualification specification.

Use various operators and expressions to further qualify your retrieval search.

## How Comparison Operators Are Used

A *comparison operator* is a symbol that informs QBF that you want to compare two values or search for a range of data.

Qualify your query with these comparison operators.

Operator	Description
=	equal to
!=	not equal to
<	less than
<=	less than or equal to



Operator	Description
>	greater than
>=	greater than or equal to

For example, the following figure shows a query that searches the Projects table for records with a budget greater than or equal to \$12,000.

The screenshot shows a window titled "Ingres - QBF". Inside, there's a section for "MANAGERS Table" with fields for "Manager:" and "Title:". Below this is the "PROJECTS TABLE:" section, which contains a table with five columns: "Project", "Description", "Dept", "Budget", and "Due Date". The "Budget" column has a search condition ">=12000" entered. At the bottom of the window, there's a status bar with the text: "Go(F9) Blank(SH-F2) LastQuery(SH-F3) Order(SH-F4) >".

All comparison operators have equal precedence.

QBF assumes that fields containing values with no comparison operator have a defacto equals sign as the comparison operator. The equals sign works differently when querying non-nullable and nullable fields. For more information, see Queries in Blank and Nullable Fields (see page 102).

Comparison operators are often combined with the AND operator within a field. For example, entering a search condition of >100 and <900 searches for all values between 100 and 900. You can also use the OR operator. For example, <100 or >900 searches for all values that are either less than 100 or greater than 900. Likewise, >=a <=d retrieves all character strings that begin with the letters a, b, c, or d.

When using a comparison operator on a character field, the search is case sensitive because QBF treats lowercase and uppercase characters as different characters. All uppercase letters come before all lowercase letters. If you enter >a, you cannot find any character strings that begin with uppercase letters, while if you enter >A <a, you can find all strings that began with uppercase letters but no strings that began with lowercase letters.

Just as capital letters are ordered ahead of lowercase letters, QBF orders numbers in a character field ahead of both uppercase and lowercase letters.

You cannot combine a greater than (>) or less than (<) comparison operator with pattern matching. For example, >Sa% is not allowed. For more information on pattern-matching, see Pattern-Matching Characters (see page 97).

### How Logical Operators Are Used

QBF allows you to use the *logical operators* AND (conjunction) and OR (disjunctive) to qualify retrievals. AND and OR are known as Boolean operators:

- When you use the AND operator, QBF retrieves only data records that meet both or all of the criteria you specify.
- When you use the OR operator, QBF retrieves data records that meet any one of the criteria you specify.

There are two ways to use the AND and OR operators:

- Within a column
- Between different columns

For more information on the use of Boolean operators, see the chapter "VIFRED Field Specifications."

## Logical Operators Within a Column

You can use logical operators within a data column. For example, if you enter Lincoln or Douglas or Tubman in a Lastname field, the query returns records with a value of either Lincoln or Douglas or Tubman in the Lastname column.

You can explicitly enter the AND operator in a field, as you do the OR operator, or you can imply the AND operator by leaving a space between words. For example, entering Delta Gamma is the same as entering Delta and Gamma.

If you want to search for a string of words separated by literal spaces, you must enclose the string with double quotation marks (" "), because QBF otherwise interprets the spaces as implied AND operators.

For most practical purposes, the AND operator is only useful in numeric fields because the AND operator requires that the data in the field meet two separate and exclusive criteria. For example, a number can be both greater than 10 and less than 100.

Because the AND operator is exclusive, it usually returns nothing in a character column. For example, entering 1776 and Adams and Street in an Address field does not return the address 1776 Adams Street because QBF looks for a single field with a value of 1776 and nothing else, a value of Adams and nothing else, and a value of Street and nothing else. Because no field could meet such impossible conditions, no data records can be found.

You can use the AND operator in a character field if you use the greater than (>) and less than (<) comparison operators. For example, entering the search condition >a and <d retrieves all character strings that begin with either b or c.

## Logical Operators Between Columns

When using logical operators between columns, you do not enter the words AND or OR in the form as you do when you use these operators in a single column. In QBF, the AND and OR operators are implied by the way the qualifying criteria is entered in the fields. Simple fields and table fields differ in how logical operators are implied:

- On a form with simple fields, the AND operator is always implied when information is entered in more than one field.

- On a form in table-field format, the AND operator is implied when information is entered in the columns across *one row* of a table field.

In the following figure, for example, QBF only retrieves data from the Project and Managers tables that have the Project ID Advertise AND a Budget of more than \$5,000.

Manager: \_\_\_\_\_ Title: \_\_\_\_\_

PROJECTS TABLE:

Project	Description	Dept	Budget	Due Date
Advertise			>=5000	

Go(F9) Blank(SH-F2) LastQuery(SH-F3) Order(SH-F4) >

- The OR operator can be implied only in table-field format. Values in *different rows* of a table field are implicitly OR. In the following figure, for example, QBF can retrieve data that has the Project ID Advertise OR has a budget of more than \$5,000.

Manager: \_\_\_\_\_ Title: \_\_\_\_\_

PROJECTS TABLE:

Project	Description	Dept	Budget	Due Date
Advertise			>=5000	

Go(F9) Blank(SH-F2) LastQuery(SH-F3) Order(SH-F4) >

## Parentheses—Group Values

When using logical operators, you can use parentheses to group values. For example, to retrieve all projects with budgets less than \$5,000 or between \$15,000 and \$20,000, enter the following expression in the Budget field:

`<5000 or (>=15000 and <=20000)`

For information on handling queries containing literal parentheses, see Character String Qualifications (see page 101).

## Pattern-Matching Characters

For further qualification of queries, you can perform pattern matching with wild card characters. Wild card characters are symbols that represent unspecified character values.

In QBF, the asterisk (\*) and the question mark (?) are the default wild card characters. Pattern-matching characters cannot be used in fields with numeric data types.

As explained in The % and \_ Pattern Matching Characters, you can specify that your system use the % and \_ character in place of ? and \*.

## ? and \* Pattern-Matching Characters

Use the question mark (?) to represent one unspecified character. For example, T?P can find the values TAP, TCP, TOP, and so on. You can use more than one question mark to indicate more than one character. For example, to search for all 5-digit identification codes beginning with 941 enter 941???

Use the asterisk to represent any number of unspecified characters or no characters at all. For example, to search for all product codes containing the characters ALPHA enter \*ALPHA\*. This returns answers such as:

122-STAR-ALPHA  
STAR-ALPHA-X4  
ALPHA-987-PROTO  
ALPHA

To search for all product codes that begin with the characters STAR you enter STAR\*. This returns answers such as:

STAR-98-BETA  
STAR-ALPHA  
STAR

You can combine the question mark and the asterisk. For example, to locate any product that begins with four characters and a dash and ends with ALPHA, enter `????-*ALPHA`. This returns answers such as:

```
STAR-986-ALPHA
PROT-BETA/ALPHA
TEST-ALPHA
```

Pattern-matching characters and logical operators interact with each other. For example, you can combine the OR operator and pattern matching characters in a column by entering `STAR*` or `ALPHA-????`. This returns answers such as:

```
ALPHA-98G5
ALPHA-TEST
ALPHA-5005
STAR-87-BETA
STAR-876-ALPHA-X
STAR
```

For most practical purposes, the AND operator is only useful in numeric fields because it requires that the data in the field meet two separate criteria. Therefore, the AND operator is rarely used in conjunction with pattern-matching characters.

Because QBF treats spaces in search qualifications as an implied AND operator, you must enclose pattern-matching qualifications containing a literal space in quotation marks. For example, in an address field the specification `1776 Adams*` returns nothing and does not find 1776 Adams St because the space is treated as an AND operator. To include a space in a character field specification you must enclose the specification in quotation marks. For example, entering `1776 Adams*` returns answers such as:

```
1776 Adams Ln.
1776 Adams St.
1776 Adams Street
1776 Adamstown Court
```

To use the asterisk or the question mark as actual characters rather than as wild card characters, precede them with the backslash character (`\`). For example, enter `A\*B` to look for the actual sequence `A*B`. If you enter `A*B` by mistake, QBF searches for any combination of any number of characters that begin with the letter A and end with the letter B.

You cannot use a wild card pattern-matching symbol in conjunction with a greater than (`>`) or less than (`<`) comparison operator. For example, `>Sm*` is not allowed.

## % and \_ Pattern-Matching Characters

You can choose to use the underscore (\_) and percent sign (%) pattern-matching characters in place of the ? and \* characters. The underscore is equivalent to the question mark and the percent sign is equivalent to the asterisk. To specify these pattern-matching characters, use the `II_PATTERN_MATCH` environment variable/logical as explained in the *System Administrator Guide*.

## Bracketed Expressions

By enclosing characters within brackets ([ ]), you can stipulate specific values in a pattern-matching search. For example, if you want to find employees whose last names begin with R or T, enter `[RT]*` in a Lastname field. This returns last names such as Randall, Rotelli, Tamatomi, and Tijerina. You can include any number of characters within brackets and they can be placed in any order.

All pattern-matching queries work in Query mode. However, if you invoke QBF with the `-e` flag (for expert mode) you cannot use bracketed pattern-matching operators for specifying selection criteria in an empty catalog in order to retrieve a set of tables or other objects for querying.

A bracketed pattern is often used when a table contains data in different cases. For example, if you entered `S*` as a query specification in the Dept field, a retrieve returns rows with Sales, but not rows with sales. However, the query specification `[Ss]*` returns all rows containing both Sales and sales. By separating characters with a hyphen and surrounding them with brackets, you can stipulate a range of characters for pattern matching. Thus, entering `[A-M]*` in the Name field retrieves the names of all employees whose last names begin with any letter in the first half of the alphabet, such as Alcott, Chung, Feldmann, King, and Moore. The wild card characters can be combined with bracketed expressions. Used either before or after the brackets, they further refine a search for patterns in data.

## Complex Queries

The following figure is an example of a complex query that utilizes comparison operators, logical operators, and wild card characters. The query searches for information from a JoinDef consisting of the Staff table and the Tasks table. Specifically, it searches for records of employees whose names begin with the letter B, whose hourly rate is less than \$50, and who have either worked in the Design phase or have worked more than 25 hours in the Implement phase.

QBF interprets the search conditions on this form as follows:

Display the data if name = B\* and hourly rate <50 and  
(task = Design or (task = Implement and hours >25))

The screenshot shows the Ingres - QBF interface. At the top, it says "Ingres - QBF". Below that, the "STAFF Table" section has input fields for "Name: B\*", "Hourly Rate: <50", "Title:", and "Manager:". Below the STAFF Table section is the "TASKS TABLE:" section, which contains a table with three columns: "Project", "Task", and "Hours". The table has two rows: "Design" and "Implement >25". At the bottom of the interface, there is a command line with the text: "Go(F9) Blank(SH-F2) LastQuery(SH-F3) Order(SH-F4) >:".

Project	Task	Hours
	Design	
	Implement	>25



## Character String Qualifications

You can use case, parentheses, and multiple words in character strings.

**Case** — Character string search conditions are case-sensitive. For example, if you enter Franklin as a search condition, QBF displays only records with Franklin, but not FRANKLIN or franklin.

**Parentheses** — If a character string contains a literal parenthesis, you must enclose the entire string in quotation marks. For example, to retrieve the record for Martin (E) Smith, enter Martin (E) Smith.

**Multiple Words in Character Fields** — If a character field contains two or more words separated by spaces, such as a street address, you must either use a pattern-matching wild card character, as explained in Using Pattern-Matching Characters, or enclose a multi-word search condition in quotation marks.

For example, suppose you want to find records with a value of 1776 Adams Street in the Address column. You could do this by entering 1776 Adams Street as a search condition in the Address field.

However, if you enter 1776 Adams Street with no quotation marks as a search condition in the Address field, nothing is found because QBF interprets the spaces between the three words as implied AND operators.

As explained in the section on pattern-matching, you could also use a wild card character. For example, entering a search condition of 1776\* finds all addresses on all streets beginning with 1776. Entering a search condition of 1776 Adams St\* finds both 1776 Adams St. and 1776 Adams Street.

## Date and Time Qualifications

Date fields can hold either a simple date, such as 22-mar-1998, or a date and time, such as 22-mar-1998 10:44:23. You cannot use wild card characters in date fields to qualify a retrieval. However, you can use comparison and logical operators to retrieve a range of dates and/or times.

For example, to retrieve all records with dates from January 1, 1998 through December 31, 1998, enter the following qualification in a date only field:

```
>=1-jan-1998 and <1-jan-1998
```

If the field contains *only* date values and no time values, you can retrieve all records for a single date, such as March 22, 1998, by entering a qualification such as:

```
22-mar-1998
```

If the field contains values in the date and time format and you want to retrieve only records with the specific date and time of March 22, 1998 10:44:23, enter the qualification:

```
22-mar-1998 10:44:23
```

If the field contains both the date and time and you want to retrieve all records for March 22, 1998, regardless of the time, enter the following date range qualification in the date and time field:

```
>=22-mar-1998 and <23-mar-1998
```

To qualify the retrieval for a specific hour of one day, enter a qualification such as:

```
>=22-mar-1998 10:00 and <22-mar-1998 11:00
```

## Queries in Blank and Nullable Fields

Query specifications differ between nullable and non-nullable fields. For general information on nulls and nullable columns, see the chapters "Using the Tables Utility" and "Working with Data Types and Data Display Formats."

In a *nullable* column, blank fields are treated as nulls (no data). In a query operation, entering the equals sign (=) into a nullable numeric column without anything else tells QBF to look for and return only rows in which the specified field is null.

In a *non-nullable* column, blank fields are treated as if they contain a value of zero (0). Entering the equals sign (=) into a non-nullable column without entering anything else returns all rows containing a value of zero (0) in that field.

## Order Operation—Sort Query Results

Use the Order operation to specify the order in which your retrieved data displays.

Choosing Order on the Retrieve frame temporarily clears all qualification specifications from the Retrieve frame form so that you can enter order qualifications. Any previously entered sort-order specifications displays.

The frame menu now displays the following operations:

### **Blank**

Clears any specifications that have been entered.

### **Cancel**

Restores the previous ordering specifications, if any, and returns to the previous Retrieve frame.

### **Help, End**

Standard operations.

## Sort Sequence and Sort Priority

You control the order in which data displays in your window by specifying the sort order. Sort order is governed by sort keys. Sort keys are data columns (fields) that have been given a sort priority number. For example, if a field called Name is the sort key, QBF presents all the data rows in alphabetic order by Name.

If you do not use the Order operation to specify a sort order, QBF displays the data according to how it is stored in the table. How data is stored in the table depends on the table's storage structure.

By default, QBF sorts columns in ascending order.

You can specify the sort order in master and detail tables, with or without table fields, for any query target.

For example, Lastname might be the first field listed on the form and Zip the fifth. But if you specify that you want zip-codes to be the primary sort key, and last names the second sort key, your data is displayed by zip-code in numeric order with the last names within each zip-code arranged alphabetically.

**To set the ordering sequence for a field**

1. Click Order on the Retrieve frame.
2. Put the cursor in a column (field) you want to sort on.
3. Type a number from 1 to 1024, which is the sort priority number.  
QBF sorts the field with the lowest number first. The first sort is called the primary sort. If you want the field displayed in descending order, follow the sort priority number by d (or D). For ascending order, you can enter a (or A) or no letter, because ascending order is the default.
4. If you want a secondary sort, repeat steps 2 and 3 in the secondary sort field. Make sure that the number you enter in the secondary field is higher than the number entered in the primary field. If you want additional sorts, continue this process until all the fields you want to sort on have been assigned a sort priority.
5. When you are finished, click End to return to the original menu for the Retrieve frame and remove the sort order specifications.
6. Click Go to execute the query.
7. If you choose Order again, the previous sort priority numbers and sort sequence letters display in the fields. To change them, type new values. Use the Blank operation to erase all sort specifications and begin with a blank form.

You can assign a sort sequence to any and all fields and table fields on the form.

The Order operation treats sort sequence numbers as relative. If you assign sequence numbers 1, 3, and 80, Order interprets them as follows:

- |    |           |
|----|-----------|
| 1  | primary   |
| 3  | secondary |
| 80 | tertiary  |

The following figure shows a JoinDef of the Projects and Tasks tables. The primary sort column is Dept. The secondary sort column is Budget. Because d is specified for Budget, the highest figure is displayed first. The third sort column (specified as 5) is Due Date, and the fourth (specified as 10) is Project.

When specifying sort order, the default is ascending order (A before B, 1 before 2). If you type d for descending order, the Order operation sorts text fields in reverse order (Z before Y, 9 before 8). If desired, you can use a colon (:) to separate the priority number from the sort direction.

For example, specifying either 2d or 2:d in the Budget field makes Budget the secondary sort key and instructs QBF to display the results in descending order according to Budget (highest budget first, lowest budget last).

Manager	Project	Description	Dept	Budget
	9		1	2d

Keep in mind that QBF sorts numbers in numeric fields differently than numbers in character fields. Numbers in numeric fields are sorted by value (1, 2, 3, 10, 11, 12, 101, 112, 121,...). When numbers are part of a character field (in an address, for example), they are sorted alphabetically (1, 10, 101, 11, 12, 121, 2, 3,...).

Nulls in a character field are sorted last.

### Sort Precedence in JoinDef Execution

QBF sorts Master/Master JoinDefs in the same way as tables. The fields can be sorted in any order.

In Master/Detail JoinDefs, the sorting sequence in a master field must have a higher precedence (lower sort priority number) than a sorting sequence assigned to a detail field. This rule ensures that the detail field remains subordinate to the master field.

### Go Operation—View Retrieved Records

After entering the data qualification and sort order specifications (if any), choose the Go operation on the Retrieve frame to display the data you have specified.

The operations available on the Retrieve frame while viewing data vary depending on the type of query target and whether the query target has a table field or simple fields.

## View Retrieve Results for a Table

If the query target is a table in simple-fields format, you view retrieve results one row at a time:

### Next

Retrieves the next record.

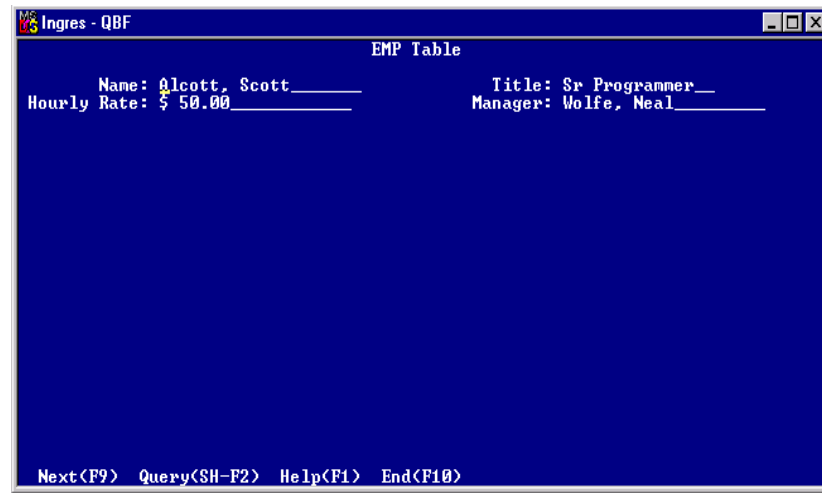
### Query

Clears the window of all retrieved information and allows you to start a new query.

### Help, End

Standard operations.

The following figure shows query results for a table displayed on a form in simple-fields format.



If the query target is in table-field format, you can view multiple rows in the table field. The operations on the menu are Query, Help, and End.

After QBF displays all rows retrieved by your query, regardless of format, it displays the following message:

No more rows in query

## View Sort Results for a Master/Master JoinDef

QBF displays the sorted results of a query on a Master/Master JoinDef with records from each table in simple fields or in table fields.

If the rows are displayed in simple-fields format, click Next to display the next single record.

The following are operations for a Master/Master join with simple fields:

### **Next**

Retrieves the next record (row) of data.

### **Query**

Clears the window of all retrieved information and allows you to make another query.

### **Help, End**

Standard operations.

If the rows are displayed in a table field, scroll through the data, if necessary, to see all rows. QBF displays an Out of Data message if you attempt to scroll past the end of the table field. You can suppress this message by changing the setting of the II\_SCROLL\_MSG environment variable/logical. For details, see the *System Administrator Guide*.

The following are operations for a Master/Master join with table fields:

### **Query**

Clears the window of all retrieved information and allows you to make another query.

### **Help, End**

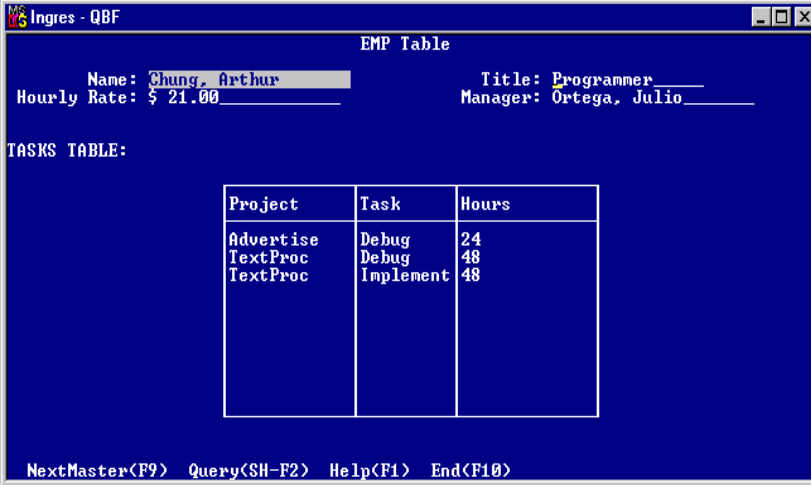
Standard operations.

### View Sort Results for a Master/Detail JoinDef

When you retrieve a master record in a Master/Detail JoinDef, QBF automatically retrieves all detail records associated with it. The master record is displayed in simple fields. The detail records display in either a table field or simple fields, depending on which format you chose when you created the JoinDef.

Regardless of the format, use the NextMaster operation to go from one master record to the next. Within each master you can view all associated detail records. If the display includes a table field for the detail records, QBF loads all detail records for a single master into the table field. If all the rows do not fit in your window, you can scroll through them. QBF displays an Out of Data message if you attempt to scroll past the end of the table field. You can suppress this message by changing the setting of the II\_SCROLL\_MSG environment variable/logical. For details, see the *System Administrator Guide*.

The following figure is an example of a retrieval for a Master/Detail JoinDef.



EMP Table

Name: Chung, Arthur Title: Programmer  
Hourly Rate: \$ 21.00 Manager: Ortega, Julio

TASKS TABLE:

Project	Task	Hours
Advertise	Debug	24
TextProc	Debug	48
TextProc	Implement	48

NextMaster<F9> Query<SH-F2> Help<F1> End<F10>

The preceding example includes a master row and its associated detail rows. The detail rows in the table field are sorted in ascending order, based on the Task column.



The following are the operations when records appear only in simple-field format:

**NextMaster**

Retrieves the next master record.

**Query**

Clears the window of all retrieved information and allows you to make another query.

**NxtDetail**

Retrieves the next detail record.

**Help, End**

Standard operations.

Use the NextMaster operation to display a master record and the first detail record associated with it. Use NxtDetail to display the next detail record associated with the same master; the master data cannot change. If you choose NxtDetail when the last detail for that master has been displayed, QBF displays the following message:

No more details for this master . . .

It then retrieves the next master and its first detail record.

When QBF has displayed all rows delivered by your query, it displays the following message:

No more masters

## No Rows Meeting Specifications

If QBF does not find any rows meeting the specifications in your query, it displays the message:

No rows found for this query

It then returns you to the Retrieve frame to enter a new query.

## Transaction Deadlock in Retrieve Mode

If a transaction deadlock occurs while you are viewing data retrieved by a query, QBF aborts the query and returns you to the Query Specification frame to enter a new query. When this occurs, QBF displays the following message:

Deadlock detected, your transaction has been aborted.

## Exit the Retrieve Operation

To set up another query specification, click End or Query. If you invoked the Retrieve operation from the command line with the qbf or query command, you can return to the operating system by choosing Quit.

## QBF Update Operation

The Update operation modifies, updates, or deletes existing data. Using Update, you fill in a form to retrieve rows of data, modify or delete the data in the form, and then save your results to the database.

## Start the Update Operation

You can start the Update operation from the operating system or from the Ingres Menu.

### To start the Update operation from the Ingres Menu

1. Choose Queries.
2. Choose QBF.

The QBF Start-Up frame displays. Choose QBFNames, JoinDefs, or Tables for the appropriate Catalog frame.

**Note:** When working with a table query target, choose SimpleFields or TableField format.

3. Place the cursor on the name of your query target in the Catalog frame and choose the Go operation. QBF displays the QBF Execution Phase frame.
4. On the QBF Execution Phase frame, choose Update.

QBF displays the Update frame, containing an appropriate query form for the query target you chose. At the top of the Update frame query form is the name of your query target.

The Update frame provides the following operations for use with the query form:

**Go**

Executes the query.

**Blank**

Clears the current entries from the frame.

**LastQuery**

Displays your last query specification for editing.

**Order**

Sets the order of rows for sorting.

**ListChoices**

Lists the available choices for the selected field on a QBFName.

**Help, End**

Standard operations.

5. Specify the row(s) you wish to retrieve for updating by filling in the appropriate fields on the Update frame query form. This procedure works in the same way as filling out the form on the Retrieve frame, as described in the chapter "Working with QBF Operations."
6. Choose Go to display the data that matches your specified criteria. You can then modify or delete this data as explained in Modifying Data and Deleting Data.

If QBF does not find any rows meeting your specifications, it displays the following message and returns you to the Update frame query form to enter a new query.

No rows found for this query

The chapter "Using QBF" discusses steps 1 through 4 in more detail.

## How Data Is Modified

QBF displays the data from your query on a data display form in the Update frame. You can edit or delete this data, with the following exceptions:

- If the table or tables you are accessing have permissions assigned to them, you need permission to perform an update.
- If JoinDef update rules specify that the table or tables you are accessing cannot be updated, you cannot update columns in that JoinDef.
- If JoinDef update rules have been applied to a join column on the frame, the cursor cannot stop on that field.
- Delete rules have been established to prevent deletion of the rows in the underlying tables.

For more information on restrictions for updating JoinDefs, see Update and Delete Rules in the chapter "Using JoinDefs in QBF."

To edit the data, type your corrections over the displayed text. QBF does not commit your changes to the actual database as you enter them, but stores them in a temporary buffer. To save your changes and update the database, choose Save on the Update frame data display form menu.

Within QBF, you can change only one row of data at a time. However, you can use the SQL update statement in the Interactive Terminal Monitor to change all qualifying rows of data at one time.

## Update Frame Data Display Form Operations

The operations for use with the Update frame's data display form vary, depending on the type of query target and whether the format is for a table field or simple fields. QBF automatically presents the operations that are appropriate for the query target and format you chose.

### **NextMaster**

Displays the next row of data from master and associated detail table fields.

### **NxtDetail**

Displays the next row of detail. (This operation is valid only for JoinDefs in simple-field format.)

### **AddDetail**

Adds a new detail record to the currently displayed master. (This operation is valid only for JoinDefs in simple-field format.)

### **Next**

Displays the next frame of data.

### **Query**

Returns to the Update frame query form and clears all fields so that you can enter a new query, including sort sequence and order values.

### **Delete**

Deletes the entry in the database, as shown on a simple-fields data display form. Displays a new menu to specify the deletion on a table-field form or Master/Detail JoinDef.

### **Save**

Saves the changes from this session in the database.

### **Help, End**

Standard operations.

## Add New Detail Rows to a JoinDef

Use the AddDetail operation on the QBF Update frame data display menu to add detail table records when both the master and detail table of a join definition are displayed as simple fields.

### To add a new detail record to a joindef

1. When a master table record and corresponding detail record are displayed, position the cursor on the first simple field for the detail table.
2. Type the data for the new record over the existing data in the currently displayed detail record.
3. Instead of *saving* the modified record, choose the AddDetail operation from the menu.

QBF writes the new detail record into a temporary buffer and redisplay the prior detail record (as it appeared before you overtyped it with the new data). If desired, you can continue to add additional records by repeating the preceding procedure. Be sure to choose AddDetail rather than Save to write the new record to the buffer; otherwise, you overwrite the existing record rather than add a new one.

4. Choose Save.

The new records are committed to the database.

*Do not* choose AddDetail until you have typed the new data into the existing record. If you choose this menu option before typing the new data, one of the following occurs:

- If duplicates are allowed, QBF enters the record that currently appears in the window as a duplicate record.
- If duplicates are not allowed, an error message displays.

## How Data Is Deleted

You use the Delete operation to erase an entire record (row) of data. The menu on the Update frame for the data display form includes this operation, regardless of the type of query target you specify.

As you delete rows, QBF writes the deletions to a temporary buffer. If you delete one row in a table field, the remaining rows scroll upward.

When you choose Delete on an Update frame with only simple fields displayed in the data display form, the menu is unchanged. In this display format, rows can only be deleted one at a time. QBF does not commit the deletions until you choose Save.

When you choose Delete for a data display form that displays a JoinDef or a form with a table field or both simple fields and a table field, the menu changes. It includes the operations appropriate to the various combinations of query target types and display formats listed next.

To delete a detail row in a JoinDef or in a form that includes a table field, place the cursor on the row to be deleted and choose Delete. When the Delete menu appears, choose DetailRow.

### **Master**

Deletes the master record currently displayed and all its detail records.

### **AllDetailRows**

Deletes all retrieved rows in the table field for the detail tables of the master currently displayed—QBF deletes all **rows** regardless of whether they are visible in the window.

### **DetailRow**

Deletes the single detail row indicated by the cursor in table-field display.

### **Detail**

Deletes the currently displayed detail row in simple-fields display.

### **ListChoices**

Lists the available choices for the selected field on a QBFName.

### **AllRows**

Deletes all retrieved rows in the table field, regardless of whether they are visible in the window.

### **Row**

Deletes the row indicated by the cursor.

### **Help, End**

Standard operations.

## How Updates Are Saved

QBF stores your modifications and deletions in a temporary buffer. To save the changes permanently, choose the Save operation on the data display form Update frame. As QBF writes your changes to the database, it displays this message:

Saving changes . . .

When the process is complete, QBF displays the following message and returns you to the Update frame query form so that you can specify a new query:

Changes saved.

When you choose the Save operation, QBF attempts to update the database with the current contents of *all* fields on the form, not just the field or fields that you changed. It also checks the contents of all fields on the Update data display form for data type consistency, integrity violations, and permission violations, and reports any errors.

## Update Operation and JoinDef Rules

When you update or delete rows using a JoinDef, be aware of the update and delete rules for that JoinDef. For more information, see Update and Delete Rules in the chapter "Using JoinDefs in QBF." If you attempt to save invalid changes, QBF does not commit your updates to those tables.

## Errors Reported During the Save Process

If you attempt to update rows for which you do not have permission, QBF does not commit your changes.

If you violate an integrity constraint (which guards against data of the wrong data type being entered), QBF displays a message to that effect. If this happens, you can:

- Press End to return to the field with the error and correct it.
- Press More to get more information about the error.



## Transaction Deadlock in Update Mode

Deadlocks can occur at two times during the QBF Update function. First, a deadlock can occur while you are editing rows of retrieved data. QBF displays the following error message:

Deadlock detected, your transaction has been aborted.

This means that you cannot update any more rows retrieved by your last query. However, any changes you have made before this point are still stored in the buffer. In this situation, QBF displays the following message:

Do you wish to save the changes made thus far?

Answering y (yes) is equivalent to selecting the Save operation. As QBF writes your changes into the database, it displays the message:

Saving Changes

The second type of deadlock can occur during the Save operation while your changes are being written from the temporary storage buffer to the database. In this case, QBF displays the message:

Deadlock detected, your transaction has been aborted.

In this situation, all changes made since the last Save operation are lost and you must begin again with a clear form.

## Exit the Update Operation

If you want to commit your modifications or deletions to the database before exiting the Update frame, choose Save from the data display form's menu on the Update frame. This also returns you to the Update frame's query form.

To leave the Update operation, choose End from the query form's menu on the Update frame. End returns you to the Query Execution frame.

If you make changes to the data and try to exit the Update frame's data display form before executing a Save operation, QBF asks:

Do you wish to leave UPDATE without saving changes?

If you type y (yes), QBF allows you to leave without saving your work. If you type n (no), QBF redisplay the Update frame's data display form, enabling you to select the Save operation.



# Chapter 6: Using JoinDefs in QBF

---

This section contains the following topics:

[JoinDefs](#) (see page 119)  
[JoinDef Rules](#) (see page 121)  
[Join Types](#) (see page 124)  
[Automatic Joins](#) (see page 128)  
[Fields on a JoinDef Form](#) (see page 128)  
[How You Create a JoinDef](#) (see page 129)  
[JoinDefs Catalog Frame](#) (see page 132)  
[JoinDef Definition Frame](#) (see page 133)  
[How Columns Are Joined](#) (see page 136)  
[Single-Table JoinDefs](#) (see page 139)  
[Update and Delete Rules](#) (see page 141)  
[JoinDef Change Display](#) (see page 143)  
[Test JoinDefs](#) (see page 145)  
[Save JoinDefs](#) (see page 145)  
[Edit JoinDefs](#) (see page 146)  
[Delete JoinDefs](#) (see page 147)

## JoinDefs

A *JoinDef*, or join definition, is a definition of a virtual table that conceptually joins together multiple tables in your database. A JoinDef does not exist as an actual table in the database, but temporarily links the information in different tables by their common columns. This enables you to work with the data in the tables simultaneously, as if it was contained in one table.

For example, in a relational database in which each table typically is devoted to information related to a single item of interest, you can have the following separate tables:

- Staff (employee's name, hourly rate, title, and manager)
- Tasks (employee's name, project identification, tasks, and the hours spent on each task)

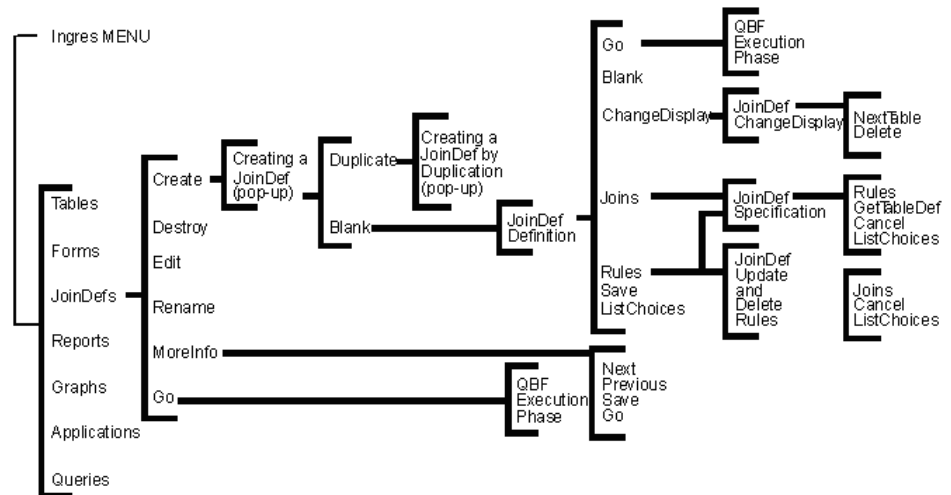
If you wanted to find out what tasks have been assigned to employees who work for a certain manager, you would need to link the information in the Staff and Tasks tables by employee name.

When you use a JoinDef to query multiple tables, you gain access to the data in all of the columns in all of the joined tables. When you use a JoinDef to append new data or update existing data, QBF writes identical information into the common columns of the different tables in the JoinDef. For example, if a common Lastname column links two tables, the same last name is simultaneously written into both Lastname columns. QBF writes changes or additions you made in non-common columns only to the table containing that column.

A JoinDef is similar to a view in that you can use either to display data contained in more than one table. However, whereas a JoinDef contains *all* of the columns from the joined tables, a view can contain only *selected* columns from the tables on which it is based. Also, you cannot use a view to append new data or modify existing data in the underlying tables, as you can with a JoinDef. You create JoinDefs with QBF, but you must use a query language to create a view. Once created, QBF lists a view as if it were a table, but always lists JoinDefs separately from tables. For additional information on views, see your query language reference guide.

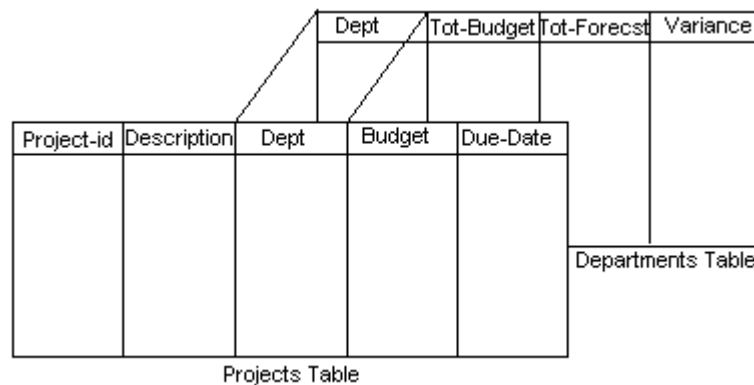
You can use JoinDefs in QBF, RBF, and VIFRED. However, you cannot manipulate or access JoinDefs from within the Tables Utility, as you can a view.

The map in the following figure illustrates the route QBF takes when you choose JoinDefs from Ingres Menu or from the QBF menu:



## JoinDef Rules

To be linked with a JoinDef, two or more tables must have at least one column with the same data type and one or more common values in those columns. For example, if a table named Projects and a table named Departments both have a column named Dept, which contains names of departments, the common values in the two Dept columns join the two tables in the JoinDef, as shown in the following figure:



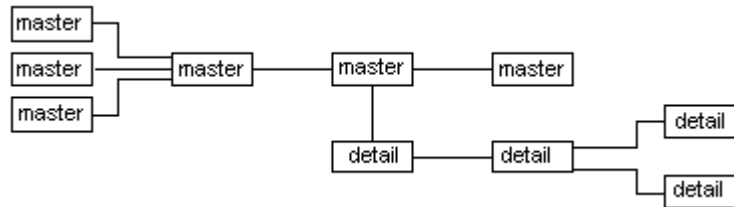
While the two columns in this example have the same name, this is not required for a JoinDef.

If you think of tables as two-dimensional grids of columns and rows, then JoinDefs are the third dimension. JoinDefs can be described as corridors linking the columns of two tables.

JoinDefs must follow these rules:

- The two join columns need not have the same name.
- Join column pairs must either be of the same basic data type, such as an integer column joined to another integer column, or be of data types that are coercible to one another.
- Except in a Master/Detail join, the data values in all the join columns must be identical in the joined tables in order for a query to return that row.
- A JoinDef can join a total of 10 tables, either Master or Detail, that together total no more than 600 columns.
- You can specify up to 50 columns in each table as join columns.
- You can create a JoinDef for a single table, which has several advantages over using the table itself as a query target. See Single-Table JoinDefs (see page 139).

The figure below shows some examples of JoinDefs. As long as you do not exceed the total limit of 10 tables, you can join one table to as many other tables as you desire. However, any given JoinDef can have only one Master-Detail join.



Queries only return those records that have identical data values in all the join columns. Thus, if the Projects and Tasks tables are joined by the Project column and they share only the single project name of Release6, a query returns only one row of data from each table. If they share the project names Release6 and Popup, then a query returns two rows from each table.

## Join Columns with Coercible Data Types

Data columns are *coercible* when their data type can be changed from one to the other. An integer data type, for instance, can be changed (coerced) into a floating-point data type. In the same way, a money data type can be changed into a floating-point data type. But a character data type cannot be coerced into a numeric type.

When you join two columns that have coercible but not identical data types, QBF displays a warning message, and QBF adopts the data characteristics of the table column listed first in the JoinDef.

For example, if two character columns of different lengths are joined, the data display on the QBF form uses the length of whichever column was specified first in the JoinDef. The same principle holds true for column nullability and data type. If two columns, one nullable and the other not, are joined, Query-By-Form treats them both as nullable or not nullable according to which was listed first in the JoinDef. If a floating-point column is joined to an integer column, QBF uses the data type of the first one.

## Multiple Join Columns

The maximum number of join columns in a JoinDef is 50. When you use multiple columns in a join other than a Master/Detail join, every set of join columns must have common values in order for the query to return a row. For example, if one set of join columns contains last names, and a second set of join columns contains first names, a query returns only the rows in which the first and last names are both identical. Thus, if two joined tables contained the first name Thomas and last name Jefferson in a particular row, a query returns data for that row from both tables. But if one table contained the first name Thomas and last name Jefferson, and the other contained the first name Thomas and last name Becket, a query cannot return any rows because only the Firstname join columns had a name in common.

## Join Types

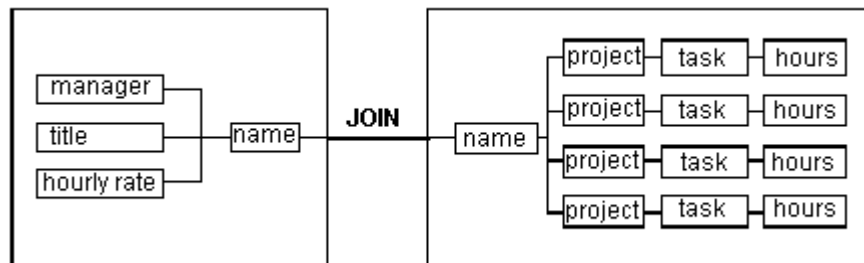
QBF allows two types of joins:

- **Master/Master** (one-to-one)—These JoinDefs correspond to an ordinary *relational equi-join*.
- **Master/Detail** (one-to-many)—This type of JoinDef corresponds to a *relational outer join*.

The terms *master* and *detail* are meaningful only as they relate to each other. The determining factor in whether a join is one-to-one or one-to-many is the nature of the data in the tables you are joining.

Each column of each table in a *Master/Master* join contains singular or exclusive properties. For example, the Staff table lists each employee only once and the Managers table lists each manager only once. If you join the Name column in the Staff table to the Manager column in the Manager table, QBF returns one row of data for every employee who is also a manager.

In a *Master/Detail* join, each detail table contains at least one column that is not restricted to one item. That is, the column contains non-singular or non-exclusive information. The join column—the common element—is the bridge between each column in the master table and the column(s) in the following detail table:



For example, in the Tasks table both the Project and Task columns contain non-singular information. Wallace Fielding, for instance, might have three rows showing that he works on the Portfolio project (Project column). Within the Portfolio project, he works on three different tasks: Design, Manage, Test (Task column). Each Portfolio task is on a separate row in the table.



If the EMP table (master) were joined to the Tasks table (detail) through the Name column in each, you could access information such as the tasks, projects, and hours (from the Tasks table) of employees who work for a certain manager (from the Staff table).

Thus, the many entries in the Task column of the Tasks table are accessible through the one entry in the Manager column of the Staff table. The following figure shows Jones, Betty, in the Manager field of a form for the EMP table.

The screenshot shows a window titled "Ingres - QBF" with a form titled "EMP Table". The form has fields for "Name:", "Hourly Rate:", "Title:", and "Manager:". The "Manager:" field contains the text "Jones, Betty". Below these fields is a section labeled "TASKS TABLE:" which contains a table with three columns: "Project", "Task", and "Hours". The table is currently empty. At the bottom of the window, there is a menu bar with the following options: "Go<F9>", "Blank<SH-F2>", "LastQuery<SH-F3>", "Order<SH-F4>", and ">".

Project	Task	Hours
---------	------	-------

The following figure displays a record retrieved for one of the employees she manages.

The screenshot shows a window titled "Ingres - QBF" with a form titled "EMP Table". The form has fields for "Name:", "Hourly Rate:", "Title:", and "Manager:". The "Name:" field contains the text "Fielding, Wallace", the "Hourly Rate:" field contains "\$ 47.00", the "Title:" field contains "Project Leader\_", and the "Manager:" field contains "Jones, Betty". Below these fields is a section labeled "TASKS TABLE:" which contains a table with three columns: "Project", "Task", and "Hours". The table contains three rows of data. At the bottom of the window, there is a menu bar with the following options: "NextMaster<F9>", "Query<SH-F2>", "Help<F1>", "End<F10>", and ">".

Project	Task	Hours
Portfolio	Design	16
Portfolio	Manage	8
Portfolio	Test	8

## Master/Master JoinDefs

When used for a query, a *Master/Master JoinDef* selects records from two or more tables based on the common values in the join columns. While the respective join columns must share at least one common value, they need not share all values. However, only records containing common values are returned in a query.

For example, suppose two tables are joined by the Lastname column that both tables have. The Lastname column of the first table contains the names Adams, Baker, Choy, and Diaz, while the Lastname column of the second table contains the names Adams, Baruch, Choy, and Donelli. In this case, a query returns only the records in both tables that contain the names Adams and Choy.

When you create a Master/Master JoinDef, you enter the table name and whether it is a master or detail table on the JoinDef Definition frame. For more information, see *How You Create a JoinDef* (see page 129).

In the following figure is an example of a Master/Master JoinDef on the JoinDef Definition frame.

Ingres - QBF

QBF - JoinDef Definition Form

JoinDef Name: staff\_home\_addr

For each table in the JoinDef, enter table name (with optional abbreviation for table name) below. For Master/Detail JoinDefs enter Master or Detail under Role. (Default is Master if blank.)

Role	Table Name	Owner	Abbreviation
MASTER	emp	ingres	e
MASTER	home	ingres	h

Table Field Format? (y/n): NO

Select the "Go" menu item to run the Join Definition.

Go(F9) Blank(SH-F2) ChangeDisplay(SH-F3) Joins(SH-F4) >

The following figure shows the data retrieved from the JoinDef displayed in the preceding figure.

The screenshot shows a window titled "Ingres - QBF" with a dark blue background. It displays data from two tables: "EMP Table" and "HOME Table".

**EMP Table**

Name: <u>McCott, Scott</u>	Title: <u>Sr Programmer</u>
Hourly Rate: \$ <u>50.00</u>	Manager: <u>Wolfe, Neal</u>

**HOME Table**

Address: <u>1212 1st St.</u>	City: <u>New York</u>
State: <u>NY</u>	Zip: <u>11787</u>
Phone: <u>516-555-5515</u>	

At the bottom of the window, there is a status bar with the text: "Next(F9) Query(SH-F2) Help(F1) End(F10)".

## Master/Detail JoinDefs

When QBF executes a query in which the query target is a *Master/Detail JoinDef*, the data appears to have come from two tables—one that is a set of all the master tables and one that is a set of all the detail tables. The master table information is displayed as simple fields, and the detail table information is displayed as either a table field or simple fields.

If there are no detail records for the master record, QBF still retrieves the master record. Thus, QBF retrieves and displays master records even if there are no corresponding identical values in the detail table join column.

A JoinDef can contain several Master/Master joins, but only one Master/Detail join. You can include multiple detail tables relative to the master table(s) in a Master/Detail JoinDef by joining them to the detail table in the Master/Detail join. For example, the second figure in this chapter shows a Master/Detail relationship combining only one pair of tables. This figure also shows several pairs joined together with Master/Master joins and several Detail tables joined to each other. For illustrations of a Master/Master JoinDef and resulting data, see the preceding two figures.

## Automatic Joins

Columns involved in a join need not have the same name, just the same data type and common values. However, when tables specified in a join have columns with the same name and data type, those columns automatically become the join column. This is known as an *automatic*, *default*, or *natural* join. QBF displays a list of join columns on the JoinDef Specification frame, described later in this chapter. You can specify different columns as the join columns or remove columns from the join columns list. For more information, see *How Columns Are Joined* (see page 136).

Keep in mind that when two tables have multiple join columns, all the data values in each join column must be identical for a query to return that row of data. For this reason, you must delete unneeded joins.

## Fields on a JoinDef Form

You can use a JoinDef to append, retrieve, or update data. The JoinDef form contains fields representing columns in all of the joined tables. On a default form, QBF marks the common columns—the join fields—by displaying them in reverse video if your monitor has that capability.

## How You Create a JoinDef

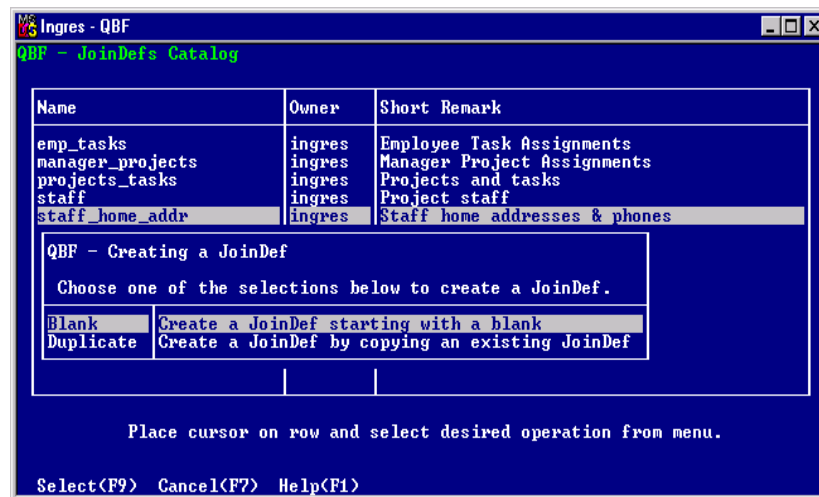
Creating a JoinDef requires specifying the tables, the common join columns, the update and delete rules, and the columns to be displayed.

The following steps describe the general process of creating, testing, and saving a JoinDef. (The steps and frames are all discussed in more detail later in this chapter.)

1. Choose JoinDefs from either the Ingres Menu or from the QBF Startup frame.
2. Choose Create from the JoinDefs Catalog frame.

The Creating a JoinDef pop-up form appears.

3. Place the cursor on one of the available options and choose the Select operation:
  - Blank (to create a JoinDef from a blank frame)
  - Duplicate (to base a JoinDef on an existing JoinDef definition)

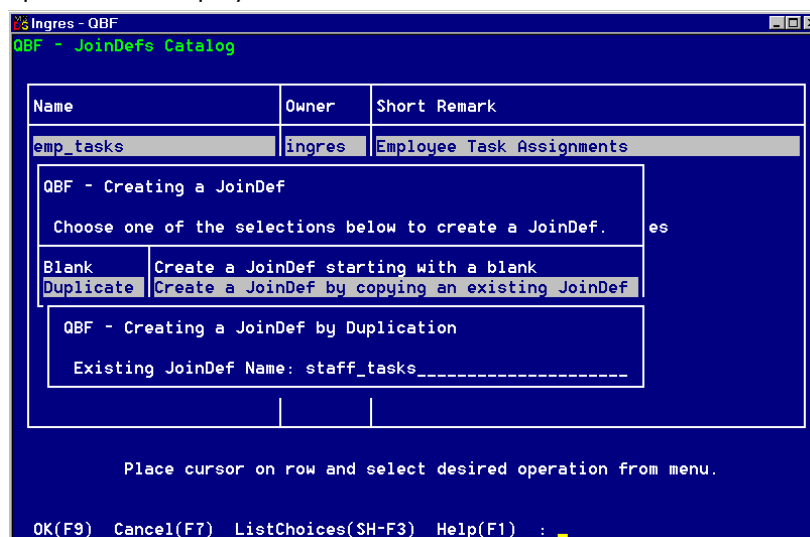


If you are basing the new JoinDef on an existing one, the Creating a JoinDef by Duplication pop-up form appears. (See the following figure.) Proceed to Step 4.

If you are creating a JoinDef from a blank frame, the JoinDefs Catalog frame appears. Proceed to Step 5.

4. Enter the name of the existing JoinDef on the Creating a JoinDef by Duplication pop-up form and choose the Select operation.

If you do not know the name of the existing JoinDef, select the ListChoices operation to display a list of the available choices.



5. When the JoinDefs Catalog frame appears, choose the Create operation.  
See JoinDefs Catalog Frame (see page 132) and JoinDef Definition Frame (see page 133).
6. Type the JoinDef name in the space following JoinDef Name and press Return.  
  
Perform the next three steps for each table you want to join. Use Tab and the arrow keys to move from column to column and row to row as needed.
7. Type the kind of table it is (Master or Detail) in the Role column. If none of the tables has a Master/Detail relationship to each other, enter them all as Master tables even if some or all of them are Detail tables on other JoinDefs. Enter as a Detail table any table that has a detail relationship (many records to one record) to any master table.
8. Type the table name in the Table Name column.  
  
If you do not know the table name, use the ListChoices operation to display a list of the available tables.
9. Type an optional abbreviation for the table name in the Abbreviation column.
10. Type **n** (no) in the Table Field Format field if you want to view all your data in simple-fields format.  
  
The default is to display the data in table-field form.

11. Choose Joins to display the JoinDef Join Specification frame. If QBF has automatically established join columns to join the tables, they are displayed. If necessary, enter or change the join specifications. For more information, see [How Columns Are Joined](#) (see page 136).
12. Return to the JoinDef Definition frame and save your JoinDef by selecting the End operation.

Before or after you save your JoinDef, you can perform either or both of the optional enhancements listed below.

## Optional JoinDef Specifications

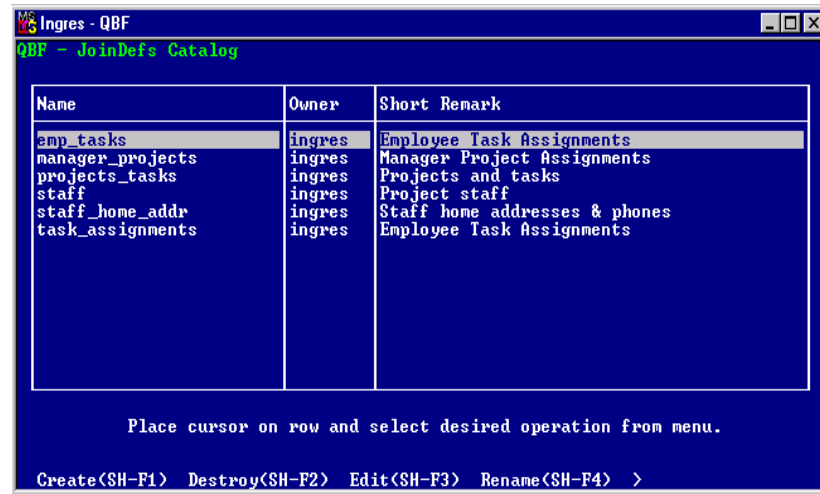
You can further enhance your JoinDef specification by performing one or both of the two optional operations listed below. (If you perform either of these operations, be sure to save the changed JoinDef when you are finished.)

- To establish update and delete rules for the join columns, choose Rules to display the JoinDef Update & Delete Rules frame. For more information, see [Update and Delete Rules](#) (see page 141).
- To hide one or more columns from view on the default query form created by the JoinDef, choose ChangeDisplay to display the JoinDef Change Display frame (see page 143).

To run a JoinDef, choose the Go operation. The QBF Execution Phase frame appears. Choose Append, Retrieve, or Update. A default form for your new JoinDef appears. Use the menu items as desired.

## JoinDefs Catalog Frame

The JoinDefs Catalog frame, shown in the following figure, displays a list of existing JoinDefs and a menu of operations for working with JoinDefs.



The menu operations for the JoinDefs Catalog are as follows:

### Create

Guides/enables creation of a JoinDef.

### Destroy

Removes a JoinDef from the database.

### Edit

Enables editing or viewing of an existing JoinDef.

### Rename

Changes the name of an existing JoinDef.

### Go

Runs a query on an existing JoinDef.

### MoreInfo

Displays additional information about the JoinDef.

### Help, End, Quit

Perform standard operations.



## JoinDef Definition Frame

Use the JoinDef Definition frame to specify the tables to be included in a JoinDef and their roles (Master or Detail), and whether the JoinDef must include table fields.

JoinDef Name: emp\_tasks

For each table in the JoinDef, enter table name (with optional abbreviation for table name) below. For Master/Detail JoinDefs enter Master or Detail under Role. (Default is Master if blank.)

Role	Table Name	Owner	Abbreviation
MASTER	emp	ingres	emp
detail	tasks	ingres	tasks

Table Field Format? (y/n): YES

Select the "Go" menu item to run the Join Definition.

Go(F9) Blank(SH-F2) ChangeDisplay(SH-F3) Joins(SH-F4) >

The menu items for the JoinDef Definition Form are as follows:

### Go

Executes the JoinDef.

### Blank

Erases the current entries in the window.

### ChangeDisplay

Drops unnecessary columns from the JoinDef specification.

### Joins

Displays the JoinDef Specification frame for viewing or changing the join columns.

### Rules

Establishes rules for deleting or updating data in the tables included in the JoinDef.

### Save

Saves the JoinDef.

### ListChoices

Lists the available choices for the selected field.

### Help, End, Quit

Perform standard operations.

## JoinDef Name

On the JoinDef Definition frame, name a new JoinDef by typing a unique name of no more than 32 letters or numbers. A database cannot have two JoinDefs with identical names. For naming conventions in databases compliant with ANSI/ISO Entry SQL-92 standards, see the chapter "Fundamentals of Using Querying and Reporting Tools."

If you create a JoinDef based on an existing JoinDef, enter a different name for the new JoinDef on this frame.

## Role Column in JoinDef Definition

In the Role column, type either Master or Detail for the role of each table. You can use m or d as abbreviations.

At least one table in every JoinDef must be entered as a Master table. Keep in mind that Master and Detail are relative terms. You can have as many Master tables in a JoinDef as you need (up to the maximum of 10). You must designate a table (or tables) as a detail table only if it has a many-to-one relation to one or more of the master tables. For example, the Staff table lists an employee only once, while the Tasks table can list that employee many times for many different tasks. Thus, the Tasks table has a many-to-one relationship to the Staff table and is designated as a detail table in a JoinDef that links Staff to Tasks.

A table that is a master in one JoinDef can be a detail table in some other JoinDef. You need to specify each table's role as it relates to the current JoinDef.

For example, the Staff table contains each employee's name, title, hourly rate, and manager. The Manager table contains the names of managers and their titles. If you joined these two tables on the Manager column in each table, the result is a Master/Detail join because one manager in the Manager table might be the manager for several different employees. Thus, this join could be used to list those employees supervised by each manager.

However, if you joined these two tables on the Manager column in the Manager table and the Name column in the Staff table, the result is a Master/Master join because those employees who were managers are listed only once in each table. You can use this join to display the title and hourly rate of those employees who were managers.

A complicated JoinDef containing many tables can have multiple master and multiple detail tables.

## Table Name Column in JoinDef Definition

In the Table Name column, enter the name of each table. You can enter a maximum of 10 names (that is, you can join a maximum of 10 tables in a JoinDef).

## Owner Column in JoinDef Definition

In the Owner column, enter the name of the schema to which the table belongs; this also substantiates ownership. If you do not enter a name here, QBF assumes the owner is the same as the current user ID. For more information on schemas, see Schemas for Owner Qualification (see page 37).

## Abbreviation Column in JoinDef Definition

An *abbreviation* is an alternate, usually shortened, name for a table. For example, the Projects table could be abbreviated to the letter p. Abbreviations are also sometimes known as *correlation names* or *range variables*. If you do not specify an abbreviation, QBF assigns the table name as the abbreviation.

Abbreviations are necessary when you want to join two identically named tables with different owners, or to join a table to itself. In these cases, use an abbreviation for one of the table names when you specify the join. For more information, see Single-Table JoinDefs (see page 139).

## Choosing Table-Field Format

The *Table Field Format?* or *Simple-Fields Format?* question controls how this JoinDef displays data:

- **Master/Master**—If the JoinDef contains all master tables, answer this question with y (yes) to specify table format for all data. Answer this question with n (no) to display all data one record at a time in simple-field format.
- **Master/Detail**—If the JoinDef contains both master tables and detail tables, answer this question with y (yes) to specify simple-field format for all data for all the master tables, and table format for all data for the detail tables. Answer this question with n (no) to display all data for all tables one record at a time in simple-field format.

Table-field format is particularly useful for Master/Detail joins, where many detail rows in the table field can correspond to a single row in the master table.

## How Columns Are Joined

When you create a JoinDef on the JoinDef Definition frame and select either Go or Save, QBF selects as the join columns the columns in the two tables that have the same name. QBF then compares the data types, leading to one of three possible situations:

- The data types are the same, in which case QBF completes the join.
- The data types are not the same, but are coercible.

This means that the data types can be changed from one to the other. For example, a money data type can be coerced into a numeric type. In this situation, QBF displays a warning message, but completes the join.

- The data types are different and not coercible.

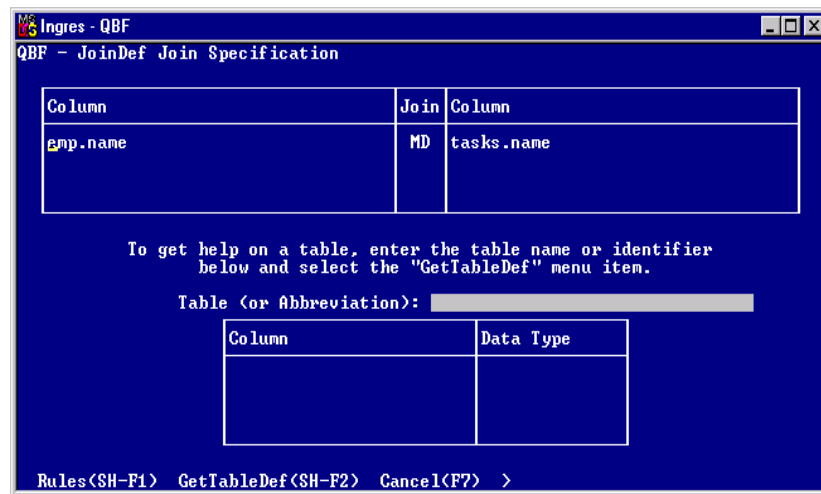
For example, a character data type cannot be coerced into a numeric type. QBF prompts you to modify your joins using the JoinDef Specification frame to establish proper joins.

## JoinDef Specification Frame—Specify Other Components of the JoinDef

After you complete the JoinDef Definition frame, you can specify other components of the JoinDef—the join columns, the update and delete rules, and the display characteristics. To reach the JoinDef Specification frame, choose the Joins operation.

The JoinDef Join Specification frame includes a list at the top of the frame that details existing joins, a table field for displaying information about table column names, and a menu of operations.

You can use the JoinDef Join Specification frame to change Join columns.



For example, in a JoinDef containing the tables Staff and Tasks, QBF selects the column called Name in each table as the join column. If there are no common columns with the same name, the JoinDef Join Specification frame is displayed empty, and you must enter the join columns you wish to use.

### Rules

Establishes rules for deleting or updating data in the tables included in the JoinDef.

### GetTableDef

Gets information on columns in a table.

### ListChoices

Lists the available choices for the selected field.

### Cancel

Cancels any changes made to the join specification and returns to the JoinDef Definition frame.

### Help, End

Perform standard operations.

## View or Change Joined Columns

In the following figure, the Name column in the Staff table is the join column with the Manager column of the Manager table. The MM between the two-table/column names indicates that this is a Master/Master join.



The table fields at the top of the JoinDef Join Specification frame are as follows:

### Column

Contains the column name of join column(s) in the table. Column name format: *abbreviation.columnname*, where *abbreviation* is the abbreviation of the table name and *columnName* is the column name. In the above figure, the column name m.manager means the name column from the Manager table. The column on the left is joined to the column on the right.

### Join

Indicates whether the join is Master/Master (MM), Master/Detail (MD), or Detail/Detail (D/D).

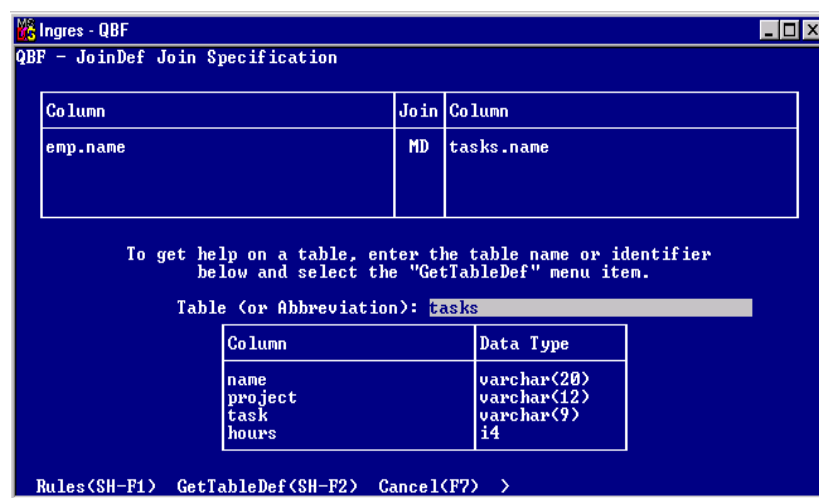
### Column

Contains the column name of join column(s) in the second table. Column name format is the same as for the first column. In the above figure, the column name s.name means the name column from the Staff table.

You can add, delete, or change join specifications by editing the table field at the top of the frame. If you alter the join specification, make sure that the two join columns in each table are of the same data type.

## Get Information on Table Column Data Types

To display a list of a table's columns and column formats, enter the name (or abbreviation) of the table and select the GetTableDef operation. That table's columns and their data types are then displayed, as shown in the following figure.



## Exit the JoinDef Join Specification Frame

When you exit the JoinDef Join Specification frame by selecting Rules or End, QBF checks the validity of your entries and reports any errors it finds. If the join information is valid, you exit the frame. If it is not valid, QBF displays an error message that indicates the problem and returns you to the frame so that you can correct it:

- Select Rules to set rules for modifying the tables in your JoinDef. For more information, see Update and Delete Rules (see page 141).
- Select the End operation to return to the JoinDef Definition frame.
- Select the Cancel operation to cancel any changes made to the JoinDef Join Specification frame and return to the JoinDef Definition frame.

## Single-Table JoinDefs

Although you can use a table *as is* for a query target, if you define a table as a single-table JoinDef you can:

- Specify that the table is displayed in table-field format rather than simple-field format.
- Set up a Master/Detail or Master/Master relationship between different fields (columns) in the same table.
- Delete columns containing sensitive or unnecessary information from the display by using the ChangeDisplay operation.
- Protect columns from updates but allow queries to display data by using the Rules operation.
- Disable deletion of rows by using the Rules operation.
- Save these formatting features in the database for later execution.

## How You Build a Single-table JoinDef

To build a single-table JoinDef, follow these steps:

1. Follow Steps 1 through 4 in Creating a JoinDef, earlier in this chapter, to get to the JoinDefs Catalog frame.
2. Choose Create on the JoinDefs Catalog frame.
3. Enter a name for the JoinDef.
4. On the JoinDef Definition Frame, enter master in the Role column, type the name of the table you wish to use in the Table Name column. In the Owner column, type the name of the schema to which the table belongs, if appropriate. Then enter an optional abbreviation for the table in the Abbreviation column.

If you are joining a table to itself in order to create a Master/Detail or Master/Master relationship between two columns in the same table, perform Step 5. Otherwise, skip to Step 6.

5. To establish a Master/Detail or Master/Master relationship between two columns in the table, move the cursor down to the second line. Enter d (detail) or m (master) in the Role column of the second line, as appropriate. After entering the role, enter the table name and, if necessary, the table's schema name. Then enter a *unique* abbreviation for the table in the Abbreviation column.
6. When you have finished specifying the tables, specify the field format. Table-field format is the default, which you can change to No if desired.

If you are simply creating a single-table JoinDef in order to establish display or update and delete rules, skip to Step 9.

7. Choose the Joins operation to display the JoinDef Join Specification frame.
8. To create a Master/Detail join between two different data columns in the table, type blank spaces over the default join specifications established by QBF to delete them.

Specify the join by entering [*schema.*]*tablename.columnname* for the join's master data column in the left side of the form and *alternateabbr.columnname* for the detail data column in the right side.

Whenever you enter a column name in the right side of the form, you always use the alternate abbreviation for the table.

9. Choose the End operation to finish specifying the join columns.
10. Choose the Go operation. The Query Execution frame is displayed.

You can choose the Rules operation to use the JoinDef update and delete rules frame to modify update and deletion rules. You can use the JoinDef Change Display frame to hide specified columns from view when you use the JoinDef form. The following sections provide details.



## Update and Delete Rules

A JoinDef allows you to modify data in single or multiple tables using a form. You can use the form to delete or update information in the tables. QBF provides update and delete rules that allow you to control the impact of changes made when executing queries on multiple tables. These rules allow you to specify the fields in which a user can perform updates and the tables from which a user can delete records.

To reach the JoinDef Update & Delete Rules frame, choose Rules on the JoinDef Definition frame or the Join Specification frame.

### Joins

Goes to the JoinDef Specification frame.

### Cancel

Cancels any changes made in this frame and returns to the JoinDef Definition frame.

### ListChoices

Lists the available choices for the selected field.

### Help, End

Perform standard operations.

## Default Update and Delete Rules

The default update and delete rules are shown in two lists on the JoinDef Update & Delete Rules frame.

Update Information: To enable modification of join fields in UPDATE mode, enter "Yes" under Update? column.

Column	Update?
emp.name	no
tasks.name	no

Delete Information: To disable deletion of rows in a table during UPDATE mode, enter "No" under Delete? column.

Role	Table Name (or Abbreviation)	Owner	Delete?
MASTER	emp	ingres	yes
detail	tasks	ingres	yes

Joins<SH-F1> Cancel<F7> ListChoices<SH-F3> Help<F1> >

The Update default is No and the Delete default is Yes for all columns in all tables in the JoinDef. This means that, by default, updates to the join column or columns are not allowed, but records can be deleted from any table in the JoinDef.

You can browse through defaults and modify them to suit your needs. Change the Yes and No values in the last column in each list to specify whether or not data in a particular table or column can be updated or deleted. To do so, move the cursor to the value and type **n** (no) or **y** (yes) over the current entry.

## Determine Update Rules

Each row in the Update Information list on the JoinDef Update & Delete Rules frame lists a table in the join and a join column in that table. If a table has more than one join column, the table name appears in more than one row.

The list in the Update column shows whether updates are permitted for the join column. The default for all join columns is No. This means that any changes to values in the join column cannot be entered into the database. Yes in the Update column allows changes to go into the database. You can allow updates to a join column in one table and yet deny them in the corresponding join column of the second table.

These update rules only apply to join columns. The other columns in a table can be updated by the user. If you want to specify that a column that is not a join column cannot be updated, you must edit the form with VIFRED.

Changing update rules can control the movement of the cursor through the fields on a form for a JoinDef. If neither table allows updates to a join column, that field becomes a display-only field on the JoinDef form and you cannot move the cursor into it.

## Determine Delete Rules

The list in the Delete Information section on the JoinDef Update & Delete Rules frame contains one row for each table in the JoinDef. This row specifies its type (master or detail), table name or abbreviation, and whether or not rows can be deleted from that table. Again, only the Delete list can be modified. Move the cursor to the entry and type the new value. Changing Yes to No specifies that rows cannot be deleted from that table. Menu choices during query execution change to fit the rules you have specified.

## Exit the JoinDef Update and Delete Rules Frame

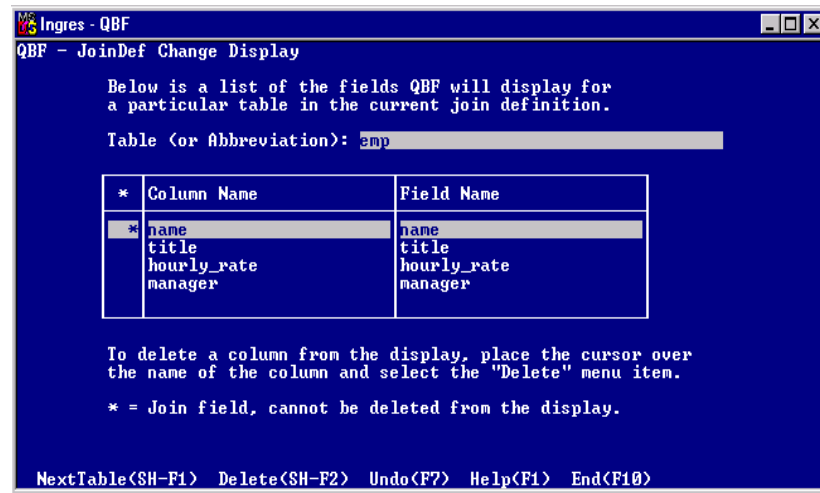
To go to the JoinDef Join Specification frame, choose Joins. Click End to finish and return to the previous frame. Click Cancel to cancel any changes and return to the JoinDef Definition frame.

## JoinDef Change Display

You can decide not to display certain columns on a form when you execute a JoinDef. For example, you can hide blank columns or columns containing confidential information in one of two ways:

- To avoid displaying certain columns, build a QBFName from a custom form edited with the VIFRED. For more information about QBFNames, see the chapter "Using VIFRED."
- Use the ChangeDisplay operation as explained below to delete the display of a column from a QBF form.

When you choose ChangeDisplay on the JoinDef Definition frame, the JoinDef Change Display frame appears, as in the following figure.



**NextTable**

Displays the next table in the JoinDef.

**Delete**

Deletes a column from the display of the JoinDef when a query is executed.

**Undo**

Backs out the effects of the last Delete or Undo action.

**Help, End**

These are standard operations.

On this form, you can scroll the Table name field to see the entire table name, *schema.tablename*, if necessary.

The Column Name column and the Field Name column both list the names of table's columns. In most cases, the names listed in each column are identical. However, when joined tables have non-join columns with the same name, QBF changes one of the internal field names for JoinDef identification purposes. In this case, the Column Name column lists the original column name as it still appears in the table, and the Field Name column gives the altered name as it appears on the JoinDef.

For example, suppose the Manager table is joined to the Staff table by joining the Manager.Manager column to the Staff.Name column. In this example, both tables have a column named Title that is not the join column. On the JoinDef Change Display frame for the Manager table, both Column Name and Field Name display title, but on the Change Display frame for the Staff table, Column Name shows title and Field Name shows title0.

## Delete Fields from JoinDef Displays

Use the Delete operation on the JoinDef Change Display frame to remove unwanted columns from the display. If the JoinDef contains more than one table, you must delete columns from each table separately.

Columns deleted on this frame are deleted from the JoinDef form only, and not from the database itself.

To delete a column from a table in the JoinDef, first display the table on the form. Select the NextTable operation to display the proper table, if necessary. Place the cursor on the row you want to delete and choose the Delete operation. JoinDef join columns are indicated by an asterisk (\*) in the leftmost column. JoinDef join fields cannot be deleted.

If you change your mind about deleting a row, click Undo before pressing any other key.

## Exit the JoinDef Change Display Frame

Choose the End operation to return to the JoinDef Definition frame.

You can then click Save to save the changes. If you click Quit without saving, QBF reminds you of this and asks whether you want to quit anyway.

## Test JoinDefs

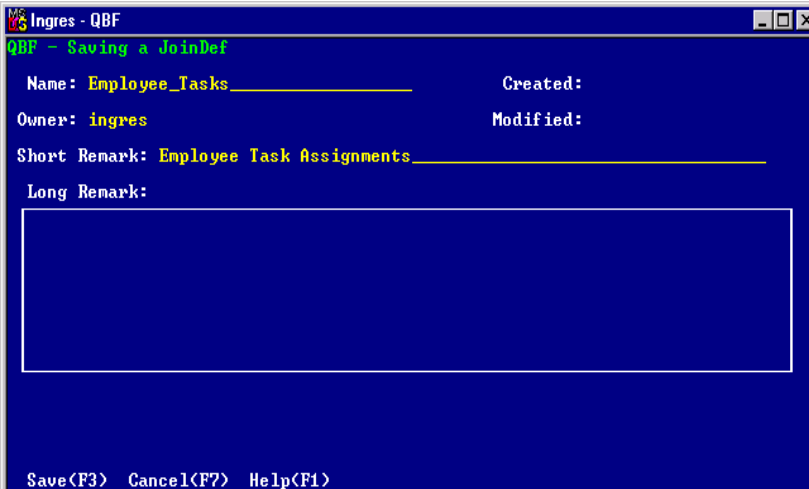
QBF provides a convenient way to test your JoinDef while you are still defining it. At the JoinDef Definition frame, choose the Go operation. QBF builds a form for the JoinDef and automatically goes into the query execution phase. In query execution, you can append, retrieve, or update data in the tables used in your JoinDef. These operations are described later in this chapter.

When you exit query execution after testing a JoinDef, QBF returns you to the JoinDef Definition frame so that you can view or modify your JoinDef entries. After making changes, you can easily switch to query execution phase by clicking Go again.

## Save JoinDefs

Saving a JoinDef places it in the JoinDef Catalog and ensures that you can run it again without having to rebuild it. Also, you can easily edit the appearance of a saved JoinDef with the VIFRED and save the edited version under a new name.

To save a JoinDef, click Save on the JoinDef Definition frame.



The screenshot shows a dialog box titled "Ingres - QBF" with the subtitle "QBF - Saving a JoinDef". The dialog contains the following fields and labels:

- Name: **Employee\_Tasks**
- Owner: **ingres**
- Short Remark: **Employee Task Assignments**
- Long Remark: (empty text area)
- Created: (empty field)
- Modified: (empty field)

At the bottom of the dialog, there are three buttons: **Save(F3)**, **Cancel(F7)**, and **Help(F1)**.

If this is a new JoinDef, QBF displays the JoinDef Save frame. Enter the name of the JoinDef. Be sure you choose a unique name. If you enter a name already in use and try to save the JoinDef, QBF informs you that a JoinDef by that name exists and asks whether you want to save the new JoinDef by the same name. If you type **y** (yes), QBF overwrites the existing JoinDef.

The JoinDef Save frame allows you to enter a short remark or description of the JoinDef and also a longer remark or description. These remarks are useful for keeping track of your JoinDefs. The Short Remark that you enter here appears on the JoinDefs Catalog frame to the right of the JoinDef's name and owner. The Long Remark that you enter here appears whenever a user chooses MoreInfo about a JoinDef.

## Edit JoinDefs

You can edit either the definition for a JoinDef on the JoinDef Definition frame, or you can edit the JoinDef form on which the JoinDef query is run.

When you run a JoinDef, QBF provides a default form in the query execution phase. You can make some adjustments to this form with the ChangeDisplay operation, described above. Use the VIFRED QBF for more extensive customization, such as adding descriptive title, validation checks, or displaying fields in reverse video. This further customization turns your JoinDef into a QBFname.

To make a JoinDef form accessible to VIFRED, be sure to save the JoinDef to store it in the JoinDefs Catalog.

To modify a saved JoinDef from within QBF, put the cursor on the name of an existing JoinDef on the JoinDefs Catalog frame and choose the Edit operation. This displays the JoinDef Definition frame for that JoinDef.

To create a new JoinDef from an existing one without affecting the original version, simply type a new name over the current one in the JoinDef Name field on the JoinDef Definition frame and then save it. QBF stores your changes under the new name when you choose the Save operation.

## Delete JoinDefs

You can delete a JoinDef with the JoinDefs Catalog frame. Place the cursor on the name of the JoinDef you want to erase and choose the Destroy operation. QBF prompts you for confirmation:

```
Do you wish to destroy JoinDef 'name'?
```

Type **y** (yes) to permanently delete the JoinDef. Type **n** (no) if you decide not to delete the JoinDef.

You can also delete a JoinDef on the command line, using the `delobj` command. For details, see the chapter "Using System Commands for the Forms-based Tools."





# Chapter 7: Using RBF

---

This section contains the following topics:

[Report-By-Forms \(RBF\)](#) (see page 149)  
[RBF Frames and Operations](#) (see page 150)  
[Start RBF](#) (see page 151)  
[Obtain Information About a Report Specification](#) (see page 153)  
[RBF PopUp Frames](#) (see page 155)  
[Preview Reports](#) (see page 156)  
[Report Specifications](#) (see page 157)  
[Sources of Report Data](#) (see page 158)  
[Sort Columns and Breaks](#) (see page 159)  
[Date, Time, and Page Number](#) (see page 160)  
[Report Styles](#) (see page 161)

## Report-By-Forms (RBF)

Report-By-Forms (RBF) is a forms-based interface for creating and producing reports from an Ingres database.

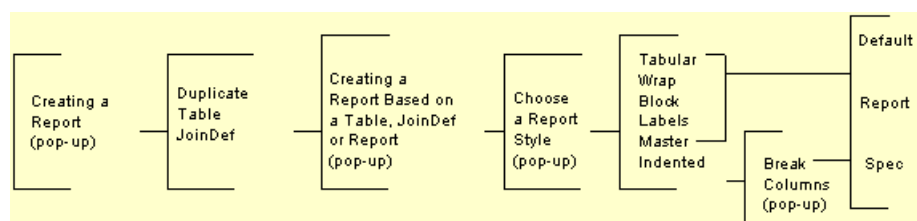
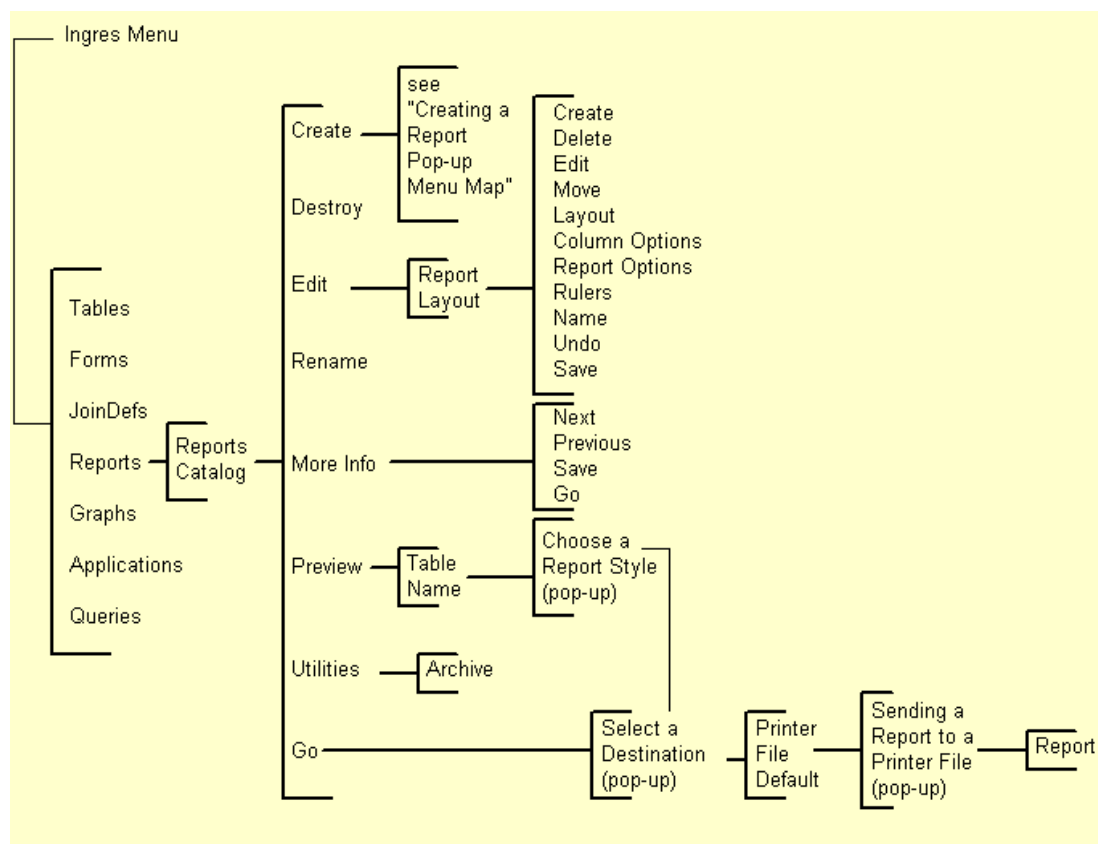
Using RBF, you can:

- Produce a report immediately with the Preview operation
- Create, edit, and save report specifications that you can use repeatedly to produce reports at any time

You can also create report specifications with Report-Writer, a command language release of RBF. With Report-Writer, you can produce more complex reports than you can with RBF, including conditional expressions, break control, and complex queries. For complete information on creating report specifications with Report-Writer, see the chapter "Using Report-Writer."

## RBF Frames and Operations

The following figures contain maps of the various RBF frames and operations.



## Start RBF

You can start RBF from the operating system or from the Ingres Menu.

### **To start RBF from the operating system command line**

Issue the rbf command:

```
rbf
```

You can optionally include parameters that:

- Display a named report specification for editing in the Report Layout Frame
- Display the Report Catalog Frame, where you can examine or select one of the listed report specifications
- Create a default report specification based on the layout style parameter you specify

For more information on the rbf command and its parameters, see the chapter "Using System Commands for the Forms-based Tools."

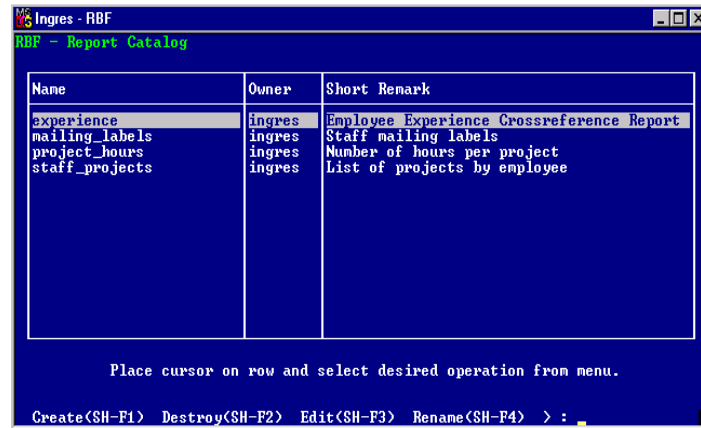
### **To start RBF from the Ingres Menu**

Choose the Reports operation from the menu.

RBF displays the Report Catalog frame.

## Report Catalog Frame

The Report Catalog frame, shown in the following figure, provides access to all the basic RBF functions. The table field portion of this frame lists all the report specifications for this database owned by you or the database administrator. Report specifications owned by other users are not listed in the Report Catalog frame.



To locate a report specification, scroll through the list or use the First Letter Find function (see page 41).

The menu items on the Report Catalog frame are as follows:

### Create

Creates a new report specification for a specified table, view, RBF report, or JoinDef.

### Destroy

Deletes a report specification that you own from the database.

### Edit

Selects a report specification for further editing. Places you in the Report Layout frame. You can edit RBF reports only; you cannot edit reports saved with the sreport command. For additional information, see the chapter "Working with RBF Report Specifications."

### Rename

Changes the name of a report specification that you own.

**MoreInfo**

Calls a second frame containing more information on the selected report specification.

**Preview**

Runs a report immediately for the specified table or view. Produces a one-time only report in the default format for the report style you selected. Contains all the data in the specified table or view. Does not save a report specification. For instructions on producing a preview report, see the chapter "Producing a RBF Report."

**Utilities**

Archives a report to a text file. For instructions, see the chapter "Working with RBF Report Specifications."

**Go**

Runs an existing report specification.

**Help, End, Quit**

Perform standard operations.

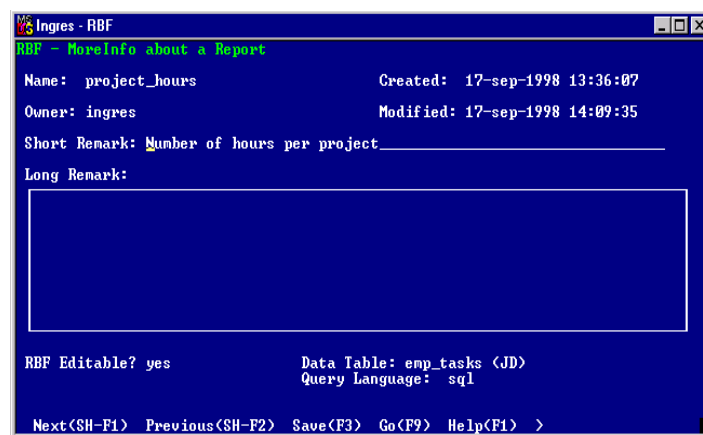
## Obtain Information About a Report Specification

**To display information about a report specification listed in the Report Catalog frame**

1. Place the cursor over the name of the report specification on the Report Catalog frame.
2. Choose the MoreInfo operation.  
The MoreInfo about a Report frame displays.

## MoreInfo about a Report Frame

The MoreInfo about a Report frame shows the name of the table, view, or JoinDef on which the report is based in the Data Table field. If the report is based on a JoinDef, the initials JD appear in parentheses after the JoinDef name.



To edit the Short Remark (which is displayed along with the report specification name on the Report Catalog frame) and the Long Remark, move the cursor into the field and edit as you would in any form field. To save the changes you made, select the Save operation.

The operations in the MoreInfo about a Report frame are as follows:

### Next

Displays information for the next report that is listed in the Report Catalog frame.

### Previous

Displays information for the previous report that is listed in the Report Catalog frame.

### Save

Saves changes to the Short Remark and Long Remark fields in the frame.

### Go

Runs the selected report.

### Help, End

These are standard operations.

## RBF PopUp Frames

When you create or edit a report specification with RBF, use a series of pop-ups. These pop-ups either provide selections for you to choose from or fields for you to complete.

Each time a pop-up appears, the menu line changes. It can contain some or all of the operations listed here:

**ListChoices**

Lists the available choices for the selected field.

**Select**

Chooses the highlighted item.

**Help, Cancel**

These are standard operations.

## Preview Reports

A preview report provides a quick method of producing a simple report containing all the data in a given table, view, or JoinDef. Each preview report is a one-time event; you cannot save a preview report.

The advantages of a preview report are:

- Ease
- Simplicity
- Speed (Producing a preview report is faster than creating and running a report specification.)

The disadvantages of a preview report are that you cannot:

- Use aggregates
- Edit the report format
- Specify data sorting
- Produce Master/Detail, labels, or indented style reports
- Save the report specification

You can produce preview reports only in tabular, wrap, or block styles. Report-sorts the data by the first column only. If your table has duplicate data in the first column, RBF reports those rows together, but in an undetermined order relative to each other.

For instructions on producing preview reports, see the chapter "Producing a RBF Report."



## Report Specifications

A report specification is a set of instructions telling RBF what data you want included in the report and how you want the data presented. With RBF, you can create report specifications:

- Based on tables, views, join definitions (JoinDefs), or other RBF reports
- In six different styles, which you can edit
- With report, page, and break headers and footers, which you can design to your specifications
- With aggregated data

Once you have named and created a report specification, you can use the same specification to produce a report as many times as you wish. The report will always reflect the most recent data in the database.

If you edit the table, view, or JoinDef on which your report is based, you must re-create your report specification. For restrictions that apply to the use of certain report data sources, see Sources of Report Data (see page 158).

**Note:** Report specifications created for this release of Ingres cannot be edited or run in earlier releases due to changes in the underlying table name formats.

## How to Produce a Report

To produce a report from a report specification, follow these steps:

1. Create a default report specification and edit it, if you wish. For details, see the chapter "Working with RBF Report Specifications."
2. Save the report specification. For details, see the chapter "Working with RBF Report Specifications."
3. Run the report specification. For details, see the chapter "Producing a RBF Report."

## Sources of Report Data

A RBF report can be based on:

- A table
- Selected columns in a table
- A view
- A joinDef
- An existing RBF report

When choosing the source for your report, be aware of the following restrictions:

- If you base the report on a table and later add a column to the table, the report will not reflect the addition.
- If you delete a column or change the data type of a column, you will not be able to edit or run the report using RBF.
- If ownership of the table changes to other than the DBA or the catalog owner, you will not be able to edit or run the report using RBF, unless the new owner grants you access.
- If you base the report on a view or JoinDef and later change the definition of the view or JoinDef, the report will not reflect those changes.
- You cannot base a report on an existing report that was saved with the sreport command, a system-level command used for saving reports in Report-Writer format. For more information about sreport, see the chapters "Working with RBF Report Specifications" or "Using Report-Writer."
- You must specify SQL as the query language if you create a report in that contains any of the following objects:
  - JoinDef
  - Table owned by a user other than yourself or the DBA
  - Table, column, or correlation name that is a delimited identifier
- You cannot run the report under earlier releases of Ingres.

## Sort Columns and Breaks

To specify the order in which your report must print the data, you can designate certain data columns as primary or subordinate *sort* columns. For instance, if you specify the Company column as the primary sort column, and the Department column as the next subordinate sort column, the report will organize the data first by Company, and then by Department within each Company.

A *break* marks a change in the value of a report column that is designated as a sort column. For instance, in the previous example, a break occurs each time RBF encounters a new department name in the Department column. You can designate how the report must indicate a break. For example, you could instruct RBF to suppress duplicate department names in the Department column and to start a new page for each new department name. Breaks are helpful for presenting data in logical divisions that are easy to read.

To indicate a break you can:

- Suppress duplicate values in a sort column and print a value only when it changes.

For instance, in the following example, the company name prints only once and is not repeated for each of its departments:

RB Associates	Development
	Personnel
	Sales
	Support
Weston, Inc.	Accounting
	Marketing

- Start a new page each time a value in a sort column changes.
- Create a break header to indicate the beginning of a group of related rows that have the same value in the sort column.
- Create a break footer to indicate the end of a group of related rows that have the same value in the sort column.
- Create an aggregate in a break footer to calculate a subtotal or to otherwise summarize information up to the break.

An example of a break is shown in the indented report style figure. The Dept (department) column is a sort column. Each time the department name changes, RBF prints a break header containing the department name.

Default RBF report styles have predefined sort columns (usually the first column only) and predefined break actions associated with each sort column. To customize the report, you must specify additional or different sort columns and break actions. For details on using breaks, see the chapter "Working with RBF Report Specifications."

## Date, Time, and Page Number

The Report Options frame allows you to specify whether or not, and in what format, you want the date, time, and page numbers to appear in your report. By default, RBF automatically places the date and time in the report and page headers and the page number in the page footer. These fields do not display on the screen when you are creating a report specification. However, they do appear on the report when it is run. If you delete these report sections, this information will not appear on your report.

When you run a report including the date or time, RBF takes the date and time from your computer's operating system. If your system's date and time are incorrect, the report date and time will also be incorrect.

The report reflects the date and time when data is taken from the database, not when the report specification was created. When you send a report to a file for future printing, the date and time on the report reflect when you take the data from the database rather than when you send the report to the printer.

For more information, see Date, Time, and Page Components in the chapter "Working with RBF Report Specifications."

## Report Styles

RBF offers the following six report styles:

- Tabular (or Column)
- Wrap
- Block
- Indented
- Master/Detail
- Labels

When you select a report style for your report, RBF provides you with the *default report specification* for the chosen style. You can then edit the default report specification to look exactly the way you want. The report style that you initially choose is not a persistent attribute of the report; rather, it is a starting point for further customization. There are some restrictions on editing labels reports, as discussed in Labels Style.

After you have created and saved a report specification, you cannot instruct RBF to convert it to a different style. For instance, you cannot take an existing tabular report and instruct RBF to convert it to block style. But, you can always change the appearance of the report by editing it, as noted above.

All styles are not always available. For example, the Master/Detail style is available only when you base your report on a Master/Detail JoinDef. RBF displays only those styles available for your report.

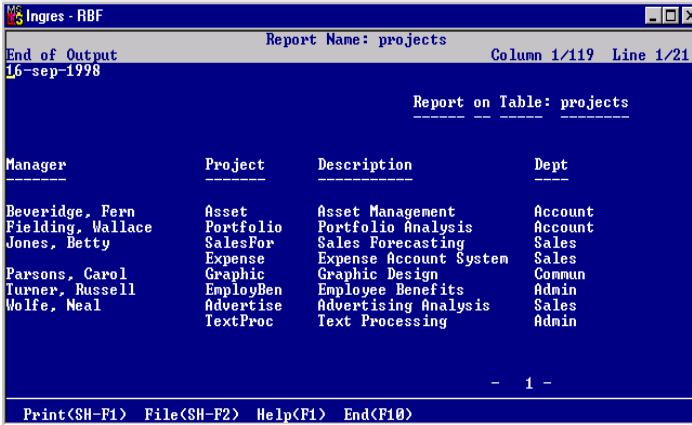
## Tabular Report Style

*Tabular* reports, also called *column* reports, list the name of each column in the table with column headings and the data in columnar format.

This style produces a report as wide as necessary to fit all the columns in your table. When you print a tabular report that is too wide to fit across a printed page, the data that cannot fit is either wrapped around to the next line or truncated (eliminated) from the report. Whether an overly wide report is truncated or wrapped depends on your output device.

By default, tabular reports break on the first sort column and do not print duplicate break values, except over page breaks.

This figure shows an example of a tabular report, which breaks on Manager, the first sort column.



End of Output  
16-sep-1998

Report Name: projects Column 1/119 Line 1/21

Report on Table: projects

Manager	Project	Description	Dept
Beveridge, Fern	Asset	Asset Management	Account
Fielding, Wallace	Portfolio	Portfolio Analysis	Account
Jones, Betty	SalesFor	Sales Forecasting	Sales
	Expense	Expense Account System	Sales
Parsons, Carol	Graphic	Graphic Design	Commun
Turner, Russell	EmployBen	Employee Benefits	Admin
Wolfe, Neal	Advertise	Advertising Analysis	Sales
	TextProc	Text Processing	Admin

- 1 -

Print(SH-F1) File(SH-F2) Help(F1) End(F10)

## Wrap Report Style

*Wrap* style is similar to tabular style, except the column headings and data are wrapped around to the next line at screen or page width.

This figure shows a wrap report with the Budget and Due Date columns wrapped around to subsequent lines of the report.

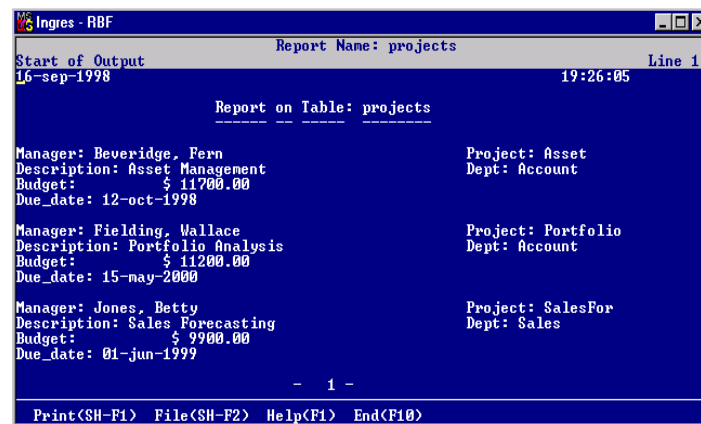


The screenshot shows a window titled 'Ingres - RBF' with a report titled 'Report Name: projects'. The report is displayed in a wrap style. The header section includes 'Start of Output 16-sep-1998', 'Line 1', and a timestamp '19:24:18'. The main data is presented in a table with columns: Manager, Project, Description, and Dept. The 'Budget' and 'Due\_date' columns are wrapped onto subsequent lines. The footer includes a page number '- 1 -' and navigation keys: 'Print<SH-F1> File<SH-F2> Help<F1> End<F10>'.

Manager	Budget	Project	Description	Dept
Beveridge, Fern	\$ 11700.00	Asset	Asset Management	Account
Fielding, Wallace	\$ 11200.00	Portfolio	Portfolio Analysis	Account
Jones, Betty	\$ 9900.00	SalesFor	Sales Forecasting	Sales

## Block Report Style

In a *block* report, each row of report data is formatted in a block. Each column heading appears immediately to the left of its data on the same detail line. The following figure shows a block report.



The screenshot shows a window titled 'Ingres - RBF' with a report titled 'Report Name: projects'. The report is displayed in a block style. The header section includes 'Start of Output 16-sep-1998', 'Line 1', and a timestamp '19:26:05'. The main data is presented in blocks for each manager, with labels like 'Manager:', 'Description:', 'Budget:', and 'Due\_date:' followed by the data. The footer includes a page number '- 1 -' and navigation keys: 'Print<SH-F1> File<SH-F2> Help<F1> End<F10>'.

Manager: Beveridge, Fern	Project: Asset
Description: Asset Management	Dept: Account
Budget: \$ 11700.00	
Due_date: 12-oct-1998	
Manager: Fielding, Wallace	Project: Portfolio
Description: Portfolio Analysis	Dept: Account
Budget: \$ 11200.00	
Due_date: 15-may-2000	
Manager: Jones, Betty	Project: SalesFor
Description: Sales Forecasting	Dept: Sales
Budget: \$ 9900.00	
Due_date: 01-jun-1999	

## Indented Report Style

The *indented* style, also called the hierarchical control break style, allows you to produce a report with multiple levels of breaks. When specifying this report style, you must specify at least one sort (break) column. The following figure shows an indented report.

The screenshot shows a window titled 'Ingres - RBF' with a report named 'projects'. The report is displayed in an indented style with multiple levels of breaks. The header information includes 'Start of Output 17-sep-1998', 'Report Name: projects', 'Column 1/96', and 'Line 1/84'. The report content is as follows:

Report on Table: projects				
Dept: Account				
Due Date: 12-oct-1998				
Manager	Project	Description	Bud	
Beveridge, Fern	Asset	Asset Management		
Due Date: 15-may-2000				
Manager	Project	Description	Bud	
- 1 -				

At the bottom of the window, there is a menu bar with the following options: Print(SH-F1), File(SH-F2), Help(F1), and End(F10).

## Master/Detail Report Style

A *Master/Detail* report presents master data in a break header and detail data in columns. The default break column is the first Master/Detail join column. If there are multiple join columns, RBF sorts the data on all the join columns. However, only the first join column contains default break actions. If you want specific break actions to occur on the other join columns, you have to add the break actions yourself. For more information, see Column Break Options (see page 203).

A Master/Detail report must be based on a Master/Detail JoinDef. This figure shows a Master/Detail report.

The screenshot shows a window titled 'Ingres - RBF' with a report named 'task\_assignments'. The report is displayed in a Master/Detail style. The header information includes 'Start of Output 17-sep-1998', 'Report Name: task\_assignments', 'Column 1/80', and 'Line 1 14:48:40'. The report content is as follows:

Report on JoinDef: task_assignments		
Name: Alcott, Scott		
Title: Sr Programmer		
Hourly Rate:	\$ 50.00	
Manager: Wolfe, Neal		
Project	Task	Hours
Advertise	Implement	5
Advertise	Test	8
- 1 -		

At the bottom of the window, there is a menu bar with the following options: Print(SH-F1), File(SH-F2), Help(F1), and End(F10).



The query language for a Master/Detail report is SQL. For additional information about the query language used for retrieving your report data, see the chapter "Working with RBF Report Specifications."

## Labels Report Style

The *labels* style produces a report similar to a mailing list. It presents blocks of data across the page.

RBF calculates the size of the label and the number of labels across the page. It bases its calculation on the size of the data fields and the location of the right margin.

Unlike the other report styles, you cannot change the basic style of a labels report. The data must appear in blocks across the page. Additionally, you cannot delete the report footer, specify break options, or specify headers in labels reports.

The following figure shows a labels report.

Report on Table: home			
Alcott, Scott	Applegate, Donald	Bee, Charles	Belter, Kwi
1212 1st St.	1916 Dublin Ave.	1415 Hive St.	4343 80th S
New York	Boston	Austin	New York
NY	MA	TX	NY
11787	24952	98989	99292
516-555-5515	781-555-5656	512-555-1212	212-555-555
Beveridge, Fern	Bluff, Clarence	Bridges, Debra	Chung, Arth
New York	Fort Lee	Westwood	St. Louis
NY	NJ	MA	MO
11788	44224	42124	12532

## Report Structure

A RBF report can contain some or all of the following sections:

### Report header

Printed on the first page of the report.

### Page header

Printed at the top of the second and subsequent pages of the report.

To print the Page Header on the first page of the report as part of the report header, use the ReportOptions operation. For instructions, see the chapter "Working with RBF Report Specifications."

### Break header

Printed before the report's detail lines. A report can have one break header for each column designated as a sort column. On the report specification, the break header contains the name of the associated sort column. For example, in the following figure, the column associated with the break header is the Dept column, so the break header is titled Dept-Break-Header.

If a report has multiple break headers, they are printed in the order designated by the sort sequence of the sort columns. For example, the first break header contains the column with a sort sequence of 1, the second break header contains the column with a sort sequence of 2, and so forth.

To define the sort sequence, use the ColumnOptions operation. For instructions, see the chapter "Working with RBF Report Specifications."

### Detail lines

Printed for each row in the table, view, or JoinDef associated with this report.

### Break footer

Printed after the report's detail lines. A report can have one break footer for each column designated as a sort column.

On the report specification, the break footer contains the name of the associated sort column. For example, in the following figure, the column associated with the break footer is the Dept column, so the break footer is titled Dept-Break-Footer. As with break headers, if a report has multiple break footers, they are printed in the order designated by the sort sequence of the sort columns.

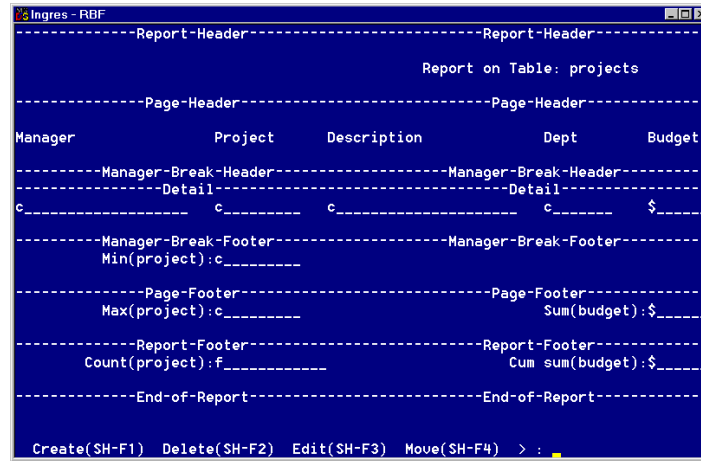
### Page footer

Printed at the bottom of each report page.

### Report footer

Printed at the end of the report.

The following figure shows a report specification with all of the report sections listed. The break, page, and report footers contain aggregates. The aggregates are discussed in the chapter "Working with RBF Report Specifications."



## Default Report Layout

The report sections that appear in your report are determined initially by the default report layout for the report style you chose.

The default layout for each report style is shown in the following table. For instructions, see the chapter "Working with RBF Report Specifications." The Layout operation, also described in that chapter, allows you to add and delete some report sections to customize the report.

Report Styles	Report Header	Page Header	Break Header	Detail	Break Footer	Page Footer	Report Footer
Tabular	yes	yes	yes	yes	no	yes	no
Wrap	yes	yes	no	yes	no	yes	no
Block	yes	yes	no	yes	no	yes	no
Indented	yes	yes	yes	yes	no	yes	no
Master/Detail	yes	yes	yes	yes	no	yes	no
Labels	yes	yes	no	yes	no	yes	yes



# Chapter 8: Working with RBF Report Specifications

---

This section contains the following topics:

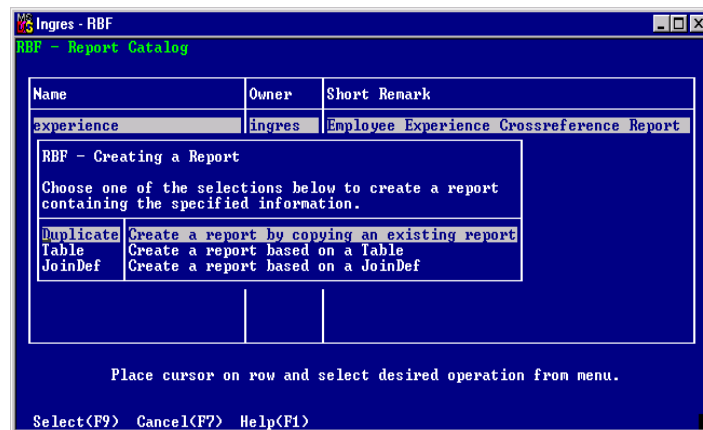
- [Create a Default Report Specification](#) (see page 169)
- [RBF Report Layout Frame](#) (see page 175)
- [Layout and Create Operations—Create New Report Components](#) (see page 181)
- [Layout and Delete Operations—Delete Report Components](#) (see page 192)
- [Edit Operation—Edit Report Components](#) (see page 194)
- [Column Display Formats](#) (see page 196)
- [Column Options](#) (see page 198)
- [Column Break Options](#) (see page 203)
- [Move Operation—Move Report Components](#) (see page 207)
- [Report Options Frame](#) (see page 211)
- [Obtain the Name of a Column](#) (see page 216)
- [Undo Operation—Undo Edits](#) (see page 216)
- [Save Operation—Save a Report Specification](#) (see page 217)
- [Archive Operation—Archive a Report Definition](#) (see page 221)
- [How to Copy Report Specifications](#) (see page 223)
- [Delobj Command—Delete a Report Specification](#) (see page 224)

## Create a Default Report Specification

### To create a default report specification with RBF

1. Start RBF. When the Report Catalog frame appears, select the Create operation.

RBF displays the Creating a Report pop-up, which lists the sources of data on which you can base your report.



The Creating a Report pop-up has the following selections:

### Duplicate

Bases your report on an existing report. A report cannot be based on one that was saved with the sreport command.

### Table

Bases your report on a table or view, as it was defined when you created the report specification.

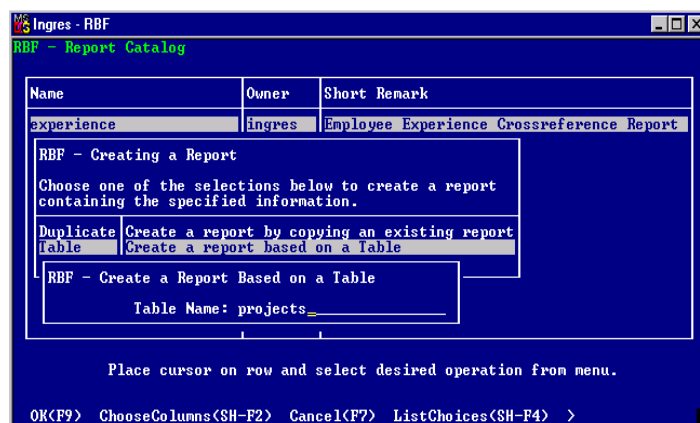
### JoinDef

Bases your report on a JoinDef, as it was defined when you created the report specification.

For restrictions and limitations on the use of tables, views, JoinDefs, and other report specifications as sources of data for your reports, see Sources of Report Data (see page 158) in the chapter "Using RBF."

2. Place the cursor on Duplicate, Table, or JoinDef and choose Select.

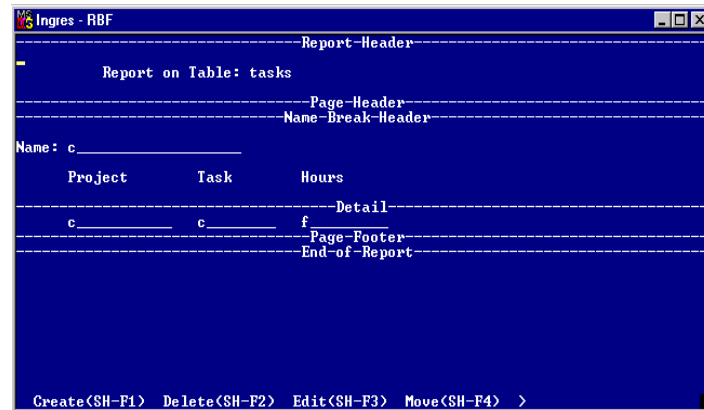
RBF displays a pop-up similar to one in the following figure.



3. Do one of the following:
  - If necessary, use the ListChoices operation to display a list of available choices and select an item from the list.
  - On the Create a Report Based on a Table pop-up, enter the name of a table and choose ChooseColumns (see page 172) to base your report only on certain columns of a database table. RBF returns to the report specification process when done.
  - On the Create a Report Based on a JoinDef pop-up, choose the Create operation to create or edit a JoinDef on which you are basing the report. Then enter the name of an existing or new JoinDef and choose the Edit operation to access QBF. RBF returns you to the report specification process.
  - On the appropriate pop-up, enter the name of a report, table, view, or JoinDef on which you want to base the report and click OK.
4. If your report is based on a table, view, or JoinDef, you must choose a report style (see page 173).

If your report is based on an existing report, or after you have chosen a report style, RBF displays the default report specification in the Report Layout frame, as shown here.

The following figure shows a default report specification for a tabular report.



5. Do one of the following:
  - Customize the report specification, as described in the remainder of this chapter.
  - Save the report specification by choosing the Save operation (see page 217).

## Choose Columns on which to Base the Report

The ChooseColumns operation on the Create a Report Based on a Table pop-up allows you to specify only certain columns on which you wish to base the report, as noted in Step 4 of the previous procedure for creating a default report specification.

### To specify particular columns in a table on which you want to base the report

1. Choose the Create operation on the Report Catalog frame; then select the Table option from the Creating a Report pop-up. RBF displays the Table Name pop-up.
2. Enter the name of a table on the Table Name pop-up and choose the ChooseColumns operation. RBF displays a pop-up containing a list of all columns in the table.

Up to 10 columns appear in the pop-up. You can scroll to view any additional columns.

3. Use the cursor to select a column and choose an appropriate operation on the menu to edit the list of columns on which to base the report. The available operations are as follows:

#### **OK**

Accepts the report column list as is, and returns the user to the Table Name pop-up.

#### **Add**

Redisplays a list of previously deleted columns. To add the column back into the report column list (above the current column), select it with the cursor and choose the Select operation.

#### **Delete**

Deletes the current column from the report column list.

#### **Move**

Displays a submenu of move operations for the current column (Above, Below, Help, and End). Choosing End cancels the Move operation.

#### **RemoveAll**

Deletes all columns from the report column list at once. You can then use the Add operation to put a few of the deleted columns back into the list.

#### **Cancel, Help**

Perform standard operations.

4. When done editing the list of columns, choose the OK operation.



RBF returns you to the Table Name pop-up.

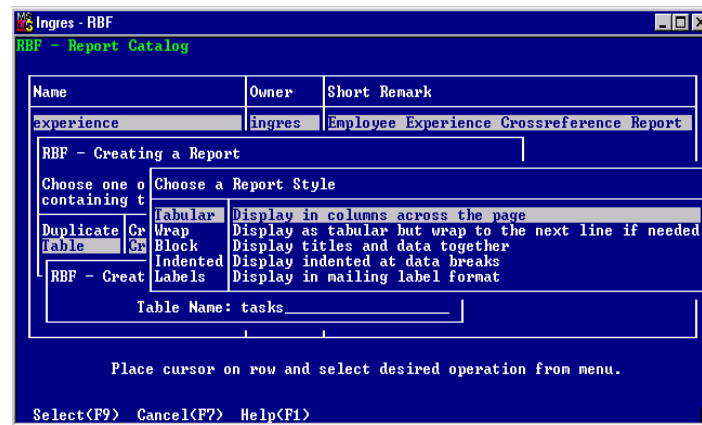
- On the Table Name pop-up, choose OK again to continue with the creation of your report specification.

RBF displays the Choose a Report Style pop-up.

## Choose a Report Style

If your report is to be based on a table, view, or JoinDef, you must choose a report style when the Choose a Report Style pop-up in the following figure appears during report specification creation.

The Choose a Report Style pop-up contains a list of the report styles available for your report. The Master/Detail style is available only if you based your report on a JoinDef.



### To choose a report style

1. Place the cursor on the desired report style and choose Select.

If you selected the indented style, go to Step 2.

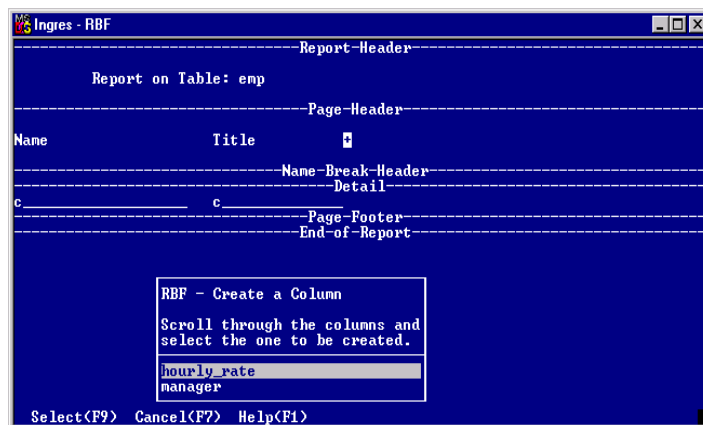
If you selected the tabular, wrap, block, Master/Detail, or labels style, the Report Layout frame appears with the default report specification. Go to step 3.

2. The Break Columns pop-up appears in the figure below. This pop-up lists all of the columns in the table or JoinDef on which your report is based.

To select break columns, enter a number between 1 and 1024 in the Sort Sequence field. Each Column Name must have a unique sort sequence number. Enter a zero (0) if you do not want to assign a sort sequence to the column.

Break columns appear in the order specified by the sort sequence; that is, the column with a sort sequence of 1 is the first break column, the column with a sort sequence of 2 is the second break column, and so forth.

You must specify at least one break for an indented report. For a discussion of breaks, see the chapter "Using RBF."

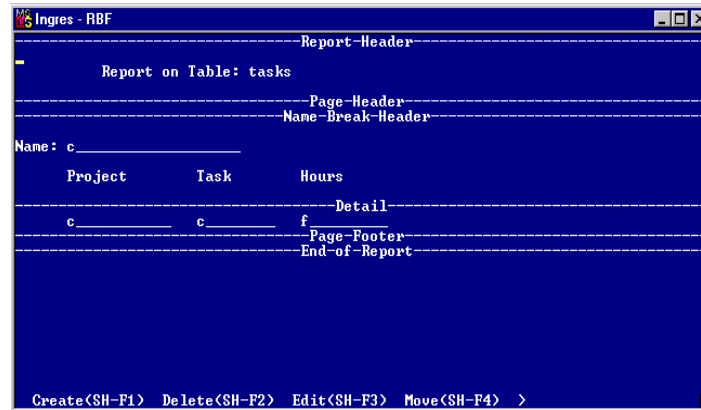


When you have chosen the report style and break columns, if appropriate, RBF displays the default report specification in the Report Layout frame in the following figure.

3. At this point, you can either:
  - Customize the report specification, as described in the remainder of this chapter
  - Save the report specification, as is, by choosing the Save operation (see page 217).

## RBF Report Layout Frame

The Report Layout frame is the primary frame for editing RBF report specifications. You can use this frame to add, edit, or delete report components. From this frame, you also can call up other frames to add and delete report sections, and to specify report sorting order, runtime selection criteria, and report output options.



The preceding figure shows a Report Layout frame for a tabular report with a break header for the Name column. A layout frame for a report in another style looks different. The operations of the Report Layout frame function the same way regardless of the report's style.

## How You Get to the Report Layout Frame

Use one of the following ways to load a default report specification into the Report Layout frame:

- Create a default report specification with the Create operation on the Report Catalog frame. RBF automatically places the new report specification in the Report Layout frame for further editing.
- Place the cursor over an existing RBF report specification name in the Report Catalog frame and choose the Edit option. RBF places the selected report specification in the Report Layout frame.
- Use the rbf command and specify a table, view, or report name as a parameter.

## Report Components in the Report Layout Frame

The Report Layout frame in the preceding figure allow you to specify the layout of the report sections and other report components, such as margins and alignment, trim, columns, and aggregate functions.

## Report Sections

A Report Layout frame is divided into some or all of the following report sections, depending on the report style you choose:

- Report header
- Page header
- Break headers
- Detail lines
- Break footers
- Page footer
- Report footer

Each report section is marked off by lines of dashes labeled with the name of the section.

You use the Layout operation to create or delete report sections. You use the Create and Line operations to add blank lines and other components to a report section, and the Delete operation to delete them from a report section. These operations are summarized in Report Layout Frame Menu.

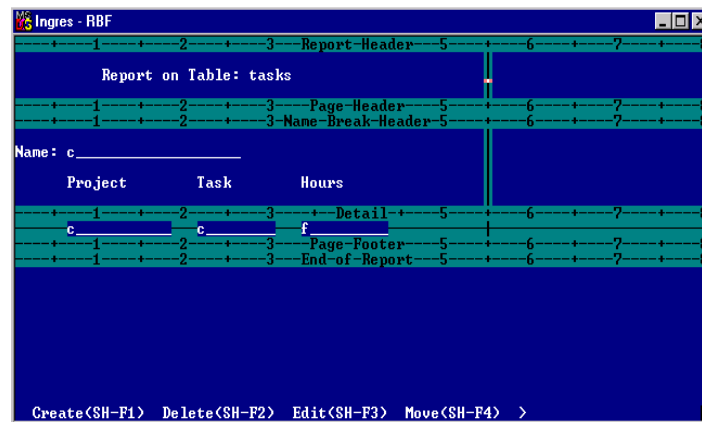
## Report Margins and Alignment Guides

The report's bottom margin is identified by the line marked End-of-Report. The report's right margin is identified by a vertical line that is marked End. Initially, the right margin line is located beyond the right edge of your screen, although in the preceding figure it has been moved so that you can see it.

Use the Move operation, as described later in this chapter, to move these margins and thus expand or contract the size of your report. Both report margins, as well as the report section dividers, optionally display ruler marks every fifth column or row, and a decimal digit every tenth column or row. Use these rulers as alignment guides to help you determine the coordinates for report components.

The Report Layout frame also provides optional horizontal and vertical straight edge alignment guides for aligning report components. If straight edges are turned on, the vertical and horizontal straight edges appear initially in the last column of the report specification and the last row of the Detail section, respectively. Move straight edges with the Move operation, as you would a piece of trim or other report component.

The following figure shows straight edges aligned on the Hours column and rulers turned on. For best results, use a monitor with a line graphics character set.



You cannot delete the section markers from the Report Layout frame. However, you can turn rulers and straight edges on or off independently of each other by choosing the Rulers operation on the Report Layout frame. On the pop-up menu, set each alignment guide to y (yes) or n (no). Regardless of whether the alignment guides are set on or off, the section markers, margin lines, and straight edges do not print on your report.

## Trim

*Trim* is a report component that consists of all lines, words, and characters, other than data or aggregates that print on your report. You can place trim in any section of the report.

*Headings* are considered trim. A heading provides a description of the contents of a column or aggregate. A heading can only be associated with a column or aggregate. You can edit, add, delete, and move headings to any section of the report. Headings are not available in Labels reports.

## Columns and Aggregate Functions

Columns and aggregate functions are report components that deal with the data in your report. *Columns* contain the actual data obtained from the table, view, or JoinDef on which the report is based. You use an *aggregate function*, such as sum or count, to calculate the value of a specified column up to the occurrence of a break.

Columns and aggregate functions are represented on the Report Layout frame by fields containing:

- Letters and symbols, which denote the data display format  
For information about these formats, see Column Display Formats (see page 196).
- Solid lines, which show the width of the column or aggregate

Fields that represent data columns appear in the Detail section of the Report Layout frame. The Report Layout frame does not initially display the names of the columns associated with the fields in the Detail section. You can use the Name operation to obtain the column name (see page 216) for a field.

## Report Layout Menu Operations

At the bottom of the frame is the Report Layout menu. It allows you to perform the following operations:

### Create

Displays a submenu with operations that allow you to create most new report components, including trim, columns, aggregates, headings, and blank lines. To create break headers, footers, and other report sections, use the Layout operation instead of Create. For instructions, see Layout and Create Operations (see page 181).

### Delete

Deletes the report component at the cursor position. To delete break headers, footers, and other report sections, use the Layout operation instead of Delete. For instructions, see Layout and Delete Operations (see page 192).

### Edit

If the cursor is on trim, edit the trim.

If the cursor is on a column or aggregate, displays a submenu with operations that allow you to change the column's data display format or other column options, including the sort order and runtime selection criteria. If the column being edited is a sort column, this submenu also includes the BreakOptions operation, which allows you to specify how RBF handles breaks in column values.

For instructions, see Edit Operation—Edit Report Components (see page 194).

### Move

Moves the current component through operations presented on submenus. For instructions, see Move Operation (see page 207).

### Layout

Creates and deletes break headers, footers, and other report sections. For instructions, see Layout and Create Operations (see page 181) and Layout and Delete Operations (see page 192).

### ColumnOptions

Establishes sort order, sort direction, and runtime selection criteria. For instructions, see Column Options (see page 198).

**ReportOptions**

Establishes a variety of report options, including page length, inclusion of form feeds, display of null values, and underlining capabilities. For more information, see Report Options Frame (see page 211).

**Rulers**

Allows you to use horizontal and vertical straight edge alignment guides for aligning report components. For instructions, see Report Components in the Report Layout Frame (see page 175).

**Name**

Allows you to obtain the name of a column for a field in the Detail section of the Report Layout frame. For more information, see Obtain the Name of a Column (see page 216).

**Undo**

Reverses the effects of the last editing operation. For instructions, see Undo Operation (see page 216).

**Save**

Saves the edited report specification into the database with the report name of your choosing. For instructions, see Save Operation (see page 217).

**Help, End**

Perform standard operations.



## Layout and Create Operations—Create New Report Components

Use the Layout operation to create these report sections:

- Report header
- Page header
- Break headers
- Detail section
- Break footers
- Page footer
- Report footer

For instructions on using the Layout operation, see [Layout and Delete Operations—Delete Report Components](#) (see page 192).

Use the Create operation to create:

- Trim
- Columns
- Aggregates
- Column headings
- Blank lines

### To use the Create operation

1. Position the cursor where you want to create the new component.
2. Choose the Create operation from the Report Layout frame menu.

RBF displays the Create submenu, which offers the following options:

#### Trim

Enters the text of a new trim element at the cursor position. For instructions, see [Create Trim](#) (see page 185).

#### Column

Creates a new column and column heading at the cursor position. In Labels reports, creates a column without a column heading. For more information, see [Create a Column](#) (see page 186).

#### Aggregate

Create an aggregate, such as sum or average, on a column. Aggregates can be cumulative or unique and must be in break, page, or report footer sections. For instructions, see [Aggregates](#) (see page 187).

**Heading**

Creates a new or additional heading line (not available for Labels reports). The new heading line must be associated with an existing heading, column, or aggregate. For details, see Create Additional Heading Lines (see page 191).

**Line**

Inserts a blank line above the line on which the cursor rests. For instructions, see Create Blank Lines (see page 192).

**Help, End**

Perform standard operations.

## Create Break Headers, Footers, and Other Report Sections

The default report specification for each style of report contains its own default report sections. You can use the Layout operation to create additional break headers, footers, and other report sections in your report specification.

Note these restrictions:

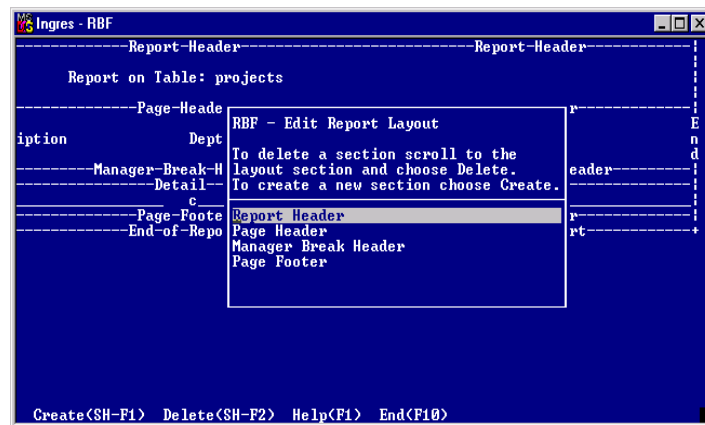
- A report can contain only one of each of the following sections:
  - Report header
  - Report footer
  - Page header
  - Page footer
- A report can contain one break header and one break footer for each sort column in the report

If the section you want to add is a header or footer for a column, you must have first designated the column as a *sort* column. For instructions on designating a sort column, see Column Options (see page 198).

### To add a report section

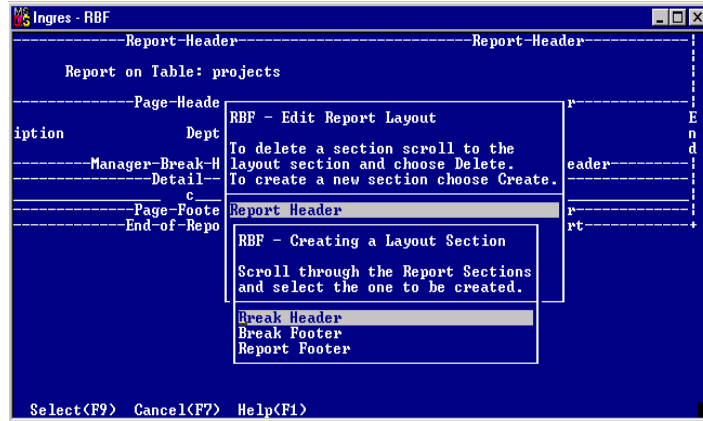
1. Select the Layout operation from the Report Layout frame menu.

The Edit Report Layout pop-up appears.



2. Select the Create operation from the menu.

The Creating a Report Layout Section pop-up appears, similar to the one in the following figure. This pop-up contains a list of the report sections you can create.

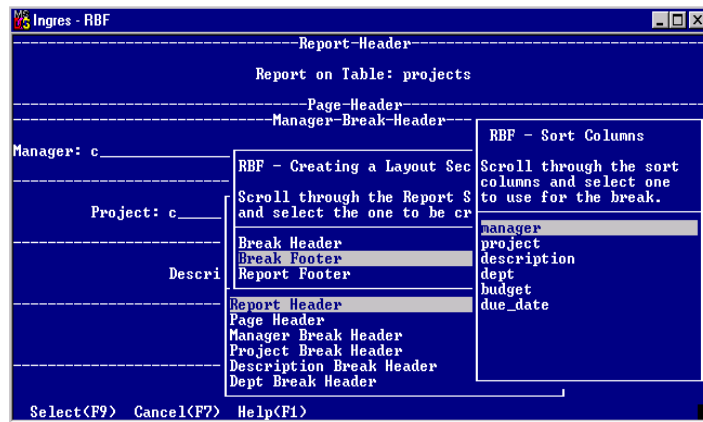


This list always includes the option to create a break header or footer, because you can create many of these. However, it only contains other report sections if they do not already exist.

3. Place the cursor on the name of the report section that you want to create and choose the Select operation.

If you are creating a report or page header or footer, proceed to Step 5.

If you are creating a break header or footer, the Sort Columns pop-up appears in the following figure. It contains a list of the sort columns available for the break header or footer.



4. Move the cursor to the desired sort column and choose the Select operation.
5. Select the End operation.

You are returned to the Report Layout frame.

The new report section appears on the Report Layout frame. If you created a break header or footer, the name of the sort column appears with the report section name.

## Create Trim

### To create trim

1. Position the cursor where you want to add the trim.
2. Select the Create operation from the Report Layout menu.
3. Select the Trim operation from the Create submenu.
4. Enter the text of the new trim.

If you want to create lines and borders for your report, use keyboard characters such as the hyphen (-) or underscore (\_).

5. Press the Menu key.

If necessary, correct the trim position with the Move operation.

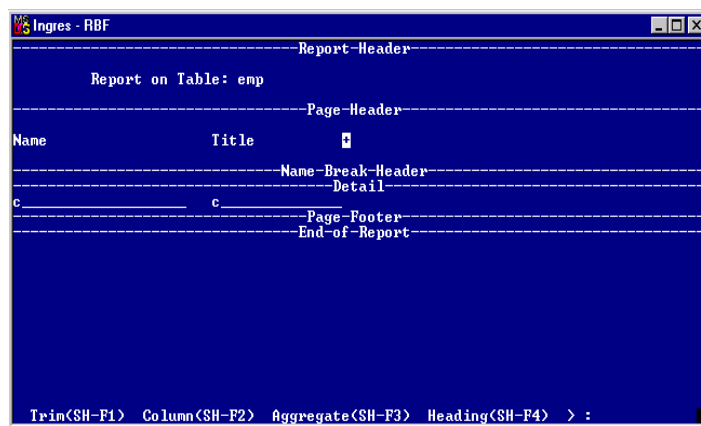
## Create a Column

Use the Column operation to create a new column and column heading in your report. Note these restrictions:

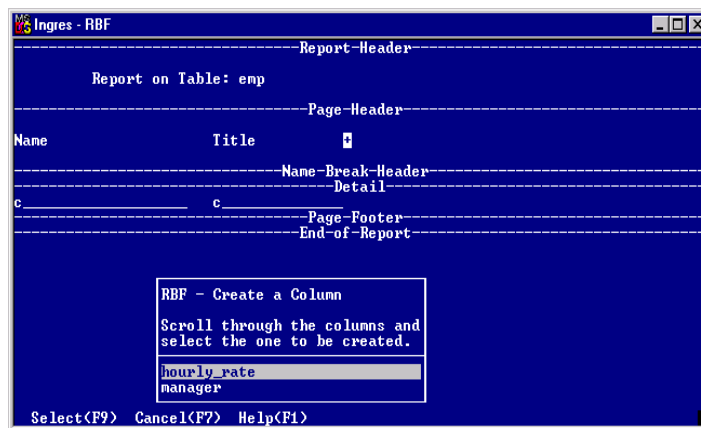
- The column must exist in the table, view, or JoinDef from which the report is generated.
- The column must not already appear on the Report Layout frame.
- In labels reports, the Column operation creates a column without a column heading.

### To create the new column

1. Move the cursor into an open spot in any section of the report.
2. Select the Create operation from the Report Layout Menu.
3. Select the Column operation from the Create submenu.



RBF displays the Create a Column pop-up (shown below), which contains a list of the columns that are available for mapping.



4. Position the cursor on the desired column and choose the Select operation.

RBF inserts the column and column heading at the cursor location and pushes existing columns to the right, if necessary.

The new column is in block style with the heading to the left of the column. To change this, use the Move operation.

For Labels reports, RBF creates the column only. It does not create a column heading. To add a column heading, use the Trim operation.

## Aggregates

You use an aggregate function, such as sum or count, to calculate the value of a specified column up to the occurrence of a break. In RBF, an aggregate must appear in a footer for a report, page, or break. RBF calculates the aggregate value each time a break occurs in the specified footer.

The cut-off point for data to be included in the calculation of an aggregate depends on whether the aggregate is simple or cumulative. For more information, see Simple and Cumulative Aggregates (see page 188).

The following table lists all of the available aggregate functions:

Aggregate Function:	Returns:
Any	1 if any data exists, 0 if none exists
Average	Mean average of all values in the column
Count	Count of all the values in the column
Minimum	Smallest value in the column
Maximum	Largest value in the column
Sum	Arithmetic sum of all the values in the column

Note these restrictions:

- If the aggregate column is numeric, all of these functions are available.
- If the aggregate column is a character or date data type, the any, count, minimum, and maximum aggregate functions are available. Additionally, you can take a sum of a date data type column, but only if date intervals (such as 1 day, 2 hours) are stored in the column. Taking a sum on absolute dates (such as December 13, 1988) results in an error.

## Simple and Cumulative Aggregates

Aggregates can be simple or cumulative.

A *simple aggregate* is determined by the type of footer in which you specify it. For example:

- If you specify the sum aggregate for the budget column in a report footer, then the aggregate contains the sum of the values in the budget column for all rows in the report.
- If you specify the sum aggregate for the budget column in the page footer, the aggregate contains the sum of the values in the budget column for the rows printed on each page of the report.
- If you specify the sum aggregate for the budget column in the break footer for the Department column, the aggregate contains the sum of the values in the budget column for all rows in each department.

A *cumulative aggregate* is a running total. It contains the aggregate of all the rows processed since the start of the report and does not depend on the location of the aggregate.

## Unique Aggregates

You can create unique aggregates for sort (break) columns only. Unique aggregates are calculated by using only the unique values of the sort columns. You cannot specify a unique aggregate for a non-sort column.

If you change a column that has a unique aggregate associated with it from a sort to non-sort column, RBF gives you the following options:

- Deleting the unique aggregate
- Changing the unique aggregate to non-unique

The unique aggregates are described in the following table:

Unique Aggregate	Returns:
Average (Unique)	Mean average of all the unique values in a sort column.
Count (Unique)	Count of all the unique values in a sort column.
Sum (Unique)	Sum of all the unique values in a sort column.



## Guidelines for Creating an Aggregate

When creating aggregates, you:

- Must create an aggregate in a footer (break, page, or report)
- Can create an aggregate on any column (the column does not have to be a sort column)
- Can create more than one aggregate in a footer and more than one type of aggregate for a particular column

For example, in a table containing the columns Project, Employee, and Salary, you can create a break footer for the Project column (Project\_Break\_Footer). Then, you can create Average and Sum aggregates (non-cumulative) for the Salary column in the Project\_break\_footer. This instructs RBF to:

- Break each time it finds a new value in the Project column
- Calculate the total salary and average salary cost of each project

After creating an aggregate, you can edit it in the same way you edit columns. You can:

- Change its display format
- Edit it or the associated headings
- Move it to another footer section
- Delete the aggregate

## Create an Aggregate

### To create an aggregate

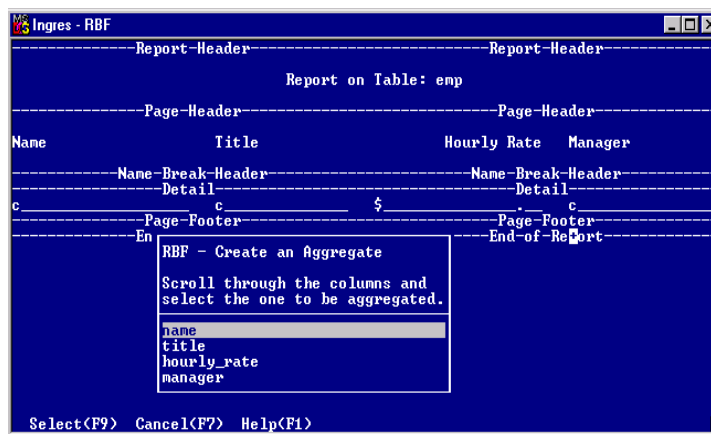
1. Place the cursor in the position in the footer section where you want to create the aggregate, and select the Create operation from the Report Layout frame menu.

(If you must create a footer section, see Create Break Headers, Footers, and Other Report Sections (see page 183).)

The Create submenu appears.

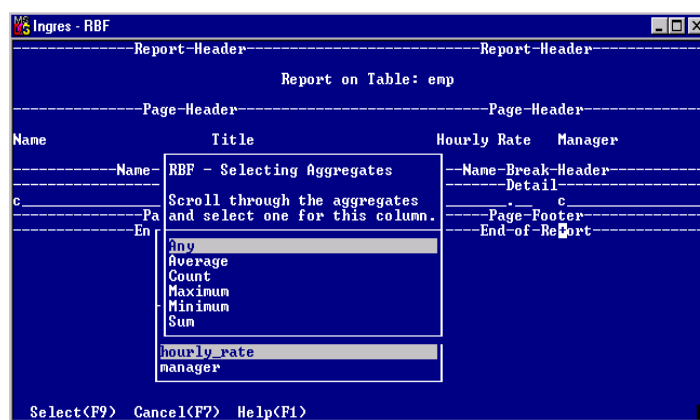
2. Select the Aggregate operation.

The Create an Aggregate pop-up appears. The pop-up contains a list of the columns in your report.



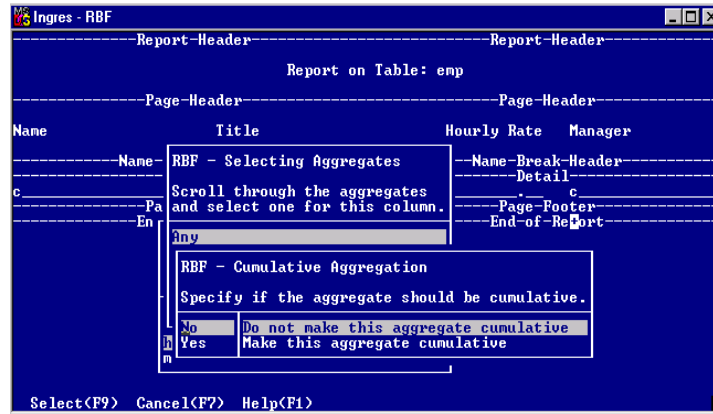
3. Position the cursor on the column for which you want the aggregate, and choose the Select operation.

The Selecting Aggregates pop-up appears. This pop-up contains a list of the aggregate functions that are available for the column to be aggregated.



- Place the cursor on the desired aggregate function and choose the Select operation.

The Cumulative Aggregation pop-up appears.



- Select Yes to make the aggregate cumulative or No to make the aggregate simple (non-cumulative). For details, see Simple and Cumulative Aggregates (see page 188).
- Choose the Select operation to finish creating the aggregate.

You are returned to the Report Layout frame.

## Create Additional Heading Lines

By using the Create and Heading operations, you can add a new heading or additional heading line to a column or aggregate, except in Labels reports.

In Labels reports, use the Create and Trim operations to add column headings. However, be aware that you cannot move columns with headings added in this manner as a unit; you must move them separately. For instructions on moving trim, columns, aggregates, and headings, see Move Operation (see page 207).

### To create a new heading line

- Place the cursor on the existing heading or, if there is no existing heading, on the column or aggregate.
- Select the Create operation from the Report Layout frame menu.
- Select the Heading operation from the Create submenu.

If there is an existing heading, RBF moves the cursor to the first blank line below the existing heading line, creating a new line, if necessary. If there is no existing heading, RBF moves the cursor to the Page Header section immediately above the column.

- At the prompt, enter the new or additional heading and press the Menu key. RBF inserts the heading.

## Create Blank Lines

When creating blank lines, keep in mind that the Line operation inserts the new blank line at the current cursor position, pushing the current line and everything below it down one line.

### To add a blank line:

1. Place the cursor at the position where you want to insert the new line.
2. Select the Create operation from the Report Layout menu.
3. Select the Line operation from the Create submenu. RBF inserts the blank line.

## Layout and Delete Operations—Delete Report Components

You use the Layout operation on the Report Layout frame to delete report sections and the Delete operation to delete other report components, such as columns and trim.

## Delete Break Headers, Footers, and Other Report Sections

The default report specification for each style of report contains its own default report sections. You can use the Layout operation to delete some of the break headers, footers, and other report sections in your report specification.

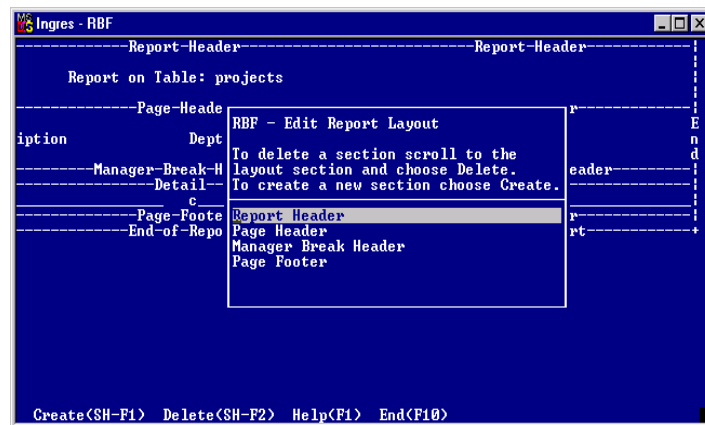
Note these restrictions on deleting report sections:

- You can never delete detail sections.
- You cannot delete sections that are needed for the options set with the BreakOptions operation. For example, if you choose to print on page breaks, you cannot delete the page header. For more information, see Column Break Options (see page 203).
- You cannot delete a section if doing so would cause the report to have no data (columns or aggregates).
- Because the report and page header contain the report date and time, and the page footer contains page numbers, your report cannot contain this information if you delete these sections.

### To delete a report section

1. Select the Layout operation from the Report Layout frame menu.

RBF displays the Edit Report Layout pop-up, which contains a list of the report sections you can delete.



2. Position the cursor on the report section you want to delete and select the Delete operation from the menu.
3. Select the End operation to return to the Report Layout frame menu.

When you return to the Report Layout frame, the deleted section and its contents disappear from the frame.

## Delete Other Report Components

Use the Delete operation on the Report Layout frame to remove any columns, aggregates column headings, or trim from your report specification.

If you delete a column or aggregate, RBF automatically removes the associated heading, but this is not true in reverse. If you delete a heading, RBF does *not* automatically delete the associated column or aggregate.

If you delete all the trim on a line, a blank line is left. You can remove blank lines by placing the cursor on the line and choosing Delete.

### **To delete a column, aggregate, heading, or trim**

1. Place the cursor on the component.
2. Select the Delete operation from the Report Layout frame menu.

## Edit Operation—Edit Report Components

Use the Edit operation to:

- Edit trim and headings
- Change a column or aggregate's display format
- Change column options, such as sort order or runtime selection criteria
- Establish how your report handles breaks in sort columns

## Edit Trim and Headings

The default editing mode is overstrike, which means that the characters you type replace the existing character at the cursor position. To change to *Insert* mode (in which the characters you type push existing characters to the right), press the Mode key for your keyboard.

When you are editing a report component, the current editing mode is displayed in the lower right corner of the frame.

When editing a report specification, you can only work on one line of trim at a time. Each line of trim is considered a separate component. You cannot extend or push a line of trim past the report's right margin, but you can use the Move operation to extend the report's right margin.

### To edit trim and headings

1. Place the cursor on the trim or heading component that you want to edit.
2. Select the Edit operation from the Report Layout frame menu.
3. Edit the component.
4. Press the Menu key.

## Edit Columns

### To edit a column

1. Place the cursor on the column's display format (the cursor must be on the actual column, *not* on the column heading).
2. Choose the Edit operation from the Report Layout frame menu.
3. When the Edit submenu appears, select one of the following operations:

#### **DisplayFormat**

To edit the data display format.

#### **ColumnOptions**

To specify sort order, sort direction, or runtime selection criteria. This is the same as the ColumnOptions operation on the Report Layout frame main menu.

#### **BreakOptions**

To suppress the printing of duplicate values in sort (break) columns or to print a new page each time the value in a sort column changes. This operation appears only if the selected column is a sort column.

## Column Display Formats

The data display formats for each of the columns in your report specification determine the way data appears in the report. You can display the data in a particular column in various ways. For example, for a column containing dates, you can choose from date formats such as January 15th, 1944 or 1/15/44.

RBF assigns a default display format for each data type. You can use the Edit operation, as described in Changing Display Formats, to change these default display formats. However, RBF does not allow you to use a display format that is incompatible with the data type. For example, you cannot use the character display format with a numeric data type.

## Representation of Display Formats

On the Report Layout frame, display formats are represented by letters, symbols and lines. For example, a simple integer display format looks like this:

i\_\_\_\_\_

The letter **i** indicates the integer display format and the underscore line shows the remaining number of characters in the column. Fixed punctuation is also shown in a display format representation. For example, a display format for a column containing financial data might look like this:

\$\_\_\_\_.\_\_\_\_.\_\_\_\_.

The total width of the representation (symbols plus underscore) equals the total width of the column.

RBF permits the same display formats as Report-Writer except for the B, q0, and c0 formats. The c0.w format, however, is valid in RBF.



## Change Display Formats

### To change a display format

1. Place the cursor on the column or aggregate you want to change.
2. Select the Edit operation from the Report Layout frame menu.
3. Select the DisplayFormat operation from the Edit submenu.

The data display component changes from graphic representation of the format definition to the symbol definition. For example, `f_____` becomes `f6`. The cursor appears on the first character of the display format definition.

4. Enter the new display format definition, following the syntax shown in the Display Format Syntax and Descriptions table in the chapter "Working with Data Types and Data Display Formats."
5. Press the Menu key.

The new format appears as a graphic representation.

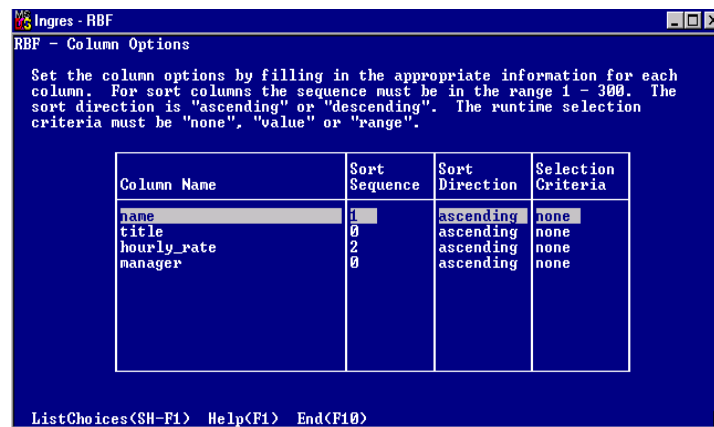
## Column Options

The Column Options frame (in the following figure) can be used to:

- Create sort (break) columns for your report
- Modify a column's sort order
- Specify runtime selection criteria for a column

For an explanation of sorts and breaks, see Sort Columns and Breaks (see page 200).

To get to the Column Options frame, you choose the ColumnOptions operation on either the Report Layout frame or the Edit Submenu of the Report Layout frame.



## Sort Order

*Sort order* determines the order in which data is presented in a report. There are two components to sort order:

- Sort sequence
- Sort direction

*Sort sequence* is the order in which the columns are sorted. You designate one column as the primary sort column. For instance, in a report of employees, you could designate the Lastname column as the primary sort column so that the report can be sorted according to last name.

You can designate additional columns as subordinate sort columns. Subordinate sort columns sort data within each sort value of the primary sort column. For example, if the primary sort column is Zipcode and the secondary sort column is Lastname, all of the data rows containing zip code 00001 are presented first. Within the 00001 group, the data rows are sorted by the alphabetic order of the names in the Lastname column.

If you do not specify a sort order, data appears in the report in whatever order it was stored in the table, view, or JoinDef on which the report is based.

## Sort Direction

*Sort direction* is another component of sort order. The *sort direction* is ascending or descending. In ascending order, the lowest values are presented first (1,2,3, a,b,c). In descending order, the highest values are given first (9,8,7, z,y,x).

## Sort Order and Data Type

Sort order is performed according to the data type of the column:

- Date columns are sorted in ascending or descending chronology.
- Numeric columns are sorted by ascending or descending numeric value.
- Character columns are sorted in ascending or descending alphabetical order.

This means that numbers entered in a character column (in an address, for instance) are sorted differently than numbers entered into a numeric column. Numbers in a character column are sorted alphabetically, as shown in the following sample table:

Numbers sorted in numeric column	Numbers sorted in character column
1	1
2	101
3	11
11	12
12	2
22	22
101	3

## Default Sort Order

When you create a default report specification or preview report, RBF sets an initial default sort order based on the values in the first column of the table, view, or JoinDef on which the report is based. Two exceptions to this are Indented reports and Master/Detail joins, which can have multiple sort columns.

The following rules apply:

- If the data type of the sort column is character, RBF sorts report rows in ascending alphabetic order on the first column.
- If the data type of the sort column is numeric, RBF sorts report rows in ascending numeric order.
- If the data type of the sort column is date, then RBF sorts report rows according to ascending chronological order.
- Except for Indented reports, if the first column contains duplicate values, RBF sorts the rows in an indeterminable manner.

For Tabular reports with duplicate values in the first column, RBF prints the duplicate value only once and does not print a value in the first column until the value changes.

## Sort Columns and Breaks

Before you create a break header or footer, you must designate the column you want to break on as a sort column. If you change the sort order of the columns, the order of the report's break headers and footers also changes.

## How to Change a Column from Sort to NonSort

If the column has an associated break header or footer, RBF does not allow you to change a sort column into a non-sort column. In this case, you must delete the associated break header or footer first with the Layout operation (see page 192).

Additionally, if you try to change a sort column into a non-sort column, you must decide what to do with any unique aggregates. You can:

- Delete all unique aggregates associated with the column
- Change the unique aggregates to non-unique aggregates

## Change the Sort Order

### To change the sort order of your report

1. Select the ColumnOptions operation from the Report Layout frame menu.  
The Column Options frame (in the seventh figure) appears, with a default sort order (see page 200) based on the first column of the table, view, or JoinDef on which your report is based.
2. Tab to the Sort Sequence field and enter the order in which you want the column sorted.  
  
You can have from 1 to 1024 sort columns, with 1 as the primary sort column, 2 as the secondary, and so forth.  
  
If you do not want to designate a column as a sort column, set the sort order to zero (0).
3. For each sort column designated, establish the sort direction:
  - a. Tab to the Sort Direction field.
  - b. If necessary, type a (for ascending) or d (for descending) over the values currently in the field and press Return. (RBF fills out the field with the complete word for you.)
4. Select the End operation to return to the Report Layout frame.

## Runtime Data Selection

Users often must limit the data that goes into a report. For example, a table can contain sales figures for the past three years, but you only want a report on sales data for the previous quarter. You can use the Column Options frame to specify that RBF ask the user to enter criteria for the data to be selected from that column when the report is run. This is called runtime data selection.

Runtime data selection is specified on a column-by-column basis. If the report has a Date column, and you specify a runtime data selection range for that column, each time a user runs the report RBF can ask the user to enter a minimum (earliest) date and the maximum (latest) date. The resulting report can only contain data from rows in which the date falls within the specified range.

You can specify runtime data selection on more than one column. For example, you can specify a runtime data selection for both the Date and Customer columns. In this case, the user could specify a range of dates for a particular customer at report runtime.

You can specify runtime data selection for both sort and nonsort columns.

Enter options in the Selection Criteria column on the Column Options frame to specify whether RBF must use runtime selection criteria. The Column Options frame provides three runtime data selection criteria options, as shown in the following table. You must specify one of these options for each column in the report.

Option	Effect at Runtime
none (or n)	No runtime selection criteria for this column. This is the default. All data is presented in the report.
value (or v)	You are asked to enter a specific value at runtime.
range (or r)	You are asked to enter a maximum and minimum range of values at runtime.

In a default report specification, all columns are set to the none option (no runtime data selection).

Each time a user runs a report for which the value or range runtime data selection option has been specified, RBF asks the user to enter the selection criteria before running the report:

- Value - If the selection criterion is a value, the user enters the value.
- Range - If the selection criterion is a range, RBF prompts the user to enter first the minimum value of the range and then the maximum value of the range.

## Hexadecimal Constants

To specify a nonprintable character, you can use a hexadecimal string constant with the following format:

`X|x'nn{nn}'`

The introductory X identifies the string as a hexadecimal string constant. You must specify the nonprintable character as two hexadecimal digits (*nn*) in the range 0-9, a-f, or A-F, and the string must contain an even number of characters. There must be no intervening white space between the **X** and the single-quoted string of hexadecimal digits. The **X** and the hexadecimal digits are case insensitive. RBF translates the hexadecimal constant into its corresponding character value.

## Null Values for Numeric Variables

If the user does not enter a value at runtime for a numeric variable with runtime qualification, RBF issues an error message. To avoid this circumstance, application developers have the following options:

- Quote the variable in the query by archiving the RBF report specification and editing it in Report-Writer.

If the user does not enter a variable, RBF interprets the quoted empty variable as a value of zero. This procedure is not recommended for fields in which this behavior is semantically incorrect—for example, in nullable fields. Once you have saved a report specification with the archive command, you can no longer edit it in RBF.

- Supply the variable value in a call report statement, or require the user to supply it on the command line, so the value is ensured to exist; then call Report-Writer to run the report.

## Column Break Options

The BreakOptions operation allows you to control how your report shows a change of value in a sort column in the following ways:

- Optionally suppress duplicate values in sort (break) columns in the Detail section of a report.
- Optionally print a new page when a sort column value changes.

You can also create break headers and footers to call attention to a change of value in a sort column. To create break headers and footers, or to use aggregate functions to summarize data in break footers, see *Create Break Headers, Footers, and Other Report Sections* (see page 183).

The BreakOptions operation is not available for Labels reports.

## Options for Showing a Change of Value

By default for most report styles, RBF prints a value in every column for every row it retrieves. If several rows contain the same value for a column, RBF prints each duplicate value. You can suppress duplicate values in sort columns to make the report more readable.

Use the BreakOptions operation to specify one or a combination of the following options for printing sort column values:

- Always print the value, including duplicates, as shown in the following figure.
- Print the value only once when the initial break occurs and suppress duplicates, as shown in the nineteenth figure.
- Print the value when a new page is printed.
- Create a new page when the sort column value changes.

Regardless of the other options you have chosen, you can instruct RBF to print a new page when the value in a sort column changes. For example, for the report shown in either of the following figures, you can instruct RBF to print a new page when the value in the Dept column changes from Admin to Account.

By default, Tabular reports always suppress duplicate values, as shown in the following figures. RBF prints the value in the Dept sort column only when it changes from Account to Admin, and from Admin to Commun.

Dept	Project	Description	Budget
Account	Asset	Asset Management	\$ 11700.00
	Portfolio	Portfolio Analysis	\$ 11200.00
Admin	EmployBen	Employee Benefits	\$ 20000.00
	TextProc	Text Processing	\$ 14000.00
Commun	Graphic	Graphic Design	\$ 18000.00
Sales	SalesFor	Sales Forecasting	\$ 9900.00
	Advertise	Advertising Analysis	\$ 9500.00
	Expense	Expense Account System	\$ 12500.00



Ingres - RBF

Report Name: projects

End of Output  
17-sep-1998

Column 1/80 Line 1/21  
18:47:39

Report on Table: projects

Dept	Project	Description	Budget
Account	Asset	Asset Management	\$ 11700.00
Account	Portfolio	Portfolio Analysis	\$ 11200.00
Admin	EmployBen	Employee Benefits	\$ 20000.00
Admin	TextProc	Text Processing	\$ 14000.00
Commun	Graphic	Graphic Design	\$ 18000.00
Sales	SalesFor	Sales Forecasting	\$ 9900.00
Sales	Advertise	Advertising Analysis	\$ 9500.00
Sales	Expense	Expense Account System	\$ 12500.00

- 1 -

Print(SH-F1) File(SH-F2) Help(F1) End(F10)

Use the Break Options Operation

To use the BreakOptions operation:

- 1. Place the cursor on a sort column and select Edit from the Report Layout menu.

The BreakOptions operation appears on the Edit submenu only if the column you select is a sort column.

- 2. Select BreakOptions from the Edit submenu.

If the selected sort column is in the Detail section of the report, the Break Column Output Options pop-up appears.

Ingres - RBF

Report-Header

Report-Header

Page-Head

Name

Name-Break-H

Detail-

c

Page-Foot

End-of-Rep

BBF - Break Column Output Options

Column: name

Specify when to print the values for this break column:

Always (yes/no): no

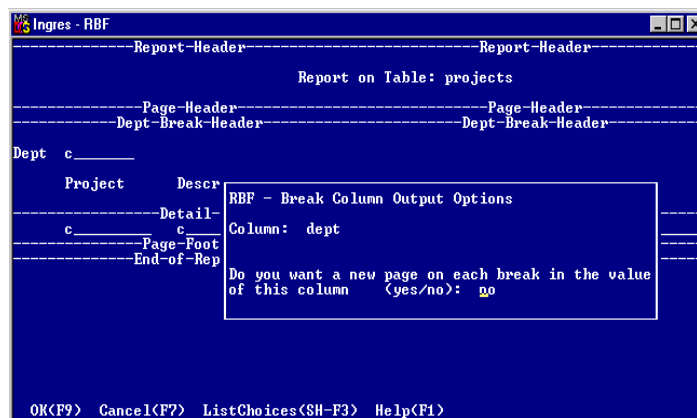
On breaks (yes/no): yes

On Page breaks (yes/no): yes

Do you want a new page on each break in the value of this column (yes/no): no

OK(F9) Cancel(F7) ListChoices(SH-F3) Help(F1)

If the selected sort column is in another section of the report, the Break Column Output Options appears.



- Specify the report's break column options, as follows:

### Always

To print a sort column value always, type yes (or y) in this field.

If you enter yes in this field, RBF automatically changes the values in the On breaks and On page breaks fields to NO.

### On breaks

To print a sort column value when it changes only, type yes (or y) in this field.

If you choose to print on breaks and the report does not have a break header for the sort column, RBF creates one.

### On page breaks

To print a sort column value when a new page is printed, type yes (or y) in this field.

If you choose to print on page breaks and the report does not have a page header, RBF creates one.

### Do you want a new page on each break in the value of this column?

To start a new page each time the sort column value changes, type yes in this field.

If you enter yes and the report does not have a break footer, RBF creates one.

- Click OK to return to the Report Layout frame.

## Move Operation—Move Report Components

Use the Move operation to move:

- Trim, columns, aggregates, and headings
- Straight edge alignment guides
- Report's right and bottom margins

Note these restrictions:

- You can only move aggregates to report, page, or break footer sections.
- The Column operation, which enables you to move columns and their associated headings together, is not available for Labels reports (you must move columns and column headings separately).

You move the straight edge alignment guides exactly as you do a piece of trim, except that none of the other report components move. The alignment guides can appear superimposed upon other report components.

## Move Trim, Columns, Aggregates, and Headings

### To move trim, columns, aggregates, and headings

1. Use the arrow keys or your mouse to position the cursor on and select the component you wish to move. You can place the cursor anywhere on the component.
2. Choose the Move operation from the Report Layout frame menu.  
The component is highlighted in inverse or blinking video if your terminal is capable of doing so.
3. Position the cursor at the new location, if appropriate. Then choose one of the operations on the Move submenu.

The Move submenu operations are as follows:

#### Place

Moves the selected component to a position indicated by the cursor. Before choosing this operation, position the cursor at the desired location. For more information, see Place and Shift Operations (see page 209).

#### Left

Moves the selected component to the current left margin of the report layout, on the indicated line.

#### Center

Moves the selected component to the center of the indicated line, as defined by the report's current margins.

#### Right

Moves the selected component to the current right margin of the report layout, on the indicated line.

#### Shift

Moves the selected component to a position indicated by the cursor, shifting other components to the right. Before choosing this operation, position the cursor at the desired location. For more information, see Place and Shift Operations (see page 209).

#### Column

This operation appears after you place the cursor on a column, aggregate, or heading. It instructs RBF to move the column or aggregate and the associated heading as a unit.

After choosing the Column operation, choose one of the other move operations to move both the column or aggregate and the associated heading. If you do not choose the Column operation, you must move columns or aggregates and the associated headings separately.

This operation is not available for Labels reports. In Labels reports, you are required to move columns and column headings separately.

**Help, End**

Perform standard operations.

**Place and Shift Operations**

The Place and Shift operations allow you to move a component to a specified location, even if it is on another line. Before choosing either of these operations, you must place the cursor where you want to move the component. The location of the cursor marks where the first (leftmost) character of the component is placed.

If the new location for the component does not affect any other component, there is no difference between the Place and Shift operations. However, if the new location causes the relocated component to overlap another component, Place and Shift handle the situation differently.

**Place**

Pushes the other component to the right. If there is not enough space for the pushed component to fit without overlapping additional components, the pushed component is dropped down one line. Thus, the Place operation affects only two components: the moved component and the component pushed out of the way.

**Shift**

Pushes the other component to the right. If there is not enough space for the pushed component to fit without overlapping additional components, the additional components are also pushed to the right to make room. If all the components cannot fit on a line, some components are forced down to the next line. A Shift operation can affect all components to the right of the moved components' new location.

**Centering**

The Center operation positions a report component in the center of the total report line width (not the width of your screen). If the report is wider than your screen, the centered component appears off-center on your screen, but is correctly centered on the printed report.

You can set report line width when you create the report specification. You can also use the Move operation to move the right margin, as explained in the following discussion.

## Move the Report Margins

You can use the Move operation to expand or contract the width or length of the report by moving the right or bottom margin of the report.

### To move a report margin

1. Place the cursor on the margin line.
2. Select the Move operation from the Report Layout frame menu.

A submenu appears with the following options:

#### Place

Places the selected boundary at the cursor position. If this causes overlap with other form components, the boundary is placed as close to the last component as possible.

#### Expand

Expands the form by a quarter of a display screen either to the right (if you are moving the right border) or down (if you are moving the bottom of the form).

#### Help, End

Perform standard operations.

3. To narrow the width or length of the report, position the cursor at the location where you want the new margin to be and choose the Place operation.

The margin moves left or up to the cursor location.

To expand the width or length of the report, choose the Expand operation. The Expand operation moves the right margin to the right by one quarter of your screen width. Expand moves the bottom margin down by one quarter of your screen's height. Each time you choose Expand, the margin expands by one quarter of your screen width or height.

RBF defines the width of a report in number of characters across the page, not in inches, centimeters, or picas. The actual width in inches of a report printed on paper can vary from printer to printer, depending on the size of the characters that the printer uses.

## Report Options Frame

The Report Options frame allows you to specify output options that apply to the report as a whole. With the Report Options frame you can specify:

- Page length in number of lines
- Character to be used for underlining
- When and where to use underlining
- Whether to insert form feeds
- How to display null values
- Whether to print the page header on the first page of the report

To call up the Report Options frame, select the ReportOptions operation from the Report Layout frame menu.

Because the Report Options frame concerns the report as a whole, cursor position on the Report Layout frame does not matter.

RBF - Report Options

Page Length <lines>: 20 (Default is 20 when report is written to a terminal and 61 to a file.)

Underlining Character: - (Default is '-' when report is written to a terminal and '\_' to a file.)

Print Page Header on first page <yes/no>: yes

Display Null Values as:

Insert Formfeeds <yes/no>: no

Report Section	Underlining
Report Header	last
Page Header	last
Name Break Header	none
Detail	none

Page Component	Include in Report	Format
Date	yes	d'03-feb-1901'
Time	yes	d'16:05:06'
Page	yes	\-zzzn \-'

ListChoices<SH-F1> Help<F1> End<F10>

## Page Length

RBF automatically formats reports in numbered pages. A *report page length* is determined by its number of lines. RBF uses two different default page lengths—one for reports displayed on a screen and one for reports sent to a file or printer:

- Default screen page length equals 20 lines
- Default file or printer page length equals 61 lines

Report page lengths include all report lines, including title, trim, headings, detail lines, headers, and footers.

You can specify a different page length for a report by moving the cursor to the Page Length field and entering a new number of lines. The number of lines you enter here is applied to reports displayed on your screen as well as reports sent to a file. If you leave this field blank, the defaults are 20 lines when the report is written to a screen and 61 when the report is sent to a file or printer. To suppress all pagination (that is, to eliminate all breaks), enter a zero (0) in this field.

Most printers print six lines to the vertical inch. Some printers print eight lines per inch and some can be set to a variable number of lines per inch.

## Underlining

The Report Options frame allows you to determine how you want to use underlining in reports written to a file or printer. Underlining is suppressed when the report appears on your screen.

The default underlining character for reports sent to files or printers is the underscore (\_). To specify a different character, such as a hyphen (-), enter it in the Underlining Character field.

Use the table field at the bottom of the Report Options frame to determine how you want to apply underlining to different types of text elements in the report. To specify how many lines of a particular report section must be underlined, move the cursor to the appropriate field and enter one of the codes listed in the following table:

Option	Action
all (or a)	Underline all lines of text in the layout area
last (or l)	Underline only the last line of text in the report section
none (or n)	Underline nothing



The default is to underline the last line in the:

- Report header (for all reports)
- Page header (for Tabular and Wrap reports only)

There is no default underlining for newly created sections.

When you are satisfied with your specifications on this frame, select the End operation to return to the Report Catalog frame.

## Page Header on First Page

If your report has a page header, it appears on the second and subsequent pages of the report. To print the page header on the first page of the report, type **yes** (or y) in the Print Page Header field.

## Display of Null Values

The Display Null Values field allows you to specify how your report presents null values. Depending on the data definition of the underlying table, a null value can be left blank or printed as some special value.

The default is to simply leave null values as blanks on the report. However, you can use this field to specify some other value. For example, you can specify that null values appear on the report as NA, Not Available, or To Be Determined.

## Form Feeds

*Form feeds* are commands sent to the printer telling the printer to advance to the next piece of paper when a report page is finished. In a report file, a form feed is usually indicated by Control-L.

By including form feeds in your report specification, you ensure that each new page of the report begins on a new piece of paper, even if the number of lines specified for the report page is not the same as the number of lines the printer can put on a page. The default is to include form feeds in the report specification. You can change this by entering **n** (no) in the Insert Formfeeds field.

If you include form feeds, you can also specify whether to print the first form feed. This initial form feed occurs at the start of the report before the first page has been printed. The default is yes. Enter **n** (no) in the Print First Formfeed field to suppress the initial form feed.

## Date, Time, and Page Components

You can choose whether to print the date, time, and page number on the pages of your report. By default, these components print in the following positions on each and every page of the report:

- Date in upper left corner
- Time in upper right corner
- Page number centered at the bottom of page

You cannot change the print position of the page number, but you can reverse the date and time or print both in the upper left or upper right corners by using certain date and time formats, as described later in this section.

You can print or suppress each component independently of the others. The default for each component is yes. Enter n (no) in the Include in Report field for any component to omit it from your report.

For each component included in the report, you can specify a print format as indicated in the following table:

Report Component	Format
Date	Date and/or time format, or a character format (cannot be a multi-line character format).
Time	Date and/or time format, or a character format (cannot be a multi-line character format).
Page	Numeric format or template. You can include characters such as hyphens or parentheses if you dereference each one by preceding it with a backslash (\).

The Date and Time components both contain the same data value, which actually includes both a date and a time value. Therefore, when specifying a date template for the Date or Time components, you can use either or both the date and time designations. The portion of the date and time string used in your template determines whether the date, time, or both appear in that component's location on the page. For instance, specifying d'16:05:06' for the Date component and d'03-feb-1901' for the Time component prints the time in the upper left corner (where the Date component would ordinarily print) and the date in the upper right corner (where the Time component would ordinarily print).

Specifying c25 for a Date or Time component displays the component as *dd-mmm-yyyy hh:mm:ss*, which is equivalent to the date template, d'03-feb-1901 16:05:06'. You cannot specify a multi-line character format such as c30.5, cj30.15, or cf30.6 for the Date and Time components.

Component	Format	Sample Result
Date	d'03-Feb-1901'	12-Sep-1993
	d'03-Feb-1901 16:05:06'	12-Sep-1993 17:07:55
	d'04:05:06 PM'	05:07:55 PM
	d'2/3/1'	9/12/93
	d'02/03/01'	09/12/93
	d'February'	September
	d'010203'	930912
Time	d'04:05:06 PM'	05:07:55 PM
	d'16:05:06'	17:07:55
	d'03-Feb-1901 16:05:06'	14-Sep-1993 17:07:55
	d'03-Feb-1901'	14-Sep-1993
Page	'\P\a\g\e zzz'	Page 123
	'nnnnn'	00123

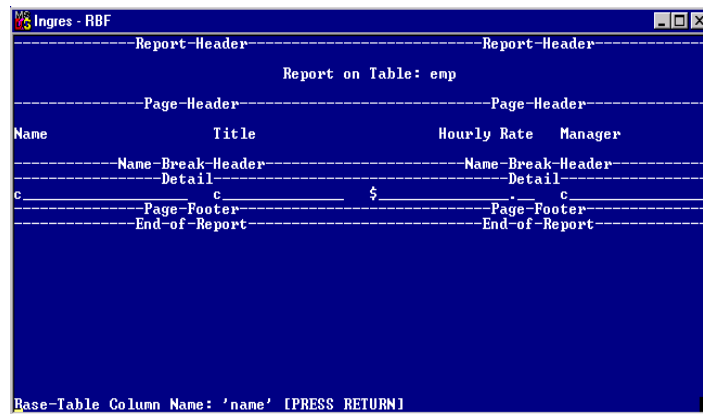
## Obtain the Name of a Column

The Report Layout frame does not automatically display the names of the columns for the data fields in your report specification. It shows the data fields only as lines and letters or symbols that indicate the maximum length and data type of the data to be included in your report. The column headings shown in the trim line can differ from the actual database table column names.

**Use the Name operation Report Layout frame to display column represented by a field in the Detail section of your report specification, as follows:**

1. Place the cursor on a field in the Detail section of your report specification.
2. Choose the Name operation.

RBF displays the name of the database column at the bottom of the screen. The following figure shows the actual column name, title, for the second column in the Detail section, whose trim heading is Position.



## Undo Operation—Undo Edits

You can reverse the effects of a Create, Delete, Edit, or Move operation by choosing the Undo operation. Undo cancels the immediately preceding operation, including another Undo operation. The report specification reverts to its original state before the operation.

Undo only reverses the last operation. Undo has no effect on the ReportOptions, ColumnOptions, BreakOptions, Save, or Layout operations.

## Save Operation—Save a Report Specification

You use the Save operation to save your report specification in RBF after creating it or to save any changes you made while editing it.

The Save operation appears on the Report Layout frame, as well as on other RBF frames and pop-up menus. When you choose the Save operation from any frame, RBF displays either the Save Report frame or the Save submenu.

The Save Report frame, as shown in the following figure, allows you to edit the report name or other general information about the report, such as a brief description in the Long or Short Remarks field. This is the same information that appears whenever you select the MoreInfo operation on the Report Catalog frame. If you are creating a new report specification, RBF always displays the Save Report frame.

If you are editing an existing report, RBF displays the Save submenu with the following operations:

### **EditInfo**

Displays the Save Report frame so you can edit the name or other information displayed on the Report MoreInfo frame.

### **Save**

Saves the report specification under its existing name.

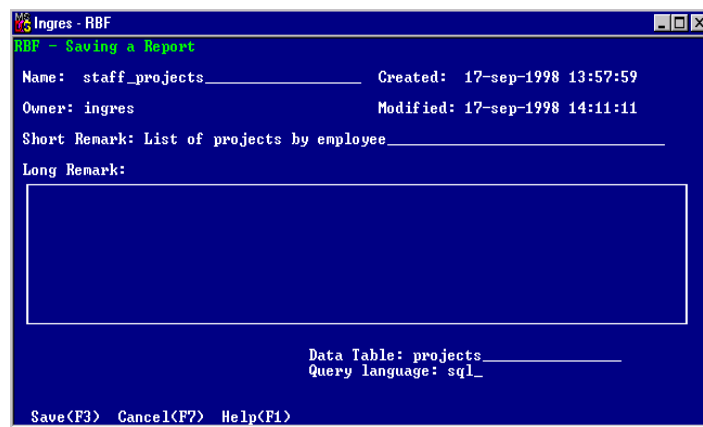
### **Cancel**

Returns you to the previous frame.

**Note:** Report specifications created for this release of Ingres cannot be edited or run in earlier releases of Ingres, due to changes in the underlying table name formats.

## Save Report Frame

The following figure shows an example of a Save Report frame.



Ingres - RBF

RBF - Saving a Report

Name: staff\_projects Created: 17-sep-1998 13:57:59

Owner: ingres Modified: 17-sep-1998 14:11:11

Short Remark: List of projects by employee

Long Remark:

Data Table: projects

Query language: sql\_

Save(F3) Cancel(F7) Help(F1)

The fields contained in the Save Report frame are as follows:

### Name

Name of the report. A newly created report specification initially has the same name as the table, view, or JoinDef on which it is based. Whenever you change the name in this field, you create a new report under the new name that includes all your changes. The old report, under its old name, is left unchanged. If you save a report without changing its name, your changes are incorporated and the old version is eliminated.

### Created

Display-only field giving the date and time the report was created. If you are saving a report for the first time, this field is blank.

### Owner

Display-only field identifying the owner of the report specification. If you copy a report owned by the DBA, your name is entered as the owner of the newly saved report.

### Modified

Display-only field showing the last date and time the report specification was saved. If you are saving a new report specification for the first time, this field is blank.

### Short Remark

Short description of the report specification. The description you enter here is displayed on the Report Catalog frame.

**Long Remark**

Long description of the report specification. The long description you enter here is displayed in the More Information about a Report frame when you choose MoreInfo from the Report Catalog frame.

**Data Table**

Name of the table, view, or JoinDef on which the report specification is based. Do not change the Data Table name to the name of a table, view, or JoinDef that does not contain the same columns as the one on which the report was originally based. If you do this, the report cannot run and you cannot be able to use RBF to edit the report specification or correct the Data Table name.

**Query Language**

Query language you want to use in retrieving the data for the report (if your installation supports more than one database language). If RBF has predetermined the appropriate query language, you cannot be able to edit this field.

The database language specified here is the one that is used when the report specification is written into the database. Additionally, if you write the report specification into a text file with the Archive operation, any .query statement generated by that operation is written in the database language specified here.

Specify SQL as the query language if your report is based on any of the following:

- JoinDef
- Table owned by a user other than yourself or the DBA
- Table, column, or correlation name that is a delimited identifier

For additional information, see Archive Operation (see page 221).

The menu operations on the Save Report frame are as follows:

**Save**

Saves the report specification in the database with the specified report name; then returns to the Report Layout frame. You can select Save as often as you want during the process of editing a report specification.

**Cancel**

Resets all field values in this frame to the original values and returns to the Report Layout frame without saving the current edits.

**Help**

Performs standard help operation.

## Use the Save Operation

### To save a report specification using the Save operation on the Save Report frame

1. Choose the Save operation from any RBF frame. If the Save submenu appears, choose the EditInfo operation.

RBF displays the Save Report frame, as shown in the preceding figure.

2. Make any changes to the report information displayed in the Save Report frame.
3. Select the Save operation.

A message near the bottom of the frame indicates that RBF is copying the edited report specification into the database, with the specified report and table names.

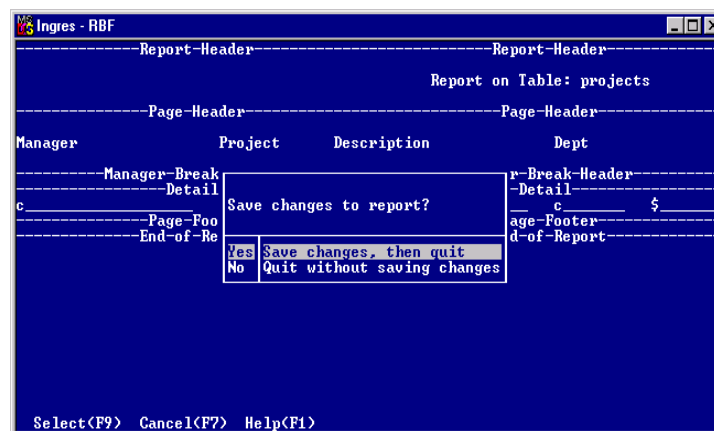
If you decide you are not ready to save the specification, select the Cancel operation instead.

When you select either Save or Cancel, RBF returns you to the Report Layout frame.

4. Continue editing the report specification, or select End to exit the frame and return to the Reports Catalog frame, or Quit to terminate your RBF session.

## Save Report Pop-up

If you select the End or Quit operations on an RBF frame without first saving the report specification, RBF displays the Save Changes to Report pop-up, shown in the following figure. Select Yes to save the report specification or No to end without saving the specification.





## Archive Operation—Archive a Report Definition

Use the Archive operation to make a text file copy of a RBF report specification that you can edit with a word processor or text editor. This allows you to use Report-Writer to add sophisticated data selection and formatting commands that are beyond the capabilities of RBF. You use the sreport command to save the edited report specification text file. Once saved, the report specification can be run at any time with the report command or the RBF Go operation to produce the report.

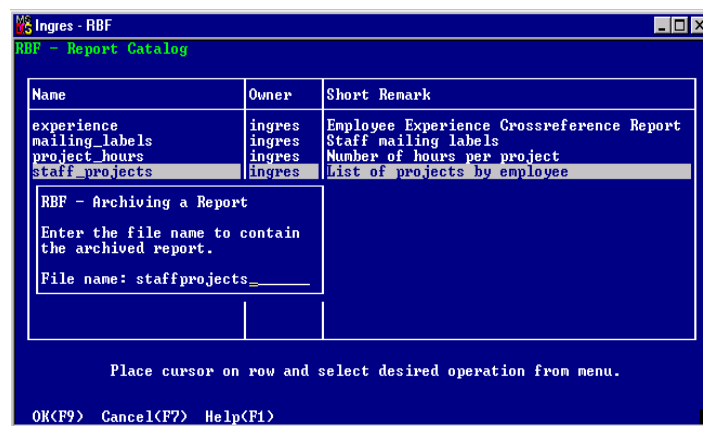
After saving a report specification with sreport, you cannot edit it from RBF. A report specification saved with sreport appears as a report title in the RBF Report Catalog frame, and you can run the report by placing the cursor on it and selecting the Go operation. However, if you choose the MoreInfo operation for a report saved with sreport, you can see that the RBF Editable Field? has been set to NO.

### Use the Archive Operation

#### To archive a saved report specification

1. On the Report Catalog frame, place the cursor over the name of the report specification that you want to archive.
2. Choose the Utilities operation.
3. From the Utilities submenu, choose Archive.

The Archiving a Report pop-up appears.



4. In the File Name field, enter a file name for the report specification text file.

The Archive operation automatically adds .rw to the end of the file name you specify (unless you add your own extension). For example, if you entered the file name staffprojects, the archived file has the name, staffprojects.rw.

## Comment Blocks in Archived Reports

RBF generates certain comment blocks in the report specification file. They are the width comment block, JoinDef comment block, and union select comment block. Comment blocks are visible when a report is archived or copied with the copyrep command.

Do not modify comment blocks in RBF reports. Changing a comment block can make the report unusable by RBF, sreport, and Report-Writer. If the report specification file has been archived, you can replace certain comment blocks with Report-Writer statements, if appropriate, as noted in the following subsections.

### Width Comment Block

Following is an example of a width comment block:

```
/* WIDTH 148
DO NOT MODIFY. DOING SO MAY CAUSE REPORT TO BE
    UNUSABLE.
*/
```

This comment block, which saves the report width, is generated for all RBF reports. If the report is archived and you want to override its width by using the .pagewidth Report-Writer statement, delete all three comment lines. For additional information, see the chapter "Report-Writer Statements."

### JoinDef Comment Block

The following is an example of a JoinDef comment block:

```
/*
DO NOT MODIFY. DOING SO MAY CAUSE REPORT TO BE
    UNUSABLE.
*
*   md
*/
```

RBF generates the JoinDef comment block for JoinDef reports only. You cannot modify or delete this comment block.

## Union Select Comment Block

The following is an example of a union select comment block:

```
/*
DO NOT MODIFY. DOING SO MAY CAUSE REPORT TO BE
    UNUSABLE.
*
The union clause is commented out because a selection
criteria on a detail table is specified.
*/
/*
union select poheader.orderno, poheader.orddate,
            poheader.vendorno, poheader.invoiceno,
            poheader.status, '' as partno, ' ' as qty,
            ' ' as unit_pr, ' ' as unit_pr, ' ' as det_tot
            from poheader poheader where not exists(select *
            from podettot podettot podettot.orderno '$det_tot'))
*/
```

RBF generates the union select comment block for Master/Detail JoinDef reports that have a selection criteria on a column in a detail table. You must not modify or delete this comment block.

## How to Copy Report Specifications

You can copy a report specification by using one of the following techniques:

- Create a report based on an existing report.
- Change the report specification name when saving it.

To create a report based on an existing report, choose the Duplicate option on the Creating a Report pop-up menu on the Report Catalog frame.

To save an existing report specification to a new file with a new report specification name, edit the name on the Save Report frame. If you have already saved the report specification under its current name, changing the name before re-saving causes RBF to save a duplicate copy under the new name.

You can also use the copyrep and sreport commands on the operating system command line to copy a report specification out of one database and into another. However, the sreport command saves the copied report specification for use with Report-Writer only, and the report cannot be edited with RBF.

## Delobj Command—Delete a Report Specification

You use the delobj command on the operating system command line to delete a report specification. With this command, you can delete a:

- Named report
- List of reports that you specify on the command line
- Several reports whose names are listed in a file
- All reports that match a wild card specification

You can also use the delobj command to delete other database objects.

# Chapter 9: Producing a RBF Report

---

This section contains the following topics:

[How You Produce Reports](#) (see page 225)

[Report Destinations](#) (see page 225)

[Background Mode](#) (see page 226)

[Produce a Preview Report](#) (see page 228)

[Report Sent to a Screen](#) (see page 230)

[Send a Report Directly to a File](#) (see page 233)

[Send a Report Directly to a Printer](#) (see page 234)

[Report Command—Run Report Specification from Command Line](#) (see page 234)

## How You Produce Reports

You can produce reports:

- With the RBF Preview operation
- By running a report specification from RBF
- By issuing the report command from the operating system

## Report Destinations

You can send a report to the following destinations:

- Default
- File
- Printer

The default destination can be a file or your screen. If the report specification contains an `.output` statement, choosing the default destination causes RBF to send the report to the file specified in the statement. Otherwise, RBF sends the report to the screen. Only report definitions created or modified with Report-Writer statements and saved with `sreport` can contain `.output` statements.

## Background Mode

**VMS:** If you send your report to a file or printer, you can either wait or not wait for it to finish running (the default is to wait). If you do not wait, you can continue using RBF while your report is running. The choice not to wait is sometimes referred to as running a report in the background or running in batch.

RBF submits the report to the batch queue, SYS\$BATCH. Be sure that your login.com file sets up your environment to run Ingres, because it is executed at the start of the batch job to run the report. ■

## Report Log

RBF logs all report statistics and errors, if any, in a report log. Because all report errors go to this log file, we recommend that you do not run reports still under development in the background. By default, RBF creates the report log in the following directory:

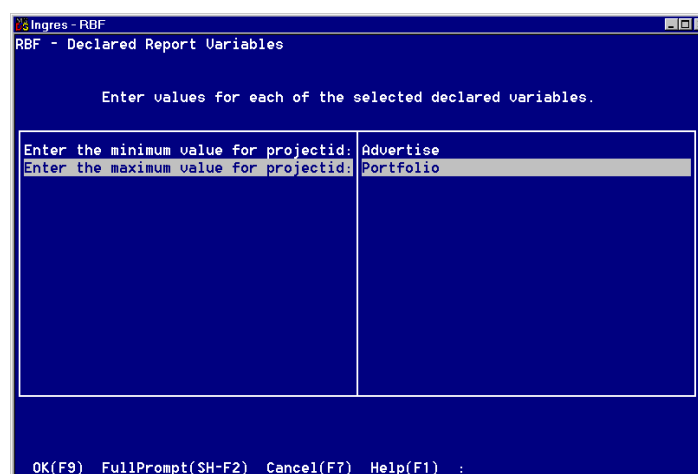
**Windows:** Your working directory.

**UNIX:** Your working directory.

**VMS:** Your home directory.

## Specify Report Variables

When running a report in the background, RBF prompts you to enter values for the report's declared variables or runtime selection criteria, if any, before sending the report to a file or printer. If the report was saved with the sreport command, the variables must have been declared with prompt. Specify values for these variables on the Declared Variables frame (in the following figure), which contains a prompt for each variable. The prompt can be up to 98 characters long, but only 31 characters are displayed. To see the entire prompt, use the FullPrompt operation.



### To specify the values

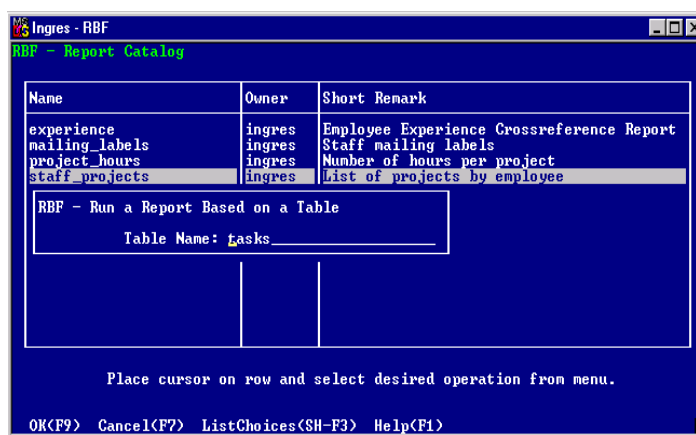
1. Enter the value in the field beside the prompt.
2. Select the OK operation.

## Produce a Preview Report

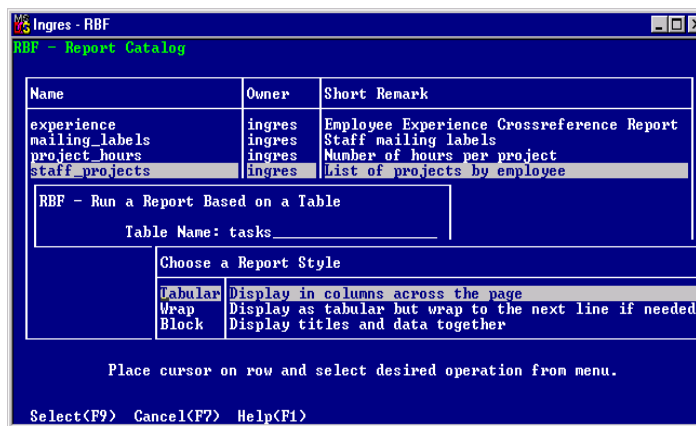
Use the Preview operation to produce a report on any table, view, or JoinDef. The Preview operation produces a report directly from the selected table or view; it does not produce a report specification.

### To run a preview report

1. Choose the Preview operation from the Report Catalog frame.
2. When the Run a Report Based on a Table pop-up appears (in the following figure), enter the name of the table, view, or JoinDef for which you want a report and select the OK operation.



3. The Choose a Report Style pop-up appears.





Place the cursor on the desired report style and choose the Select operation. You can choose:

- Tabular
- Wrap
- Block

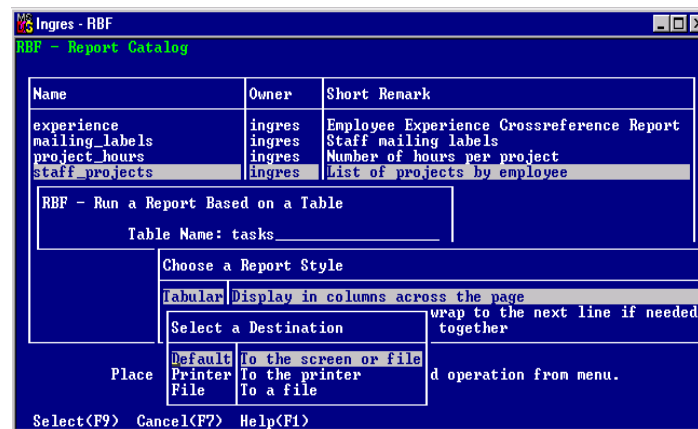
The Master/Detail, Label, and Indented report styles are not available for Preview reports.

4. When the Select a Destination pop-up appears, place the cursor on the desired destination and choose the Select operation. You can choose:
  - **Default**—The report appears on your screen (see page 230).
  - **File** (see page 233)
  - **Printer** (see page 234)

## Produce a Report from a Report Specification

### To use a report specification to produce a report from RBF

1. Go to the Report Catalog frame and position the cursor on the name of the report specification that you want to run. RBF displays the Select a Destination pop-up.



2. Place the cursor on the desired destination and choose the Select operation. You can choose:
  - **Default**—If you choose Default, the report either appears on your screen or is sent to a file, as described previously in Report Destinations.
  - **File** (see page 233)
  - **Printer** (see page 234)

## Report Sent to a Screen

When you send a report to your screen, RBF displays the first page of the report (in the following figure). If the report contains more than a single page, you can scroll through the report by using the ScrollUp, ScrollDown, ScrollLeft, and ScrollRight keys.

Report Name: emp  
Start of Output  
17-sep-1998  
Column 1/83 Line 1/63  
18:59

Report on Table: emp

Name	Title	Hourly Rate	Manager
Alcott, Scott	Sr Programmer	\$ 50.00	Wolfe, Neal
Applegate, Donald	Analyst	\$ 51.00	Wolfe, Neal
Bee, Charles	Sr Programmer	\$ 43.00	Fielding, Wallace
Belter, Kris	Programmer	\$ 33.00	Alcott, Scott
Beringer, Tom	Programmer	\$ 41.00	King, Richard
Beveridge, Fern	Project Leader	\$ 57.00	Wolfe, Neal
Bluff, Clarence	Programmer	\$ 24.00	Jones, Ashley
Bridges, Debra	Sr Programmer	\$ 48.00	Parsons, Carol
Chung, Arthur	Programmer	\$ 21.00	Ortega, Julio
Downing, Susan	Programmer	\$ 29.00	Bee, Charles

- 1 -

Print<SH-F1> File<SH-F2> Help<F1> End<F10>

After sending a report to a screen, you can:

- Use the Print operation to send the report to a printer. If you choose the Printer destination, RBF sends the report directly to a printer (either the default printer or a printer whose name you provide). If the report specification contains an .output statement, as previously described, RBF also sends the report to the file named in the statement.
- Use the File operation to send the report to a file. If you choose the File destination, RBF sends the report to a file whose name you provide. If the report specification contains an .output statement, as previously described, the file name you provide overrides the file name in the statement.
- Use the FRS printscreen command to send the contents of the current window to a file or printer.

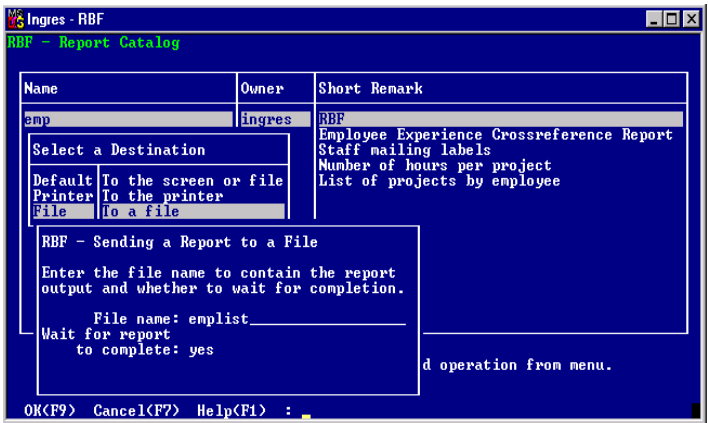
When sending a report from the screen to a .file or printer, you can send the entire report or only the executed portion. The executed portion is the portion that you have viewed on your screen. RBF formats the report to 20 lines per page. To format the report to 61 lines per page, send the report *directly* to a printer or file.

When you send a report to the screen, Report-Writer ignores any underline characters. Similarly, if you send a report from the window to the printer or to a file, underlining cannot occur in the printed or stored report output. To preserve underlining, send the report to the printer or file directly, rather than from the screen.

## Send Reports from a Screen to a File

### To send a report from your screen to a file

1. Select the File operation from the report menu. The Sending a Report to a File pop-up appears.



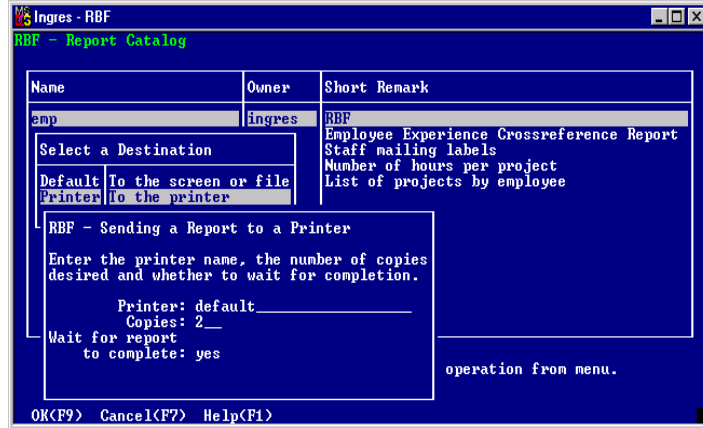
2. In the File Name field, enter the name of the file to which you want to send the report.  
You can enter a file name or full file name specification. If you enter a file name only, RBF sends the report to your working directory.
3. To file the entire report, choose the OK operation. (If you have not run the report to completion, RBF executes the entire report before sending it to the file.)  
To file the executed portion of the report only, choose the FilePartial operation.

## Send Reports from a Screen to a Printer

### To send a report from a screen to a printer

1. Select the Print operation from the report menu.

The Sending a Report to a Printer pop-up appears.



2. To send the report to a printer other than the default, type the printer name in the Printer field.
3. In the Copies field, type the number of copies to print. The default is 1 and the maximum is 999.

To send the report to the default printer, tab past this field. The default is to send the report to the default printer, which is determined by the system administrator.

4. In the File Name field, type the name of a temporary file to send the report to. You can enter a file name or full file name specification. If you enter a file name only, RBF sends the report to your working directory.

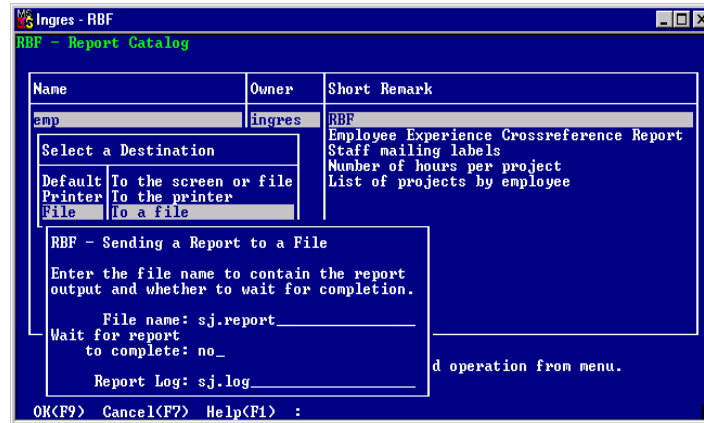
RBF first sends the report to this file and then sends it to a printer. After RBF sends the report to the printer, it deletes this file.

5. To print the entire report, choose the OK operation. (If you have not run the report to completion, RBF executes the entire report before sending it to the printer.)

To print only the executed portion of the report, choose the PrintPartial operation.

## Send a Report Directly to a File

If you send the report directly to a file, RBF displays the Sending a Report to a File pop-up.



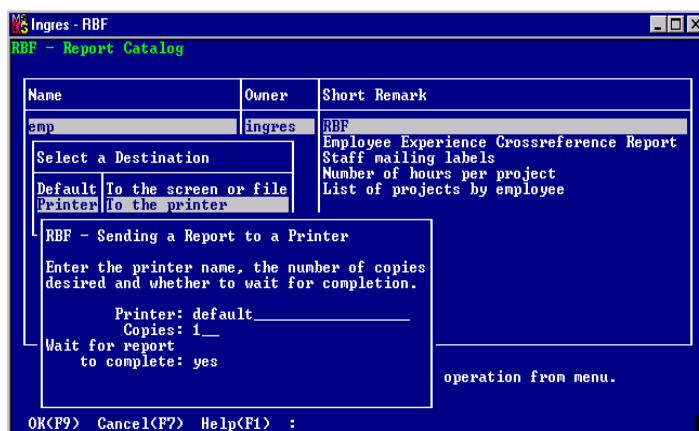
### To send a report to a file, complete the fields on this pop-up:

1. In the File Name field, type the name of the file to send the report to. You can enter a file name or full file name specification. If you enter a file name only, RBF sends the file to your working directory.
2. To run the report in the background while you continue working in RBF, type no (or n) in the Wait for Report to Complete field. The default is yes; that is, *wait* for the report.

If you enter NO, the Report Log field appears. Type the name of a file to receive report statistics and errors. (After the report runs, check this file for report errors.) If the report has variables, the Declared Variables frame appears. For instructions, see Background Mode (see page 226).

## Send a Report Directly to a Printer

If you send the report directly to a printer, RBF displays the Sending a Report to a Printer pop-up.



### To send the report to a printer, complete the fields on this pop-up

1. To send the report to a printer other than the default printer, type the printer name in the Printer field. The default is to send the report to the default printer, which the system administrator determines.
2. Type the number of copies you want to print in the Copies field. The default is 1 and the maximum is 999.
3. To run the report in the background while you continue working in RBF, type no (or n) in the Wait for Report to Complete field. The default is yes; that is, *wait* for the report.

If you enter no, the Report Log field appears. Type the name of a file to receive report statistics and errors. (After the report runs, check this file for report errors.) If the report has variables, the Declared Variables frame appears. For instructions, see Background Mode (see page 226).

## Report Command—Run Report Specification from Command Line

You can run a report from the command line using the report command.

For details on report command parameters, see the *Command Reference Guide*.

## Passing Parameters on the Command Line

You can pass parameters to Report-Writer from the command line for reports based on SQL queries. You cannot pass parameters that contain delimited identifiers on the command line if your report specification contains a QUEL query.

If necessary, for details on debugging your query, see the section on the `II_EMBED_SET printqry` option in your *System Administrator Guide*.

## Passing Numeric Variables

Suppose your report specification contains the following code:

```
.declare deptno = integer
.query select firstname, lastname, deptid from employees
       where deptid = $deptno
```

To pass in a value for the numeric variable, `$deptno`, on the command line, enclose the `variablename=value` clause in parentheses ( ):

### Windows and VMS:

```
report personnel emp (deptno=504)
```

### UNIX:

```
report personnel emp "(deptno=504)"
```

### Note for QUEL Users

Regardless of whether your report specification uses an SQL or a QUEL query, you pass values for numeric variables in the same way. Your report specification code would look like this for QUEL:

```
.declare deptno = integer
.query
range of e is emp
retrieve (e.firstname,e.lastname,e.deptid)
       where e.deptid = $deptno
```

## Passing String and Date Variables

The report specification must enclose the string and date variables, *\$dname* and *\$ddate*, in quotes that are appropriate for your query language (in this case, single quotes for SQL), as shown in the following example:

```
.declare  dname = varchar(20),
          ddate = date
.query select firstname, lastname, deptid from employees
       where deptname = '$dname' and hiredate >= '$ddate'
```

The previous example retrieves data for employees in the named department, *\$dname*, who were hired on or after a specified date, *\$ddate*.

If the variable is quoted appropriately for your query language (single quotes for SQL or double quotes for QUEL) in the report specification, use the following syntax for specifying a string or date value on the report command line:

```
report dbname tablename (variablename='value')
```

Each string or date value must be enclosed in single quotes to identify it to Report-Writer as a string. Report-Writer strips off these quotes before assigning the value to the variable in the query. Additionally, enclose the entire *variablename=value* clause in parentheses ( ). If it contains any characters treated specially by your operating system (such as parentheses in Windows NT and UNIX or slashes in VMS), the parenthetical clause must be enclosed within double quotes to pass it through the operating system.

For example:

```
report personnel emp "(dname='BL', ddate='01/01/98')"
```

If the report specification omits the query language-specific quotes around the variable (for example, where *deptname* = *\$dname* instead of where *deptname* = '\$dname'), these quotes must be supplied to the query. To do so, include them in the string or date value on the command line within the single quotes required to identify the string to Report-Writer. You must dereference the embedded single quotes required by SQL. To dereference single quotes within a single-quoted string you double them ("). Additionally, if the parenthetical clause contains characters special to your operating system, enclose the entire parameter within double quotes. For example:

```
report personnel emp "(dname='''BL''',
                      ddate='''01/01/98''')"
```

### Note for QUEL Users

The report specification must enclose the string and date variables, *\$dname* and *\$ddate*, in double quotes for QUEL, as shown in the following example:



```
.declare    dname = varchar(20),
           ddate = date
.query
range of e is emp
retrieve    (e.firstname,e.lastname,e.deptid)
           where e.deptname = "$dname" and
           e.hiredate >= "$ddate"
```

This example retrieves data for employees in the named department, *\$dname*, who were hired on or after a specified date, *\$ddate*.

If the report specification omits the query language-specific quotes around the variable (for example, where `e.deptname = $dname` as opposed to where `e.deptname = "$dname"`), these quotes must be supplied to the query by including them within the string or date value on the command line, inside the single quotes that identify the string to Report-Writer. Additionally, if the parenthetical clause contains characters special to your operating system, enclose the entire parameter within double quotes and escape any double quotes within the parameter according to the rules for your operating system.

**Windows:** In Windows, you escape double quotes by preceding them with a backslash (\). For example:

```
report personnel emp "(dname='\"BL\"' ,
  ddate='\"01/01/91\"')"
```

**UNIX:** In UNIX, you escape double quotes by preceding them with a backslash (\). For example:

```
report personnel emp "(dname='\"BL\"' ,
  ddate='\"01/01/91\"')"
```

**VMS:** In VMS, you escape double quotes by preceding them with another double quote. For example:

```
report personnel emp "(dname='\"\"BL\"\"' ,
  ddate='\"\"01/01/91\"\"')"
```

## Passing an Entire Where Clause

In the following examples, the where clause itself is a variable for the report. You pass the entire where clause to Report-Writer from the operating system command line in the *variablename=value* clause. The .query section in the report specification might look like this:

```
.QUERY select * from tablename where $wherevar
```

In this case, the *value* in the *variablename=value* syntax is a string containing the search qualifications. Enclose *valuestring* in single quotes to identify to Report-Writer as a string, as shown in the following example:

```
report accounting recpay (wherevar='valuestring')
```

If the parenthetical clause contains characters special to your operating system, enclose the entire parameter within double quotes:

```
report accounting recpay "(wherevar='valuestring')"
```

The *valuestring* can include a variable and a value. If the value in the *valuestring* is a character string or date, it must be enclosed in quotes that are appropriate for your query language (single quotes for SQL). Dereference single quotes within the single-quoted *valuestring* by doubling them (").

For instance, in the following example enclose the date string, 12/31/98, in a double set of single quotes to dereference them inside the quotes surrounding the *valuestring*:

```
report accounting recpay  
  (wherevar='date = ''12/31/98''')
```

If your operating system requires it, enclose the entire parameter within double quotes:

```
report accounting recpay  
  "(wherevar='date = ''12/31/98''')"
```

The resulting *valuestring* is:

```
date = '12/31/98'
```

### Note for QUEL Users

The entire where clause is passed to Report-Writer from the operating system command line in the *variablename=valuestring* clause. The .query section in the report specification might look like this for QUEL:

```
.QUERY retrieve tablename.all where $wherevar
```

The *valuestring* can include a variable and a value. If the value in the *valuestring* is a character string or date, it must be enclosed in quotes that are appropriate for your query language (double quotes for QUEL). For instance, in the following example enclose the date string, "12/31/93" in double quotes within the single-quoted *valuestring*:

```
report accounting recpay
    (wherevar='date = "12/31/98"')
```

If your operating system requires it, enclose the entire parameter within double quotes and escape any embedded double quotes.

**Windows:** To dereference double quotes in Windows, precede them with a backslash (\). For example:

```
report accounting recpay
    "(wherevar='date = \"12/31/98\"')"
```

**UNIX:** To dereference double quotes in UNIX, precede them with a backslash (\). For example:

```
report accounting recpay
    "(wherevar='date = \"12/31/98\"')"
```

**VMS:** To dereference double quotes in VMS, double them. For example:

```
report accounting recpay
    "(wherevar='date = \"\"12/31/98\"\"')"
```

The resulting *valuestring* is:

```
date = "12/31/98"
```

## Passing Multiple Parameters

To pass multiple parameters in a where clause, you follow the same rules for the use of parentheses and quotes as described in the Passing an Entire Where Clause section. For example, suppose your report specification contains the following code:

```
.declare wherevar = varchar(100)
.query select * from employees
      where $wherevar
```

You would enter the following command on the command line:

```
report personnel emp (wherevar='deptid=504 and
      mgr='Jones' and hiredate>='01/01/95')
```

If your operating system requires it, enclose the entire parameter within double quotes:

```
report personnel emp "(wherevar='deptid=504 and
      mgr = 'Jones' and hiredate >= '01/01/95')"
```

## Passing Delimited Identifiers

Suppose the *value* in the *valuestring* is a delimited identifier. For example, you want a list of employees with some other information about each employee that is to be determined at runtime by entering a column name.

To accomplish this, your report code looks like this:

```
.declare info = varchar (20)
.query select firstname, lastname, $info as otherinfo
      from emp
```

The information you want to select from the database is in a column whose name is a delimited identifier.

To specify the delimited identifier as a value on the command line, you must:

1. Specify the column name as a delimited identifier in editable format. For more information, see Delimited Identifiers (see page 51).

```
"phone #"
```

2. Enclose the delimited identifier and its surrounding quotes within single quotes to identify it to Report-Writer as a string value in the *variable=value* clause:

```
info=' "phone #" '
```

3. Enclose the *variable=value* clause in parentheses. If the parameter contains any characters treated specially by your operating system (such as parentheses in Windows NT), enclose the entire parameter within double quotes and escape any embedded double quotes to pass them through the operating system.

**Windows:**

Escape the double quotes surrounding a delimited identifier by preceding them with a backslash (\):

```
report personnel emp "(info='\"phone #\"')"
```

**UNIX:**

Escape the double quotes surrounding a delimited identifier by preceding them with a backslash (\):

```
report personnel emp "(info='\"phone #\"')"
```

**VMS:**

Escape the double quotes surrounding a delimited identifier by preceding each with another double quote:

```
report personnel emp "(info='\"\"phone #\"')"
```

For more information on how to pass delimited identifiers on the command line, see the *System Administrator Guide*.

## Passing String Values with Embedded Quotes

Suppose the database value for which you are entering a comparison *value* contains embedded single or double quotes, as for example:

```
"Big John's" Barbecue
```

Suppose the where clause is specified in the report as:

```
where $wherevar
```

You want to specify a value for the variable so that Report-Writer produces the following SQL query:

```
where clientname="'Big John's" Barbecue'
```

Follow this procedure:

1. Determine how the value appears in the database:

```
"Big John's" Barbecue
```

2. Enclose the string value in quotes that are appropriate for your query language (single quotes in SQL). Dereference any embedded quotes (including apostrophes) according to the rules of your query language (in SQL, precede a single quote or apostrophe with another single quote):

```
'"Big John's" Barbecue'
```

3. Enclose the entire *valuestring* within single quotes to identify it to Report-Writer as a string. Dereference any embedded single quotes within this string by preceding each single quote with another single quote. This includes any single quotes (or apostrophes) from the original value, as well as those required by your query language in the previous step.

```
'clientname="'Big John'''s" Barbecue'''
```

4. Enclose the entire parameter within parentheses. If the parameter contains any characters treated specially by your operating system (such as parentheses in Windows NT and UNIX or slashes in VMS), enclose the entire parameter within double quotes and escape any embedded double quotes to pass them through the operating system.

### Windows:

```
report accounting receivables "(wherevar=  
'clientname='\"Big John'''s\" Barbecue\"')"
```

### UNIX:

```
report accounting receivables "(wherevar=  
'clientname='\"Big John'''s\" Barbecue\"')"
```

### VMS:

```
report accounting receivables "(wherevar=  
'clientname='\"\"Big John'''s\"\" Barbecue\"')"
```

**Note for QUEL Users**

To specify a value for the variable so that Report-Writer produces the following QUEL query:

```
where clientname="\Big John's\ Barbeque"
```

Follow this procedure:

1. Determine how the value appears in the database:

```
"Big John's" Barbeque
```

2. Enclose the string value in quotes that are appropriate for your query language (double quotes in QUEL). Dereference any embedded quotes according to the rules of your query language (in QUEL, precede a double quote with a backslash):

```
"\"Big John's\" Barbeque"
```

3. Enclose the entire *valuestring* within single quotes to identify it to Report-Writer as a string. Dereference any embedded single quotes (or apostrophes) within this string by preceding each one with a single quote.

```
'clientname="\"Big John's\" Barbeque"'
```

4. Enclose the entire parameter within parentheses. If the parameter contains any characters treated specially by your operating system (such as parentheses in Windows NT and UNIX or slashes in VMS), enclose the entire parameter within double quotes and escape any embedded double quotes to pass them through the operating system.

**Windows:**

```
report accounting receivables "(wherevar= 'clientname=\\\"\\\"Big  
John's\\\"Barbeque\\\"')"
```

**UNIX:**

```
report accounting receivables "(wherevar= 'clientname=\\\"\\\"Big  
John's\\\"Barbeque\\\"')"
```

**VMS:**

```
report accounting receivables "(wherevar= 'clientname=\\\"\\\"Big  
John's\\\"Barbeque\\\"')"
```

## Prompted Runtime Variables as Parameters

Use variables in your report specification that prompt the user for a value at runtime. For string or date values, the report specification must be coded to include quotes around the variable that are appropriate for the query language. If these quotes are included in the report specification, the user can enter the value, as is, without the surrounding quotes. For example, suppose the report specification contains either the following statements:

```
.declare clientname with prompt 'Enter client's name:'  
.query select balance from receivables  
      where name = '$clientname'
```

Or, these QUEL statements:

```
.declare clientname with prompt 'Enter client's name:'  
.query  
range of e is receivables  
retrieve (e.balance)  
      where e.name = "$clientname"
```

At runtime, the user can enter:

```
report accounting receivables
```

Report-Writer displays the prompt:

```
Enter client's name:
```

The user can respond:

```
Enter client's name: Lakeside Inn
```

If the report specification omits the query language-specific quotes around a string or date variable, the user must enter these quotes along with the value on the command line. For example, suppose the report specification contains the following where clause for an SQL or QUEL query:

```
where name = $clientname
```

The user must respond to the prompt:

```
Enter client's name: 'Lakeside Inn'
```

If the value is a delimited identifier or contains embedded quotes, the user must follow their query language's rules for dereferencing quotes within the string.



## Send Reports to a Screen using Report Command

Instead of sending your report directly to a printer, you can send it to the screen, as described in Report Sent to a Screen (see page 230).

### To send a report to your screen using the report command

Specify the `-f` flag on the report command, without specifying an output file on the command line nor in an `.output` statement within the report specification.

## Report Command Examples

1. Run a default report based on the vendor table in the purchasing database, using a default format and redefining the page width, and send the output to the named printer:  

```
report purchasing vendor -m -180 -olaser2
```
2. Run the report contained in the source file, `po_rep.rw`, against the purchasing database and store the results in the `po_sum_out` file:  

```
report purchasing -ipo_rep.rw -fpo_sum.out
```
3. Run a default report in column format on the clients table owned by `mktgmgr` in the sales database and eliminate duplicate rows:  

```
report sales mktgmgr.clients -mcolumn -6
```
4. Run the report named `recpay` against the accounting database and pass in the value of the variable, `title`:

### Windows and VMS:

```
report accounting recpay  
  (title = 'Accounts Receivable')
```

### UNIX:

```
report accounting recpay  
  "(title = 'Accounts Receivable')"
```



# Chapter 10: Using Report-Writer

---

This section contains the following topics:

[What Is Report-Writer?](#) (see page 247)

[Report Preparation](#) (see page 248)

[How You Produce a Report](#) (see page 252)

[Considerations for the Finished Report](#) (see page 255)

[Types of Report Specification Statements](#) (see page 256)

[Format of Report Specification Statements](#) (see page 262)

[Sample Report](#) (see page 266)

[Report Setup and Format](#) (see page 268)

## What Is Report-Writer?

Report-Writer provides a high-level report language that allows you to quickly create sophisticated reports without having to write an application program. With Report-Writer, you can create regular production reports, as well as special reports for your application, when needed.

Report-Writer contains features and capabilities to create reports that meet your own special needs. Some broad categories of features include:

- Powerful tools to help you extract the data you want to print  
For a simple report, you can easily specify which table to access and how to sort the information. For a complex report, you can use a database query to retrieve selected rows of data to be used in the report.
- Support for nulls  
Your Report-Writer specifications can include use of logical operators, null variables, dynamic definition of null strings, and complex null expressions for specifying how null data is to be reported.
- Complete control over the appearance of the report  
You have complete control over the appearance of titles, headings, and placement of the data on the page. A powerful set of formatting commands allows you to specify exactly how you want the numbers and text information to print. Text formatting features include many features similar to word-processing capabilities, such as centering, justification, and automatic pagination.

- Powerful arithmetic capabilities

There are many functions to help you compute totals and averages over ranges of data in your report; many arithmetic functions make almost any kind of computation possible.

- Expressions and variables for report flexibility

You can customize reports using expressions and variables as arguments to Report-Writer statements. Users can assign values to variables interactively by responding to a prompt that you specify.

- Direct running of reports

You can use the report command with appropriate flags to run reports from a report specification that you saved to a text file or to the database. Running a report directly from a text file is more efficient for interactive testing.

- Parameters to dynamically change the report each time it is printed

You can specify ranges of data, table names to operate on, or any other information at the time the report is run. This allows you to use the same report formatting commands for many different reports.

## Report-Writer and RBF

Report-Writer alone provides a sophisticated language for creating and customizing reports with detailed formats and multiple data sources. Report-Writer used in conjunction with RBF provides a way for you to create a report specification in RBF, use its archive operation to save the file as a text file, and then edit the file to add additional Report-Writer features.

RBF is an interactive, visually oriented, forms-based approach to creating simple reports. It offers its own default report styles in addition to the ones available in Report-Writer. In RBF, you can edit the column headings, formats, aggregates, and other parts of a default report in an on-screen display that shows the overall layout of your report.

Although RBF is not as flexible as the full Report-Writer formatting language described in this guide, it is adequate to create your report.

## Report Preparation

Before creating a report with Report-Writer, you must decide what data and what kinds of summary information (such as subtotals) you want in your report and how it is laid out on the printed page (or screen). These factors are important in determining how the data must be extracted from the database, how the data must be ordered, and where you must place *breaks* (logical divisions between groups of data) in your report.

## How You Obtain Data for a Report

Depending on the intended use of the report, obtain the data for your report in one of three ways:

- Specify a report directly from an existing table (or view) with the `.data` statement.
- Specify a query to retrieve a specific subset of the data in the tables each time the report is run.
- Create one or more temporary tables based on one or more existing tables.

If you obtain data for your report with the `.data` statement, Report-Writer reads all rows and columns in the table each time you run the report.

You can limit the data for the report by specifying conditions in a `where` clause in the query. You can also specify a query that contains variables (generally in the query's `where` clause) to be specified each time the report is run.

Another way you can obtain data for the report is to create a temporary table by specifying a series of query language statements in a `.setup` section. Create this section with the optional `.setup` statement in conjunction with the required `.data` or `.query` statement in your report. The `.setup` section can also be customized at runtime, using variables.

## Sorted Data in a Report

Most reports display sorted data. It is often desirable to sort the data for the report for the sake of readability and usability. For instance, if you want a list of employees in order by job title within each department, you must sort the data in the table. Reports with subtotalling require sorted data.

## Breaks in a Report

*Breaks* are divisions between parts of a report (such as page breaks) or between groups of data in your report (for instance, between data for Employee 1 and Employee 2). You specify breaks between groups of data by designating certain columns in a report as *break columns*. A break occurs when Report-Writer encounters a change of value in a break column while reading the data.

You can instruct Report-Writer to perform some action after a break has occurred by placing formatting instructions, called *break actions*, in a *header* or *footer* section associated with the break column. For example, you can instruct Report-Writer to print heading information for the next group of data rows, print summary information for the data rows associated with the last break column value, or skip to a new page and print a page header.

### Automatic Report Breaks

When running a report, several types of breaks occur automatically:

- **Start-of-report (report header)**—When the report begins, a *start-of-report* (or report header) break occurs. This break can be thought of as a change of value from no data to some data. Use this break to specify titles and other heading information that appears once at the top of the report.
- **End-of-report (report footer)**—When the report finishes, an end-of-report (or report footer) break occurs. This break can be thought of as the change from some data to no data. Use this break to specify information that is only printed once, at the end of the report, such as grand totals, footnotes, and so forth.
- **Detail**—When each data row is read by the report, a *detail* break occurs. For the detail break, detailed printing of data items is most commonly specified.

### Page Breaks in a Report

You can also specify break actions to occur at the top and bottom of pages. Page breaks occur when the report comes within a specified number of lines at the end of the page. You can define the page size to fit your needs, or specify a page footer to be printed at the page break, followed by a page header at the top of the next page. You might want to print page numbers, the current date or time, values of data items currently being processed by the report, or any number of other items.

## Headers and Footers in a Report

*Headers* and *footers* are sections of your report code where you put instructions to tell Report-Writer what actions it must perform at the break for that part of the report.

You can specify headers and footers for the report itself, for page breaks, and for those columns you have designated as break columns. The page header appears at the top of each page, except the first. The only header you can specify at the top of the first page is the report header.

If you specify break columns, the header section is executed each time the column value changes, and the footer section is executed before the new value of the column is retrieved.

Report-Writer performs the specified footer actions after it has processed all rows in a group of data (indicated by the break designation). For instance, you can specify a footer action to occur after Report-Writer has read all rows for the entire report, all rows which can fit on a page, or all rows having the same value in the specified break column.

The footer section often contains instructions for calculating and printing subtotals or other summary information at the designated break. The special functions used to calculate this information are called *set functions* or *aggregates*, which you specify in print statements. A header action, if specified, can occur at the start of the report, at the start of a new page, or before the next group of data is processed.

You can specify both footer and header actions to occur in response to a change of value in a break column. Report-Writer performs the footer actions on the previous group of data rows and the header actions for the group yet to come. At the end of the report, however, Report-Writer performs only footer actions, because there is no more data. Similarly, at the start of the report, a break in each of the break columns occurs and Report-Writer can perform header actions for each of the major to minor break columns. However, because there is no previous group of data, footer actions are not relevant.

## Detail Section

*Detail* instructions are Report-Writer statements that format, position, and print a row of data retrieved from the data table. The detail instructions are grouped together in a *detail section*.

## How You Produce a Report

The following steps are required to produce a report with Report-Writer. These steps are described in detail in the sections below.

1. Create the report specification by entering your Report-Writer statements in a text file.
2. Save the resulting text file, or store the report specification in the database using the sreport command.
3. Execute the report using the report command.

## Ways to Create a Report Specification

The report specification is a text file consisting of Report-Writer statements that define the content and format of your report.

You can create a report specification in one of the following ways:

- Create Report-Writer source code from scratch in a text file using any text editor and the language described in this guide.
- Use RBF to create a default report based on available default styles and the table or view of your choice; then edit the report in Report-Writer, if necessary.

Use the RBF Archive operation to save a RBF default report specification as a text file that you can edit in Report-Writer. The Archive operation allows you to choose a name for the text file. You can then edit the file to customize the report to your needs by adding or deleting Report-Writer statements.

**Note:** Once a RBF report specification has been archived for use with Report-Writer and stored in the database with the sreport command, the report specification can no longer be edited in RBF.



## Sreport Command—Save a Report Specification

The sreport report specifier command is used to compile and store the report specification in the database. The sreport command reads the text file containing your report specification statements, which are described in this document, and performs the following tasks:

- Performs rudimentary syntax checking
- Loads the report specifications into your Ingres database from your base text file, including any additional Report-Writer code in files specified in .include statements

**Note:** If your database and database server reside on another system called the remote host, whenever you specify a database, indicate the v\_node (virtual node) name of the remote host.

- Enters the report name in the Reports Catalog

If a report of the same name does not already exist in the Reports Catalog, sreport adds the specification into the database. If a report with the specified name already exists, sreport replaces the old report specification with the new one. The sreport command only adds or replaces a report if it detects no errors.

After you have stored a report specification with sreport, it appears in the RBF Report Catalog frame under the report name in your specification. You can run the report from RBF to produce it, but it cannot be edited from RBF.

**Note:** The sreport command compiles and stores a report for later use—it does not actually run it. You execute a report with the report command. If you want, you can execute a report directly from a text file without compiling and saving it to the database. For details, see the following section.

## Report Command—Execute a Report Specification

After you have successfully stored the report, you can run Report-Writer using the report command to create the desired output. Or, if you choose to run the report without compiling and storing the specification, you can use the report command with the `-i` parameter to read the report specification directly from a text file.

Executing a report directly from a text file is useful if you are testing the report for content and appearance and do not want to save the specification until it meets your standards. However, once you are satisfied with the report, you must store the specification in the database, using the `sreport` command.

After a satisfactory specification has been stored in the database, execute the report command each time you want to produce a new report. You need not execute `sreport` before issuing the report command, unless you modify the report specification.

Executing the report command performs the following tasks:

- Loads the report specification by reading in the report specification created by RBF or stored by an `sreport` command and performing additional error checking
- Runs the database query to extract the data (if specified)
- Writes the formatted report either to a file, to your screen, or sends the output to a printer

The report command offers the option of producing a default report from a specified table or view name. You can specify one of three available styles as the format for the default report:

### **Wrap**

The wrap style lists the data in columns across the page, with column headings at the tops of the pages. For tables with too many columns to fit across the page, Report-Writer calculates a convenient place to wrap the remaining column headings or entries to the next line, directly beneath the first line of headings or entries. In this style of report, the column entries for a single record (row) can be displayed on consecutive rows of the report, with data in columns from the far right of a table wrapped beneath data in the initial columns.

### **Tabular (also known as column)**

The tabular style, like the wrap style, also lists the data in columns across the page, with column headings at the top of the page. However, depending on your output device, Report-Writer either truncates lines wider than the page or wraps them around to the next line.

**Block**

The block style lists data without column headings, preceding each data value with the name of the column from which it comes and organizing the columns into blocks of data for each record.

If you do not specify a report style with the table or view name, Report-Writer chooses as the default style either the tabular (column) or block style, whichever is best suited for the report. Additional styles are also available through RBF.

## Considerations for the Finished Report

Before developing your report specifications, you must know how you want the finished report to look. Consider some of the following issues:

- What data is needed from your database to create the report?

If you must run a database query to get the data, design the query and run it from a terminal monitor to make sure it retrieves the desired data.

- How do you want the data sorted?

If you want headings or footings for subgroups of your data, the data must be sorted on the columns that define the subgroups. You must choose the sort.

- What do you want the various headers and footers to look like?

Decide whether you want titles, subtotals, or other aggregates, extra blank lines, or other types of headers or footers in your report. Sketch the report layout on a piece of paper to see how it looks.

- What information must be printed for each specific data row of the report, and in what format must the information appear?

For numbers, you must think about the number of significant digits to print, and the number of decimal places.

- What kind of page headers and footers do you want?

## Types of Report Specification Statements

To specify a report, you must create a text file containing report formatting and structure statements. These statements define the data to be reported, the order of the data, the page layout, explanatory text to be inserted in the report, and the position and format of titles and data items.

There are seven types of report specification statements:

- Report Setup Statements
- Page Layout and Control Statements
- Report Structure Statements
- Column and Block Statements
- Text Positioning Statements
- Print Statements
- Conditions and Assignment Statements

## Report Setup Statements

Report setup statements are used for setting up the overall report environment:

- The `.name` statement names the report.
- The `.delimitedidentifier` statement enables recognition of delimited identifiers for table, view, user, and column names in report specifications.
- The `.setup/.cleanup` statements perform data preparation and cleanup tasks (specified through query language statements). The `.setup` tasks occur before processing the query, whereas `.cleanup` tasks occur after processing the query, but just before exiting the report.
- The `.shortremark` statement allows you to provide a brief description about the report. This information is included in the Reports Catalog and the Save window in RBF.
- The `.longremark` and `.endremark` statements allow you to enter a large amount of descriptive text to describe the report specification. This text is displayed in the RBF Save window and on the Reports Catalog frame in RBF when you choose the MoreInfo operation.
- Comments can be placed in the text file of report specifications if preceded with `/*` and followed with `*/`. Comments are ignored in report processing.
- The `.output` statement sets up an external file to receive the report output.
- The `.data` or `.query` statements define the data to be used by the report output.
- The `.sort` statement defines the sort order of the data for the report.
- The `.break` statement specifies the break columns for the report and the order in which to process occurrences of breaks.
- The `.declare` statement declares variables to be a given data type and allows definition of a prompt string or value string if required for that variable.
- The `.include` statement specifies the inclusion of Report-Writer code residing in different files, and is executed when you run the `sreport` command to store the report specification.
- The `.cleanup` statement performs initialization tasks, (specified through query language statements) just before exiting the report.

## Page Layout and Control Statements

You can specify the layout of report pages with the following statements:

- The `.pagelength` statement defines the page length, in lines.
- The `.pagewidth` statement defines the page width, in characters.
- The `.formfeeds` statement instructs Report-Writer to insert formfeed characters to force a page break at the start of the report and at the end of each page.
- The `.noformfeeds` statement suppresses the addition of formfeed characters to the end of pages in the report.
- The `.leftmargin` statement sets up a left margin as the default start of all new lines and for use with the `.left` and the `.center` statements.

If the left margin is not explicitly specified, Report-Writer determines this default automatically from an analysis of the other report formatting statements. For more information, see Automatic Determination of Default Settings (see page 282).

- The `.rightmargin` statement sets the right margin of the report for use with the `.right` and the `.center` statements.
- If the right margin is not explicitly specified, Report-Writer determines this default automatically from an analysis of the other report formatting statements. For more information, see Automatic Determination of Default Settings (see page 282).
- The `.need` statement tests for a given number of lines on a page to see if a page break is appropriate.
- The `.newpage` statement skips to a new page, and optionally sets a page number.

## Report Structure Statements

The following statements are used in setting up the structure of the report:

- The `.header` statement designates a group of formatting statements for the heading of the report, the page, or one of the break or sort columns.
- The `.footer` statement designates a group of formatting statements for the report footer, the page, or one of the break or sort columns.
- The `.detail` statement designates a group of formatting statements for each data row in the report.

## Column and Block Statements

You can use column and block statements to set up an explicit print position, column width, and format for the values contained in the named database column or for a report *block* (as defined by a `.block` statement).

For instance, you can use the `.position` and `.width` statements to assign the starting print position and column width for a column or block, which are used in conjunction with text positioning statements (see page 260) such as `.tab` and `.right` justification. If you do not explicitly specify column print positions, column widths, and column formats with these statements, then Report-Writer assigns defaults automatically (see page 282) from an analysis of the other report formatting statements.

Use the following column and block statements:

- The `.format` statement explicitly specifies a print format (such as character string or standard decimal notation) for a column.
- The `.tformat` statement temporarily changes the print format for a column, only for the next value to be printed.

This statement is used for such functions as printing a value of a column on the first line of a page or of a group only, or including a currency symbol only on the first printing of a column value.

- The `.position` statement sets up an explicit starting print position for a column, which can be used with the `.tab`, `.right`, `.left`, or `.center` statements.
- The `.width` statement sets up an explicit width for a column, to be used with the `.right` or `.center` statements.
- The `.block` and `.endblock` statements allow you to treat sections of the report as blocks.

This enables you to refer to positions on previous as well as on subsequent lines in the report. These statements can be used in conjunction with the `.top` and `.bottom` statements to align blocks of data horizontally adjacent to each other rather than in vertical sequence.

- The `.top` statement, used while in block mode, moves the current position to the top line of the current block.
- The `.bottom` statement, used while in block mode, moves the current position to the bottom line of the current block.
- The `.within` and `.endwithin` statements allow you to temporarily set the report margins to the confines of a specific column, using its position and width.

## Text Positioning Statements

Text positioning statements are used to specify a print position—absolute or relative to other positions—for any text to be printed. Most of these statements also accept the name of a column for which a print position or column width has been set with the `.position` or `.width` statements as value.

Use the following text positioning statements:

- The `.tab` statement tabs to an exact or relative position before continuing printing.  
It can be used with the name of a column to tab to the specified or default print position for that column.
- The `.newline` statement prints out the current line and skips to the start of a new line.
- The `.center` statement centers text around the center of the page or around a specified alternate position.  
It can be used with the name of a column to center the text within the specified or default margins for that column.
- The `.right` statement right justifies text to the right margin or to a specified position.  
It can be used with the name of a column to right justify text within the specified default margins for that column.
- The `.left` statement left justifies text to the left margin or to a specified position.  
It can be used with the name of a column to left justify text within the specified or default margins for that column.
- The `.lineend` statement tabs to the end of the text on the current line before continuing to print.
- The `.linestart` statement tabs to the left margin before continuing to print.



## Print Statements

Use these statements to print text or data values in a report:

- The `.print` statement prints text or values at a default position, or at a position which was previously specified with the column and block and/or text positioning statements.
- The `.nullstring` statement specifies a string of characters you want to print in the report whenever a null value is encountered in the data.
- The `.underline` and `.nounderline` statements control underlining for sections of text.
- The `.ulcharacter` statement sets up a different underline character than the default, for use with Report-Writer underlining statements.

Use an *expression* in the `.print` statement syntax to specify the text or value you want to print. Expressions can include any (or any combination) of the following:

- Column names from the data retrieval statement
- Variables
- Constants
- Functions
- Aggregates
- Special report variables, such as the `current_time`, `current_date`, or `page_number`

You can optionally indicate the print format within the syntax of the `.print` statement, or you can specify it in a separate `.format` or `.tformat` statement for column values.

## Conditional and Assignment Statements

Use these statements to specify alternative blocks of statements or to assign values to variables:

- The `.if`, `.then`, and `.else` statements specify alternative blocks of statements to be executed under specified conditions.
- The `.let` statement assigns a value to a variable, which can be used in subsequent computations.

## Format of Report Specification Statements

Specify report formatting statements with a keyword proceeded by white space and a period (.), and optionally followed by parameters. The general format of a report formatting (specification) statement is:

```
.statement {parameters}
```

where:

### ***statement***

Is one of the text formatting statements, such as .data or .tab. You can specify statements in uppercase, lowercase, or mixed case letters.

### ***parameters***

Are optional parameters. Parameters take many different forms, depending on the specific statements. In many cases, parameters to the statement can also be variables and expressions. To obtain the value of a variable, precede the variable with a dollar sign (\$).

Here are some examples of report formatting statements; they include a .tab, .newline, .header, .println, and a .sort statement:

```
.tab $first_col  
.newline  
.header report  
.println 'This is the value of:',abc(f10.2),  
        ' Sum:',sum(def)  
.sort a,b,c
```

The sample reports in the appendix, "Report-Writer Report Examples" demonstrate the correct specification of statements.

## Statement and Parameter Delimiters

You must precede and follow a report formatting statement with white space, either by explicitly entering spaces or tabs before and after a statement, or by using line breaks to separate the statements. White space following a statement must occur between the statement and any numeric parameter following the statement, as in:

```
.pagewidth 80
```

Lack or incorrect placement of white space statement delimiters can result in incorrect statement interpretation and could cause an error.

Statements can span any number of lines. Except where otherwise noted, commas separate multiple values for a parameter within statements, such as in the .sort or .print statements.

If a parameter is a user name, column name, or table name (including view names and synonyms), you can delimit it with double quotes (") to include spaces or other characters that are usually disallowed in these names. For more information, see Delimited Identifiers in the chapter "Report-Writer Expressions and Formats."

## Schemas for Owner Qualification

A *schema* is a collection of database objects, such as tables. Each table, view, or synonym belongs to a schema that is determined when the object is created. The schema name corresponds to the user who owns the object. The schema name helps distinguish between objects with identical names and different owners.

In report specifications containing a SQL *.query* or *.data* statement, you can qualify a table name, view name, or synonym by specifying the schema to which it belongs (which also implies its owner), using the following construct:

*schema.objectname*

This allows you to access a table, view, or synonym owned by a user other than yourself or the DBA, if you have the correct permissions to access it.

A period (.) must immediately follow the schema name, although white space following the period is allowed. For example, Report-Writer allows the following construction:

*schema. objectname*

Both the schema name and the object name (table name, view name, or synonym) can be variables or delimited identifiers. A *schema.objectname* construct in which both the schema and table name are variables would take the form:

*\$schema.\$objectname*

A *schema.objectname* construct in which both the schema and object name are delimited identifiers would take the form:

*"schema name"."object name"*

A separate set of double quotes must surround each delimited identifier. For more information on delimited identifiers and variable substitution, see the chapter "Report-Writer Expressions and Formats."

You can use the *schema.objectname* qualification within the following statements:

- .cleanup*
- .data*
- .declare* (in the with value string)
- .query* (SQL only)
- .setup*

If you do not qualify the table, view, or synonym with a schema, Report-Writer searches for the specified object in the following order:

1. Objects owned by the current user.
2. Objects owned by the DBA to which you have been granted access.
3. Objects in the system catalogs.

### QUEL User Notes

If you are a QUEL user, you cannot use schemas or owner qualification for a table name, view name, or synonym in a report specification that contains a QUEL query. If you use the *schema.objectname* construct in reports with QUEL queries, Ingres tries to interpret the construct as a Report-Writer statement, which generates a runtime error.

## Summary of Report-Writer Specifications

In summary, the general specification of a report can be defined as a collection of distinct groups of related statements. Some of these statements relate to the overall composition of the report and some relate to major action groups within a report:

- **The Report Header**—At the start of the report, you can specify some textual information to print and set up many of the report layout specifications, such as page size and margins. The report header precedes the page header on the first page of the report.
- **Page Headers and Footers**—At the top of each page, except the first page, you can specify that Report-Writer print a page header, and at the bottom, a page footer. These usually include titles, page numbers, the date and time the report was printed, and so on.
- **A Break Header**—When Report-Writer detects a change of value in any of the designated break columns, a break occurs. Before Report-Writer processes a new group of data rows, it performs the break header actions. Break headers often highlight information such as the value of the break column, as well as textual information.
- **Detail Section**—Report-Writer processes this group of statements as it reads each new row of data. These statements generally include the instructions necessary to format and print specific data items. The detail break is the only break that does not include a header and a footer.
- **A Break Footer**—Report-Writer processes the break footer at the end of a group of data rows (as determined by the next break). A break footer often prints the current value of the break column just before the break, or prints a subtotal associated with the data rows just processed.
- **The Report Footer**—At the end of the report, you can specify some textual information to be printed.

## Sample Report

The following example demonstrates a simple report using Report-Writer. This report specification was created with a text editor, stored within the database with the `sreport` command, and run with the `report` command.

The report shows a titled listing of data from an existing table, `edat` in a database. The `jobcat` column is displayed only once for each job category value.

```
/* Sample report */
.NAME          jobcat
.OUTPUT        jobcat.out
.DATA          edat
.SORT          jobcat, name
.HEADER REPORT
  .NEWLINE 2
  .CENTER
  .PRINT 'Sample Report'
  .NEWLINE 2
.HEADER jobcat
.TFORMAT jobcat(' zzzz ')
.DETAIL
.PRINT jobcat(b8), name(c15), dept,
      code, age, sales(f12.3)
.NEWLINE
```

The report output looks like this:

					Sample Report	
10	Adams,Joe	toy	0	22	10500.000	
	Green,James	toy	0	34	43645.000	
	Smith,Tony	acct	0	48	8690.000	
20	Davis,Miles	music	0	56	234987.000	
	Tanhous,Karl	music	0	20	18765.000	
30	Jones,Mary	acct	1	34	34599.000	
	Maney,Sikkim	none	1	51	15333.000	
	Mellon,Tim	toy	0	24	45098.000	
	Mellon,Tim	any	0	44	67876.000	
	Norris,Bill	acct	0	22	23988.000	

This list briefly explains each statement's function:

- The `.name` statement gives a name to the report, which is placed in the Reports Catalog by the `sreport` facility and is used by the `report` command to locate the report specifications.
- The `.data` statement identifies an existing table or view in the database that contains the data to report.
- The `.sort` statement indicates the order the data is to be displayed on the report.
- The `.header report` statement begins the section of Report-Writer statements to be performed at the start of the report.

- The `.newline`, `.center`, and `.print` statements are used in positioning and printing a title.
- The `.header jobcat` statement begins the section of Report-Writer statements to be performed any time the value in the `jobcat` column changes. The `.tformat` statement is used to temporarily change the normal print format of the `jobcat` column on the next printing of `jobcat`, which occurs in the `.detail` section.

Normally `jobcat` is not printed. Its format is `(b8)`, which means "print 8 blank spaces." The `.tformat` statement makes a change to the format to enable a one-time printing of this value, so the actual value of the `jobcat` column is output. The `.detail` statement begins the specification of Report-Writer statements to be performed on each row in the data table.

The `.print` statement within the detail section prints out the values of the columns in the formats given after the column names, or the default format for that type of data item, if no format is specified. The chapter "Report-Writer Expressions and Formats" describes the format specifications that appear in the parentheses following the column names in detail.

This table shows the table definition data on which the report was run:

Column Name	Type	Length	Nulls	Defaults
jobcat	integer	4	yes	no
name	char	15	yes	no
dept	char	6	yes	no
code	integer	1	yes	no
age	integer	2	yes	no
sales	money		yes	no

This table shows the data for the sample report:

jobcat	name	dept	code	age	sales
10	Adams, Joe	toy	0	22	\$10500.00
10	Green, James	toy	0	34	\$43645.00
10	Smith, Tony	acct	0	48	\$8690.00
20	Davis, Miles	music	0	56	\$234987.00
20	Tanhaus, Karl	music	0	20	\$18765.00
30	Jones, Mary	acct	1	34	\$34599.00

jobcat	name	dept	code	age	sales
30	Maney, Sikkim	none	1	51	\$15333.00
30	Mellon, Tim	toy	0	44	\$67876.00
30	Mellon, Tim	any	0	24	\$45098.00
30	Norris, Bill	acct	0	26	\$23988.00

## Report Setup and Format

At the beginning of your report specification file, you must include some statements to perform the following setup tasks:

- Name the report.
- Set up a report results file.
- Specify SQL statements to perform setup and cleanup tasks that occur before the main query is processed.
- Specify the table, view, or query from which data is to be obtained.
- Specify the optional inclusion of Report-Writer formatting statements residing in different files.
- Define the order in which the data is to be sorted.
- Define the break columns for the report.
- Declare any variables used in the report specification.
- Enter optional remarks and comments.

Use the `.name` statement to name the report, the `.output` statement to set up the report results file, and the `.declare` statement to declare variables for creating custom runtime query or formatting criteria.



Additionally, you can use the optional `.setup` statement to perform set up tasks (such as setting the lock mode, or creating a temporary table), and the `.cleanup` statement to perform clean up tasks (such as dropping a temporary table).

To obtain the data for your report, use *either* the `.data` or the `.query` statement. These statements are mutually exclusive. Use the `.data` statement to name a table or view in the database from which data can be obtained. Use the `.query` statement to retrieve a subset of the available data, based on the results of the query. By including parameters in the query, you can allow users to choose the criteria for the report at runtime.

To sort the data for your report, you must include a `.sort` statement in your report specification. The `.sort` statement lists the columns, in order of precedence, on which the data can be sorted. You also must specify the break columns, using the `.break` statement, if you want breaks to occur between data items in columns other than those specified in the `.sort` statement.

To ensure standardization between reports and to take advantage of repeated report formatting statements, you can create commonly used Report-Writer formatting statements in text files for inclusion in report specifications. The `.include` statement allows you to specify the text file containing the formatting statements. This statement can appear anywhere in your specification, if it is logically correct and is executed when you run the `sreport` command to store the report specification.

In addition to the report setup statements described above, you can include in your report specification some descriptive text about your report. The `.shortremark` and `.longremark` statements can be used to include text that appears on the RBF Save frame and in the MoreInfo display on the RBF Reports Catalog frame. Additionally, you can include comments anywhere in the report specification by enclosing them with the comment delimiters `/*` and `*/`. The chapter "Report-Writer Statements" discusses the use and syntax of remark statements and comments in detail.

## How Variables Are Used in a Report

For more dynamic report specifications, you can design a report using variables. You can use variables in queries or report formatting specifications. The values of the variables are specified at runtime by:

- Assigning initialization values using the `.declare` statement
- Responding to prompts for values
- Assigning values using the `.let` statement
- Passing values as parameters on the command line

You are *strongly* encouraged to define all variables with the `.declare` statement. Although Report-Writer recognizes any name preceded by a dollar sign (\$) as a variable, undeclared variables assume default types and characteristics that are often incompatible with their intended use. By defining variables with `.declare`, you can use the variables in all of the ways described previously. If a variable has *not* been defined through the `.declare` statement, you can assign it a value only by using a command line parameter or by entering a value in response to a runtime prompt. A value cannot be specified for an undeclared variable with a `.let` statement. In addition, unless the variable has been declared, attempting to pass a parameter with a null value to Report-Writer can produce incorrect results.

During execution of the report, the value assigned to a variable from data retrieval can vary, depending on the placement of the assignment statement in various `.header`, `.footer`, or `.detail` sections.

**Note:** A report cannot be run in the background if it contains undeclared variables.

Many Report-Writer statements accept variables, alone or in expressions, to allow users flexibility in determining report specifications.

For information on these specific statements: Report Setup Statements, Page Layout and Control Statements, Report Structure Statements, Column and Block Statements, Text Positioning Statements, Print Statements, and Conditional and Assignment Statements, see the chapter "Report-Writer Statements."

Procedures for using variables are common throughout Report-Writer statements. Exceptions are the `.declare` and `.let` statements, which are used to define the variables.

Using variables in the `.query` statement allows the end user to retrieve data that meets particular needs. For instance, the user can obtain a report on a single employee or on all employees in a specified department by entering the employee name or the department name at runtime.

To create a report with user-specified variable values, precede the variable with a dollar sign (\$) when specifying it in the query. At report runtime, the user can either put the value for the variable on the command line when invoking the report, or respond to a program prompt.

For example, suppose you have a banking database in which you keep a table of customer accounts. In this table you have fields for customer names (custname), customer account numbers (custno), checking account balances (checking), and savings account balances (savings). You want to create two reports. They must be identically formatted, but must present different information; one report must provide checking account balances, and the other must provide savings account balances. To accomplish this task you can write a query like this:

```
.declare account_type = c10 with prompt
'Please enter "savings" or "checking":'
.query
  select custno, custname,$account_type as val
    from account
```

The value of the variable *account\_type* is the column name checking or savings rather than a string value.

As Report-Writer generates your report, it prompts you to enter an account type (savings or checking). Your response tells Report-Writer which kind of information it must retrieve with the query. If you respond to the prompt with checking, the completed query looks like this:

```
select custno, custname, checking as val
from account
```

For variables declared with the `.declare` statement, you can create a customized prompt string by using the `with prompt` clause, or you can specify an initial value for the variable by using the `with value` clause. For those variables that are undeclared, Report-Writer uses a default prompt string when prompting the user for the variable value.

You can also use variables in titles and other places within the report. For details on using variables in reports, see the `.query`, `.declare`, and `.let` statements in the chapter "Report-Writer Statements."

## Reports that Use Multiple Tables

At times, you must use Report-Writer to produce a report from related information scattered across several tables that share one or more column definitions. You can do this several ways. You can specify the tables in a query in the `.query` section of your specification. Alternatively, you can build a report on an existing view or you can create a temporary table or view based on multiple tables using query language statements in the `.setup` section. You can then drop the table or view, or update a status, in the `.cleanup` section of the same report.

For a discussion and example of joining tables for a report, see the appendix "Report-Writer Report Examples."

## Sorts and Breaks in Reports

To produce a report that is orderly and easy to read, you must sort the retrieved data based on one or more of the columns. The data *must* be sorted if you want to include subtotals or other summary information in your report. In addition, you must specify the break columns in which a change of value signals Report-Writer to look for subtotaling or other special statements. For example, the Population Example in the appendix, "Report-Writer Report Examples" is a 1970 U.S. population report by region and state. To generate the regional population subtotals, the states must first be grouped by the value of the region column in the database, and breaks must occur at each change of value in the region column.

The easiest way to group rows is to sort them on the column that is used as the grouping column (in this example, region). Often, a report is sorted on more than one column. In such cases, the rows are first grouped on the basis of the first sort column (called the *major* sort column) and, within those groups, on the basis of the next sort column (called a *minor* sort column), and so forth. The sort order is specified by naming the columns in the `.sort` statement in a section containing report set-up statements. If you have a `.query` statement with an order by clause, you cannot use the `.sort` statement.

By default, Report-Writer assumes the break columns are the same as the sort columns. In the above example, for instance, no other breaks need be specified. However, you can override the default breaks by specifying break columns with the `.break` statement. Use the `.break` statement to specify your break columns if you have a `.query` statement with an order by clause.

The currently active list of break columns (specified by either the `.sort` or `.break` statement) is known as the *break list*. The first column in the break list indicates a *major* break column, while those that follow are considered *minor* break columns. A break on one break column automatically produces a break on all subsequent break columns in the currently active break list.

In the Account Example of the appendix "Report-Writer Report Examples," break columns are not explicitly specified, so breaks can occur on the sort columns. Report-Writer sorts the data based on `acctnum` (the major sort column) and, within `acctnum`, based on `date`. When a change occurs in the value of `date`, the date break occurs and Report-Writer looks for some of your formatting instructions to process. When a change in value occurs in the `acctnum` column, breaks in both `acctnum` and `date` occur.

You need not specify actions for every break in your report. You can specify sort columns (which produce breaks) simply for the appearance of the report. In the Population Example, in the appendix, "Report-Writer Report Examples," breaks in `region` invoke a number of summary and heading actions, whereas breaks in `state` do not.

Under certain conditions, such as with numeric columns of rounded values, breaks occur by default when the formatted value changes, *not* when the actual value changes. For example, assume a column is rounded to the first decimal place. There can be no break between the actual values of 35.87 and 35.92, because each rounds to 35.9.

You have control over how numeric values are rounded through the format specification. For more information, see Format Specifications in the chapter "Report-Writer Expressions and Formats." To force breaks to occur on the actual values rather than on the formatted values, specify the `-t` flag on the report command line. For more information on the `-t` flag, see the *Command Reference Guide*.

## Pagination in Reports

Pagination in the report is controlled by a number of statements. The `.pagelength` statement specifies the vertical size of pages, in lines, while the `.pagewidth` specifies the horizontal size of pages, in characters. Statements are used in the page header and footer sections to define actions to be taken at the beginning and end of each page. Use the `.newpage` and `.need` statements to force page breaks, and the `.formfeeds` statement to instruct Report-Writer to send a formfeed character to the printer after printing all lines that fit on the defined page. Line numbering begins at 1 (top line).

Before Report-Writer begins to print a report, it calculates the number of lines in the page header and footer you have specified. After printing each line, it compares the specified or default page length with the number of lines already printed. If there are only enough blank lines left to write the page footer, Report-Writer prints the page footer, issues a formfeed character (if specified) for a page break, updates the page number, and prints the page header for the next page.

If the `.formfeeds` statement is in effect, Report-Writer inserts the formfeed character at the start of the report and at the end of each page. In some cases, the `.formfeeds` statement is not needed. For instance, the `.print` statement automatically inserts formfeeds appropriate for 11-inch paper if you use the default page length (61 lines).

The following command, issued at the operating system prompt, sends the output of "myreport" to the specified file and to the default printer, assuming the default value of 61 lines per page. It does not require the `.formfeeds` statement.

```
report mydb myreport -frepfile.lis -o
```

For a format that uses 66 lines per page, you can add the flag `-v66` at the end of the report command line, or you can use the `.pagelength` statement in the report specification.

For a printer that is 80 characters wide, you can add the flag `-l80` or use the `.pagewidth` statement.

For special forms and other printers, use the `.formfeeds` statement to instruct Report-Writer to insert its own formfeeds, or the `.noformfeeds` statement to prevent Report-Writer from inserting them.

You can use the `.newpage` statement to force an immediate page break at any point in the report. This statement causes Report-Writer to skip enough lines to get to the first line of the page footer (if specified), and to print the page footer before going to the top of the next page.

The `.need` statement forces a page break to occur if the remaining available lines on the page are less than the number of lines specified in the `.need` statement. It is used to keep lines of text together on the same page. For instance, this statement can be used prior to a break header to insure that enough lines remain on the current page to print the entire break header.

For details on using variables in reports, see Page Layout and Control Statements in the chapter "Report-Writer Statements."

## Report Margins

Report-Writer can determine report margins by analyzing your report code (see page 282). In most cases, the default settings generated by Report-Writer are quite adequate. However, in some cases you must define these settings explicitly, using the `.leftmargin` and `.rightmargin` page layout statements to indicate the starting and ending character positions. Horizontal character positions start at 0 (left margin).

In some reports, the right and left margins are changed dynamically to achieve different effects. See Dictionary Example in the appendix "Report-Writer Report Examples." In cases such as these, you must keep the margins for the page header and footer independent of the margins for the rest of the report, because the report margins have been set to values inappropriate for the footer and header when a page break is encountered.

Because the margins for the page header and footer can be independent of the margins for the rest of the report, the page header margins are stored separately. These can be determined automatically in the same way that default margins for the report are determined, or you can specify the margin setting statements, `.leftmargin` and `.rightmargin`, in the formatting statements for the page header.

For detailed information on margin setting statements, see Page Layout and Control Statements in the chapter "Report-Writer Statements."

## Data Positioning, Formatting, and Printing

Report-Writer relies on three different groups of statements to print data in the correct place and format. These are:

- Column and block default setting statements
- Text positioning statements
- Print statements

You use these statements to:

- Set default print positions and widths for columns
- Position text explicitly, or left justify, right justify, or center column values within the margins defined by the column defaults
- Define the print format (for instance, character string or decimal) for the value to be printed
- Print an explicit value or print the next value in a column at the previously defined position, in the designated format

The following discussions describe the process of positioning, formatting, and printing data in more detail.

**Note:** Column defaults can be explicitly defined with the column and block or other statements noted previously. If defaults are left undefined, Report-Writer automatically determines the defaults from an analysis of your report code. See Automatic Determination of Default Settings. Explicitly set defaults override any automatically determined ones.



## Default Print Positions for Columns

Before you can print a value, indicate *where* it must be printed. As noted previously, Report-Writer automatically determines default column print positions from an analysis of the other report formatting statements. However, if you want to set up your own defaults, you can do so with the following column and block statements:

- `.position`
- `.width`

Using the `.position` statement, you can effectively set up margins for each column. This statement allows you to set the starting print position for a column and optionally, the width of the printed column in number of characters. You can also set the width of a column with the `.width` statement. All horizontal print positions start at 0 (left margin).

To print columns horizontally adjacent to each other, you must reference the column names within the same `.print` statement, separated by commas. If possible, Report-Writer can print the columns next to each other, at the positions specified in the `.position` statement(s) or at default print positions.

In some cases, however, the following block statements can be used to exercise more control over the printing of horizontally adjacent text:

- `.block` and `.endblock`
- `.top`
- `.bottom`
- `.within` and `.endwithin`

The `.block` and `.endblock` statements allow you to define a block of formatting and print statements to be treated as a unit. Then, you can use the `.top` or `.bottom` statements to reset the current line to the top or bottom of the defined block before processing the next statement. The `.within` and `.endwithin` statements temporarily set the report margins to the margins for a referenced column. This allows you to print text (such as the caption, Total) within the column margins without having to calculate the exact print position.

For details, see Column and Block Statements in the chapter "Report-Writer Statements."

## Text Positioning

In addition to the column and block default-setting statements, you generally use another group of statements, called text positioning statements, to tell Report-Writer how to position the text or data in relation to the default position. The text positioning statements are:

- .tab
- .newline
- .left
- .center
- .right
- .lineend
- .linestart

You can use the .tab statement with a column name to tab to the assigned print position for that column before issuing a .print statement. In addition to tabbing, text positioning statements allow you to center or justify text within the default column margins, or to position text at the beginning or end of a line or on another line.

You can also use the text positioning statements with explicit values (instead of column names) or variables for the tab setting, left and right justify positions, and so on. Explicitly set positions override column defaults.

For details, see Text Positioning Statements in the chapter "Report-Writer Statements."

## The Print Format

The appearance of the text or data in your report is controlled by the format specification. For instance, the `c` format indicates a character string format and the `e` format causes a value to be printed in scientific notation. You can also specify the format with a template such as `$zz,zzz.nn`, containing characters with special meanings, which define the way a value is to be printed.

The print format can be specified in the `.print` statement, or can be used in a `.format` statement to set a default print format for a column, as in the code fragment:

```
.format emp (c12), sal ('$zz,zzz,zzn.nn')
.print emp,sal
```

The results might look like this:

Jones	\$ 109,224.00
Smith	\$ 32,575.00

You can temporarily override a default column format with the `.tformat` statement to print the *next value only* in a different format. After the value is printed, the format returns to the original default type. This is useful for printing a dollar sign only once at the start of a page, for instance.

You can also override a default format by specifying the format as a parameter in the `.print` statement, as in this example:

```
.print salary ('$zz,zzz,zzn.nn')
```

This code fragment causes Report-Writer to print salary values in the specified format, without the dollar sign, until it encounters another format or print statement for this column. For more information on print formats, see Format Specifications in the chapter "Report-Writer Expressions and Formats."

To indicate underlining of text or values, use the `.underline` and `.nounderline` statements. Any `.print` statements located between the `.underline` and `.nounderline` statements can produce underlined text. By default, the underline character is the underscore (`_`) for reports written to a file (reports written to the screen do not display underlining). You can change the default to any other character, using the `.ulcharacter` statement. All underline characters are printed on the line below the text, except for the underscore (`_`) character, which appears on the same line as the text. For more information on underlining, see Print Statements in the chapter "Report-Writer Statements"

## How the Printed Value Is Determined

The actual text or value to be printed is specified as an expression in the `.print` statement syntax. The expression can be a column name, a constant, a function, an aggregate, a runtime report variable such as the current date and time, or a variable whose value is specified in a `.declare` or `.let` statement or on the command line, with or without a prompt.

By default, Report-Writer prints an empty string when it encounters a null value. You can change this default to any string of characters, using the `.nullstring` statement. For instance, you can tell Report-Writer to print the string "none" wherever it finds a null value in the data.

For more information on the `.print` and `.nullstring` statements, see [Print Statements](#) in the chapter "Report-Writer Statements."

## Use of Conditional and Assignment Statements

Use the conditional `.if`, `.then`, and `.else` statements to tell Report-Writer to execute alternative blocks of statements, under specified conditions. For example, you can test for the current line number or character position using one of the special report variables (discussed in the chapter "Report-Writer Expressions and Formats"), and then issue a `.newpage` or `.newline` statement. Or, you can execute alternative `.print` statements to suppress or print confidential data, based on a user's ID number stored in a declared variable.

The condition in an `.if` statement is a boolean expression that evaluates to true or false. Each of the following is a condition:

- a clause
- a boolean function
- not condition
- condition or condition
- condition and condition
- (condition)

Some examples of conditions in `.if` statements are:

```
age <= 50
not (age <= 50)
(age <= 50) and (salary >= 40000) and
(job = 'programming')
age > avage
```

You can use the `.let` statement to assign a value to a declared variable. For instance, you could calculate the number of years which have elapsed since an employee was hired, and assign the result to a variable for a report on employee longevity. The `.let` statement is often used in conjunction with the `.if`, `.then`, and `.else` statements.

Note that the `.declare` statement, used to define variables, can also be used to assign an initial (first-time) value to the variable during the loading of the report specification. See the `.Declare` statement in the chapter "Report-Writer Statements."

For more information, see Conditional and Assignment Statements in the chapter "Report-Writer Statements."

## Summary Data Calculation and Printing

You can use aggregate functions such as sum or count, as well as arithmetic and other built-in functions, to calculate subtotals and other summary values to be printed in a report. Many Report-Writer statements accept expressions. For example, you can specify an aggregate, arithmetic operation, or function in the .print statement for immediate printing of the calculated value. Alternatively, you can use an expression containing the operation in a .let statement to assign the calculated value to a variable prior to printing, in case you want to use the result in additional calculations.

## Automatic Determination of Default Settings

If you have not explicitly specified them, Report-Writer calculates default settings for the right and left margins of the report, for the starting position and width of each column (for use with statements such as .tab and .right), and/or for the formats to use when printing columns. Report-Writer determines the default settings on the basis of an analysis of the other report formatting statements, which are performed after the report set up and page layout statements (such as .leftmargin), but before the first printing of the report.

## Analysis of Report Formatting Statements

To determine default values, Report-Writer analyzes the formatting statements in reverse hierarchical order, from the *innermost* (detail level) statements to the *outermost* (report level) statements, as shown below:

1. .detail section statements
2. .footer statements for innermost sort column
3. .header section for innermost sort column
4. .footer and .header sections for the next to the last sort column, and so on
5. .footer and .header text for the page
6. .footer and .header text for the report

By analyzing the report code in this way, Report-Writer attempts to determine the innermost references to columns in the report, and to determine the leftmost and rightmost print positions indicated by the specified report formatting statements.

### How Default Page Width Is Determined

For default reports—that is, a report based on a table, as in the command, `report mydb tblname`—the page width default is 80 characters if the report is displayed on the screen, or 132 characters if the report is written to a file or sent to the printer. You can override the default with the `.pagewidth` statement.

### How Default Margins Are Determined

If you explicitly specify the margins for the report with the `.leftmargin` and `.rightmargin` statements, these values are used. Otherwise, Report-Writer determines the minimum and maximum print positions for a line in the report while scanning the report formatting statements. If only one of the margins is explicitly stated, Report-Writer determines the other one during the scan. Report-Writer uses margins derived in this manner to determine line positions for the `.center`, `.right`, or `.left` statements, when these statements are used without specified parameters.

### How Default Column Positions Are Determined

If you do not specify a `.position` statement for a column, Report-Writer determines the column's default position for use with the `.tab`, `.right`, `.left`, or `.center` statements from the analysis of report formatting statements. Report-Writer determines default column positions by the first print position it encounters that has been specified for the printing of a value in that column or for an aggregate of that column.

Reports are usually set up in such a way that the innermost printing of column values occurs in the `.detail` statements of the report. Items such as column headers and aggregates, which print in the header or footer text for a break, can then use the `.tab` or another positioning statement in relation to the default position for the innermost position of a column. If you want to change the position of a column and its associated heading and/or aggregates, you must change the innermost print position for the column. Because all references to headers and aggregates are given in relative terms, Report-Writer changes their positions automatically.

As an example, see Account Example in the appendix "Report-Writer Report Examples." The default position for the `amt` column is determined by the cumulative aggregate for `amt`.

## How Default Column Formats Are Determined

If you do not specify a `.format` statement for a column, Report-Writer determines the column's default format in a manner similar to that used for determining the default column position. Report-Writer uses the innermost reference to a format for a column, or to an aggregate for a column, as the default format for a column. If you do not specify any formats for a column, Report-Writer determines defaults from the data type of the column, as described in Default Formats in the chapter "Report-Writer Expressions and Formats."

The default format for a column is best used in situations where you specify the format in the reference to a column in the `.detail` formatting statements. Aggregates of that column are specified in the footers for some of the breaks. Report-Writer then correctly uses the format you specified in the `.detail` section for the aggregates.

However, the `.format` statement is often quite useful for specifying a series of columns that are given the same format. For a good illustration of the use of the `.format` statement for this purpose, see Population Example in the appendix "Report-Writer Report Examples."

## How Default Column Widths Are Determined

If you do not specify a `.width` statement or width parameter to the `.position` statement for a given column, Report-Writer determines the default column width by the default format for that column, as specified by the `.format` statement or as determined from the analysis of report formatting statements. The default width of a column is the width required by the column format to print a value. Report-Writer uses the column width to determine the print positions for the `.right` or `.center` statements.



# Chapter 11: Report-Writer Expressions and Formats

---

This section contains the following topics:

[Expressions in Report-Writer](#) (see page 285)

[Object Naming Conventions for ANSI/ISO Entry SQL-92 Compliant Databases](#) (see page 287)

[Recognition of Delimited Identifiers](#) (see page 288)

[Reserved Words](#) (see page 292)

[Types of Data in Expressions](#) (see page 293)

[Operations](#) (see page 309)

[Format Specifications](#) (see page 316)

[Expressions and Formats Syntax Summary](#) (see page 342)

## Expressions in Report-Writer

In Report-Writer, expressions are used to obtain data values in your report. Expressions are composed of constants, variables, columns, aggregates, and functions that are combined with operators to produce a single value. Each of these types of expressions is described in this chapter.

Many Report-Writer statements accept expressions that provide users control and flexibility in determining report values. In general, Report-Writer statements can be divided into the following two categories:

- Statements that accept only variables and are evaluated once during the loading of the report specification before retrieving the report data.
- Statements that accept any expression appropriate in the context of the statement and are evaluated each time the statement is executed.

For ways to use expressions in a given statement, see each individual statement in the chapter "Report-Writer Statements."

The .print and .query statements are two statements in which you often use variables and expressions. One way you use variables in the .query statement is in the where clause to retrieve a subset of the data. In the .print statement, you can use expressions to print multi-column functions, such as adding columns together, or to determine the printing format.

You can compare expressions to other expressions in conditions for the `.if` statement, or use them in the `.let` statement to specify a value to be assigned to a variable. For details on conditional and assignment statements, see the chapter "Report-Writer Statements."

The following example uses several expressions. The example uses a database, which has a table of shipments featuring part number, number of defective parts in a shipment, and the total number of parts in a particular shipment. Suppose you want a report of the shipments grouped by part number, with the calculated percentage of defective parts for all the shipments of that part. The following report fragment would accomplish this:

```
.sort partno
.
.
.
.footer partno
.print partno, ' IS '
.print (sum (defective)/sum (total)) * 100,
' % DEFECTIVE '
.newline
```

In the previous example, the following are expressions:

```
partno
' IS '
(sum(defective)/sum(total)) * 100
'% DEFECTIVE'
```

Because no print formats have been explicitly specified in the preceding report code, Report-Writer automatically uses predetermined ones.

## Object Naming Conventions for ANSI/ISO Entry SQL-92 Compliant Databases

For reports based on tables or views in ANSI/ISO Entry SQL-92 compliant databases, observe the following conventions regarding object names when referencing them within Report-Writer statements and commands:

- Names consist of 18 or fewer characters.
- Regular identifiers begin with an alphabetic character (a - z) and contain only alphabetic, numeric, or underscore (\_) characters.
- Regular identifiers are case insensitive.
- Delimited identifiers are case sensitive.

These conventions differ somewhat from standard Ingres conventions for regular and delimited identifiers. Also, in standard Ingres databases, regular identifiers can include the non-alphanumeric characters #, @, and \$, and delimited identifiers can be case insensitive, depending on how the database was created.

## Recognition of Delimited Identifiers

Using delimited identifiers allows you to reference database objects such as tables and columns that are identical to reserved words or that contain spaces or other characters disallowed in regular identifiers. If the database was created as case sensitive, you can also use delimited identifiers to distinguish between identical names with different cases (for example, "SALES" as opposed to "Sales").

Report-Writer statements accept delimited identifiers as table names, view names, synonyms, column names, correlation names, and schemas under the following conditions:

- Recognition of delimited identifiers has been turned on with the `.delimid` statement.
- Statement syntax permits a table, view, column, or schema.
- SQL is used in `.query` statements.

Report-Writer also accepts delimited identifiers as user identifiers in system level commands.

In order for Report-Writer to recognize delimited identifiers within a report specification, the `.delimid` statement must be included in the report specification file. For details, see the chapter "Report-Writer Statements." Otherwise, any use of delimited identifiers within the report specification results in errors when compiling the report.

If you attempt to run a report with delimited identifiers against a database created with an earlier release, Report-Writer interprets the delimited identifiers as string constants, which can result in unpredictable behavior.

Ingres supports delimited identifiers in `.query` statements if the query language is SQL. Delimited identifiers cannot be used in QUEL queries.

### QUEL User Notes

Delimited identifiers are not available for use in QUEL queries. If you have specified the `.delimid` statement in a report specification that uses a QUEL query, Report-Writer suppresses recognition of delimited identifiers during the query. Although it is possible to use delimited identifiers in other parts of your report specification, we do not recommend that you do so if you are using a QUEL query, as this can cause errors in some circumstances. For example, if you create temporary tables in the `.setup` section that have delimited identifiers as table names or columns, those tables or columns cannot be accessed in a QUEL query.

## How to Specify Delimited Identifiers

You can specify delimited identifiers for database objects in the following circumstances:

- Within the report specification code
- On the command line within a command to run a report
- On the command line in response to a prompt for runtime substitution of a variable

To specify a delimited identifier in the report specification code, as a parameter on the command line, or in response to an Ingres prompt on the command line, enclose the delimited identifier within double quotes (") and escape any embedded double quotes by doubling them. For convenience, we refer to this as *editable format*. You can specify delimited identifiers as follows:

Editable Format	Stored Format
"Dave's table"	Dave's table
"Dave's ""Expert"" Witness table"	Dave's "Expert" Witness table

You can use delimited identifiers in the following circumstances:

- As a table, view, or synonym:
 

```
.data "Dave's table"
```
- As a column name:
 

```
.print "Stocks & Bonds"
```
- As a correlation name and column name in *correlation\_name.column\_name* constructs:
 

```
.query
  select "t-1"."col 1" as col1, "t-2"."col 2"
        as col2
  from table_one "t-1", table_two "t-2"
```
- When using schemas for owner qualification, for either or both the schema name and/or object name, as follows:
 

```
"schema 1".table2
dave."Dave's table"
"schema 1"."view table1 & table2"
```
- On the command line, as a *username* for the -u flag, *groupid* parameter for the -G flag, or database object in a command parameter

If your operating system requires additional delimiting and dereferencing quotes for delimited identifiers on the command line, see the *Command Reference Guide*.

When using schemas for owner qualification or when specifying a user ID on the command line, the schema or user ID must be enclosed in double quotes if it does not conform to the conventions for regular identifiers. If a schema or user ID contains characters unacceptable in a regular identifier, such as in Da Vinci or O'Neil, then all objects created by that user can have a schema name that must be specified as a delimited identifier in double quotes.

In addition, the DBA can specify at database creation time whether Ingres stores user IDs in uppercase, lowercase, or mixed case when impersonating that user with the `-u` flag on the command line.

Also, any schema name that Ingres creates by default based on that user ID is stored in mixed case and must be specified as a delimited identifier, in double quotes, when qualifying an object with an owner name.

For more information on schema names or specifying case conventions for user identifiers in a particular database, see the *Database Administrator Guide*. For information about specifying embedded quotes and other special characters in delimited identifiers, see the *SQL Reference Guide*.

### Case Sensitivity of Delimited Identifiers

When specifying delimited identifiers, follow the rules for case as defined for your database. In standard Ingres databases, delimited identifiers can be either case sensitive or case insensitive, as determined by the DBA when creating the database. By default, standard Ingres databases are case insensitive.

In databases compliant with ANSI/ISO Entry SQL-92 standards, delimited identifiers are case sensitive.

## Multiple Delimited Identifiers

Separate multiple delimited identifiers on the same line with at least one space, because Ingres perceives two adjacent double quotes (") as an escaped double quote ("). For example, suppose you specify:

```
.print "abc""def"
```

it is interpreted as:

```
.print abc"def"
```

Separating delimited identifiers with white space prevents confusion. For example, include a space between the delimited identifiers "abc" and "def":

```
.print "abc" "def"
```

It is correctly interpreted as:

```
.print abc def
```

## Precedence over String Constants

When enabled, delimited identifiers take precedence over double-quoted string constants. For example, the following statement causes Ingres to attempt to print the value of a *column* named "a b c" rather than the *string* "a b c".

```
.print "a b c"
```

If it does not find a column matching that description, Ingres issues an error.

To avoid any potential confusion between delimited identifiers and string constants or format templates, we strongly recommend that you use single quotes (') to delimit quoted strings and format templates within Report-Writer. The only exception to this rule is in .query statements if the query language is QUEL, in which case double quotes for string constants and format templates are required. This does not pose a problem because delimited identifiers are not allowed within QUEL queries.

## Reserved Words

The identifiers in the following table are reserved for use as keywords by Report-Writer. If delimited identifiers have been turned on, you can use reserved words as delimited identifiers for table, view, column, schema, or user names by enclosing them within double quotes ("). Using reserved words in QUEL is not allowed. Follow all rules for using delimited identifiers, as described in the section, Delimited Identifiers. Using reserved words in any other way is likely to produce incorrect results when Report-Writer prints the report.

abs	current_time	locate	page_width
and	date	log	position_
ascii	decimal	lowercase	numberreport
atan	detail	max	right
average	dow	maximum	right_margin
averageu	exp	maximumu	run
avg	float	maxu	shift
avgu	float4	min	sin
break	float8	minimum	smallint
cnt	int4	minimumu	sqrt
cntu	integer	minu	squeeze
concat	integer1	mod	sum
cos	integer4	not	sumu
count	interval	null	table
countu	left	or	trim
cum	left_margin	page	uppercase
cumulative	length	page_length	w_column
current_date	line_number	page_number	w_name
current_day			

If you use one of the reserved words in the preceding table as a column name without delimiting it, Report-Writer issues a warning message. It also supersedes the definition of the built-in function with the column name you specify. All further references to the reserved word relate to the column, not to the Report-Writer function, which could produce unexpected results.

All SQL keywords are also reserved words in Report-Writer. For a complete list of these keywords, see the *SQL Reference Guide*.



## Types of Data in Expressions

Expressions can contain any of the following data elements:

- String Constants
- Numeric Constants
- Date Constants
- Columns
- Variables
- Special Report Variables
- Aggregates
- Operators
- Boolean Functions
- Format Specifications

### String Constants

Many reports have lines of text that appear in the body of the report. You can specify these string constants by enclosing them in single quotation marks (').

The syntax for specifying any character string in most Report-Writer statements is:

*'string'*

Within a .query statement, however, use the quotes appropriate to your query language. As a convention, this document uses single quotes ('), as required in SQL queries, to delimit string constants.

Using double quotes for string constants outside of SQL queries does not necessarily generate an error in Report-Writer. However, if you activate delimited identifiers with the .delimid statement, using double quotes for string constants produces unexpected or incorrect results, because Ingres interprets the double quotes as signifying a delimited identifier.

If you want to include a single quotation mark within the text of a single-quoted string constant, enter it as two adjacent single quotes, together on a single line, so that Report-Writer does not assume it has found the end of a string. Ingres automatically interprets a backslash (\) within a single-quoted string as a literal backslash, unless it precedes a wild card character (see page 311).

For information on dereferencing double quotes within double-quoted strings, see the QUEL User Notes section below.

Examples of valid strings delimited by single quotes are:

```
'This is a string'
```

```
'This string has extra   blanks   '
```

```
'This string has one \ backslash in it'
```

```
'This string has a ''single-quoted'' string in it'
```

```
'This string has a "double-quoted" string in it'
```

### **QUEL User Notes**

Within a .query statement, the following must be enclosed in double quotes ("):

- String constants
- Dates, specified as string constants
- Numeric templates

You need to dereference a literal double quotation mark (") or backslash (\) within a string constant by preceding it with a backslash (\).

Examples of valid strings delimited by double quotes are:

```
"This is a string"
```

```
"This has extra   blanks"
```

```
"This has a \"quoted\" string in it"
```

```
"This has one \\ backslash in it"
```

The syntax for a valid absolute date and time format for use in a QUEL query is:

```
"mm/dd/yy hh:mm:ss"
```

An example of a valid numeric template for use in a QUEL query is:

```
"$zz,zzz,zzn.nn"
```

## Hexadecimal Strings

To specify a nonprintable character, you can use a hexadecimal string constant with the following format:

`X|x'nn{nn}'`

The introductory X identifies the string as a hexadecimal string constant. You need to specify the nonprintable character as two hexadecimal digits (*nn*) in the range 0-9, a-f, or A-F, and the string must contain an even number of characters. There must be no intervening white space between the X and the single-quoted string of hexadecimal digits. The X and the hexadecimal digits are case insensitive. Report-Writer interprets hexadecimal constants as data type varchar.

You can use hexadecimal string constants anywhere you use a string constant. Ingres translates the hexadecimal constant into its corresponding character value. The following example uses hexadecimal string constants in a .query statement:

```
.query
  select binary_key, X'414243' as abc
  from   my_table
  where  binary_key = X'5A7B0034'
```

You can also use hexadecimal string constants as expressions in .print statements. Hexadecimal strings are intended for use primarily with the q0 format, which sends the string directly to the output device as is, without interpretation. Hexadecimal strings in .print statements *without* the q0 format are interpreted by the formatting routines and can produce undesirable results or cause Report-Writer to fail. For more information on the q0 format, see Control Character Format Q0 (see page 339).

### QUEL User Notes

Hexadecimal string constants are not supported within QUEL .query statements.

## Numeric Constants

Numeric constants consist of an integer, a decimal point, and a fraction or exponential (scientific) notation. You can specify numeric constants with the following format, where *d* is a digit:

`[+|-] {d} [. {d} [e|E[+|-] d[d]]]`

Some examples of valid numeric constants are:

23  
8.97327  
4.7 e-2

Numeric constants can range from -10\*\*38 to +10\*\*38 (\*\* denoting "to the power of") with precision to 17 decimal places.

Report-Writer interprets a numeric constant as follows:

- Integer, if it has no decimal point or exponent. A numeric constant of type integer is treated as type decimal if the value exceeds i4 (4,294,967,295)
- Decimal, if it has a decimal point but no exponent. Under the following circumstances, a numeric constant of type decimal is treated as type float:
  - If the total number of digits exceeds 31
  - If you have set the II\_NUMERIC\_LITERAL environment variable/logical to float
- Float, if it has an exponent, or if the release is Ingres 6.4 or earlier and it has a decimal point.

## Date Constants

Dates are referenced as single-quoted character string constants. However, as with string constants within a .query statement, the quotation marks appropriate to your query language must be used.

Report-Writer accepts the following variations as date expression. You can specify a date template for the date expression in the .print, .format, or .tformat statement. For details, see Format Specifications (see page 316).

- Absolute dates - Examples of expressions of the date November 15, 1998 are:

```
'11/15/98'
'15-nov-98'
'15-nov-1998'
'11-15-98'
'98.11.15'
'111598'
'11/15'
'11-15'
```

The string 'today' is a legal absolute date, which returns today's date as its value. The string 'now' is a legal absolute date and time, which returns today's date and the current time as its value.

- Absolute times - Examples of expressions of the time 10:30:00 are:

```
'10:30:00'
'10:30:00 pst'
'10:30'
```

**Note:** Report-Writer supplies the appropriate time zone designation. Time formats are assumed to be on a 24-hour clock. However, time entered with a designation of "am" or "pm" is automatically converted to 24-hour internal representation. Any such designation must follow the absolute time and precede the time zone, if included. If you do not specify a date with an absolute time, today's date (that is, the current day) is supplied.

- Absolute date and time - Examples of expressions of the date and time, November 15, 1998, 10:30:00 are:

```
'11/15/98 10:30:00'
'15-nov-98 10:30:00'
'11/15/98 10:30:00 pst'
'15-nov-98 10:30:00 pst'
'11/15/98 10:30'
'15-nov-98 10:30'
'11/15/98 10:30 pst'
'15-nov-98 10:30 pst'
```

- Date intervals - Examples of date interval expressions are:

```
'5 years'
```

```
'8 months'  
'14 days'  
'5 yrs 8 mos 14 days'  
'5 years 8 months'  
'5 years 14 days'  
'8 months 14 days'
```

- Time intervals - Examples of time interval expressions are:

```
'23 hours'  
'38 minutes'  
'53 seconds'  
'23 hrs 38 mins 53 secs'  
'23 hrs 53 seconds'  
'28 hrs 38 mins'  
'38 mins 53 secs'  
'23:38 hours'  
'23:38:53 hours'
```

## Columns

To reference a column value in a data row currently being processed, you can:

- Specify the database column name directly
- Reference the column by a name you give it in the as clause of a select statement within the .query statement

You can use the select as construct in a .query statement to select a column by its correlation name, which is not recognized by Report-Writer, and then give it another name, which you can use in Report-Writer statements. You can also use this construct to select a column whose name is a delimited identifier, and give it another name that is easier to reference in other Report-Writer statements.

To reference a column from a database table by a name other than its actual database column name, use the following construct in a .query statement:

```
select columnname as resultcolumn from tablename
```

Thereafter, refer to the column by its result name, as in the following print statement:

```
.print resultcolumn
```

After assigning a result name in the query, any references to *columnname* causes an error.

Use a delimited identifier (see page 51) to reference a column that contains spaces or other non-alphanumeric characters, or that is identical to a reserved word.

Columns are comprised of one of the following types of expressions:

- Numeric
- Character
- Abstract

This table shows the SQL data types that belong to each of these categories. See the QUEL User Notes for QUEL data types.

<b>Numeric</b>	<b>Character</b>	<b>Abstract</b>
decimal	char	date
float	c	money
float4	text	
float8	varchar	
integer1		
integer2 (smallint)		
integer4		

Report-Writer perceives and treats a user-defined (UDT) data type as a character string. It does not recognize columns of data types long varchar, byte, byte varying, and long byte. If you specify a column of this data type in the query, Report-Writer silently ignores and does not print values for that column.

**Note:** If Report-Writer encounters subsequent references to a column of the long varchar, byte, byte varying, and long byte data types—for example, in sort operations—it issues an error message and terminates the report.

### QUEL User Notes

The QUEL data types that belong to these categories are listed in the following table:

Numeric	Character	Abstract
f4	char	date
f8	c	money
i1	text	
i2	varchar	
i4		



## Variables

Variables are user-defined symbol names that represent a data value that can change with each run of the report. Many Report-Writer statements accept variables as parameters. For example, in a `.query` statement you can use variables as substitutes for any part of a query. See `.Query` in the chapter "Report-Writer Statements."

In Report-Writer, assign variable values in the following ways:

- At report runtime, as a parameter on the command line or in response to a default or custom prompt
- In a `.declare` statement, as an initial value for a variable (useful for variables evaluated during the loading of a report specification)
- In a `.let` statement, where you can assign a value to a variable for use within the body of the report. See `.Let` in the chapter "Report-Writer Statements."

You define variables through the `.declare` statement, which allows you to name a variable. The `.declare` statement also allows you to:

- Create your own custom prompt
- Specify the data type and nullability of the variable

You are *strongly* encouraged to define all variables with the `.declare` statement. By declaring variables, there are no limits to the ways you can use them in your report. Although Report-Writer recognizes any name preceded by a dollar sign (\$) as a variable, undeclared variables assume default types and characteristics that are often incompatible with their intended use.

If you use a variable in your report without declaring it, you can assign it a value only by using a command line parameter or by entering a value in response to a runtime prompt.

You *cannot* specify a value for an undeclared variable with a `.let` statement. In addition, unless the variable has been declared, attempting to pass a parameter with a null value to Report-Writer can produce incorrect results. For more information, see the `.Declare` statement in the chapter "Report-Writer Statements."

**Note:** If you run reports from the RBF catalog, they cannot be run in the background if they contain undeclared variables.

When you specify variables for runtime substitution of values in expressions, precede the variable name with a dollar sign (\$). The dollar sign cannot be embedded within the variable (for example, `var$name`). In addition, variable names must follow these rules:

- Can be up to 32 bytes long. Valid characters are letters, digits, and underscore (\_). For ANSI/ISO Entry SQL-92 compliant database restrictions, see Object Naming Conventions for ANSI/ISO Entry SQL-92 Compliant Databases (see page 287).
- Must begin with a letter
- Cannot match any of the reserved words listed in the Reserved Words section
- Any commas (,), parentheses ( ), or colons (:) required in the statement syntax must be explicitly stated and cannot be part of the variable.

Some examples of variables are:

```
$myvar  
$your_name  
$salary  
$start_date
```

A variable used in a query can be referenced in other parts of the report specification as well, but it must always be preceded by a dollar sign (\$). The variable must logically correspond to its intended value. For example, if the variable is used as a number, its value must be a legal Ingres number. If the variable is used as a date, its value must be a legal Ingres date. Otherwise, Ingres interprets the variable as a character string.

## Special Report Variables

You can use the following predefined report variables to generate and print such items as page numbers and the date and time a report is run, or to control the report layout.

**page\_number**

Current page number in the report—pages number from 1

**line\_number**

Current line number on the page—starts at 1

**position\_number**

Current column position on the page—starts at 0

**page\_length**

Current length of the page

**page\_width**

Current width of the report

**left\_margin**

Current left margin column position.

**right\_margin**

Current right margin column position

**current\_date**

Date when report is run. This does *not* include the time component. For full date and time, use *current\_time*.

**current\_day**

Day of the week when report is run, in the form of a three-character string (for example, Mon or Fri).

**current\_time**

Complete date and time of day when report is run

**w\_name**

Name of the column currently being used in a within block, in the form of a string.

**w\_column**

Value of the *w\_name* column in the data row currently being processed.

## Aggregates

You use an *aggregate function*, or *set function*, such as sum or count, to perform a calculation on data read in from one column, up to the occurrence of a break in another column. For instance, in the section, Population Example in the appendix, "Report-Writer Report Examples," the regional population subtotals represent use of the sum aggregate on each of the columns tot, tot\_18 to 65, tot\_under18, and tot\_over65 up to a break in region. Additionally, the population totals at the end of the report represent use of the sum aggregate for the same columns up to a break in "report."

You specify *which* data must be used in the calculation by naming the column containing that data as a parameter of the aggregate function. In the POPULATION example, the columns containing the relevant data are tot, tot\_18 to 65, tot\_under18, and tot\_over65. You indicate the cut-off point for the data to be included in each calculation by placing the aggregate function within the footer section for a particular column or section of the report. The aggregate value is calculated each time a break occurs in the specified footer.

Aggregates can be *non-unique* or *unique*, *simple*, or *cumulative*. A *non-unique* aggregate performs a calculation based on *every* value read in from the aggregate column up to a break in the specified footer. A *unique* aggregate performs a calculation on each *break value* in the aggregate column, up to a break in the specified footer. (Depending on how the data is sorted and where the aggregate is specified, the *break values* can or cannot be the actual *unique* values in a column.)

A *simple* aggregate produces a single value, calculated on all the values in the aggregate column up to a break in the specified footer. A *cumulative* aggregate calculates a running total for each value in the aggregate column up to the break containing the aggregate instruction. Simple and cumulative aggregates can be either non-unique or unique. Aggregate types are discussed in more detail later in this section.

The following aggregates are allowed:

### **avg**

Finds the average value of a numeric column up to a break in the specified footer. You can take the average value of a date data type column that has date intervals. Taking an average of absolute dates generates a DBMS error.

### **avgu**

Finds the average value of the unique or break values for a numeric column up to a break in the specified footer. You can specify avgu only for a break column. For additional details, see Unique Aggregates (see page 188) . You can take the average value of a date data type column that has date intervals. Taking an average of absolute dates generates a DBMS error.

**count**

Counts the number of rows up to a break in the specified footer.

**countu**

Counts the number of unique or break values up to a break in the specified footer. You can specify countu only for break columns. For more details, see Unique Aggregates (see page 188).

**min**

Finds the minimum value of a numeric or date column up to a break in the specified footer.

**max**

Finds the maximum value of a numeric or date column up to a break in the specified footer.

**sum**

Calculates the sum of a numeric column up to a break in the specified footer. In columns of data type date, you can use the sum aggregate only if the column contains time intervals. Taking a sum on absolute dates generates a DBMS error.

**sumu**

Calculates the sum of the unique or break values in a numeric column up to a break in the specified footer. You can specify sumu only for break columns. For additional details, see Unique Aggregates (see page 188). In columns of data type date, you can use the sumu aggregate only if the column contains time intervals. Taking a sum on absolute dates generates a DBMS error.

## Syntax of Aggregates

The basic syntax of an aggregate specification is:

```
[cumulative|cum [(breakname)]] aggname  
(columnname [, preset])
```

where:

### ***breakname***

Specifies the name of a break in the report (either a sort column name, or report or page ). It is optionally used as a parameter to the cumulative function to indicate when to reset the cumulative. The value of a cumulative represents the aggregate since the last break in *breakname*. The default value for *breakname* is report (that is, the value represents the cumulative value of an aggregate since the start of the report).

### ***aggname***

Specifies the name of the aggregation to be executed. Valid *aggnames* and synonyms are average (avg), decimal, count (cnt), minimum (min), maximum (max), and sum.

### ***columnname***

Specifies the column name in the data being reported. Values of this column are aggregated. Therefore, the column must be of the correct type (that is, numeric or date columns only for all aggregates except count). Note that a *columnname* must be specified for the count aggregate even though all columns result in the same value.

### ***preset***

Specifies either a constant value or the name of a column that is used for pre-setting the aggregate before calculations begin. This is used primarily with the cumulative function to set an aggregate to a non-zero value before starting.

For example, if to print an account balance next to each transaction in an account, use the cumulative sum aggregate with a *preset* to the starting balance of the account. For an example, see Account Example in the appendix "Report-Writer Report Examples." If *preset* is a constant, the aggregate is set to that value. It can be a numeric or date constant.

If *preset* is a valid numeric or date column name, the aggregate is set to the value in that column at the start of the break over which the aggregate is defined. In addition, *preset* is not allowed with the average aggregate.

## Simple Non-Unique Aggregates

The scope of a simple non-unique aggregate is determined by the context in which it is specified. For example, if you specify `sum (salary)` in the footer for the report, it refers to the sum of salary for all rows read in the report. If you specify `sum (salary)` in the page footer, it refers to the sum of salary for all rows that were processed during the printing of each page. If specified in the footer for a break in department, `sum (salary)` refers to the sum of salary for all rows in each department.

You can specify simple aggregates only in the `.footer` section for breaks, because these calculations are intended to provide summary information.

## Unique Aggregates

You specify a *unique* aggregate by following the aggregate name with the letter "u," as in `sumu`, `countu` or `avgu`, respectively. The difference between a unique and a non-unique aggregate is that a unique aggregate performs an operation only when the value in the aggregate column *changes*, while a non-unique aggregate performs the operation for *every* value in the aggregate column. Therefore, a unique aggregate performs its calculation only on the break values in the specified column, up to the break containing the aggregate instruction.

For example, if you specify the aggregate, `count(region)`, in the report footer for the sample report in the Population Example section of the appendix, "Report-Writer Report Examples," the result would be 51 (remember District of Columbia), because there are 51 rows in the report. However, if you specify `"countu(region)"` instead, the result would be 9, because nine breaks occur on region.

The number of breaks is not necessarily the same as the actual unique values in the column. This result depends on the break in which the aggregate instruction is placed, and on whether the data in the aggregate column has been sorted or not. For instance, `countu` would produce a result of 3 on the following unsorted data in Column 1, even though the data contains only two unique values, because three breaks would occur:

Column 1

AAA  
BBB  
AAA

## Cumulative Aggregates

Preceding an aggregate name with the keyword *cumulative* or *cum* indicates that the cumulative value of an aggregate is to be calculated and printed. As such, you can specify cumulatives in any context (for instance, in detail sections), because you use them to provide running totals. You can apply a cumulative to any of the other aggregates. Cumulatives are particularly useful for applications that need to use running totals, such as account balance applications.

If you do not specify a *breakname* after the cumulative keyword, or if you specify a *breakname* of report, Report-Writer assumes that the cumulative aggregate refers to all data rows processed since the start of the report. If a *breakname* of page is specified, the cumulative aggregate refers to all data rows processed since the last page break. If a *breakname* is specified which is one of the break columns, the cumulative aggregate refers to all data rows processed since the last break in that column.

You can specify the *preset* parameter to set the cumulative function to a constant value or to the value of a column when it is initialized (that is, at the start of the break in *breakname*). For example, in the Account Example section of the appendix, "Report-Writer Report Examples," the aggregate, `cum(acctnum) sum(amt,balance)`, in the detail block indicates a common use of the *preset* parameter. When a break occurs in *acctnum*, Report-Writer sets the cumulative function to the value of *balance*. As each new transaction is processed, Report-Writer adds the value of *amt* to the cumulative aggregate. Because deposits are positive and withdrawals are negative, the cumulative aggregate reflects the running balance.

## Rounded or Actual Values

By specifying the `+t` flag on the report command line, aggregates utilize the rounded values for any floating point column whose format has been specified in a `.format` statement with a template or as numeric F (for additional information, see Format Specifications (see page 316)). That is, the value of the aggregate for such a column is derived from the rounded values for the individual column rows. To force the aggregate to utilize the actual, rather than the rounded, values, specify the `-t` flag on the report command line. For more information, see the *Command Reference Guide*.



## Examples of Aggregates

Here are some examples of aggregates:

```
min(salary)
```

Specified in footer for dept, this element gives the minimum value of salary for all data rows in a dept.

```
average(age)
```

Specified in the footer for class, this element gives the average age for all data rows in a class.

```
count(name,200)
```

Specified in the footer for the report, this element gives the count of the number of data rows in the report + 200.

```
sum(transact,oldbal)
```

Specified in the footer for acct, this element gives the sum of transact, initialized by the value of oldbal at the start of each acct.

```
cumulative avg(height)
```

Specified in the detail text, this element gives the cumulative average of height since the start of the report.

```
cum(acctnum) sum(amt,balance)
```

Specified in the detail text, this element gives the cumulative sum of amt since the last change in acctnum and initialized by the value of balance at the last change of value in acctnum.

## Operations

The following operators can be used in expressions:

- Arithmetic
- Comparison
- Logical

## Arithmetic Operators

Numeric expressions can be combined arithmetically to produce other (compound) expressions. The following arithmetic operators are supported (in descending order of precedence):

Operator	Description
<code>+, -</code>	plus, minus (unary)
<code>**</code>	exponentiation
<code>*, /</code>	multiplication, division
<code>+, -</code>	addition, subtraction (binary)

Unary operators group from right to left, while binary operators group from left to right. You can force the order of precedence of operations using parentheses. For example, the following is an expression with no ambiguity as to precedence of operations.

```
(salary + 1000) * 12
```

Some arithmetic operations on date expressions are available:

Date Addition:

`interval + interval` —> `interval`

`interval + absolute` —> `absolute`

Date Subtraction:

`interval - interval` —> `interval`

`absolute - absolute` —> `interval`

`absolute - interval` —> `absolute`

Report-Writer does not support multiplication or division of date values. For example, suppose `birthdate` is an absolute date column in the data table. The following constructs give tomorrow's date and the age of the person with that `birthdate`, respectively:

```
current_date + date('1 days')
```

```
current_date - birthdate
```

## Comparison Operators

A comparison operator is a binary operator that takes two expressions as operands. Both expressions must be of the same type—numeric, string, or date. The following operators are recognized:

Operator	Description
=	equal to
!= or <>	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

All comparisons are of equal precedence. When comparisons involving character strings are made, all blanks are ignored.

## Conditional Expressions

A conditional expression has the form:

```
expr comp_op expr
```

The *expr* is an expression, and *comp\_op* is a comparison operator.

An expression can be enclosed in parentheses without affecting its interpretation, as in the following examples:

```
(age < 50)
((salary * 12) >= 20000)
```

A conditional expression evaluates to true or false. It can contain partial match specification characters.

## Pattern Matching with Wild Cards

You can indicate partial matches of character string data in a conditional clause in an .if statement and in the where clause of a query by using special *wild card characters* with the comparison operators. The character string data must be delimited by single quotes (except when used in a QUEL query).

## Wild Cards in an .If Clause

When used in a string within an .if condition, the special meaning of wild card characters can be disabled by preceding them with a backslash (\) character. Report-Writer then interprets the wild card character literally. Thus, \\* refers to the asterisk character. When used outside of an .if condition, wild card characters have no special meaning and are *always* interpreted literally.

You can use the following wild card characters within a conditional clause in an .if statement for the purpose of comparing character string data:

Character	Description
*	Matches any string of zero or more characters
?	Matches any single character
[..]	Matches any of the characters in the brackets

Any of these special characters can be used alone or in combination to specify partial match criteria:

Example	Description
ename = '*'	Matches all values in the "ename" column
ename = 'E*'	Matches any value beginning with "E"
ename = '*ein'	Matches any value ending with "ein"
ename = '*[aeiou]*'	Matches any value with at least one vowel
ename = 'Br???'	Matches any five-character value beginning with "Br"
ename = '[A-J]*'	Matches any value beginning with A, B, C, ..., J
ename = '[N-Z]???'	Matches any four-character value beginning with N, O, P, ..., Z

You cannot use blanks in bracketed expressions such as "[A-J]\*" or "[N-Z]???".

## Wild Cards in Queries

When a string appears within the where clause of a .query statement, the wild card conventions must follow those of the database query language you are using to retrieve the data.

## Logical Operators

Report-Writer recognizes the following Boolean logical operators:

Operator	Description
not	logical not - negation
and	logical and - conjunction
or	logical or - disjunction
is null	test to see if value is null
is not null	test to see if value is null

These operators take Boolean expressions as operands and evaluate to true or false. The not operator has the highest precedence of the operators; and/or have equal precedence. You can use parentheses for arbitrary grouping. Logical operators group from left to right.

## Built-in Functions

You denote a function by a function name, followed by an operand (or two operands) in parentheses. When valid expressions are substituted for the operands, the result itself is an expression that evaluates to a number, string, or date.

The resulting expression assumes the default print format appropriate for that type of expression (number, string, or date), as described in . For example, if you specify the following:

```
.print uppercase(column1)
```

The expression, `uppercase(column1)`, evaluates to a string. When printing the string, Report-Writer uses the default format for strings (`c0`), *not* the format for the specified column. If you want to print the resulting expression in a format other than the default for number, string, or date, specify the format explicitly in a `.print` or `.format` statement; for example:

```
.print uppercase(column1) (cf30.6)
```

A faster and more efficient way to unconditionally print the column value in uppercase for all retrieved rows would be:

```
select uppercase(column1) as column_a
.format column_a (cf30.6)
.print column_a
```

You can nest functions to any level.

All of the Ingres conversion, numeric, string, and date functions are syntactically allowed in the `.cleanup`, `.query`, and `.setup` statements. For all other Report-Writer statements the following table lists the supported functions. To determine if a function is appropriate for use in a particular context within Report-Writer, see its description in the *SQL Reference Guide*, or if you are using an Enterprise Access product, the *OpenSQL Reference Guide*.

The following table lists the built-in functions:

Conversion	Numeric	String	Date
<code>c(expr)</code>	<code>abs(n)</code>	<code>charextract(c1,n)</code>	<code>date_gmt(date)</code>
<code>char(expr)</code>	<code>atan(n)</code>	<code>concat(c1,c2)</code>	<code>date_part(unit,date)_</code>
<code>date(expr)</code>	<code>cos(n)</code>	<code>left(c1,len)</code>	<code>date_trunc(unit,date)_</code>
<code>dow(expr)</code>	<code>exp(n)</code>	<code>length(c1)</code>	<code>interval(unit,date_interval)_</code>
<code>float4(expr)</code>	<code>log(n)</code>	<code>locate(c1,c2)</code>	<code>_date(s)</code>
<code>float8(expr)</code>	<code>mod(n,b)</code>	<code>lowercase(c1)</code>	<code>_time(s)</code>
<code>hex(expr)</code>	<code>sin(n)</code>	<code>pad(c1)</code>	
<code>int1(expr)</code>	<code>sqrt(n)</code>	<code>right(c1,len)</code>	
<code>int2(expr)</code>		<code>shift(c1,nshift)</code>	
<code>int4(expr)</code>		<code>size(c1)</code>	
<code>money(expr)</code>		<code>squeeze(c1)</code>	
<code>object_key(expr)</code>		<code>trim(c1)</code>	
<code>table_key(expr)</code>		<code>uppercase(c1)</code>	
<code>text(expr)</code>			

Conversion	Numeric	String	Date
<code>varchar(<i>expr</i>)</code>			
<code>vchar(<i>expr</i>)</code>			

## Boolean Functions

A Boolean function is like a built-in function except that it returns a value of true or false instead of number, string, or date. The result of a Boolean function cannot be printed; it can only be used as a condition. A Boolean function is composed of a function name followed by an operand in parentheses.

The `break` function is the only Boolean function found in Report-Writer. For more information, see Breaks in the chapter "Using Report-Writer." Its syntax is:

```
break (columnname)
```

The *columnname* parameter must either be a break column (that is, it must be in the sort list) or the value report.

If you specify a break column, Report-Writer returns a value of *true* if the current value for that column has changed from the previous value *or* if the current value in any column of higher precedence than *column* has changed. If neither the current value for *column* nor the current value of any column of higher precedence in the sort list has changed, it returns a value of *false*. If report is specified, it returns *true* if the end of the report is reached; otherwise, it returns *false*.

The following example illustrates the use of Boolean functions:

```
.sort dept, empno
/* Other Report Writer statements */
.footer empno
  .if not break(dept) .then
    .newpage
  .endif
```

This generates a new page when the employee number breaks, but only if no break occurs on the department.

## Format Specifications

You can give special format specifications to expressions in the report in the .print statement or in a .format or .tformat statement. The format determines whether the data is printed as a character string, decimal value, date, or in some other format. Be sure to use the right type of format depending on the type of expression. If no format is specified, Report-Writer determines a default format from an analysis of your other statements.

The following formats are allowed:

- B format specifies that the value be blanked out. This is a special format used for blanking out a field, for use with temporary formats in conjunction with the .tformat statement.
- C format specifies character strings.
- Date templates are formats that allow you very detailed control over the appearance of dates and times in your reports.
- E format specifies numeric expressions printed in scientific notation.
- F format specifies numeric expressions.
- In the F format, you can control the placement of the decimal point or suppress it entirely.
- G format specifies numeric expressions. This format chooses either F or E format, depending on what fits in the field width. This format also guarantees that decimal points align, whether printed in F or E format.
- I format specifies numeric expressions printed in integer format.
- N format specifies numeric expressions like G format, but decimal points do not necessarily align.
- Numeric templates are complex formats for numeric data that allow you to control placement of dollar signs, commas, or other punctuation within the number.
- Q0 format specifies character strings like the C format, except that control characters can be part of the character string without affecting the layout of the report.
- T format specifies character strings like the C format, except that it displays certain unprintable characters in a visible format.

Each of these formats is discussed in more detail in its own section later in this chapter.



You can precede any of the preceding format specifications with a sign character to indicate that the value printed is to be either right justified, left justified, or centered. The following list describes each of the valid sign characters:

- A minus sign (-) indicates that the data is to be left-justified in the specified field width.
- An asterisk (\*) indicates that the data is to be centered in the width of the field.
- A plus sign (+) indicates that the data is to be right-justified in the specified field width.

The behavior of the sign characters is different for each data type. The discussions for each data format type later in this section contain examples of the effect of each sign character. If no sign is given, justification defaults to left justification for character fields and right justification for numeric fields.

## Default Formats

If you do not specify a format after an expression, Report-Writer uses a default format.

### Default Format for Strings

Report-Writer prints any string expression without a specified format in its entirety. That is, the default format for strings is `c0`.

### Default Format for Columns

If you do not specify a column format with the `.format` statement, Report-Writer uses the default format for the column. The default format is based on the data type of the column. For more information, see *Determining Default Column Formats* in the chapter "Using Report-Writer."

The following table lists default formats for SQL data types:

SQL Data Type	For Reports Other Than Block Style	For Block Style Reports
c1 - c35	c1 - c35	c1 - c35
c36 - cn	cj0.35	cj0.35
where <i>n</i> is the lesser of the maximum configured row size and 32,000		
char(1) - char(35)	c1 - c35	c1 - c35

SQL Data Type	For Reports Other Than Block Style	For Block Style Reports
char(36) - char( <i>n</i> ) where <i>n</i> is the lesser of the maximum configured row size and 32,000	cj0.35	cj0.35
text(1) - text(35)	c1 - c35	c1 - c35
text(36) - text( <i>n</i> ) where <i>n</i> is the lesser of the maximum configured row size and 32,000	cj0.35	cj0.35
varchar(1) - varchar(35)	c1 - c35	c1 - c35
varchar(36) - varchar( <i>n</i> ) where <i>n</i> is the lesser of the maximum configured row size and 32,000	cj0.35	cj0.35
integer1	f6	f10
smallint (integer2)	f6	f10
integer (integer4)	f13	f10
decimal (31.31 - 31.0)	Based on size. For example, decimal (5.1) defaults to f7.1, allowing additional digits for the decimal point and an optional +/- sign.	
float( <i>n</i> )	n10.3	f10.3
float4	n10.3	f10.3
float8	n10.3	f10.3
date	c25	c25
money	'\$-----.nn'	'\$-----.nn'

**Note:** All character data types are fully supported in non-Ingres databases accessed by way of an Enterprise Access product, in which case the column size limit can be greater than the lesser of the maximum configured row size and 32,000.

### Default Format for Special Report Variables

The following non-string report variables have the corresponding default formats:

Report Variable	Default Format
page_number	f6

Report Variable	Default Format
line_number	f6
position_number	f6
left_margin	f6
right_margin	f6
page_length	f6
page_width	f6
current_date	d' 3-feb-1901'
current_time	d' 3-feb-1901 16:05:06'
w_column	The default format for the column currently being used in a .within block. See the default column formats in the previous table.

### Default Format for Aggregates

The default format for all the aggregates except count(u) is the format of the column being aggregated. For count(u), Report-Writer uses the default format Nw, where w is the width of the column being counted.

### Default Format for Numbers

Any other numeric expressions such as numeric constants, numeric functions, numeric parameters, and arithmetic operations have a default format based on its data type.

### Default Format for Dates

Any other date expressions, such as the date function, date parameters and date arithmetic operations, have a default format of c0, which appears in the report as if specified as "d' 3-feb-1901'" for an absolute date, "d' 3-feb-1901 16:05:06'" for an absolute date and time, or as the needed portion of the template "d' 1 yrs 2 mos 3 days 4 hrs 5 mins 6 secs'" for a time interval.

### QUEL User Notes on Default Formats

The default column formats used for reports based on QUEL data are identical to those listed in the Default Formats for Columns table, but with the following data types designated differently in QUEL:

In place of:	Use QUEL data type:
integer1	i1

In place of:	Use QUEL data type:
smallint	i2
integer	i4
float4	f4
float	f8

## Blanking Format B

The B format can be used with any type of data to print blanks in place of the value of the specified variable.

The syntax of the B format is:

`bw`

*where w specifies the desired field width.*

You can specify either an uppercase B or lowercase b (they are identical). Report-Writer ignores the value of the expression, and prints *w* spaces in the output instead.

You can use this format in conjunction with the .tformat statement (see page 406), which temporarily changes a column format, to suppress printing of unchanged values in break columns.

## Character String Format C

Use the C format to print string expressions.

The syntax of the C format specification is:

`[-|*|+] c[f|j]n[.w]`

where:

### **f**

Instructs Report-Writer to fold (wrap) text by breaking between words when printing strings that span multiple lines. Tab and carriage return characters produce tab and carriage return actions within the string. Must be *specified* in conjunction with either *w* and/or *n*.

### **j**

Instructs Report-Writer to fold (wrap) text by breaking between words, and to right justify text by padding the line with blanks between words, when printing strings that span multiple lines. Tab and carriage return characters produce tab and *carriage* return actions within the string. Must be specified in conjunction with either *w* and/or *n*.

### **n**

Specifies the maximum number of characters to print. If there are more than *n* characters in the string, it truncates the excess. If there are fewer, it pads the rest of the string with blanks until *n* characters have been printed. Use a value of 0 for *n* if the entire string is to be printed, *regardless* of its length.

### **w**

Specifies the number of characters to print on each line. If *n* is greater than *w*, then more than one line is written, in newspaper-style column format. By default, *w* is set to *n*.

When specifying the previous options, you can use either uppercase or lowercase letters interchangeably. The optional field width *n* can be specified to give an exact width. If *n* is specified and the character string is less than *n* characters wide, blanks are added to the output to assure *n* characters. If the string is longer than *n* characters, only the leftmost *n* characters are printed and the rest are ignored.

If you specify a value for *w* as well as *n*, you can print text in column format (that is, newspaper style). The *f* and *j* modifiers can be used to assure breaking of words at blanks, or right justification of text. If neither is specified, simple wrap-around of text occurs, regardless of where the line break might occur.

Report-Writer has a default maximum of 310 lines for printing a character string. If you exceed this maximum, some of the text can not appear in your report. You can override the default by specifying the *-wmxwrap* parameter on the command line.

If you use the C format statement, the following unprintable characters are printed as spaces:

- Horizontal tab
- Vertical tab
- Line feed
- Form feed
- Carriage return

Note that tabs and carriage returns cause tabs and carriage return actions if you are using multi-line formats, such as cf or cj.

To print a visual representation of unprintable characters, use the T format statement (see page 341).

## C Format Examples

These examples illustrate the use of the C format.

### Example 1:

Suppose your report contains a character column called name that you want to print, and a value for name is Jones, J. Then, suppose you issue the following six .print statements:

```
.print 'First :', name (c15), ':First' .nl
.print 'Second:', name (c4), ':Second' .nl
.print 'Third :', name (c0), ':Third' .nl
.print 'Fourth:', name (-c15), ':Fourth' .nl
.print 'Fifth :', name (+c15), ':Fifth' .nl
.print 'Sixth :', name (*c15), ':Sixth' .nl
```

The print statements produce, respectively, the following six lines of output:

```
First :Jones, J.      :First
Second:Jone:Second
Third :Jones, J.:Third
Fourth:Jones, J.      :Fourth
Fifth :      Jones, J.:Fifth
Sixth :   Jones, J.   :Sixth
```

### Example 2:

If your data includes the character string, "Now is the time for all good men to come to the aid of their party.", the following table shows the effect of three different format specifications used to print the output on multiple lines in three different ways:

c0.15	cf100.15	cj0.15
Now is the time	Now is the time	Now is the time
for all good m	for all good	for all good
en to come to t	men to come to	men to come to
he aid of their	the aid of	the aid of
party.	their party.	their party.

Because the second format specification, cf100.15, specifies an actual number of character positions to print, Report-Writer prints out two blank lines after the text, to pad out the 75 characters already printed to the full 100-character column width specified.

After Report-Writer prints a string in column format, it sets the position for the next text output to the top line of the column, at the end of that line (in the example, this would be after "time").

## Date Format D

You can output date expressions using a D format specification. The date format specification is a D or d, followed by a quoted string *template* indicating exactly how a specific date would be printed. Ingres requires that you delimit the template with single (') or double (") quotation marks; single quotes are recommended for consistency with requirements for quoting string constants.

The date is left justified by default, but by specifying the optional plus sign (+), you can right justify the date. The template must be surrounded with single quotes.

The syntax of a date template is:

```
[ - | * | + ] D | d ' template '
```

where:

**'*template*'**

Specifies a string of characters representing a sample absolute date and time



## Absolute Date and Time Templates

Specify the absolute date and time format by a string containing one of many possible representations of a sample date and time, such as "SUN Feb 3 04:05:06 p.m." or "FEB 03 16:05." The selection and arrangement of the sample date and time elements within the template indicate the way you want all dates and times to be displayed or printed.

The following date and time must be used as the basis for your template:

Sunday, 1901 February 3 at 4:05:06 p.m.

**Note:** This specific date and time was chosen as the sample for the template because Sunday is the first day of the week, and arguments 1, 2, 3, 4, 5, and 6 are the year, month, day, hour, minute, and second, respectively. This makes it easy to interpret the elements of the template correctly. For instance, in the template '2/3/01,' the 2 indicates the month (February), the 3 indicates the day (3), and 01 indicates the year (1901).

Your template can contain only the weekday of Sunday and the month name of February (or their accepted abbreviations), the day of the month as 3, the year 1901, the time 04:05:06 p.m. in various formats, and the time designation p or p.m. (including the periods for p.m.). Report-Writer prints any other word exactly as entered.

You can specify 24-hour (military) time by using 16 instead of 4 for the hour in your template. Do not use p or p.m. with 24-hour time.

You can use all or only some of the arguments in your template, and you can arrange the arguments in any order. You can use any combination of lowercase and uppercase letters, and you can abbreviate the day of the week as Sun or Su and the month as Feb or Fe. The resulting output assumes the same format as your template.

You can create ordinal numbers from numbers by suffixing them with the appropriate "st", "nd", "rd", or "th," as in:

d'3rd day of the 2nd month of 1901'

To print the day of the year, you specify the day and year, but leave out the month:

d'3/1901'

The following examples demonstrate the use of absolute date and time templates:

Format	Value	Output
d' 2/ 3/01'	25-oct-1998	10/25/98
d' 2/ 3/01'	5-jun-1909	6/ 5/09
d'03-02-01'	5-oct-1998 07:24:12	05-10-98
d'2/3/1'	25-oct-1998	10/25/98
d'2/3/1'	5-jun-1909	6/5/9
d'010203'	5-oct-1998	981005
d'1\ 2\ 3'	5-oct-1998	98 10 5
d'FEBRUARY, 1901'	1-sep-2134 09:13:02	SEPTEMBER 2134
d'FEBRUARY, 1901'	7-may-1962 13:08:42	MAY, 1962
d'Sunday'	5-oct-1998	Wednesday
d'SUN Feb 3 16:05 1901'	13-oct-1998 07:24:03	THU Oct 13 07:24 1998
d'FEB 03 4:05:06 p.m.'	12-dec-1998 22:13:03	DEC 12 10:13:03 p.m.
d'04:05:06 PM'	5-oct-1998 14:08:45	02:08:45 PM
d'04:05:06 PM'	5-oct-1998 07:29:12	07:29:12 AM
d'16:05 pst'	5-oct-1998 14:08:45	14:08 pst
d'3/01'	5-oct-1998	278/98
d'February 3rd'	29-jul-1954	July 29th
d'3rd day of 1901'	11-may-1999	131st day of 1999

If you use the special report variables, *current\_date*, *current\_day*, or *current\_time*, to print an absolute date, day, or date and time, be sure to use only that portion of the date template that matches the returned variable value. Specifying more of the template than you need causes Report-Writer to print zeros (0) or empty strings for the unneeded portions of the template. For example, suppose you specify the following:

```
.print current_date(d' FEB 03 1901 16:05:06')
```

Report-Writer returns the current date, but with a time of 00:00:00, because the *current\_date* report variable returns only the date and not the time.

Numbers requiring more than one digit replace preceding blanks or zeroes in the template, as needed. If there are no more available preceding blanks or zeroes, the number expands to the right. Ingres retains in the output any single blank following a letter, word, or number in the template; it does not replace such a blank with a succeeding number.

You can align columns of numbers by preceding them with an appropriate number of blanks or zeroes (as shown by the first three examples in the preceding examples of output).

Because full month and weekday names (as well as numbers without preceding blanks or zeros) are of differing lengths, date columns using either of these components in the format rarely line up.

Following February or Sunday with a vertical bar (|) specifies that for shorter month names or weekdays, an appropriate number of blanks are substituted for the vertical bar to line up the components. Similarly, if you place a vertical bar after a single digit number in your template, Report-Writer prints a blank before each single-digit number it encounters (unless the digit is already preceded by a blank or zero).

For example, the template Sunday,| February | 3,| 1901 produces dates like:

```
Friday,    January 15, 1998
Wednesday, May      4, 1998
Saturday,  November 20, 1998
```

Report-Writer prints any character preceded by a backslash exactly as it appears.

## Date and Time Interval Templates

Date and time interval templates show the amount of elapsed time between two dates or times, rather than an absolute date or time.

Specify a time interval with a quoted string containing one of many possible representations of a sample time interval, such as 1 year or 1 yr 3 day. The selection and arrangement of the time interval elements within the template indicate the way you want time intervals to be displayed or printed.

Use the following time interval as the basis for your template:

1 year 2 months 3 days 4 hours 5 minutes 6 seconds

You can use all or only some of these units in your template, and the units can be arranged in any order. Use the plural or singular form of any unit, as well as the singular or plural form of the abbreviations, yr and mo, hr, min, and sec.

To specify a time interval, use the format `d'template'`, where *template* contains one or more time interval keywords (for example, minutes) preceded by the appropriate digit from the representative time interval string, as in:

`d'5 minutes'`

This format displays results followed by the keyword; for example:

9 minutes

The following examples demonstrate the use of the time interval templates:

Format	Value	Output
D'1 year'	3 yrs 5 mos 16 days	3 years
d'2 MONTHS, 3 DAYS'	3 yrs 5 mos 1 days	41 MONTHS, 1 DAY
d'1 yr 3 day'	1 yrs 5 mos 16 days	1 yr 168 days
D'4 hours 6 seconds'	23 hrs 8 mins 53 secs	23 hours 533 seconds
d'04:05 \hours'	23 hrs 0 mins 53 secs	23:01 hours
d'3 days 4 hours'	23 hrs 8 mins 53 secs	0 days 23 hours
d' 1 yr 2 mos 3 days'	200 yrs 11 mos 28 days	200 yr 11 mos 28 days
d' 1 yr 2 mos 3 days'	5 yrs 1 mos 3 days	5 yr 1 mos 3 days

For the purpose of date interval calculation, Report-Writer assumes there are 30.4376875 days in a month and 365.2425 days in a year. The smallest unit is rounded up.

Numbers requiring more than one digit replace preceding blanks or zeroes in the template, as needed. If there are no more available preceding blanks or zeroes, the number expands to the right. Report-Writer retains in the output any single blank following a letter, word, or number in the template; it does not replace such a blank with a succeeding number. You can align columns of numbers by preceding them with an appropriate number of blanks or zeroes (as shown by the last two examples in the preceding examples of output).

For English-language releases, if the number preceding a completely spelled template word is 1, Report-Writer prints the template word in its singular form (for example, 1 year); otherwise, it prints the word in its plural form (for example, 2 years). Report-Writer makes this adjustment only if you spell out the template word completely. For example, specifying 2 month would produce appropriate singular and plural results, such as 1 month, 5 months, and 9 months. Specifying 2 mo would produce only singular results such as 1 mo, 5 mo, or 9 mo.

Report-Writer prints any character preceded by a backslash (\) exactly as it appears.

## Numeric Format E

The E format prints numeric expressions in scientific notation. Numbers output in E format take the form of

`[ - ] m . nnnnn E | e [ + | - ] ppp`

where *m* is the mantissa, *n* is the number of decimal digits, and *p* is an exponential digit.

For example, 10.456e+03 means 10.456 times 10 raised to the 3rd power. Numbers output in E format are right justified in the field, unless preceded by a .left statement or the minus sign (-) in the format designation.

The syntax of an E format specification is:

`[ - | * | + ] ew[ . d ]`

where:

***w***

Specifies the maximum field width.

***d***

Specifies the precision or the number of digits to print after the decimal point.

You can specify either an uppercase or lowercase e, which also determines the case of the "e" in the output. Specify the field width *w*, which refers to the maximum number of spaces in the field. Be sure to include spaces in the field width to account for the exponential part of the printout—that is, include five spaces for E+*ppp*. If the value can be printed in fewer than *w* spaces, Report-Writer right justifies it in the field. If this width is too small to fit the value to be printed, Report-Writer fills the entire field with asterisks (\*) instead.

If you specify *d*, Report-Writer prints a decimal point, followed by *d* digits to the right of the decimal point.

If you do not specify *d*, or if you specify a value of "0" for *d* (for example, E20.0), Report-Writer does not print the decimal point and rounds off any fractional digits (it *does* print the exponential part).

This table illustrates the E format specification. The carets (^) in the Output column are used here only to show where blanks occur in the output; they do not print in your report.

Format	Value	Output
e10.3	22.3	2.230e+001
E10.2	-.123	-1.23E-001
e10	123.789	^1238e-001
E4.2	22.34	****
+E10.2	22.34	^2.23E+001
-e10.2	22.34	2.23e+001^

## Numeric Format F

The F format causes Report-Writer to print numeric expressions in standard decimal notation, with or without a decimal point. Numbers in F format are right justified in the field, unless preceded by a .left statement or by the minus sign (-) in the format designation.

The syntax of an F format specification is:

`[ - | * | + ] f w [ . d ]`

where:

**w**

Specifies the maximum field width.

**d**

Specifies the precision or the number of digits to print after the decimal point.

The minus sign (-) and plus sign (+) characters, when used as prefixes for the format specification, specify how the entire text must appear in the field—as either right or left justified. They *do not* refer to the placement of the *sign* of the number, as is the case in some programming languages.

You can specify this format with either uppercase or lowercase letters. Specify the field width  $w$ , which refers to the maximum number of printing positions in the field. If the value can be printed in fewer than  $w$  spaces, Report-Writer right justifies it in the field. If the value cannot be printed in  $w$  or more spaces, Report-Writer fills the entire field with asterisks (\*).

If you specify  $d$ , Report-Writer prints a decimal point followed by  $d$  digits to the right of the decimal point. The number of digits to the left of the decimal point is a maximum of  $(w - (d + 1))$ , because you need to account for the fractional part in the field-width specification.

If you do not specify  $d$ , or if you specify a value of "0" for  $d$  (for example, F20.0), Report-Writer does not print a decimal point and rounds off any fractional digit.

The following table illustrates the F format specification. The carets (^) in the Output column are used here only to show where blanks occur in the output; they do not print in your report.

Format	Value	Output
f10.2	22.3	^^^^22.30
F10.2	-.123	^^^^-0.12
f10	123.789	^^^^^^124
f4.2	22.34	****
+f10.2	22.6	^^^^22.60
-f10.2	22.6	22.60^^^^



## Numeric Format G

The G format uses an F format specification if there is enough room in the field, or E format if there is not enough room.

The syntax of the G format specification is:

`[ - | * | + ] g $w$ [ .  $d$ ]`

where:

**$w$**

Specifies the maximum field width.

**$d$**

Specifies the precision or the number of digits to print after the decimal point.

You can specify either an uppercase G or lowercase g, which determines the case of the "e" if the value is printed in scientific notation. For a full description of the meanings of  $w$  and  $d$ , see the descriptions of Numeric Format E and Numeric Format F.

Report-Writer aligns data on the decimal point and then justifies the entire column right or left, according to the sign specified. By default, the column of numbers is right justified. To align F format numbers with E format numbers, Report-Writer right justifies F format numbers to a location several spaces in from the right edge of the field to account for the space taken up by the E format's exponential power designator—that is, five spaces for E+|-*ppp*.

The following table of examples illustrates the G format. The carets (^) in the Output column are used here only to show where blanks occur in the output; they do not print in your report.

Format	Value	Output
g11.2	123.456	123.46^^^^
G11.2	123456	^12.35E+004
g12.2	-134.65	-134.65^^^^
g9	-123	^^^^-123
+g11.2	123.45	^^^^^123.45
-g11.2	123.45	123.45^^^^

## Numeric Format I

The I format prints numeric expressions in integer format. The syntax of the I format specification is:

`[-|*|+] i $w$`

where:

**$w$**

Specifies the maximum field width.

Specifying i9 prints a number up to nine characters long in integer format, including an optional sign, and left justifies it in the column by default. Specifying a plus sign (+i9) right justifies the number in the column.

Format	Value	Output
i10	22,000	22000^^^^
I10	-120,300,000	-120300000
i4	123.789	124^
i4	22,800	****
+i8	22.4	^^^^^22
-i8	22.6	23^^^^^

Numeric Format N

The N format specification is identical to the G format specification except that the field is right justified, whether printed with E or F format. Of course, if you specify the optional minus sign (-) before the format designation, the value is left justified.

The syntax of the N format specification is:

`[-|*|+] nw[.d]`

where:

**w**

Specifies the maximum field width.

**d**

Specifies the precision or the number of digits to print after the decimal point.

You can specify either an uppercase N or lowercase n, which determines the case of the "e" if printed in scientific notation. For a full description of the meanings of w and d, see the descriptions of Numeric Format E and Numeric Format F.

Numbers printed with N format are right justified in the output field. Unlike G format, the decimal points are not always aligned.

The following table of examples illustrates the N format. The carets (^) in the Output column are used here only to show where blanks occur in the output; they do not print in your report.

Format	Value	Output
n10.2	123.456	^^^123.46
N10.2	12345678.01	^1.23E+007
n10.2	-2345678.01	-2.35e+006
n8	-123	^^^123
+n10.2	123.79	^^^123.79
-n10.2	123.79	123.79^^^

## Numeric Templates

If you need more complex numeric formats than the standard formats offer, you can specify a numeric format in a *template* form. Essentially, a template is an example of what the formatted output appearance. However, instead of specifying an actual sequence of digits, you use specially designated characters to indicate what must be printed at that position in the template. For instance, a Z indicates that the next digit in the number (or a space, if no digits remain) must be printed. A comma (,) in the template causes a comma to be printed in that position. So the template Z,ZZZ would cause the number 1000 to be printed as 1,000. In addition to the specially defined characters listed below, you can include any other character directly in the numeric template by preceding it with a backslash.

The general syntax of a numeric template is:

```
[ - | * | + ] ' { c } '
```

where:

**c**

Is one of several special characters that can be repeated any number of times, as follows:

**n or N**

Prints a digit if the number contains a digit in that position. Otherwise, prints a zero. If a field is specified *without* n in the numeric positions and Report-Writer encounters a value of zero (0), Report-Writer enters blanks in the field.

**z or Z**

Prints a digit if the number contains a digit in that position. Otherwise, prints a space. This is used for standard blank-padded numeric fields.

**\$**

(Dollar sign) Prints a digit if the number contains a digit in that position. If no digits remain, prints a floating dollar sign immediately to the left of the last evaluated digit. Report-Writer displays a dollar sign only once in the output field. If a dollar sign has already been printed, Report-Writer prints a space in this position. This can be used to print a dollar sign directly to the left of the number, or to place a dollar sign in a fixed position in the field when used with other template characters.

**-**

(Minus sign—Preceding or Trailing)

For preceding: Prints a digit if the number contains a digit in that position. If no digits remain and if the number is negative, prints a floating minus sign immediately to the left of the last evaluated digit. Report-Writer prints a minus sign only once in the output field. If a minus sign has already been printed, or if the number is positive and no digits remain, Report-Writer prints a space in this position.

For trailing: Prints a minus sign in the position if the number is negative, or if the number is positive, prints a space.

**+**

(Plus sign—Preceding or Trailing)

For preceding: Prints a digit if the number contains a digit in that position. If no digits remain, prints a floating sign (+ or -). Report-Writer prints a plus or minus sign only once in the output field. If one has already been printed, Report-Writer prints a space in this position.

For trailing: Prints a plus sign in the position if the number is positive or a minus sign if the number is negative.

**,**

(Comma) Prints a comma if the number contains any digits to the left of this position. If no digits remain, prints a space.

**.**

(Decimal point) Prints a decimal point in this position. The template can contain only one decimal point.

**\***

(Asterisk) Prints a digit if the number contains a digit in this position. If no digits remain, prints an asterisk. This is useful to fill a number on the left with asterisks (for example, for checks).

**space**

Prints a blank space in this position. This is identical to specifying a backslash followed by a space, and is provided for convenience only. Do not use spaces as thousands separators in place of commas and a decimal point if your template contains floating characters (+ - \$ [ ] ( ) < > { }). Floating characters work correctly only when used with commas and the decimal point as separators.

**\c**

(Backslash) Prints in the specified position any character c preceded by a backslash. This allows you to insert hyphens, slashes, or other characters into the number. (The backslash is not printed.)

**CR**

(Two characters) Inserts the characters "CR" (for credit) if the number is negative, or two blanks if positive. The letters "CR" appear exactly as specified, in uppercase and/or lowercase letters.

**DB**

(Two characters) Inserts the characters "DB" (for debit) if the number is negative, or two blanks if positive. The letters "DB" appear exactly as specified, in uppercase and/or lowercase letters.

**( ) or [ ] or < >**

(Parentheses, brackets or angle brackets) If the number is negative, prints it within the specified symbols.

The numeric template, like any number, is right justified by default. You can left justify the template by specifying the optional minus sign (-) as the first character in the template. Surround the template with single (') or double (") quotes; single quotes are recommended for consistency with requirements for quoting string constants.

Ingres evaluates the data from right to left, as it compares the number to the template.

If Report-Writer encounters a value of zero, it prints blanks (spaces) in the output field, unless the template contains n in the numeric positions. Report-Writer prints a floating symbol (\$, -, or +) only once in the output field, regardless of the number of times it appears in the template.

The following examples demonstrate the use of numeric templates. The carets (^) in the Output column are used here only to show where blanks occur in the output; they do not print in your report.

<b>Format</b>	<b>Value</b>	<b>Output</b>
'zzzzz'	123	^^123
'zZzZz.Zz'	0	^^^^^^^
'zzzzz.nn'	0	^^^^^.00
'+++ ,+++ ,+++ '	23456	^^^^+23,456
'--- ,--- ,---.NN'	23456.789	^^^^^23,456.79
'--- ,--- ,---.zz '	-3142.666	^^^^^-3,142.67^
'zzz,zzz,zzz.zz-'	-3142.666	^^^^^^3,142.67-
'\$\$\$\$,\$\$\$\$,\$\$\$\$.nnncr'	235122.21	^^^\$235,122.21^^
'\$\$\$\$,\$\$\$\$,\$\$\$\$.nnDb'	-235122.21	^^^\$235,122.21Db
'\$zz,zzz,zzn.nn'	1234.56	\$^^^^^1,234.56
'\$**,***,***.nn'	12345	\$*****12,345.00
'+\$\$,\$\$\$\$,\$\$\$\$. '	54321	^^^+\$54,321.00

Format	Value	Output
' nnnnnnnnnn '	023243567	^023-24-3567^
-'zzzzz'	123	123^^
'(zzzzz)'	-123	(^^123)
'[[[[[z]'	-123	^^[123]

## Control Character Format Q0

The q0 (q zero) format allows you to print a character string without the string taking up any actual space in the report. Report-Writer treats the string as if it had a length of zero. This format is designed to allow the printing of control character sequences within a report, although it can be used for other purposes. A typical use for the q0 format in a report might be to send an initialization sequence to an output device to change fonts or write bar codes.

The syntax of the q0 format is:

q0

where:

**0**

(Required) Permits any string length to be printed in this format. The parameter assumes a control sequence or a non-printable string is being specified.

Any printable character or hexadecimal string can be printed with this format. You must specify non-printable characters, such as TAB, SPACE, LF, FF, CR, and ASCII NUL (X'00'), as hexadecimal strings in the format, X|x'nn{nn}'. The q0 format sends the string directly to the output device as is, without interpretation. Hexadecimal strings in .print statements *without* the q0 format are interpreted by the formatting routines and can produce undesirable results or cause Report-Writer to fail. For more information, see Hexadecimal Strings (see page 295).

**Important!** Sending a report with a q0 character string to a device other than the one intended can halt your output device. For example, if your report uses q0 to highlight specific characters on your screen, your printer halts because it does not understand the initialization string.

Also, if you are sending a report with a q0 format to the screen, place positioning commands (such as .tab or .right) on a separate line from the q0 format and its string. For example:

```
.tab 14
.println $start_hilite (q0), 'This is
    hi- lighted', $end_hilite (q0)
```

or

```
.println $start_hilite (q0)
```

Placing positioning commands between the Q0 format and its string can produce incorrect positioning of the string output.

The following example initializes the printer to use a special font. The initialization sequence has been previously assigned to the variable *start\_font*. After Report-Writer prints the string, the variable *end\_font* resets the printer back to the original font.

```
.print $start_font (q0)
.print 'This will be printed in a different font.'
.print $end_font (q0)
```



## Character String Format T

The T format is nearly identical to the C format, except that the T format translates characters outside the normal character set into visible representations. For more information, see Character String Format C (see page 321).

This format is useful when you want to produce output that looks exactly like that of a terminal monitor, which expands unprintable characters into visible representations.

The syntax of a T format specification is:

`[-|*|+] tn[.w]`

The value for *n* is the width of the field that the expanded output occupies on the page. It does not refer to the number of characters of data that are translated.

Assume the data you wish to print is the character string "John?Smith,\Esq.", where the question mark (?) actually stands for a non-printing *formfeed* character. You might enter a print statement such as:

```
.print 'Output:', user_data(t0), ':Output'
.newline
```

The preceding statement would produce the following output:

```
Output:John\fSmith,\\Esq.:Output
```

The character representation of each unprintable character translated by the T format is as follows:

- Newline becomes \n.
- Horizontal Tab becomes \t.
- Backspace becomes \b.
- Carriage Return becomes \r.
- Form Feed becomes \f.
- Backslash becomes \\\.
- Null becomes \0.
- Any other unprintable character is printed as the character string \nnn, where *nnn* is the three-digit octal number equivalent to the character.

## Expressions and Formats Syntax Summary

The following lists useful expressions and formats syntax.

### Special Report Variables

current\_date  
current\_day  
current\_time  
left\_margin  
line\_number  
page\_length  
page\_number  
page\_width  
position\_number  
right\_margin  
w\_name  
w\_column

### Arithmetic Operators

+, - plus, minus (unary)  
\*\* exponentiation  
\*, / multiplication, division  
+, - addition, subtraction (binary)

### SQL Conversion Functions

c(*expr*)  
char(*expr*)  
date(*expr*)  
dow(*expr*)  
float4(*expr*)  
float8(*expr*)  
hex(*expr*)  
int1(*expr*)  
int2(*expr*)  
int4(*expr*)  
money(*expr*)  
text(*expr*)  
vchar(*expr*)  
varchar(*expr*)  
table\_key(*expr*)  
object\_key(*expr*)

## QUEL Conversion Functions

`c(expr)`  
`char(expr)`  
`date(expr)`  
`dow(expr)`  
`float4(expr)`  
`float8(expr)`  
`hex(expr)`  
`int1(expr)`  
`int2(expr)`  
`int4(expr)`  
`money(expr)`  
`text(expr)`  
`vchar(expr)`  
`varchar(expr)`

## Numeric Functions

`abs(n)`  
`atan(n)`  
`cos(n)`  
`exp(n)`  
`log(n)`  
`mod(n,b)`  
`sin(n)`  
`sqrt(n)`

## SQL and QUEL String Functions

`charextract(c1,n)`  
`concat(c1,c2)`  
`left(c1,len)`  
`length(c1)`  
`locate(c1)`  
`lowercase(c1)`  
`pad(c1)`  
`right(c1, len)`  
`shift(c1,nshift)`  
`size(c1)`  
`squeeze(c1)`  
`trim(c1)`  
`uppercase(c1)`

## Date Functions

`date_trunc(unit,date_)`  
`date_part(unit,date_)`  
`date_gmt(date)`  
`_date(s)`  
`interval(unit,date_interval)`  
`_time(s)`

## Boolean Function

`break (columnname)`

## Aggregates

`cumulative [(breakname)] sum[u] (columnname [preset])`  
`cumulative [(breakname)] count[u] (columnname [preset])`  
`cumulative [(breakname)] minimum (columnname [preset])`  
`cumulative [(breakname)] maximum (columnname [preset])`  
`cumulative [(breakname)] average[u] (columnname) [(format)]`

## Formats

[+ | \* | -]bw

where *w* is width

[+ | \* | -]c[f | j ] *n* [.w]

where *n* is total string length, and *w* is the column width

[+ | \* | -]ew[.d]

where *w* is width, *d* is *precision*

[+ | \* | -]fw[.d]

where *w* is width, *d* is *precision*

[+ | \* | -]gw[.d]

where *w* is width, *d* is *precision*

[+ | \* | -]nw[.d]

where *w* is width, *d* is *precision*

[+ | \* | -]tn[.w]

where *n* is total string length, and *w* is the columnwidth

[+ | \* | -] '{c}'

valid characters are: n or N z or Z \$ + ' , . \* CR [ ] ( ) DB \c or space

[+ | \* | -] D[d ] '*template*'

where *template* is a string of characters specifying the absolute date and time or specifying a time interval

q0

where 0 is a required parameter that permits any string length to be printed



# Chapter 12: Report-Writer Statements

---

This section contains the following topics:

[Report Setup Statements](#) (see page 347)

[Page Layout and Control Statements](#) (see page 380)

[Report Structure Statements](#) (see page 393)

[Column and Block Statements](#) (see page 398)

[Text Positioning Statements](#) (see page 414)

[Print Statements](#) (see page 428)

[Conditional and Assignment Statements](#) (see page 436)

[Statement Syntax Summary](#) (see page 438)

## Report Setup Statements

Report setup statements are used for setting up the overall report environment.

### .Break Statement—Specify Break Columns

The .break statement specifies the break columns for the report and the order in which they should break.

The .break statement has the following format:

```
.break | .brk columnname{, columnname
```

The parameters for the .break statement are:

#### ***columnname***

Name of a column in the table used as the basis for your report, or the label for a column in the result column list of the specified query.

The *columnname* can be expressed optionally as a delimited identifier or a variable. The comma (,) must be stated explicitly and cannot be part of a *columnname* variable. You can specify *columnname* as a delimited identifier by enclosing it in double quotes ("), if you have previously specified the .delimid statement.

#### ***\$columnvariable***

Variable whose value is a column name. The variable must be preceded with a dollar sign (\$).

## Description

You can use the optional `.break` statement to specify the break columns if no `.sort` statement has been specified, or to override the default break columns created by the `.sort` statement. The order in which Report-Writer processes the break statements is the order in which they appear in the specified break list. A break on one column in the list produces a break on all subsequent columns in the list.

If a variable is specified as the column, Report-Writer evaluates the *columnname* during the loading of the report specification, before retrieving the data.

If a `.sort` statement is not specified, all columns that have `.header` or `.footer` statements must be included in the break list.

If you specify an order by clause in a `.query` statement, a `.break` statement that lists the columns in the order by clause must also be specified. The `.query` statement does *not* create default column breaks as does the `.sort` statement.

**Note:** When using a variable for *columnname*, the variable must be specified identically in corresponding `.sort`, `.header`, and `.footer` statements.

## Examples

1. The following example breaks on the columns, state and city of employment (a delimited identifier). The order to sort the rows retrieved from the database appears in the `.query` statement. The `.break` statement is required to identify the sort columns to Report-Writer.

```
.query
  select *
    from emp
   order by state, "city of employment"
.break state, "city of employment"
```

2. The following example overrides the original sort order specified in the `.sort` statement by using the `.break` statement. This might be done so that a change in the second sort column does not trigger a break on the third column. Note the consistent usage of variable names.

```
.sort    $first_col :a,
         $second_col :a,
         $third_col :a
.break   $first_col, $third_col
```



## .Cleanup Statement—Embed SQL Statements that Clean Up

The .cleanup statement embeds SQL statements that do not involve data retrieval into Report-Writer sections. Report-Writer executes the statements *after* the main report query is processed.

The .cleanup statement has the following format:

```
.cleanup SQL_statement; {SQL_statement};
```

The parameters for the .cleanup statement are:

### **SQL\_statement**

One or more action SQL statements that do not involve data retrieval, separated by semicolons (;).

**Note:** The select statement cannot be used.

For a complete explanation of all available statements, see the *SQL Reference Guide*.

## Description

Use the optional .cleanup statement to embed groups of SQL statements that perform cleanup tasks such as dropping temporary tables created in the .setup section. Report-Writer executes the .cleanup section after it substitutes Report-Writer variables into the SQL statements. For information on embedding groups of SQL statements before the report is processed, see the .Setup statement.

Use as many lines as you need to specify the .cleanup. You can also include embedded declared variables within an SQL statement in the .cleanup section. The end of .cleanup is detected by the start of a new Report-Writer statement or an end-of-file indicator.

The following rules apply to the .cleanup section:

- The .cleanup section only supports SQL statements. The language of your .query section determines the query language of the report. If you have a QUEL .query section, the query language is QUEL; otherwise it is SQL. You can have a QUEL query and SQL .setup and .cleanup sections.
- If the report query language is SQL, the default value of autocommit is off. If it is QUEL, the default is on. To override the default commit behavior, set autocommit off or on as the first statement in your .setup section.

- Only statements that are compatible with execute immediate are permitted in a .cleanup section. For a list of compatible statements, see the *SQL Reference Guide*. Neither the select statement nor any statement requiring embedded semicolons (;) or colons (:), such as create procedure, are allowed. Semicolons within quoted strings *are* allowed. Therefore, you can specify a table for selection by using the expression:  
`create table tablename as select...`
- Report-Writer evaluates variables in the .cleanup section only once, before running the report. Therefore, you can set the value of the variable only at report runtime, as follows:
  - On the command line in a *variablename=valuestring* clause
  - On the command line in response to a prompt
  - In the with value or with prompt clause of a .declare statement associated with the variable

While actually executing the .cleanup section, Report-Writer does not recognize any changes that have been made to .cleanup section variables while the report was being run.

- In the .cleanup section, Report-Writer evaluates variables before sending them to the Database Management System (DBMS) and evaluates SQL statements at report runtime. It generates error messages at runtime from the DBMS. If it does not detect any errors in the report or in the .cleanup and .setup sections, and if autocommit is off, Report-Writer executes an explicit commit at the close of report processing.
- Use the -d flag with the report command to run the report as if there were no errors. This flag causes DBMS errors in the .setup and .cleanup section to be ignored. The failure of the .setup statement, however, can affect data availability in the .query. For example, if you run the report on a temporary table created improperly in the .setup section, the report fails.
- When you specify the -d flag, error messages continues to display on the screen. Although it is not recommended, you can also run the report with the -s flag if you do not want error messages to display.
- If you do not use the -d flag, errors and transaction handling follow these rules:
  - If Report-Writer detects an error in the .cleanup section, the .setup section (if it exists) and the report have already run. The transaction is rolled back if autocommit is off.
  - If Report-Writer detects an error in the .setup section, neither the report nor the .cleanup section runs. The transaction is rolled back if autocommit is off.
  - If Report-Writer detects an error in the query, or if there is a fatal error in the report, neither the .cleanup section nor the report runs. The transaction is rolled back if autocommit is off.

## Examples

For .cleanup examples, see .Setup Statement (see page 374).

## Comments

You can include comments for your own documentation within a report specification saved as a text file.

Commented text has the following format:

```
/* {any_text} */
```

where

***any\_text***

Is any text, except the characters \* and /, which close the comment.

## Description

To create internal documentation for a report specification saved as a text file, you can include comments within the report specification by surrounding them with the /\* and \*/ characters. Report-Writer ignores all text between these characters during report processing. If you save your report specification with the sreport command, Report-Writer ignores the comments and does not save them when storing your report specification in the database.

Comments can be nested; that is, you can have a set of comments within another set of comments. You can place comments anywhere within your file.

## Examples

```
/* this is an example
   of a comment...
   */

/* You can also have
   /* nested comments */
   in Report-Writer specifications */
```

## .Data Statement—Specify Table for Report

The .data statement specifies the table or view in the database containing data for the report.

The .data statement has the following format:

```
.data | .dat | .table | .view [schema.] tablename|viewname|synonym
```

The parameters for the .data statement are as follows:

### ***schema***

Collection of database objects to which the specified object belongs. The schema also implies the user that owns the object.

The *schema* can be expressed as a variable or delimited identifier.

### ***tablename* | *viewname* | *synonym***

Name of a table or view in the database. You can also use a synonym for *tablename*. Report-Writer reads all rows and columns in the specified table or view each time the report is run. Rules for the use of *tablename* and *viewname* are the same as in all other Ingres tools.

The *tablename*, *viewname*, and *synonym* can be expressed as variables or delimited identifiers.

### ***\$tablevariable***

Variable whose value is the name of a table, view, or synonym for a table in the database. The variable must be preceded with a dollar sign (\$).

## Description

The .data statement identifies a table or view in the database that is used in its entirety in the report. The four statement names shown in the syntax are synonymous and can be used interchangeably. Each time you run a report with the report command, all of the rows and columns in the table or view are available for use in the report specification. One exception to this is that Report-Writer silently ignores and does not print values for columns with unsupported data types such as long varchar, byte, byte varying, and long byte.

**Note:** If Report-Writer encounters subsequent references to a column of an unsupported data type, such as within sort operations, it issues an error message and terminates the report.

You can use a delimited identifier (see page 51) for *tablename* if you have previously specified the .delimid statement.

If you specify a variable as the table, Report-Writer evaluates *tablename* during the loading of the report specification, before retrieving the data.

An SQL language report specification includes duplicate rows in the data for reports that use the `.data`, `.table`, or `.view` statements. To specify distinct rows, you can specify the `-6` flag on the report command line.

Either the `.data` or the `.query` statement is a required statement for each report. The `.data` and `.query` statements are mutually exclusive; both cannot appear in the same report specification.

## Examples

1. Use table "repdat" for the report.  
`.data repdat`
2. Use view "myview" for the report.  
`.table myview`
3. Use the value of variable `rep_table` as the table for the report.  
`.data $rep_table`
4. For the report, use the table whose name is the delimited identifier "my table" and whose schema is "robert."  
`.data robert."my table"`

## .Declare Statement—Declare Variables

The `.declare` statement declares variables that can be assigned values and used in expressions. More than one `.declare` statement can be specified in a report.

The `.declare` statement declares variables that can be assigned values interactively or in Report-Writer code. You can assign a value to the variable in any of the following ways:

- On the command line during runtime
- On the command line in response to a runtime prompt, which you define in the `with prompt` clause of a `.declare` statement
- In `.let` assignment statements placed in any `.header`, `.footer`, or `.detail` sections.
- In the `with value` clause of a `.declare` statement

Declared variables can also be used in a query block to specify runtime substitution of text in the query. For details, see the `.Query` statement.

The `.declare` statement has the following format:

```
.declare variablename = datatype
  [with null | not null]
  [with prompt 'promptstring']
  [with value 'valuestring']
  {, variablename = datatype...}
```

The parameters for the `.declare` statement are as follows:

***variablename***

A valid variable name. In standard Ingres databases, the variable name can be up to 32 bytes long and must begin with an alphabetic, or underscore (\_) character. Following characters must be alphanumeric or underscore. Also see Object Naming Conventions for ANSI/ISO Entry SQL-92 Compliant Databases (see page 287).

**Note:** To reference the *value* of a declared variable within a report specification, the variable must be preceded with a dollar sign (\$); otherwise, the *name* of the variable is referenced. Do not use the dollar sign (\$) when referencing the name of the variable in the `.declare` statement or on the left side of an assignment in the `.let` statement. For example:

```
.declare var = c3 . . .
.let var = 'abc'
.print $var
```

***datatype***

A legal data type.

The `.declare` statement declares each variable to be the given data type. You can specify whether a data type is nullable or not nullable by including the `with null` or `not null` option.

- If the variable is declared as nullable with the `with null` option, it is automatically initialized to the null value.
- If the variable is declared with the `not null` option, it is initialized to the default value for the data type.

If you specify neither option, the variable data type defaults to null or not null, depending on the query language (SQL or QUEL) used in the `.query` statement. If you specify a `.data` statement instead of the `.query` statement, the installation default language determines default nullability.

***promptstring***

A string constant up to 100 characters in length including quotes. The constant must be enclosed in single quotes to preserve spaces. For more information, see String Constants (see page 293).

The `with prompt` option instructs Report-Writer to prompt for the initial value of the variable at runtime, using the specified prompt string. The `with value` option instructs Report-Writer to use an initial value that you specify for the variable in the `.declare` statement. Using both the `with prompt` and `with value` options allows you to specify a default value if the user fails to enter a value at the prompt. This value remains unchanged by any `.let` statement until after the query. For more information, see the `.query` and `.let` statements.

### ***valuestring***

An initial string value for the variable up to 100 characters in length including quotes. The value must be expressed as a constant in character, numeric, or date format and must be enclosed in quotes to preserve spaces. It can contain delimited identifiers (enclosed in double quotes) or `schema.objectname` constructs.

You can use the `schema.objectname` construct in the `with value` clause, as appropriate. You can also use a delimited identifier as the schema name, table name, or name of a column in the `with value` clause, if you have previously specified the `.delimid` statement.

For example, the following code fragment declares the variable, `default_table`, with an initial default value of a table belonging to the schema "dave" and whose name is the delimited identifier, "my table." The value string parameter for the `with value` clause is enclosed in single quotes.

```
.declare default_table=varchar(65) with value
'dave."my table"'
.query
select * from $default_table
```

If you do not specify an initial value or prompt, and you reference the variable outside of a query block, the initial value is null (or the default value for that data type, if not null was used). When you reference a declared variable within a query block, its initial value *must* be entered either in the `.declare` statement, on the command line, or in response to a prompt string specified in the `.declare` statement.

**Note:** Some statements (such as `.query`) accept variables that can be executed before the `.header report` statement. However, variable values cannot be assigned with the `.let` statement before the `.header report` statement. In these cases, the person running the report must specify the value of a variable as a parameter on the command line or in response to a prompt.

The `with value` option, used in conjunction with the `.include` statement, allows greater reporting flexibility. For example, you can create various include files to define date formats for various languages. Instead of entering the initial value of the date format interactively, the date formats are initialized in the `.declare` statement during the loading of the report specification. For more information, see Example 2 in the following section, Examples.

If both the `with prompt` and `with value` options are included in a declaration, the *promptstring* overrides the *valuestring*. No warning is issued about the override. The user is prompted with the specified *promptstring* for an initial value at runtime.

## Examples

1. Declare variables using `with prompt`, `with null`, and `not null`.

```
.declare
  counter = integer,
  salary = money with prompt
    'Please enter the salary:',
  spouse = c30 with null,
  dept = i4 not null with prompt
    'What department?'
```

2. Declare variables using `with value` in `.include` files "BR\_fmts.rw" and "AM\_fmts.rw," and then include the appropriate definition for that version of your report.

```
.declare date_fmt = c30 with value 'd\'03/20/01\''
```

or

```
.declare date_fmt = c30 with value 'd\'02/03/01\''
```

The main body of the report can be written independently of the exact value of the date format; for example:

```
.print current_date ($date_fmt)
```

3. Declare a variable with a value that is an schema-qualified table name.

```
.declare var1 = varchar(65) with value 'mike.table_abcd'
```

The schema and table name can be delimited identifiers, if you have specified the `.delimid` statement. The variables have been declared with the maximum size to accommodate a compound identifier in which each part is a delimited identifier.

```
.delimid
```

```
.declare var2 = varchar(133)
  with value 'jane."table efg"'
```

```
.declare var3 = varchar(133)
  with value '"c bradley".table_hijk'
```

```
.declare var4 = varchar(133)
  with value '"r panzer"."table xyz"'
```



## .Delimid Statement—Enable Recognition of Delimited Identifiers

The .delimid statement allows recognition of delimited identifiers for table, view, column, schema, and user names.

The .delimid statements has the following format:

```
.delimid
```

### Description

Specify the .delimid statement in the Report-Writer source file to enable Report-Writer to recognize delimited identifiers in your report specification code. For a detailed discussion of delimited identifiers, see the chapter "Report-Writer Expressions and Formats."

You only need to specify this statement once in the source file. However, Report-Writer accepts multiple occurrences in a report specification to support use of the statement in individual and independent .include files for which delimited identifiers must be enabled.

**Note:** If you use the .delimid statement in an .include file, be sure the included file actually contains code that requires this statement. Otherwise, Report-Writer can produce unexpected results.

Observe the following rules regarding placement of this statement in your report specification:

- The .delimid statement must precede any statement that allows delimited identifiers. Therefore, only the following statements can precede the .delimid statement:
  - .name
  - .shortremark
  - .longremark ... .endremark
  - .[no]formfeeds
  - .nullstring
  - .output
  - .ulcharacter

The .include statement cannot precede the .delimid statement because the included file can contain statements that allow delimited identifiers.

- If your report specification contains a .query statement specifying QUEL as the query language.

## Example

The following example shows use of the `.delimid` statement in a report specification that creates a temporary table with columns "aaaa" and "delim col," then selects and prints values from column "aaaa" that match the value entered in response to a prompt for the variable "my\_smallint."

```
.name sql_setup
.delimid
.declare my_smallint = smallint with prompt 'Enter
      selection key numeric value: '
.setup
      create table my_temp_table (aaaa smallint,
      "delim col" varchar(4));
      insert into my_temp_table values (123, 'abcd');
.query
      select aaaa from my_temp_table where aaaa =
      '$my_smallint'
.detail
      .println 'Key Value: ', aaaa
.cleanup
      drop table my_temp_table;
```

## .Delimid with QUEL Query

Delimited identifiers cannot be used in a QUEL query. However, you can use delimited identifiers and the `.delimid` statement in non-query portions of a report specification that contains a QUEL query, as shown in the following example.

This example creates a temporary table with columns "aaaa" and "delim col," then retrieves and prints values from column "aaaa" that match the value entered in response to a prompt for the variable "my\_smallint."

```
.name sql_setup
.delimid
.declare my_smallint = smallint with prompt 'Enter
      selection key numeric value: '
.setup
      create table my_temp_table (aaaa smallint,
      "delim col" varchar(4));
      insert into my_temp_table values (123, 'abcd');
.query
      range of mtt is my_temp_table
      retrieve (mtt.aaaa, quote = "\"")
      where mtt.aaaa = "$my_smallint"
.detail
      .println 'Key Value: ', aaaa
.cleanup
      drop table my_temp_table;
      create table my_temp_table (aaaa smallint,
      "delim col" varchar(4));
      insert into my_temp_table values (123, 'abcd');
      drop table my_temp_table;
```

## .Include Statement—Import Files that Contain Report-Writer Code

Imports Report-Writer code residing in more than one file when the report is saved.

The .include statement has the following format:

```
.include filename
```

The parameter for the .include statement is as follows:

### ***filename***

The name of a text file containing a Report-Writer construct to be included in a report. If necessary, specify the full path name for the file. If you do not explicitly specify an extension for the file, the system assumes ".rw" as the default extension.

## Description

Use the optional .include statement to specify one or more files containing Report-Writer code to be imported into the command text file. By using this statement, you can avoid retyping commonly used constructs that you need in more than one of your reports. You can nest .include statements within an .include statement.

Because Report-Writer constructs do not have to appear in any particular order, the .include statement can appear anywhere within Report-Writer code. However, the .name statement must still be the first statement in your specification.

Report-Writer incorporates the contents of the file specified in an .include statement each time you execute the sreport command to save your report specification. This means that when you update included files, the report specifications that include those files must be saved again using sreport to incorporate the changes. As Report-Writer executes sreport, it displays a message notifying you of each file that is included in your specification. This message can help diagnose report errors if any should occur.

**VMS:** If you give the full path name for a file, enclose the name in single quotes. If it is the name of a file in the current directory, no quotes are needed. ■

The following rules apply to the .include section:

- The .include statements cannot contain variables. This is because variables are only available at runtime and Report-Writer only executes .include statements when you save the report specification with the sreport command (rather than when the report is executed with the report command).

- An included file can contain more than one statement. However, a single Report-Writer statement must be contained completely within one file. You cannot use `.include` statements in the middle of another Report-Writer statement.
- Within an `.include` file, you can have nested `.include` statements.
- The maximum number of nested `.include` statements permitted is based on the number of open files allowed in your system.
- A corresponding `.if` `-.then-` `.else[if]` `-.endif` statement must all be contained in the same `.include` file, but statements following the condition can be other `.include` statements.

### Examples

1. Include a file in current directory.  
`.include otherrep.rw`
2. Include a file name with full path name.

#### **Windows:**

```
include \direct\subdirect\otherrep.rw
```

#### **UNIX:**

```
.include /direct/subdirect/otherrep.rw
```

#### **VMS:**

```
.include '[direct.subdirect]otherrep.rw'
```

## .Longremark/.Endremark Statements—Mark a Block of Descriptive Text

Begins and ends a multi-line block of text that describes the report.

The .longremark and .endremark statements have the following format:

```
.longremark | .lrem  
    remark_text  
.endremark | .endrem
```

The parameter for the .longremark statement is as follows:

### **remark-text**

Any number of characters or lines of text.

## Description

The .longremark and .endremark statements are an optional pair that specify a lengthy description of the report. Indicate the start of the block of descriptive text with the .longremark statement and denote the end of the block with the .endremark statement. This long description appears in the Save frame and MoreInfo display of the Catalog frame in RBF.

The long remark differs from a Report-Writer comment (*/\* comment text \*/*). Report-Writer stores the descriptive text in the .longremark statement in the database when you save the report with the sreport command, so that it is thereafter available to other Ingres application development tools. By contrast, you can only save Report-Writer comments in the text file. The sreport command ignores comments of this type and does not save them in the database.

There can be only one .longremark statement in a report specification. Should you attempt to enter two .longremark statements in one report specification, the second is flagged with a syntax error. You can enter as much remark text as you like. However, only the first 600 characters of remark text is saved in the DBMS and appear in the RBF windows; the rest is ignored. Report-Writer ignores leading spaces that separate the .longremark statement from the first character of the remark text. Tab characters convert to blank characters.

## Examples

### Example 1

```
.longremark  
This report correlates information from the sales order header, the sales order  
detail, and the inventory files, to produce the customer backlog by part number  
report.  
.endremark
```

### Example 2

```
.lrem  
Stock Analysis Report  
8 1/2" x 11" output  
10 minutes runtime  
Input: Begin/End date  
.endrem
```

## .Name Statement—Assign a Name to a Report

The .name statement assigns a name to a report.

The .name statement has the following format:

```
.name | .nam reportname
```

The parameter for the .name statement is as follows:

### ***reportname***

The name of a report to which the next set of formatting statements apply. The *reportname* is a standard object which must comply with object naming rules. See Naming and Name Use Conventions (see page 49) or Object Naming Conventions for ANSI/ISO Entry SQL-92 Compliant Databases (see page 287).

## Description

The .name statement is required and must be the first statement specified for a report. The program sreport stores the report in the database under the report name rather than the file name of the file containing the report specification statements.

Specifications for several reports can be stored in one text file by using several .name statements. Each occurrence of a .name statement signals the end of the previous report's specification statements and the beginning of a new report. You can then save the reports in the database by using sreport.

When executing a report directly by using the -i flag with the report command, you must have exactly one .name statement specifying one report only. The .name statement cannot appear in a file specified by an .include statement.

## Examples

1. Denote the start of the report, abc:  

```
.name abc
```
2. Denote the start of report, my\_rep:  

```
.name my_rep
```

## .Output Statement—Specify File Name for a Report

The .output statement specifies the file name where the report is written.

The .output statement has the following format:

```
.output | .out filename
```

The parameters for the .output statement are as follows:

### ***filename***

A file to which the formatted report is written each time the report is run. The *filename* parameter must follow all conventions for valid file names in the operating system.

The *filename* can be expressed as a variable.

### ***\$filevariable***

Variable whose value is the file name. The variable is preceded by a dollar sign (\$).

## Description

The .output statement is an optional statement that specifies the name of a file where the report is written.

If you specify a variable for *filename*, Report-Writer evaluates the variable during the loading of the report specification, before retrieving the data.

If you don't use the .output statement in your report specification, Report-Writer either directs the output to the screen, or to a file name specified on the command line for the report command with the -f flag. The -f flag takes precedence over the .output statement. If both the -f flag and the .output statement are present, the report results are sent to the file specified by the -f flag, rather than of the .output statement. If the .output statement is not specified, and no file is specified with the -f flag, the report appears on your screen.

**VMS:** If you give the full path name for a file, enclose the name in single quotes. If it is the name of a file in the current directory, no quotes are needed. ■

## Examples

1. Write to file in current directory.  

```
.output myreport.lis
```
2. Write to file with full path name.

### **Windows:**

```
.out \direct\subdirect\myreport.lis ■
```

**UNIX:**

```
.out /direct/subdirect/myreport.lis
```

**VMS:**

```
.out '[direct.subdirect]myreport.lis'
```

## **.Query Statement—Indicate Start of a Query**

The .query statement specifies an SQL or QUEL query to be used to generate data for a report. The statement syntax and description provided here apply to SQL only. If you are using the QUEL language, see .Query Statement for QUEL Users (see page 369).

The .query statement has the following format:

```
.query | .quer
select [all | distinct] column_list
[as resultcolumn_list]
from [schema.] table | view | synonym [corr_name]
    {, table | view | synonym [corr_name]}
[where search_condition]
[group by column {, column}]
[having search_condition]
    {union select_statement }
[order by ...]
```

For a complete explanation of the syntax and parameters for the SQL select statement within the Report-Writer .query statement, see the *SQL Reference Guide*.

### **Description**

The .query statement indicates the start of a valid SQL query that creates the data to be reported. This query follows the same rules as any other SQL select statement, although it can also contain variables. You can use as many lines as you need to specify the query. The end of the query is indicated by the start of a new report formatting statement.

Specify either the .query or the .data statement (but not both) for every report. Only one .query statement is permitted for a report, and only one data retrieval statement is permitted within the .query statement. Both a .query with an order by clause and a .sort statement cannot be in the same report specification.



You can use the optional *select column\_list as resultcolumn\_list* construct to reference a column in subsequent statements by a name other than its database table column name. For example, if you selected A\_column as B\_column, thereafter you need to refer to B\_column in all statements referencing that column. This construct is useful for printing the value of a column that is specified as a runtime variable. When Report-Writer encounters a .print statement that directly references a runtime variable for a column name (for example, .print \$account\_type), it prints the *name* of the column as entered at runtime, rather than its *data*. To print the data rather than the column name, you use the *select column\_list as resultcolumn\_list* construct in the query and reference the *resultcolumn\_list* name in the .print statement.

For example:

```
.query
      select $account_type as val
.print val
```

Because the .query statement generates a standard query, the standard limits apply to any report's query. For Ingres databases, these limits are 1024 columns and 2008 bytes per row. These limits are extended for some Enterprise Access products. For details, see your Enterprise Access product guide.

If any column you specify with *select \** is of an unsupported data type, such as long varchar, byte, byte varying, and long byte, Report-Writer silently ignores and does not print any values for that column. If you explicitly specify a column with an unsupported data type in the *column\_list* of a select statement in a query, Report-Writer additionally issues a warning message. If it encounters any subsequent reference to a column with an unsupported data type, regardless of the method used to select the column, Report-Writer issues an error message and terminates the report.

You can specify the table, view, or synonym as *schema.objectname*. You can also use delimited identifiers for table, column, or schema names if you have previously specified the .delimid statement. This includes use of delimited identifiers (see page 51) as either or both the correlation name and the column name in a *correlationname.identifier as resultcolumn* construct. For more information, see the .delimid statement.

To avoid any potential confusion with delimited identifiers, which are double-quoted, enclose string constants with the standard SQL string delimiter, the single quote (').

## Variables

For standard Ingres databases, variable names:

- Must begin with a letter
- Can use letters, digits, and underscore (\_), and otherwise must comply with the valid object naming rules described in Naming and Name Use Conventions (see page 49).
- Cannot match any of the reserved words listed in Reserved Words (see page 292).

For variable naming conventions in databases that comply with ANSI/ISO Entry SQL-92 standards, see Object Naming Conventions for ANSI/ISO Entry SQL-92 Compliant Databases (see page 287).

You can use variables as runtime substitutes for any part of a query (that is, as a field name, table name, or even in where clauses). You indicate variables in a query by preceding the name with a dollar sign (\$). For example, you can specify a query as follows:

```
.query
  select empname, salary, manager
  from emp
  where salary>$minsal
```

Subsequently, you can invoke the report with a statement such as:

```
report mydb myrep (minsal = 20000)
```

If the parenthetical clause contains characters that are treated specially by your operating system (such as parentheses in Windows NT), enclose it in double quotes:

```
report mydb myrep "(minsal = 20000)"
```

Report-Writer converts the query to:

```
... where salary>20000
```

You can also assign an initial value for the variable using the with value option in the .declare statement. Report-Writer initializes the variable with your assigned value during the loading of the report specification.

If the value of a variable is not assigned on the command line or during the loading of the report specification, Report-Writer prompts you for the value, using the custom string specified in the with prompt option in the variable declaration. If you did not specify a custom string prompt, Report-Writer uses a default prompt.

You are *strongly* encouraged to define all variables with the .declare statement. Although Report-Writer recognizes any name preceded by a dollar sign (\$) as a variable, undeclared variables assume default types and characteristics that are often incompatible with their intended use.

By defining variables with .declare, there is no limit to the ways you can use the variable in your report. You can assign a value to an undeclared variable only by using a command line parameter or by entering a value in response to a runtime prompt. A value for an undeclared variable cannot be specified with a .let statement. In addition, unless the variable has been declared, attempting to pass a parameter with a null value to Report-Writer can produce incorrect results.

Specify as many variables as you want in a query by defining a unique name for each variable. If the same variable is to be substituted more than once within the query, specify the name, prefixed by a dollar sign (\$), at each place where substitution is to be done.

Variables can be specified anywhere in the query, or example, within the column list for a select statement or as a value to which the contents of a column is compared. For example, the following query phrases are legal:

```
... select $var, ... from emp ...  
... where name = '$employee_name' ...
```

If the variable is a character string used for comparison purposes in a where clause, enclose the variable within quotes that are appropriate for your query language (single quotes for SQL), as shown in the preceding example. If you omit these quotes in the query, the user must be made aware of the need to include them in the string value for the variable at runtime.

If you actually want to include the dollar sign (\$) as a literal part of the query, precede it with a backslash (\). For example:

```
... where symbol = '\$' ...
```

You can also use variables specified in the .query statement in the body of the report. Report-Writer prints the value of the variable if the variable name is preceded by a dollar sign (\$). For more information, see the Population Example in the appendix "Report-Writer Report Examples."

## Examples

1. You set up the query as:

```
.query  
  select *  
    from emp  
   where salary > $sal  
   and dept = '$dept'
```

You invoke the command:

```
report mydb myrep (sal = 50000, dept = 'toy')
```

Enclose toy within single quotes to identify it to Report-Writer as a string. Report-Writer strips off these quotes before passing it to the query.

**Note:** If your operating system requires it, enclose the parenthetical clause within double quotes:

```
report mydb myrep "(sal = 50000, dept = 'toy')"
```

Report-Writer executes the following query:

```
select *
  from emp
 where salary > 50000
 and dept = 'toy'
```

2. Consider a table called account owned by davis with columns including customer number, customer name, checking, and savings. You have separate fields for checking and savings accounts on one row, because most customers have both a savings and a checking account with the bank.

If you want to write one report specification that prints either the savings or checking account balances with a single query, you could code a .query statement similar to the following, using the select *column\_list* as *resultcolumn\_list* construct:

```
.delimid
.declare account_type = c10 with prompt
'Please enter the type of account:'
.query
  select "customer number", "customer name",
         $account_type as val
  from davis."bank accounts"
```

The column names and table name in the preceding code must be double-quoted, according to standard rules for specifying delimited identifiers.

You can invoke the preceding query with the command:

```
report otherdb rename
```

At execution, Report-Writer issues the following prompt and, in this example, the user enters the column name, savings:

```
Please enter the type of account: savings
```

The following query would be executed, which selects values from the savings column and assigns them to the result column name, val:

```
select "customer number", "customer name", savings as val
  from davis."bank account"
```

To print the value of the column, savings, refer to the result column name, val:

```
.println val
```

## .Query Statement for QUEL Users

This statement specifies a QUEL query to be used to generate data for a report.

The .query statement for QUEL has the following format:

```
.query | .quer  
{range_statement(s)}  
retrieve [unique] (target_list) [where qual]  
    [sort by | order by...] sort-list
```

The parameters for the QUEL .query statement are as follows:

### ***range\_statement(s)***

Valid QUEL range statements used to identify the tables used in the query.

### ***target\_list***

A valid QUEL target list that creates data for the report. The retrieve into form of the retrieve statement is not used.

### ***qual***

A valid QUEL qualification to a query.

### ***sort\_list***

A valid set of sort columns according to the rules for constructing QUEL retrieve statements. Note that you cannot use both a .sort statement and a sort by or order by clause in the same report specification.

## Description

The .query statement indicates the start of a valid QUEL query that creates the data to be reported. All range statements needed to designate the row markers for the query must be specified. This query follows the same rules as any other QUEL query, although it can also contain parameters. You can use as many lines as you need to specify the query—sreport detects the end of the query by the start of a new report formatter statement.

Either the .query or the .data statement (but not both) must be specified for every report. Because the .query statement generates a standard Ingres query, the usual limits of 1024 columns and a maximum row width of 2008 bytes apply to any query issued through the Report-Writer.

If any column you specify with `retrieve all` is of an unsupported data type, such as `long varchar`, `byte`, `byte varying`, and `long byte`, Report-Writer silently ignores and does not print any values for that column. If you explicitly specify a column with an unsupported data type in the *column\_list* of a `retrieve` statement in a query, Report-Writer additionally issues a warning message. If it encounters any subsequent reference to a column with an unsupported data type, regardless of the method used to select the column, Report-Writer issues an error message and terminates the report.

You can use the optional `retrieve (resultcolumn=table.columnname)` construct to reference a column in subsequent statements by a name other than its database table column name. For example, if you retrieved `B_column=table1.A_column`, in subsequent statements refer to the column as `B_column`.

Only one `.query` statement is permitted for a report, and only one data retrieval statement is permitted within the `.query` statement.

You cannot have both a `.query` with an `order by` or `sort by` clause and a `.sort` statement in the same report specification, because their functions are mutually exclusive.

Neither delimited identifiers nor the *schema.objectname* construct are allowed in QUEL queries.

String constants must be enclosed by the standard QUEL string delimiter, the double quote (`"`). The double quote string delimiter is required only within the `.query` statement; within other Report-Writer statements, the string delimiter is the single quote.

Variable names:

- Must begin with a letter
- Can use letters, digits, and underscore (`_`), and otherwise must comply with the valid object naming rules described in *Naming and Name Use Conventions* (see page 49).
- Cannot match any of the reserved words listed in *Reserved Words* (see page 292).

Variables in the Query

You can use variables as runtime substitutes for any part of a query (that is, field names, table names, or even where clauses). You indicate variables in a query by preceding the name with a dollar sign (`$`). For example, you can specify a query as follows:

```
.query
  range of e is emp
  retrieve (e.empname,e.salary,e.manager)
  where e.salary > $minsal
```

Subsequently, you can invoke the report with something like the following:

```
report mydb myrep (minsal = 20000)
```

If the parenthetical clause contains characters that are treated specially by your operating system (such as parentheses in Windows NT and UNIX or slashes in VMS), enclose it in double quotes:

```
report mydb myrep "(minsal = 20000)"
```

Report-Writer converts the query to:

```
... where e.salary > 20000
```

You can also assign an initial value for the variable using the `with value` option in the `.declare` statement. Report-Writer initializes the variable with your assigned value during the loading of the report specification.

If the value of a variable is not assigned on the command line or during the loading of the report specification, Report-Writer prompts you for the value, using the custom string specified in the `with prompt` option in the variable declaration. If you did not specify a custom string prompt, Report-Writer uses a default prompt.

We *strongly* encourage you to define all variables with the `.declare` statement. Although Report-Writer recognizes any name preceded by a dollar sign (\$) as a variable, undeclared variables assume default types and characteristics that are often incompatible with their intended use. By defining variables with `.declare`, there is no limit to the ways you can use the variable in your report. You can assign a value to an undeclared variable only by using a command line parameter or by entering a value in response to a runtime prompt. You cannot specify a value for an undeclared variable with a `.let` statement. Also, unless the variable has been declared, attempting to pass a parameter with a null value to Report-Writer can produce incorrect results.

You can specify as many variables as needed in a query by defining a unique name for each variable. If the same variable is to be substituted more than once within the query, specify the name, prefixed by a dollar sign (\$), at each place where substitution is to be done.

You can specify variables anywhere in the query—for example, within the column list for a retrieve statement or as a value to which the contents of a column is compared. For example, the following query phrases are legal:

```
... where e.name = "$employee_name" ...  
... retrieve (e.$var, ...) ...
```

When the variable is a character string used for comparison purposes in a where clause, enclose the variable within quotes that are appropriate for your query language (double quotes for QUEL), as shown in the preceding example.

If you actually want to include the dollar sign (\$) as a literal part of the query, precede it with a backslash (\). For example:

```
... where e.symbol = "\$" ...
```

You can also use variables specified in the .query statement in the body of the report. Report-Writer prints the value of the variable if the variable name is preceded by a dollar sign (\$). For more information, see the Population Example in the appendix "Report-Writer Report Examples."

## Examples

1. You set up the query as:

```
.query
    range of e is emp
    retrieve (e.all)
        where e.salary > $sal
        and e.dept = "$dept"
```

You invoke the command:

```
report mydb myrep (sal = 50000, dept = 'toy')
```

Enclose toy within single quotes to identify it to Report-Writer as a string. Report-Writer strips off these quotes before passing it to the query.

**Note:** If your operating system requires it, enclose the parenthetical clause within double quotes:

```
report mydb myrep "(sal = 50000, dept = 'toy')"
```

Report-Writer executes the following query:

```
range of e is emp
retrieve (e.all) where e.salary > 50000
    and e.dept = "toy"
```

2. Consider a table called, account, with columns including custno, custname, checking, and savings. You have separate fields for checking and savings accounts on one row, because most customers have both a savings and a checking account with the bank. To write one report specification that prints either the savings or checking account balances with a single query, you can code a .query statement similar to the following:

```
.declare account_type = c10 with prompt
    "Please enter the type of account:"
.quel
    range of a is account
    retrieve(a.custno, a.custname, val=a.$account_type)
```

You can invoke the query with the command:

```
report otherdb repname
```

At execution, Report-Writer issues the following prompt:

```
Please enter the type of account:
```



Suppose you respond with:

savings

The following query would be executed:

```
range of a is account  
retrieve(a.custno, a.custname, val=a.savings)
```

To print the value of the savings column, use:

```
.println val
```

This query selects values from the database savings column, not the string constant, savings.

## .Setup Statement—Embed SQL Statements that Perform Setup

The .setup statement embeds SQL statements that do not involve data retrieval into Report-Writer sections. Report-Writer executes the statements *before* it processes the main report query.

The .setup statement has the following format:

```
.setup SQL_statement; {SQL_statement;}
```

The parameter for the .setup statement is as follows:

### **SQL\_statement**

One or more action SQL statements that do not involve data retrieval, separated by semicolons (;). (Note that the select statement cannot be used.)

For a complete explanation of all available SQL statements, see the *SQL Reference Guide*.

## Description

Use the optional .setup statement to embed groups of SQL statements to perform tasks such as creating and qualifying temporary tables for use in the report and setting up lock modes. Report-Writer executes the .setup statement after substituting variables into the SQL statements. You can use as many lines as you need to specify the .setup. You can also include embedded declared variables within an SQL statement. The end of .setup is indicated by the start of a new Report-Writer statement. For information on embedding groups of SQL statements after the report is processed, see the .cleanup statement.

If you are running reports with an Ingres database and you are not updating the data in your report, we recommend that you set up your report to be read only. To do this, provide the following statement in the .setup section:

```
set lockmode session  
  where readlock=nolock;
```

**Note:** Keep in mind that when you set the lockmode to be read only, other users can still update the table used to produce the report. In this case, your report can not always match actual database values. If it is critical to keep the table consistent throughout the printing of the report, lock the table. For details on using lockmodes, see the *Database Administrator Guide*.

The following rules apply to the .setup section:

- The .setup section only supports SQL statements. The language of your .query section determines the query language of the report. If you have a QUEL .query section, the query language is QUEL; otherwise it is SQL. It is acceptable to have a QUEL query and SQL .setup and .cleanup sections.
- If the report query language is SQL, the default value of autocommit is off. If it is QUEL, the default is on. To override the default commit behavior, simply set autocommit off or on as the first statement in your .setup section.
- Only statements that are compatible with execute immediate are permitted in a .setup section. For a list of compatible statements, see the *SQL Reference Guide*. Neither the select statement nor any statement requiring embedded semicolons (;) or colons (:), such as create procedure, are allowed. Semicolons within quoted strings *are* allowed. Therefore, you can specify a table for selection by using the expression:  
  
create table *tablename* as select...
- In the .setup section, Report-Writer evaluates variables only once, before running the report. Therefore, you can set the value of the variable only at report initiation time as follows:
  - On the command line in a *variablename=valuestring* clause
  - On the command line in response to a prompt
  - In the with value or with prompt clause of a .declare statement associated with the variable
- In the .setup section, Report-Writer evaluates variables before sending them to the Database Management System (DBMS) and evaluates SQL statements at report runtime. It generates error messages at runtime from the DBMS. If it has detected no errors in the report or in the .cleanup and .setup sections and autocommit is off, Report-Writer executes an explicit commit at the close of report processing.
- Use the -d flag with the report command to run the report as if there were no errors. This flag causes Report-Writer to ignore DBMS errors in the .setup and .cleanup sections. The failure of the .setup statement, however, can affect data availability in the .query. For example, if you run the report on a temporary table created improperly in the .setup section, the report fails.
- If you specify the -d flag, error messages continue to display on the screen. Although it is not recommended, you can run the report with the -s flag if you do not want error messages to display.
- If you do not use the -d flag, errors and transaction handling follow these rules:
  - If there is an error in the .setup section, neither the report nor the .cleanup section runs. The transaction is rolled back if autocommit is off.

- If there is an error in the .cleanup section, the .setup section (if it existed) and the report have already run. The transaction is rolled back if autocommit is off.
- If there is an error in the query or a fatal error in the report, neither the .cleanup section nor the report runs. The transaction is rolled back if autocommit is off.

### Example

```
.NAME books2
.OUTPUT books2.out
.LONGREMARK
    The BOOKS2 report demonstrates using setup
    and cleanup to produce temporary tables.
.ENDREMARK
.SETUP
    set lockmode session where readlock=nolock;
    create view tempbooks as
        select b.id, b.title, a.name, '' as subject,
            1 as code
        from book b, author a
        where b.id = a.id
    union
        select b.id, b.title, '' as name, s.subject,
            2 as code
        from book b, subject s
        where b.id = s.id;
    create table subj_count as
        select id, count(subject) as num_sub
        from tempbooks
        where code = 2
        group by id;
    create table auth_count as
        select id, count(name) as num_auths
        from tempbooks
        where code = 1
        group by id;
.CLEANUP
    drop tempbooks;
    drop subj_count;
    drop auth_count;
.QUERY
    select b.id, b.title, b.name, b.subject, b.code,
        a.num_auths, s.num_sub
    from tempbooks b, subj_count s, auth_count a
    where b.id = a.id and b.id = s.id
```

## .Shortremark Statement—Specify Remark

The .shortremark statement specifies a one-line remark describing the report.

The .shortremark statement has the following format:

```
.shortremark | .srem remark_text
```

The parameter for the .shortremark statement is listed below:

### ***remark\_text***

A string of characters on the same line as the statement keyword.

## Description

The .shortremark statement is an optional statement that specifies a one-line description of the report. You can use this short description to help document your report specifications, but its primary purpose is to provide information that appears on the Catalog and Save frames of RBF.

There can only be one .shortremark statement in a program. The second .shortremark statement is flagged with a syntax error. Only the first 60 characters of the descriptive text are stored in the database. All characters past 60 are ignored.

## Examples

Here are two examples of the .shortremark statement:

```
.shortremark Monthly Accounts Receivables
```

```
.srem customized emp & dept report tables
```

## .Sort Statement—Specify Sort Order of Reported Data

The .sort statement specifies the ordering of rows to be reported.

The .sort statement has the following format:

```
.sort | .srt {columnname[:sortorder] [, columnname[:sortorder]]}}
```

The parameters for the .sort statement are as follows:

### ***columnname***

The name of a column in the table to be reported, or the label for a column in the result column list of the specified query. You can express *columnname* as a delimited identifier by enclosing it in double quotes ("), if you have previously specified the .delimid statement.

The *columnname* can be expressed as a variable. The colon (:) and the comma (,) must be explicitly stated and cannot be part of the *sortorder* variable.

### ***sortorder***

Either ascending (also a or asc or ascend) or descending (also d or desc or descend), depending on how you want the rows to be ordered. If neither is specified, the default is ascending.

The *sortorder* can be expressed as a variable. The colon (:) and the comma (,) must be explicitly stated and cannot be part of the *sortorder* variable.

### ***\$columnvariable***

The variable whose value is the name of a column. Precede the variable with a dollar sign (\$).

### ***\$sortorder***

The variable whose value is a sort order that evaluates to an acceptable sort direction. For more information, see the description of *sortorder* in this table. Precede the variable with a dollar sign (\$).

## Description

The optional .sort statement specifies the ordering that applies to the rows of data to be reported. Report-Writer initially sorts rows on the first column in the list. If several rows have the same value in the first sort column, Report-Writer then sorts them on the second column in the list, and so forth. If there is exactly one sort column, and there are duplicate values for the sort column, all rows with that value appears together, but in an undetermined order relative to each other.

By default, the `.sort` statement also specifies the columns used as break columns in the report. You can specify break headers and footers for each column, using the `.header` and `.footer` statements. A break on one column in the sort list produces a break on all subsequent columns in the sort list.

If you want, you can use the `.sort` statement to order rows for appearance in the report only, without specifying these columns as break columns. To override the default break specifications in the `.sort` statement, specify break columns in a `.break` statement.

An SQL language report specification includes duplicate rows in the data for reports that use the `.sort` statements. To specify distinct rows, you can specify the `-6` flag on the report command line.

If you specify variables as the *columnname* and/or *sortorder* parameters, Report-Writer evaluates these variables during the loading of the report specification, before retrieving the data.

You can have either a `.sort` statement or an order by clause in a `.query` statement but not both in a report specification.

**Note:** When using a variable for *columnname*, specify the same variable identically in corresponding `.break`, `.header`, and `.footer` statements.

## Examples

1. Sort two columns of a table (whose names are specified by delimited identifiers), with both columns in ascending order:

```
.sort "last name","first name"
```

2. Sort three columns of a table, with first and last columns in descending order, and the second column in ascending order.

```
.sort dept:descending,  
      jobcode,  
      name:d
```

3. Sort the column specified by the value of the variable *sort\_col* in the order specified by the value of the variable *sort\_order*. For example, the value of *sort\_col* might be *last\_name* and the value of *sort\_order* might be for ascending.

```
.sort $sort_col:$sort_order  
...  
.header $sort_col  
...  
.out \direct\subdirect\myreport.lis  
.include \direct\subdirect\otherrep.rw
```

## Page Layout and Control Statements

Page layout and control statements are used for specifying the layout of report pages.

### **.Formfeeds/.Noformfeeds—Force or Suppress Formfeed Characters**

The .formfeeds and .noformfeeds statements force and suppress the addition of formfeed characters to the end of each page written in the report.

These statements have the following format:

```
.formfeeds | .ffs | .ff  
.noformfeeds | .noffs | .noff
```

#### **Description**

For printers that support formfeeds, it is often most convenient to use the ASCII formfeed character to support pagination. You can use the .formfeeds and .noformfeeds statements to override the default setting for formfeeds, which is determined by the operating system.

The default for formfeeds is:

**Windows and UNIX:** .formfeeds

**VMS:** .noformfeeds

You specify the .formfeeds or .noformfeeds statement at the start of your report specification statements, before any .header or .footer statements.

Specify .formfeeds to override a .noformfeeds default. This forces Report-Writer to send formfeed characters at the start of the report and at the end of each page in the report. You can suppress the initial formfeed by specifying the .nofirstff statement in your report specification or by including the -nofirstff flag when using the report command.

If the report contains page footer formatting statements, Report-Writer sends the formfeed character after the page footer. Otherwise, Report-Writer sends the formfeed character after the last line of the page, as determined from the default or specified page size. For details on specifying page size, see the .Pagelength statement. When writing to a screen, Report-Writer ignores the .formfeeds statement.



Specify `.noformfeeds` to override a `.formfeeds` default and suppress formfeeds.

You can override these statements and the formfeed defaults at runtime by specifying the `-b|+b` flag on the report command line.

### Examples

1. To turn on formfeeds and override a `.noformfeeds` default, specify:  
`.formfeeds`
2. To turn off formfeeds and override a `.formfeeds` default, specify:  
`.noformfeeds`

## .Leftmargin Statement—Set a Left Margin

The .leftmargin statement sets a left margin for the report.

This statement has the following format:

```
.leftmargin | .lm [[+|-] n | expression]
```

The parameters for the .leftmargin statement are listed below:

**+ | -**

If sign is present, the new position is calculated relative to the current position. If no sign is present, it is set to the absolute position.

The optional plus sign (+) or minus sign (-) must be stated explicitly and cannot be a part of the *expression*.

***n***

The position of the new left margin of the report. The default value is discussed in Automatic Determination of Default Settings.

***expression***

A numeric expression that evaluates to the position of the new leftmargin of the report. All variables that are part of the expression must be preceded by a dollar sign (\$).

### Description

The .leftmargin statement sets the left margin of the report to a specific position or a position that is the value of a numeric expression. Subsequently, when new lines are written, new text output begins at the left margin position. To set the left margin for the entire report, place the statement in the .header report section. The .leftmargin value is also used by the .left and .center statements to determine the default position for those statements.

Report-Writer evaluates any expressions used as parameters to the .leftmargin statement during runtime.

**Note:** If you do not specify a left margin position, Report-Writer determines the default value for the left margin from your other specifications for the writing of the report, as discussed in Automatic Determination of Default Settings.

The value specified for the .leftmargin statement must be greater than or equal to zero (0), less than the specification for the right margin and less than the page width (as specified with the -l flag on the report command, or with the .pagewidth statement).

## Examples

1. Have new lines begin at position 5 (the sixth character position).  
`.lm 5`
2. Move the left margin to the right by the number of characters specified by the value of the variable *width*.  
`.lm +$width`
3. Set the left margin to be the sum of the value of variable *width* and 3.  
`.lm $width+3`

## .Need Statement—Keep Lines Together

The `.need` statement ensures that a specified number of text lines is printed together on the same page.

```
.need | .ne nlines | expression
```

The parameters for the `.need` statement are as follows:

### ***nlines***

The number of lines in the text block to remain together.

### ***expression***

A numeric expression that evaluates to the total number of lines in the text block to remain together. Precede all variables that are part of the expression by a dollar sign (\$).

## Description

The `.need` statement allows you to ensure that groups of text lines are not broken across pages of the report. Report-Writer performs conditional page breaks, to keep the specified or variable-specified lengths of text blocks together. You can use this statement to make sure that Report-Writer keeps all lines of text in the headers, and such, on the same page. For examples of placing the `.need` statement, see the appendix "Report-Writer Report Examples."

Report-Writer evaluates any expressions used as parameters to the `.need` statement during runtime.

## Examples

1. Make sure that the break header is on the same page.

```
.need 3
.print 'Header for account:',acct
.newline
.print '-----'
.newline 2
```

2. In this example, the report ensures the label information can be printed on varying sized label forms. The value of `label_size` is the number of lines per label. The printing of the company name is dependent on a record meeting the criteria of the condition.

```
.need $label_size
.if company != '' .then
    .println company
    .let address_lines = 4
.else
    .let address_lines = 3
.endif
.println name
.println address
.println city, state, zip
.newline $label_size - $address_line.
```

## .Newpage Statement—Insert Page Break

The .newpage statement causes an immediate page break, with an optional change in the page number.

This statement has the following format:

```
.newpage | .np [[+|-] pagenumber | expression]
```

The parameters for the .newpage statement are as follows:

**+ | -**

If a sign is present, the new position is calculated relative to the current position. If no sign is present, it is set to the specified value.

. The plus sign (+) or minus sign (-) must be stated explicitly and cannot be a part of the expression.

***pagenumber***

The page number to be assigned to the next page in the report.

***expression***

A numeric expression that evaluates to the page number to be assigned as the next page in the report. All variables that are part of the expression must be preceded by a dollar sign (\$).

### Description

You can place the .newpage statement at any point in your report specification. Report-Writer performs an immediate page break by skipping enough lines to get to the end of the page, then prints the page footer, sets the new page number, and writes a page header at the top of the new page. Report-Writer determines the new page number by incrementing the old page number, or by setting, incrementing, or decrementing the old page number to the specified value. If *pagenumber* is not specified, Report-Writer determines the default page number by incrementing the current page number by one.

At the end of the report, Report-Writer performs a .newpage statement automatically, if a page footer is specified (in this case, no page header appears on the next page). Also, if Report-Writer encounters a .newpage statement as the first printing action of the report, it does not print a page footer for that page.

Report-Writer evaluates any expressions used as parameters to the .newpage statement during runtime.

## Examples

1. Skip to a new page, incrementing the current page number by 1.  
`.newpage`
2. Reset the page number to 22.  
`.newpage 22`
3. Reset the page number to the current page plus the value of *skip\_page*.  
`.newpage +$skip_page`

## .Nofirstff Statement—Suppress Initial Formfeed

The `.nofirstff` statement suppresses the initial formfeed prior to the start of the report when formfeeds are enabled.

This statement has the following format:

```
.nofirstff
```

## Description

When ASCII formfeeds have been enabled by default, by the `.formfeeds` statement, or by the `+b` flag for the report command, Report-Writer generates an initial formfeed in addition to one at the end of each page of the report. The initial formfeed is useful for printers with continuous-form paper. To avoid generating this blank page at the beginning of your report, you can use the `.nofirstff` statement in your report specification to suppress the initial form feed.

You need only specify this statement once in your report source file, preferably immediately after any `.formfeed` statement and prior to all statements that generate printed text (such as `.print`, `.println`, `.newline`, and `.newpage`). This statement should not appear in any `.detail`, `.footer`, or `.header` section other than `.header` report.

If specified in an `.include` file, its placement in the file must follow the same placement rules as in the report specification source file.

You can omit this statement and use the `-nofirstff` flag on the report command line at runtime instead. You can also override the `.nofirstff` statement at report runtime by specifying the `-firstff` flag.

## Example

To suppress the initial formfeed when formfeeds have been explicitly enabled:

```
.formfeeds  
.nofirstff
```

## .Pagelength Statement—Set Page Length

The .pagelength statement sets a new default page length in number of lines per page.

This statement has the following format:

```
.pagelength | .pl {nlines | expression}
```

The parameters for the .pagelength statement are as follows:

### ***nlines***

The number of lines per page. The default is 61 lines per page if the report is written to a file, or the length of your screen if the report is written to the screen. To suppress all pagination (eliminate all page breaks), specify .pagelength as zero (0).

### ***expression***

A numeric expression that evaluates to the new page length of the report. Precede all variables that are part of the expression with a dollar sign (\$).

## Description

The .pagelength statement controls where page breaks occur. The report formatter subtracts the number of lines in the footer (specified in the .footer page section of the specifications) from the total page length of *nlines*. Page length is the total number of body text lines for the page.

During the running of the report, Report-Writer checks to see if the total count of body text lines has been written to the current page. If not, no action is taken. If so, Report-Writer writes a page footer and header, and the report continues. If the .formfeeds statement or formfeed default is in effect, Report-Writer performs a formfeed at the end of each page footer.

The value used in the .pagelength statement should be greater than the combined number of lines specified in the heading and footing for the page.

You can override the .pagelength statement at runtime by specifying the *-vpagelength* flag on the report command line. If both the *-v* flag and the .pagelength statement are present, the *-v* flag overrides the .pagelength value.

Report-Writer evaluates any expression used as a parameter to the .pagelength statement during runtime.



## Examples

1. Set a new page length for screens with 24 lines.  
`.pl 24`
2. Set the page length to be the value of variable *page\_size*.  
`.pl $page_size`
3. Suppress report pagination.  
`.pl 0`

## .Pagewidth Statement—Set Page Width

The .pagewidth statement sets a new default page width in number of characters or positions per line.

This statement has the following format:

```
.pagewidth | .pw width
```

The parameters for the .pagewidth statement are listed below:

### ***width***

The number of characters per line (*n* characters). The default is 132 characters per line if the report is written to a file, and the width of your screen if the report is written to the screen.

The *width* can be expressed as a variable.

### ***\$widthvariable***

Variable whose value is the report page width (*n* characters). Precede the variable with a dollar sign (\$).

## Description

The .pagewidth statement controls **where** line breaks occur. With the .pagewidth statement, you specify total number of characters specified per line through a constant or variable.

If you specify a variable for *width*, Report-Writer evaluates the variable during the loading of the report specification, before retrieving any data.

The value used in the .pagewidth statement must be greater than or equal to the width determined from the .leftmargin and .rightmargin values. The .rightmargin value must be greater than the .leftmargin value. If you are using a .position statement in your specification, its value must be less than the value of .pagewidth.

While running the report, Report-Writer checks to see if the total count of characters goes beyond the .pagewidth value. If so, an error message appears on the screen, or if possible, the report executes within the assigned width.

You can override this statement at runtime by specifying the -l flag on the report command line. Otherwise, the report reflects the .pagewidth setting. If there is no .pagewidth statement and the -l flag is not specified, Report-Writer uses 132 as the default width, and tries to execute the report within that width. If this is not possible, an error message appears on the screen.

All reports created and saved through RBF automatically generate a page width comment that RBF uses to set the page width when you run the report. Delete the page width comment from an archived RBF report specification and add an explicit `.pagewidth` statement instead. If you fail to delete the page width comment when specifying the page width explicitly, Report-Writer can truncate the text.

## Examples

1. Set a new page width for a printer.  
`.pw 132`
2. Set page width for a screen where the width is the value of variable *term\_width*.  
`.pw $term_width`

## .Rightmargin Statement—Set the Right Margin

The .rightmargin statement sets a right margin for the report.

This statement has the following format:

```
.rightmargin | .rm [[+|-] n | expression]
```

The parameters for the .rightmargin statement are as follows:

**+ | -**

If sign is present, the new position is calculated relative to the current position. If no sign is present, it is set to the absolute position *n*.

The plus sign (+) or minus sign (-) must be stated explicitly and cannot be a part of the expression.

***n***

The position of the new right margin of the report. If signed, the new position is calculated relative to the current position. If unsigned, it is set to absolute position *n*.

***expression***

A numeric expression that evaluates to the position of the new rightmargin of the report. All variables that are part of the expression must be preceded by a dollar sign (\$).

## Description

The .rightmargin statement sets the right margin of the report to a specific position or a position that is the value of a numeric expression. To set the right margin for the entire report, place the statement in the .header report section. The .rightmargin value is used by the .right and .center statements to determine the default position for those statements. If text would ordinarily go beyond the right margin, it is wrapped around to the start of the next line.

Report-Writer evaluates any expressions used as parameters to the .rightmargin statement during runtime.

If you do not specify a value, Report-Writer determines the default value for the right margin by the printing statements in the report, as discussed in Automatic Determination of Default Settings. The value specified for the .rightmargin statement must be greater than the specification for the left margin and less than the page width (as set by the -l flag on the report command).

## Examples

1. Specify margins such that the default position used by `.center` is 50.

```
.leftmargin 10
.rightmargin 90
...
.center
.print 'This title is centered on column 50'
```

2. Move the right margin left by the number of characters specified by the value of the variable *width*.

```
.rm $width
```

3. Set the right margin to be the sum of the value of variable *width* and 3.

```
.rm $width+3
.noformfeeds
```

## Report Structure Statements

Report structure statements are used to set up the structure of the report.

### .Detail Statement—Begin Data Formatting Statements

The `.detail` statement specifies the start of action to be taken when each data row is processed.

This statement has the following format:

```
.detail | .det
```

### Description

The `.detail` statement signifies the start of a group of report formatting statements that Report-Writer executes each time it processes a data row for the report. Report-Writer executes these formatting statements after any break headers that can be caused by the data row, but before any break footers.

Report-Writer also uses the formatting statements specified in the .detail block as the basis for determining the default margins of the report and the default positions of columns. For more information, see Automatic Determination of Default Settings.

```
.detail
.print acctnum(b16), tdate(b16),
.tab +8
.print transnum ('nnnn'), deposit, withdrawal
.tab +5
.print cum(acctnum), sum(amt.balance)
.newline
```

## .Footer Statement—Begin Formatting Statements for the Footer

The .footer statement identifies the start of a block of formatting statements that are executed at the end of a break.

This statement has the following format:

```
.footer | .footing | .foot  
report | page | columnname
```

The parameters for the .footer statement are as follows:

The following table shows the results obtained by specifying each of the options in the .footer statement:

### **report**

Prints the footer at the end of the report.

### **page**

Prints the footer at the bottom of each page.

### ***columnname***

Specifies a break column name in the list of the .sort or .break statement. Specify the column name as a delimited identifier by enclosing it in double quotes ("), if you have previously specified the .delimid statement.

Prints the footer at the end of a group of data rows with identical values for that break column. The *columnname* can be expressed as a variable.

### ***\$columnvariable***

Variable whose value is the name of a column. Precede the variable with a dollar sign (\$).

## Description

The .footer statement starts the block of text formatting statements that define the action to be taken at the end of a break in the report.

If you specify a variable for *columnname*, Report-Writer evaluates the variable during the loading of the report specification, before retrieving any data.

Report-Writer considers all statements between one .footer statement and any subsequent .header, .footer, or .detail statement to be part of the first footer action.

**Note:** When using a variable for *columnname*, specify the same variable identically in corresponding .break, .sort, and .header statements.

## Examples

1. Specify printing a page number at the bottom of each page

```
.footer page
.newline
.center
.print '-', page_number (f2), '-'
.newline
```

2. Print the sum of column named "wholesale cost" (a delimited identifier), at the end of the report

```
.footer report
.tab 10
.print 'TOTAL COSTS: ', sum("wholesale cost"),
      ('$-----.zz')
.newline
```



## .Header Statement—Begin Formatting Statements for the Header

The .header statement identifies the start of a block of text formatting statements that define the action to be taken at the start of a break in the report.

When you create a report header, put the .header report statement and the formatting statements associated with it *before* any other .header statements. Report-Writer considers all statements between one .header statement and any subsequent .header, .footer, or .detail statement as part of the first header action.

This statement has the following format:

```
.header | .heading | .head  
report | page | columnname
```

The parameters for the .header statement are as follows:

### **report**

Prints the header before the start of the report.

### **page**

Prints the header at the top of all pages except the first page.

### ***columnname***

A break column name specified in the .sort or .break statements. Specify the column name as a delimited identifier by enclosing it in double quotes ("), if you have previously specified the .delimid statement.

The *columnname* can be expressed as a variable.

Prints the header before a new value of a break column.

### ***\$columnvariable***

Variable whose value is a name of a column specified in a .sort or .break statement. Precede the variable must with a dollar sign (\$).

If you specify a variable for *columnname*, Report-Writer evaluates the variable during the loading of the report specification, before retrieving any data.

**Note:** When using a variable for *columnname*, specify the same variable identically in corresponding .break, .sort, and .footer statements.

## Examples

1. Specify a page header.

```
.header page
.tab 10
.print 'Accounts Receivable Aging
      Report by Client'
.newline
```

2. Specify a report header that prints the current date, the current time and the name of the report at the start of the report.

```
.header report
.leftmargin 10
.rightmargin 70
.left
.print current_date
.right
.print current_time
.newline 2
.underline
.center
.print 'Annual Costs'
.nounderline
.newline
```

3. Specify a break header for the column "last name."

```
.delimid
.query
    select "last name", "first name"
        from names_tbl
.sort "last name"
.header "last name"
.newpage
```

## Column and Block Statements

Column and block statements are used to set up an explicit print position, column width, and format for the values contained in the named database column or for a report *block* (as defined by a `.block` statement).

## .Block/.Endblock Statements—Enable/Disable Block Mode

The .block and .endblock statements enable and disable block mode, which lets you refer to positions on previous and subsequent lines in the report.

This statement has the following format:

```
.block | .blk  
      formatting statements  
.endblock | .endblk | .end block
```

### Description

The .block and .endblock statements switch Report-Writer into and out of block mode, allowing you to use advanced capabilities of Report-Writer in formatting complex reports. While in block mode, you can move not only across the page (through the .tab statement) and down the page (through the .newline statement), but also back up the page (through the .top statement).

Block mode gives you the capability of logically printing information in your report, and then putting summary information ahead of the detailed information. Do this by switching Report-Writer into block mode, printing out some number of lines, moving to the top of the block to add summary information, and then printing out the entire block by leaving block mode.

By using this statement in conjunction with the .within and .endwithin statements, described later in this chapter, you can describe column headings and subtotalling in a more natural and convenient fashion than is possible if you had to describe each line completely before going to the next line.

All formatting statements are allowed within block mode, except for the .newpage and .need statements. Additionally, you can use the .top and .bottom statements only while in block mode to move the current position within the block.

Report-Writer permits a default maximum of 310 explicit .newline statements within any one block, as protection against misspecified columns. You can override this default by setting the -wmxwrap parameter in the report command.

## Examples

1. Assume the following sequence of Report-Writer statements:

```
.block
    .print 'Line 1' .newline
    .print 'Line 2' .newline
    .top
    .tab 10 .pr 'more line 1' .newline
.endblock
```

You would get the following output:

```
Line 1 more line 1
Line 2
```

2. Assume the following sequence of Report-Writer statements:

```
.sort region, state
.header region
    .need 4
    .block
        .print 'Region: ', region .nl
    .detail
        .tab 5
        .print state(c15)
        .tab 30
        .print tot_18to65('n,nnn,nnn')
        .newline
    .footer region
        .top
        .lineend
        .tab + 5
        .print 'Count of states: '
        .println count(state) (f3)
    .end block
```

You would get the following output:

Region: East South Central	Count of states: 4
Alabama	2,528,938
Kentucky	2,971,232
Mississippi	1,393,283
Tennessee	3,283,432
Region: Mountain	Count of states: 8
Arizona	1,604,948
Colorado	2,112,352
Idaho	0,698,802
Montana	0,663,043
Nevada	0,448,177
New Mexico	0,915,815
Utah	1,031,926
Wyoming	0,323,024

## **.Bottom Statement—Make Current Line the Bottom Line**

The `.bottom` statement changes the current output line to the bottom line in the current block.

You can use the `.bottom` statement only while block mode is in effect (that is, after a `.block` statement, but before the corresponding `.endblock` statement). It moves the current output line to the current bottom line in the block. The character position on that line is one space beyond the last character printed on that line.

This statement has the following format:

```
.bottom | .bot
```

### **Example**

Assume the following sequence of Report-Writer statements:

```
.block
  .print 'Line 1' .newline
  .print 'Line 2' .newline
  .top
  .tab + 2 .pr 'more line 1' .newline
  .bottom .lineend
  .print 'Last line in block' .newline
.endblock
```

You would get the following output:

```
Line 1 more line 1
Line 2 Last line in block
```

## .Format Statement—Set Default Printing Format

The .format statement sets up a default printing format for a column or set of columns.

This statement has the following format:

```
.format | .fmt columnname{, columnname} (format)
      {, columnname{, columnname} {format}}
```

The parameters for the .format statement are as follows:

### **columnname**

The name of a column in the report. You can specify the column name as a delimited identifier by enclosing it in double quotes ("), if you have previously specified the .delimid statement.

The *columnname* can be expressed as a variable. The comma (,) and the parentheses ( ) must be stated explicitly and cannot be part of the *columnname* or *format* variable.

### **format**

A valid printing format, as described in Format Specifications. The format must be the correct type for the column(s).

The *format* can be expressed as a variable. The comma (,) and the parentheses ( ) must be stated explicitly and cannot be part of the *columnname* or *format* variable.

### **\$columnvariable**

Variable whose value is a name of a column. Precede the variable with a dollar sign (\$).

### **\$formatvariable**

The variable whose value is a name of a printing format described in Format Specifications. The format must evaluate to the correct type for the columns. Precede the variable with a dollar sign (\$).

## Description

The .format statement sets up a default format associated with a column to be used whenever Report-Writer prints the column or an aggregation of a column.

If you specify variables for the column and/or format, Report-Writer evaluates *columnvariable* and *formatvariable* during the loading of the report specification, before retrieving any report data.

Because the format for a column can determine the default width for that column (used in the `.center`, `.right`, and `.left` statements), you can use the `.format` statement to control the default width of a column. Report-Writer uses this statement to determine the default width *only in the absence* of the `.width` or `.position` statements, which also specify the default width for a column. You can use `.tformat` to override the format of a column during the run of the report.

If you do not specify a `.format` statement for a column, Report-Writer determines the print format by scanning the printing statements in the report. If the printing statements do not specify a format, Report-Writer determines the default.

By default, breaks use the formatted values for any column whose format has been specified in a `.format` statement. That is, a break occurs for such a column only when the formatted value changes. Otherwise, no break occurs, even if the unformatted value changes. To force Report-Writer to use the actual, rather than the formatted, values to determine breaks, specify the `-t` flag on the report command line. For more information, see the *Command Reference Guide*.

## Examples

1. This example shows a `.format` statement that declares formats for several columns, followed by a `.print` statement that uses the formats specified in the `.format` statement to print the information.

```
.format trans, balance ('$$$,$$$,$$$,nn')
.print trans,balance
.newline
```

2. This example shows a `.format` statement that declares formats for date columns where `date_fmt` evaluates to a date format and column names are delimited identifiers.

```
.format "trans date", "cur date" ($date_fmt)
```

## .Position Statement—Set Position and Width of Column Output

The `.position` statement sets a default output position and optional width associated with a column.

This statement has the following format:

```
.position | .pos columnname {, columnname} (position [,width])  
      {, columnname{, columnname} (position [, width])}
```

The parameters for the `.position` statement are as follows:

### ***columnname***

Name of a column in the report. You can specify the column name as a delimited identifier by enclosing it in double quotes (""), if you have previously specified the `.delimid` statement.

### ***position***

Numeric location on the output line where the default column position should be. This value must be less than the maximum line size (as set by the `.pagewidth` statement in the report specification or by the `-l` flag on the report command line) and greater than or equal to zero (0).

### ***width***

The default width which is the total number of characters in the column to be used when calculating the positioning for `.center` and `.right` statements. If not specified, Report-Writer determines this numeric by looking at the default format for the column.

The *columnname*, *position*, and *width* can be expressed as variables. The comma (,) and parentheses ( ) must be explicitly stated and cannot be part of the *columnname*, *position*, or *width* variable.

### ***\$columnvariable***

Variable whose value is the name for a column in the report. Precede the variable with a dollar sign (\$).

### ***\$positionvariable***

Variable whose value is the numeric location on the output line where the default column position should be. This variable must evaluate to a position less than the maximum line size (as set by the `.pagewidth` statement in the report specification or by the `-l` flag on the report command line) and greater than or equal to zero (0).

### ***\$widthvariable***

Variable whose value is the default width, which is the total number of characters in the column to be used when calculating the positioning for `.center` and `.right` statements. If not specified or evaluated, Report-Writer determines this value by looking at the default format for this column.



## Description

The `.position` statement sets a default position in the output line associated with a column name for use with statements such as:

```
.left
.right
.center
.tab
```

You can also use this statement to set an optional default width (the total number of characters in a column) when calculating positions in the `.center` and `.right` statements.

If you specify a variable for *columnname*, *position*, and/or *width*, Report-Writer evaluates the variable during the loading of the report specification, before retrieving any report data.

Normally, you do not need this statement because Report-Writer determines default positions and widths by analyzing the report formatting statements. If the determined default position for a column is not convenient, or you would like a different position associated with a *columnname*, you can override the default with the `.position` statement. Subsequently, you can use the `.tab`, `.right`, `.left`, or `.center` statements with a *columnname* to refer to this position.

If you do not specify a `.position` statement for a column, and *columnname* is not printed in the report, the default position is zero (0). If you specify a position, but no width is specified or evaluated for a column, Report-Writer determines the default width by looking at the default format for the column. You can optionally use the `.width` statement to specify the width of a column.

## Examples

1. The following example sets up default positions for columns, and prints out the data. The column "acct num" is a delimited identifier.

```
.position "acct num"(5), transact(20), balance(35)
.format transact, balance ('$,$$$,$$$.$nn')
.format "acct num"('nn-nnnnn-n')
...
.tab acct .print acct
.tab transact .print transact
.tab balance .print balance
.newline
```

This results in a printout like this:

```
01-02234-4      $1,345.24      $11,429.32
02-41989-1       $876.24       $10,553.08
```

2. To vary the column positions and widths, use variables:

```
.position "acct num"($acct_pos, $acct_width),
      transact ($transact_pos, $transact_width),
      balance ($balance_pos, $balance_width)
```

## .Tformat Statement—Change Format of Column Output Temporarily

The .tformat statement changes the format temporarily for the output of a column.

This statement has the following format:

```
.tformat | .tfmt columnname{, columnname} (format)
{, columnname{, columnname} {format}}
```

The parameters for the .tformat statement are as follows:

### **columnname**

The name of a column in the report. You can specify the column name as a delimited identifier by enclosing it in double quotes ("), if you have previously specified the .delimid statement.

### **format**

A valid printing format, as described in Format Specifications. The format must be the correct type for the column(s).

The *columnname* and *format* can be expressed as variables. The comma (,) and the parentheses () must be stated explicitly and cannot be part of the *columnname* or *format* variables.

### **\$columnvariable**

Variable whose value is a name of a column in the report. Precede the variable with a dollar sign (\$).

### **\$formatvariable**

Variable whose value is a name of a printing format described in Format Specifications. The format must evaluate to the correct type for the column(s). Precede the variable with a dollar sign (\$).

## Description

The .tformat statement temporarily changes the format used to print out the value of a column. After Report-Writer prints the column using this format, the effect of the temporary format is discarded, and when Report-Writer prints the next value in the column, it uses the default format.

For example, reports often include columns containing currency data. If you want to print a leading dollar sign for the currency figure only the first time it appears on a page, you could use the .tformat statement to specify a format of \$\$\$,\$\$\$,\$\$n.nn for the column in the header action for page breaks. If the normal format for printing the column is zzz,zzz,zzn.nn, then Report-Writer prints the column with a leading dollar sign only the first time it is printed on each page.

Another common use of the `.tformat` statement is for blanking out the unchanged values of break columns in the detail action for a report. You use the B type format (described in the Blanking Format B section of the chapter "Report-Writer Expressions and Formats") to accomplish this. Specifying a B format, with an appropriate field width as the standard format for printing a column in the detail section, causes Report-Writer to print blanks instead of the value of that column as the default action.

To ensure that Report-Writer prints each *new* value in the column, specify a printing format in a `.tformat` statement in the break header for that column. Refer to the use of the `.tformat` statement for the date column in Account Examples or the examples in this section.

If you specify a variable for *columnname*, *position*, and/or *width*, Report-Writer evaluates the variable during the loading of the report specification, before retrieving any report data.

## Examples

1. Print out the value of a break column only when it changes. The acct num column is a delimited identifier.

```
/*
** In the detail section, blank out the
** account number. When the account number
** changes, print it.
*/
.header report
    .format "acct num"(b10),
        transact('$$$,$$$,$$$.$nn')
.heading "acct num"
    .tformat "acct num"(c10)
.detail
    .print "acct num"
    .tab +2
    .print transact
    .newline
```

This is the sample report output for the above specification:

```
01-34567-8      $345.21
                $14.10
                $1,143.23
04-35999-2      $1.99
                $177.00
```

2. Print dollar sign at top of page only. The variable *thous\_dollar* evaluates to '\$\$\$,\$\$n' and *thous* evaluates to 'ZZZ,ZZn':

```
.declare thous_dollar = c8 with value '\ '$$$,$$n\'' ,
        thous_dollar = c8 with value '\ 'zzz,zzn\''
.header page
    .print 'Top of page' .nl 2,
        .tformat salary($thous_dollar)
.detail
    .print name(c14), salary($thous)
    ...
```

This is the sample report output for the above specification:

```
Top of page
Jones, A.   $23,145
Jones, B.   16,145
Jost, C.    32,143
```

## .Top Statement—Make Current Line the Top Line

The .top statement changes the current output line to the top line in the current block.

This statement has the following format:

```
.top | .tp
```

### Description

You can use the .top statement only while block mode is in effect (that is, after a .block statement, but before the corresponding .endblock statement). It moves the current output line to the first (topmost) line in the block.

The character position on the topmost line is the same as its previous position on the line when the last .newline statement affected the topmost line. To get to the left margin of the top line, use the .tab statement with no parameters. To get to the last nonblank character on the line, use the .lineend statement.

### Example

Assume the following sequence of Report-Writer statements:

```
.block
    .print 'Line 1' .newline
    .print 'Line 2' .newline
    .top
    .tab + 2 .pr 'more line 1' .newline
.endblock
```

You would get the following output:

```
Line 1  more line 1
Line 2
```

## .Width Statement—Set Output Width of a Column

The .width statement sets a default output width associated with a column.

This statement has the following format:

```
.width | .wid columnname{, columnname} (width)  
      {, columnname}{, columnname} (width)}
```

The parameters for the .width statement are as follows:

### ***columnname***

The name of a column in the report. You can specify the column name as a delimited identifier by enclosing it in double quotes ("), if you have previously specified the .delimid statement.

### ***width***

The width, which is the total number of characters in the column to be used when calculating the positioning for the .center and .right statements. If not specified, Report-Writer determines this numeric by looking at the default format for the column.

The *columnname* and *width* can be expressed as variables. The comma (,) and parentheses ( ) must be explicitly stated and cannot be part of the *columnname* or *width* variable.

### ***\$columnvariable***

Variable whose value is the name for a column in the report. Precede the variable with a dollar sign (\$).

### ***\$widthvariable***

Variable whose value is the default width, which is the total number of characters in the column to be used when calculating the positioning for .center and .right statements. If not specified or evaluated, Report-Writer determines this value is determined by looking at the default format for this column.

## Description

The .width sets the default width, which is the total number of characters in a column when calculating positions in the .center and .right statements. Alternatively, you can specify the default width for a column as a parameter to the .position statement. If you specify variables for *columnname* and *width*, Report-Writer evaluates the variables during the loading of the report specification, before retrieving any report data.

Normally, you do not need the .width statement because Report-Writer determines default widths by analyzing the report formatting statements, as described in Automatic Determination of Default Settings.

If the determined default width for a column is not convenient, or you would like a different width associated with a *columnname*, you can override the default with the `.width` statement. Subsequently, you can use the `.right` or `.center` statements with a *columnname* to use this width, in conjunction with the default position for this column, in calculating the placement of text.

If no width is specified for a column, Report-Writer determines the default width by looking at the default format for the column.

## Example

To print the following columns of salaries, set up desired position, formats and widths:

New Salary	Old Salary
\$50,000.00	\$45,000.00
\$32,000.00	\$28,800.00
\$35,000.00	\$31,500.00
\$100.00	\$90.00
\$35,000.00	\$31,500.00
\$25,000.00	\$22,500.00
\$5,000.00	\$4,500.00

```
.declare salary_fmt = c14 with value '$$,,$$.nn'
.declare salary_wid = integer with value '12'
.header report
  .position sal1(0), sal2(13)
  .format sal1, sal2 ($salary_fmt)
  .width sal1, sal2 ($salary_wid),
  .underline
  .center sal1 .print 'New Salary'
  .center sal2 .print 'Old Salary'
  .newline
  .nounderline
.detail
  .left sal1 .print '|'
  .right sal1 .print sal1
  .print ' |'
  .right sal2 .print sal2
  .print ' |'
  .newline
```

## .Within/.Endwithin Statements—Enable/Disable Column Formatting Mode

The .within and .endwithin statements enable and disable column formatting mode in Report-Writer.

This statement has the following format:

```
.within | .wi columnname{, columnname} | all  
    other formatting statements  
.endwithin | .endwi | .end within
```

The parameters for the .within and .endwithin statements are as follows:

### ***columnname***

The name of a column in the report within which other formatting statements are to be used. You can specify the column name as a delimited identifier by enclosing it in double quotes ("), if you have previously specified the .delimid statement.

### ***all***

Indicates that all columns in the report are to be used.

The *columnname* or all can be expressed as a variable. The comma (,) must be stated explicitly and cannot be part of *columnname* variable.

### ***\$columnvariable***

Variable whose value is a column name or all. Precede the variable with a dollar sign (\$).

## Description

The .within and .endwithin statements switch Report-Writer into and out of column formatting mode. In column formatting mode, Report-Writer temporarily sets the margins of the report to the left and right margins for a given column, determined either by default (as described in Automatic Determination of Default Settings) or through the use of the .position, .width, and .format statements.

If you specify a variable as *columnname*, Report-Writer evaluates the variable during the loading of the report specification, before retrieving data.

Report-Writer processes all statements between the .within and the corresponding .endwithin statement using the margins for the specified column, rather than the margins for the report. If more than one column is specified or evaluated on the .within statement, or if the keyword all is used, Report-Writer applies the set of statements to each of the columns in turn.

When using the `.within` and `.endwithin` block of statements for a set of columns, you can invoke a slightly different set of formatting statements within each column, differing only in the column referenced by a formatting statement. To accomplish this, two special names are available for use in formatting statements while in column formatting mode. You can use them to refer to the column that is currently being used:

### **w\_column**

Can be used anywhere *columnname* would normally be used on a formatting statement, such as in `.print w_column` or `.print sum(w_column)`.

### **w\_name**

Refers to the name of the column currently being used in the `within` block. You can use it to print out the actual column names.

When used in a `.within` block, these special names can also be values of variables where appropriate. For instance, the value of the variable *colname* is `w_name` in the following example:

```
.tab $colname
```

In another example, the value of the variable *colval* is `w_column`:

```
sum($col)
```

For examples of the use of these special names, see the examples in this section or the appendix "Report-Writer Report Examples."

Because Report-Writer temporarily changes the margins of the report to the margins for a column while the `.within` statement is in effect, the positions referred to by the default values for the `.left`, `.right`, and `.center` statements are those of the column, rather than the full width of the report.

If the text you want to print spans more than one line per column in the `.within` section—that is, the text has more than one `.newline` command—then surround the `.within` section with the `.block` and `.endblock` statements. Report-Writer automatically executes a `.top` statement immediately before the `.endwithin` statement to simplify this type of specification.

Once you start to use the `.within` and `.endwithin` statements, you can find that the `.position`, `.width`, and `.format` statements take on additional usefulness.



## Examples

1. Assume the following sequence of Report-Writer statements, where two of the column names are delimited identifiers:

```
.position "last name"(0), "first name"(16), address(32)
    ...
.within "last name", "first name", address
    .pr w_name
.end within
```

Report-Writer prints the names of the three columns similar to this:

```
last name      first name      address
```

2. In the following example, the .within statements allow Report-Writer to print each column of the report in a similar format.

In the report header and footer section, a .block statement allows the centered titles and sums to use more than one line. In the detail section, only one line is used so a .block statement is not necessary. In the detail and the footer section for the report, the special name w\_column is used.

Total	
Population	18 to 65
11,113,976	9,600,381
5,193,669	4,820,324
8,875,083	7,833,474
10,652,017	9,646,997
712,567	698,802
694,409	663,043
19,953,134	17,761,032
[detail omitted]	
-----	-----
203,165,702	177,612,309

The following report example was used to create the two columns:

```
.name withinex
.query select tot_18to65 + tot_under18 + tot_over65
      as totpot, tot_18to65
      from pop
.header report
.position totpot (2,15), tot_18to65(20,15)
.format totpop, tot_18to65 ('zzz,nnn,nnn')
.block
  .within totpop
    .right .println 'Total'
    .underline
    .right .println 'Population'
    .nounderline
  .endwithin
  .within tot_18to65
    .newline
    .underline
    .right
    .println '18 to 65'
    .nounderline
  .endwithin
.endblock
.detail
  .within totpop, tot_18to65
    .right .println w_column
  .endwithin
.footer report
.block
  .within totpop, tot_18to65
    .right .println '-----'
    .right .print sum(w_column)
  .endwithin
.endblock
```

## Text Positioning Statements

Text positioning statements are used to specify a print position—absolute or relative to other positions—for any text to be printed. Most of these statements also accept as a value the name of a column for which a print position or column width has been set with the `.position` or `.width` statements.

## .Center Statement—Center the Text

The .center statement centers the next text to be printed.

This statement has the following format:

```
.center | .cen | .ce [[+|-] n | columnname | expression]
```

The parameters for the .center statement are listed below:

**+ | -**

If sign is present, the position is moved *n* positions relative to the last output position. If unsigned, the position is the absolute position in the output line.

***n***

The position around which the next block of text is centered. The default value is the halfway point between the left and right margins of the report.

### ***columnname***

The name of a column in the report. You can specify the column name as a delimited identifier by enclosing it in double quotes ("), if you have previously specified the .delimid statement.

Report-Writer determines the position for the column either explicitly through the use of the .position statement, or implicitly as described in Automatic Determination of Default Settings. Report-Writer positions the text in the next .print statement around the center for the column. For more information, see the discussion following this table.

### ***expression***

A numeric or string expression. If the expression is numeric, it must evaluate to the position around which the next block of text is centered. If the expression is a string, it must evaluate to the name of a column in the report. Precede all variables that are part of the expression with a dollar sign (\$).

The plus sign (+) and minus sign (-) must be explicitly stated and cannot be part of the expression.

## Description

The .center statement centers the text printed in the next .print statement. Report-Writer removes all leading and trailing blanks from the text before it places it in the output line.

Report-Writer evaluates any expressions that are used as parameters to the .center statement during runtime.

If you specify *n* (either relative or absolute), Report-Writer centers the text around that position. If you specify nothing, Report-Writer calculates the center of the page as the halfway point between the left and right margins of the report. If you specify `.leftmargin` and `.rightmargin` statements, you can calculate the center by the same method. However, if you are using the default values for the right and left margins (the right in particular), see Automatic Determination of Default Settings in the chapter "Using Report-Writer" for a discussion of how Report-Writer determines the margins.

If you specify centering with the *columnname* parameter or with an expression that evaluates to a *columnname*, Report-Writer centers the text in that column. Report-Writer determines the center of the column through both of the following:

- Default position of the column, as determined by the `.position` statement or by default
- Width of the column, as determined by default or by the width of the format specified in the `.format` statement, or as specified in the `.width` or the `.position` statements

The `.center` statement centers text around a position calculated as:

$$\text{centering position} = \text{default column position} + (\text{default format width} / 2)$$

Report-Writer rounds the position to the next highest number if there is any fraction.

The `.center` statement has a somewhat different meaning when executed in column formatting mode (that is, inside a `.within` statement with default column widths and positions assumed). Because the `.within` statement temporarily resets the report margins to the left and right margins of a specified column's width and position, a `.center` statement so executed centers a text string within the column width, not within the report page margins. For more information about column formatting mode, see the `.within/.endwithin` statements.

## Examples

1. Output the date centered on the page.

```
.center
.print 'Report Executed On:', current_date
```

2. Output a heading for a column centered above the value of that column, where the column is the delimited identifier, bank balance.

```
.position "bank balance" (20)
.format "bank balance"('+++,...nn')
...
.center "bank balance" .print 'Balance'
...
.detail
...
.tab "bank balance" .print bal ...
```

## .Left Statement—Left Justify the Text

The .left statement left justifies the next text to be printed.

This statement has the following format:

```
.left | .lft [[+|-] n | columnname | expression]
```

The parameters for the .left statement are as follows:

**+ | -**

Optional sign that moves the output position *n* positions relative to the last position output. If sign is not present, the position is the absolute position in the output line.

***n***

Position to which the next text is left justified. The default value is the left margin of the report (set by the .leftmargin statement).

***columnname***

Name of a column in the report. You can specify the column name as a delimited identifier by enclosing it in double quotes ("), if you previously specified the .delimid statement.

The position for the column is determined either explicitly through the use of the .position statement, or implicitly as described in Automatic Determination of Default Settings. If *columnname* is specified, Report-Writer left justifies the next output text and places it at the position associated with the named column.

***expression***

Numeric or string expression. If the expression is numeric, it must evaluate to the next print position on the line. If the expression is a string, it must evaluate to the name of a column in the report. Precede all variables that are part of the expression with a dollar sign (\$).

The plus sign (+) and minus sign (-) must be explicitly stated and cannot be part of the expression.

## Description

The .left statement left justifies the text printed in the next .print statement to one of the following locations:

- Specified position
- Position corresponding to a specified column
- Value of a numeric expression that evaluates to a specified position
- Value of a string expression or variable that evaluates to a column name

Report-Writer evaluates any expressions that are used as parameters to the `.left` statement during runtime.

You can specify all the values for these parameters as either absolute or relative to the last output position. Report-Writer removes all leading and trailing blanks from the text before placing the text in the output line.

The `.left` statement is the same as the `.tab` statement for all output except for text that contains leading blanks, such as formatted numbers.

The meaning of the `.left` statement is slightly changed when executed in column formatting mode (that is, when the `.within` statement is in effect and default column widths and positions are assumed). When executed under these circumstances, the `.left` statement positions text at the left margin of the column indicated in the `.within` statement. For more information about column formatting mode, see the `.within/.endwithin` statements.

### Example

Output the title as centered and the value of balance as left justified to position 40.

```
.position balance (40, 10)
. . .
.center balance
.print 'Balance'
. . .
.detail
  .left balance
  .print balance (f10.2)
```

## **.Lineend Statement—End a Line**

The .lineend statement begins the next text to be printed following the last non-blank character on the current line.

This statement has the following format:

```
.lineend | .lnend
```

### **Description**

The .lineend statement changes the current position in the output line so that Report-Writer places the text in the next .print statement immediately after the last non-blank character on the line. This is useful in some advanced reports that use the .tab statement extensively. The .lineend statement always moves the current position marker to a position within the current margins of the report.

### **Example**

To print a list of items across the page on the first line of a block:

```
.block
  .println 'ITEMS:'
  ...
.detail
  .top
  .lineend
  .println ', ', item
```



## **.Linestart Statement—Print Next Text on New Line**

The .linestart statement begins the next text to be printed at the current left margin.

This statement has the following format:

```
.linestart | .lnstart | .linebegin
```

### **Description**

The .linestart statement changes the position of the current marker for the output line so that the next text printed by the .print statement appears at the current left margin. The left margin is set either by the .lm statement, by default, or by the left edge of the column currently in use while in a .within block. The .linestart statement is useful in reports that use the .tab statement extensively. The .linestart statement always restores the current position marker to a known position, at the beginning of the line.

### **Example**

With .linestart you can return to the beginning of the current line.

```
.center  
.print 'Accounts Receivable for', dept_name  
.linestart  
.print '(', dept_code, ')'
```

## .Newline Statement—Advance to a New Line

The `.newline` statement writes out the current line and optionally advances a number of lines on the output page.

This statement has the following format:

```
.newline | .nl [nlines | expression]
```

The parameters for the `.newline` statement are as follows:

### ***nlines***

The number of lines to advance. If you are advancing to the next line, you can specify the `.newline` statement without the *nlines* parameter.

### ***expression***

A numeric expression which evaluates to the number of lines to advance from the current line. Precede all variables that are part of the expression with a dollar sign (\$).

## Description

The `.newline` statement must be specified to advance to a new line on the output page. Unlike some programming languages (for example, Fortran), a `.print` statement does not imply a new line at its completion. However, you can use the `.println` statement for this purpose. If you do not specify *nlines* or a numeric expression that evaluates to *nlines*, the default value of *nlines* is one (advance to the next line).

After Report-Writer executes `.newline`, it begins the next text output at the left margin, unless another text positioning statement overrides the default.

Report-Writer evaluates any expressions that are used as parameters to the `.newline` statement during runtime.

If the output of a new line reaches the end of the current page, or if there are fewer than *nlines* left on the current page, Report-Writer prints the page footer and page header, if they have been specified.

If the current line includes multi-line format strings, (Cn.w), the `.newline` statement advances to the bottom of the longest column printed during the formation of the line. For the Dictionary Example in the appendix "Report-Writer Report Examples," the `.newline` statement in the footer for "word" causes an advance to the line following the end of the definition.

When you invoke column formatting mode, `.newline` causes an advance to the next line at the left margin, as determined by the `.within` statement. For more information on column formatting mode, see the `.within/.endwithin` statements.

## Examples

1. Print out one line of text:

```
.print 'This is a line'  
.newline
```

2. Print out a variable number of newlines:

```
.println name  
.println address  
.println city, state, zip  
.nl $label_size - 3
```

## .Right Statement—Right Justify the Text

The .right statement right justifies the next text to be printed.

This statement has the following format:

```
.right | .rt [[+|-] n | columnname | expression]
```

The parameters for the .right statement are as follows:

**+ | -**

If sign is present, the position is moved *n* positions relative to the last output position. If no sign is present, the position is the absolute position in the output line.

***n***

The position to which the next block of text is right justified. The default value is the right margin of the report (set by the .rightmargin statement).

***columnname***

The name of a column in the report. You can specify the column name as a delimited identifier by enclosing it within double quotes ("), if you have previously specified the .delimid statement.

Report-Writer determines the column's position either explicitly through the use of the .position statement, or implicitly as described in Automatic Determination of Default Settings. Report-Writer right justifies the text in the next .print statement to the right edge of that column, as determined from the default position and width of that column. For more information, see the discussion following this table.

***expression***

A numeric or string expression. If the expression is numeric, it must evaluate to the next print position on the line. If the expression is a string, it must evaluate to the name of a column in the report. Precede all variables that are part of the expression with a dollar sign (\$).

The plus sign (+) and minus sign (-) must be explicitly stated and cannot be part of the expression.

## Description

The .right statement right justifies the text printed in the next .print statement to one of the following locations:

- Specified position
- Position corresponding to a specified column
- Value of a numeric expression that evaluates to a specified position
- Value of a string expression or variable that evaluates to a column name

Report-Writer evaluates any expressions that are used as parameters to the `.right` statement during runtime.

You can specify all the values for these parameters as either absolute or relative to the last output. Report-Writer removes all leading and trailing blanks from the text before placing the text in the output line.

If you specify *n* or a numeric expression that evaluates to *n* (either relative or absolute), Report-Writer right justifies the text to that position. If you specify nothing, Report-Writer right justifies the text to the right margin of the report. Report-Writer determines the right margin either from the `.rightmargin` statement, if specified, or by default as described in Automatic Determination of Default Settings.

If you specify right justification with the *columnname* parameter or through a string expression that evaluates to a *columnname*, Report-Writer right justifies the text to the right edge of that column, as determined from the following:

- Default position of the column, as determined from the `.position` statement or by default
- Width of the column, as determined by the default width or the width of the format specified in a `.format` statement for that column, or by the width specified in `.width` or the `.position` statements

The `.right` statement justifies to a position calculated as:

justification position =  
default column position + default width

The meaning of the `.right` statement is slightly changed when the `.right` statement is executed within column formatting mode (that is, when the `.within` statement is in effect and default column widths and positions are assumed). When the `.right` statement is so executed without a parameter, the current position becomes the right margin as defined by the `.within` statement, not the right margin of the report. For more information about column formatting mode, see the `.within/.endwithin` statements.

## Examples

1. Output a page number, right justified on the right margin.

```
.right  
.print 'Page ', page_number('zn')
```

2. Output a heading for column "bal," right justified to the right edge of the column.

```
.position bal (20)  
.format bal('+++,.nn')  
.right bal .print 'Balance'  
...  
.detail  
...  
.tab bal .print bal ...  
...
```

## .Tab Statement—Specify Tab Position

The .tab statement specifies the position on the line where the next text is printed.

This statement has the following format:

```
.tab | .tb | .t [[+|-] n | columnname | expression]
```

The parameters for the .tab statement are listed below:

### **+ or -**

If sign is present, the new position or column is calculated relative to the current position. If sign is not present, it is set to the evaluated position *n*.

### ***columnname***

The name of a column in the report. You can specify the column name as a delimited identifier by enclosing it within double quotes ("), if you have previously specified the .delimid statement.

Report-Writer determines the position for the column either explicitly through the use of the .position statement, or implicitly as described in Automatic Determination of Default Settings. If *columnname* is specified, Report-Writer begins the next output text at the position associated with the named column.

### ***expression***

A numeric or string expression. If the expression is numeric, it must evaluate to the next print position on the line. If the expression is a string, it must evaluate to the name of a column in the report. Precede all variables that are part of the expression with a dollar sign (\$).

The plus sign (+) and minus sign (-) must be explicitly stated and cannot be part of the expression.

## Description

The .tab statement moves the current position marker to one of the following locations:

- Specified position
- Position corresponding to a specified column
- Value of a numeric expression that evaluates to a specified position
- Value of a string expression or variable that evaluates to a column name

Report-Writer evaluates any expressions that are used as parameters to the .tab statement during runtime.

You can specify numeric values for these parameters as either absolute or relative to the last output position.

If you do not follow the `.tab` statement with *n*, a *columnname*, or an *expression*, then the `.tab` statement works like a `.linestart` statement, with the next text beginning at the left margin of the report. The `.linestart` statement is described further in this chapter.

The `.tab` statement takes on a slightly different meaning when executed in the column formatting mode sections of a report (that is, when the `.within` statement is in effect and default column widths and positions are assumed). When the `.tab` statement is executed without a parameter in column formatting mode, Report-Writer moves the current position to the left margin of the current line. The left margin is determined by the `.within` statement.

For more details on column formatting mode, see the `.within/.endwithin` statements.

## Examples

1. To output "HERE" in character position 12 on a line, use:

```
.tab 12
.print 'HERE'
```

2. To output Summary in column position `$title_col`, use:

```
.tab $title_col
.println 'Summary'
```

3. To output the value of the daily balance column (a delimited identifier) in position 30, use:

```
.position "daily balance"(30)
...
.tab "daily balance"
.print "daily balance"('+++++.NN')
```

4. To print two columns of figures separated by a bar:

```
.tab credit_sum
.right salary1
.print salary1, ' | '
.right salary2
.print salary2
```

5. To tab forward the amount specified by the value of `$col_width`, use:

```
.tab +$col_width
.print
```

## Print Statements

Print statements are used to print text or data values in a report.



## .Nullstring Statement—Specify String for Null Value

The .nullstring statement specifies an alternate null string.

This statement has the following format:

```
.nullstring | .nullstr 'null_string' | expression
```

The parameters for the .nullstring statement are as follows:

### ***null\_string***

Any string of characters. Enclose the string in single quotes, so Report-Writer can properly handle leading and trailing blanks, which are important in some format specifications.

### ***expression***

A string expression that evaluates to any character string. Precede all variables that are part of the expression with a dollar sign (\$). For more information, see the chapter "Report-Writer Expressions and Formats."

## Description

The .nullstring statement allows you to specify a string to print whenever a null value is to appear on the report. The string can be an expression that evaluates to a string at runtime. Because a data value of *null* means that there is really no data present to print, you can use the .nullstring to print a designated string that signifies the absence of the data.

**Note:** Ensure that the length of the *null\_string* is less than or equal to the width of printed nullable columns. If a column is not wide enough to contain the *null\_string*, then the empty string is printed instead.

If you do not specify a .nullstring statement, Report-Writer uses the default value of the II\_NULL\_STRING environment variable/logical, if defined. If not defined, it uses a default of the empty string (a string with no characters) to print a null value. You can specify several .nullstring statements in a report specification. The system uses the current .nullstring until another .nullstring statement is executed.

## Example

Suppose phone\_number is a nullable integer column whose value is null. If you issued the following print statements:

```
.nullstring      'N/A'
.print          ' Phone number = ', phone_number .nl
.nullstr        '?'
.print          'Phone number = ', phone_number .nl
```

Report-Writer would print the following if the value of phone-number were null:

```
Phone number = N/A
Phone number = ?
```

## **.Print and .Println Statements—Print Text of Report**

The .print and .println statements print literal text strings, columns from the database, or expressions on the report.

These statements have the following formats:

```
.print | .pr | .p expression[(format)]{, expression[(format)]}

.println | .prln | .pln expression[(format)]{, expression[(format)]}
```

The parameters for the .print statement are as follows:

### ***expression***

Any legal expression. For more information, see the chapter "Report-Writer Expressions and Formats." You can specify a column name in an expression as a delimited identifier by enclosing it in double quotes ("), if you have previously specified the .delimid statement.

### ***format***

An optional printing format for the expression, as described in Format Specifications. The format must be the correct type for the expression. If you do not specify a format, Report-Writer uses one of the default formats listed in the chapter "Report-Writer Expressions and Formats."

The *format* can be expressed as a variable (*\$formatvariable*). The parentheses ( ) must be stated explicitly and cannot be part of the *format* variable. The comma (,) must be stated explicitly and cannot be part of the *expression*.

### ***\$formatvariable***

Variable whose value is the name for a printing format described in Format Specifications. The format must evaluate to the correct type for the column(s). Precede the variable with a dollar sign (\$). Report-Writer evaluates *\$formatvariables*, but not expressions, at load time.

## Description

The `.print` statement specifies text to be included in the body of the report. Text can be character strings printed directly, data items from the data table, variables, aggregates, or a combination of these. Report-Writer includes the text at the place in the report where it encounters the `.print` statement. By preceding the `.print` statement with the positioning statements such as `.newline`, `.tab`, `.center`, `.right`, or `.left`, you can specify the location of the text. By default, Report-Writer includes the text immediately after the last text output with the `.print` statement.

Report-Writer evaluates any print expressions that are used as parameters to the `.print` statement during runtime. You can include as many expressions as you want in the `.print` statement; Report-Writer adds them to the report in the order specified.

If the expression is a runtime variable for a column name (for example, `.print $account_type`), Report-Writer prints the *name* of the column as entered at runtime, rather than its *data*. To print the data rather than the column name, you use the select *column\_list* as *resultcolumn\_list* construct in the query and reference the *resultcolumn\_list* name in the `.print` statement. For more information, see the `.Query` statement.

If you use the optional `.println` form of the statement, the current print position advances to the next line after the specified text is printed.

Data that formats into a single logical line can wrap to yield a default maximum of 310 physical lines as protection against omitted explicit `.newline` and/or `.newpage` statements. You can override this limit by specifying the `-wmxwap` parameter in the report command.

**Important!** Embedding tabs in a string can truncate the string if it is printed with a default format or with a format that is not large enough to allow the tab to be expanded into spaces. To print string sthats contain tabs, use a specified format wide enough for the expanded tab(s).

## Examples

1. Assume a report specification with the following literals:

```
.print 'Text may'  
      ' span several lines.'
```

It would print the following output:

```
Text may span several lines.
```

Because there was no specification statement such as a `.tab` or a `.newline` to separate the fields, the two text strings printed immediately adjacent to each other on the same line of the output.

2. In this example, assume that `page_number` is equal to 3.

```
.pr 'Page number:', page_number(zz)
```

Report-Writer would print the following text:

Page number: 3

3. The following example shows the specifications you need to print a data value (represented by a delimited identifier for the column name) and an aggregate, using a numeric template for the aggregate where the value of *millions\_fmt* is 'nnn,nnn,nnn.nn':

```
.print "acct bal", sum("acct bal")($millions_fmt)
```

4. A complex .print statement that displays a large number of data items can look like the following:

```
.print 'Values of the data are: ', var1,  
      var2(e20.4) cvar1(c40), ' and finally',  
      lastvar (' $$$,$$$,$$$.$nnCR')
```

Note that in the previous example, the field, *var1*, was listed without a format. Report-Writer prints the value with the default format for the data type, according to the table in Default Formats, in the chapter "Report-Writer Expressions and Formats." It is acceptable to mix the default data formats with extremely complex templates. Using an acceptable default format saves you the time and effort of specifying every format in detail.

## .Ulcharacter Statement—Set Underline Character

The .ulcharacter statement sets the underlining character to any single character when output is to a file or printer.

This statement has the following format:

```
.ulcharacter | .ulchar | .ulc 'c' | expression
```

The parameters for the .ulcharacter statement are as follows:

### **c**

Any single character, subsequently used as the underlining character. The default underlining character is the underscore (\_) for reports written to a file or printer, and none for reports written to the screen.

The character *c* must be a single character enclosed in quotes.

### ***expression***

A character expression that evaluates to any single character, subsequently used as the underlining character. Precede all variables that are part of the expression with a dollar sign (\$). For more information, see the chapter "Report-Writer Expressions and Formats."

## Description

With the .ulcharacter statement, you can specify an alternate underlining character or an expression that evaluates to an underlining character during report runtime. Underlining occurs only in reports written directly to a file or printer. Report-Writer ignores underlines when displaying a report on the screen and in reports sent to a file or printer from the screen.

The character remains in effect until Report-Writer encounters another .ulcharacter statement in the report.

Report-Writer prints underscoring (\_\_) on the same line as the text. If any other character, such as a hyphen (-), is specified with the .ulcharacter statement, Report-Writer prints underlining as a second line immediately below the underlined text.

## Example

To produce the following:

```
Underline me
----- --
and me
=== ==
```

Use the following specifications:

```
.underline
      .ulcharacter '-' .pr 'Underline me' .newline
      .ulcharacter '=' .pr 'and me' .newline
.nounderline
```

## .Underline and .Nounderline Statements—Underline Text

The .underline and .nounderline statements let you underline text.

These statements have the following format:

```
.underline | .ul | .u
    any printing statements
.nounderline | .noul | .nou
```

### Description

To underline text in a report, put an .underline statement immediately before the spot where underlining begins, and .nounderline at the spot where it stops. You can underline anything that can be printed, including character strings, column values, parameter values, or aggregate values. Underlining occurs only in reports written directly to a file or printer. Report-Writer ignores underlines when displaying a report on the screen and in reports sent to a file or printer from the screen. By default, the underlining character is an underscore (\_). This can be changed with the .ulcharacter statement.

When underlining is in effect, only letters and digits are underlined. The .underline statement ignores all other characters, such as blanks, commas, and periods. Underscores print on the same line as the text. If the underlining character is anything other than an underscore, Report-Writer prints the underlining on the line below the one containing the text to be underlined.

**Note:** Printers that interpret carriage returns as a combination of both a carriage return and a line feed is not able to use the underscore (\_) as the default underlining character. If your printer is configured this way, you should use the .ulcharacter command to reset the underline character to a hyphen (-) or some other character; otherwise, if you use the underscore as the underlining character, Report-Writer prints the underline above the text instead of below it.

### Example

To produce the following line:

```
Numbers - 123,456 are underlined,
but punctuation is not!
```

Use the following specifications:

```
.underline
.print 'Numbers - 123,456 are '
.print 'underlined,'
.newline
.println 'but punctuation is not!'
.nounderline
```

## Conditional and Assignment Statements

Conditional and assignment statements are used to specify alternative blocks of statements or to assign values to variables.

### .If Statement—Specify Alternative Statements

The .if statement specifies alternative blocks of statements to be executed under specified conditions.

This statement has the following format:

```
.if condition .then {statement}  
  {.elseif condition .then {statement}}  
  [.else {statement}]  
.endif
```

The parameters for the .if statement are as follows:

#### ***condition***

A boolean expression that evaluates to true or false.

#### ***statement***

Any action statement, including the .if statement (this excludes the setup and structure statements in the sections, Report Setup Statements and Report Structure Statements).

Both the expression in the *condition* and/or the parameters to *statement* can contain column names that are delimited identifiers enclosed in double quotes ("), if you have previously specified the .delimid statement.



## Description

The `.if` statement specifies alternative blocks of statements to be executed depending upon the value of the specified condition.

Report-Writer evaluates the conditions in the `.if` and `.elseif` clauses one after another. When a condition is met, Report-Writer executes the statements following the subsequent `.then` statement. If *none* of the specified conditions is met, Report-Writer does nothing. If none of the conditions is met and there is an `.else` clause included in the `.if` statement, Report-Writer executes the statements following the `.else` statement.

## Examples

1. This example illustrates the use of the `.if` statement to take different actions based on the current condition of the Report-Writer environment. It tests the current character position, and starts a new line if the current position is past the end of a line:

```
.if position_number > 80 .then
    .newline
.endif
```

2. This example tests the data and prints different things, depending on the value of some of the report data:

```
.if balance < 0 .then
    .print '(-,-balance, ' )'
.else
    .tab +1
    .print balance
.endif
```

3. This example tests a column value and uses `.if` statements to translate a numeric code number from the database to a text string for the report, and to selectively print column values accordingly. Notice that both the conditional expression and the `.then` statements reference column names that are delimited identifiers.

```
.if "dept code" = 1 .then
    .print 'Books: ', "book sales"
.elseif "dept code" = 2 .then
    .print 'Furniture: ', "furniture sales"
.elseif "dept code" = 3 .then
    .print 'Jewelry: ', "jewelry sales"
.else
    .print 'Misc: ', "misc sales"
.endif
```

## .Let Statement—Assign Expression Value to a Variable

The .let statement assigns the value of an expression to a declared variable.

This statement has the following format:

```
.let variablename [:] = expression  xe "=" (equals sign):assignment operator"
```

## Statement Syntax Summary

The following table provides a summary of Report-Writer statement syntax.

Statement Category	Accepts Variables or an Expression*	Default, If Not Specified
.block   .blk <i>other formatting statements</i> .endblock   .endblk   .end block	No	None
.bottom   .bot	No	None
.break   brk <i>columnname</i> {, <i>columnname</i> }	No	None
.center   .cen   .ce [[+ -] <i>n</i>   <i>columnname</i>   <i>expression</i> ]	Expression	Page center**
.cleanup <i>SQL_statement</i> ; { <i>SQL_statement</i> ;}	No	None
.data   .dat   .table   .view [ <i>schema.</i> ] <i>tablename</i>   <i>viewname</i>   <i>synonym</i>	Variables	None
.declare <i>variablename</i> = <i>datatype</i> [with null   not null] [with prompt ' <i>promptstring</i> '] [with value ' <i>valuestring</i> '] {, <i>variablename</i> = <i>datatype</i> ...}	No	None
.delimid	No	Delimited identifiers not recognized
.detail   .det	No	None
.footer   .footing   .foot report   page   <i>columnname</i>	Variables	None
.format   .fmt <i>columnname</i> {, <i>columnname</i> } ( <i>format</i> ) { <i>columnname</i> {, <i>columnname</i> } ( <i>format</i> )}	Variables	Determined by data type of column
.formfeeds   .ffs   .ff .noformfeeds   .noffs   .noff	No	
.if <i>condition</i> .then { <i>statement</i> } {.elseif <i>condition</i> .then { <i>statement</i> }}	No	None

Statement Category	Accepts Variables or an Expression*	Default, If Not Specified
[.else { <i>statement</i> }] .endif		
.include <i>filename</i>	No	None
.left   .lft   .lt [[+   -] <i>n</i>   <i>columnname</i>   <i>expression</i> ]		Left margin**
.leftmargin   .lm [[+   -] <i>n</i>   <i>expression</i> ]	Expression	0
.let <i>variablename</i> [:] = <i>expression</i>	Expression (right of the equal sign)	None
.lineend   .lnend	No	None
.linestart   .lnstart   .linebegin	No	None
.longremark   .lrm <i>remark_text</i> .endremark   .endrem	No	None
.header   .heading   .head report   page   <i>columnname</i>	Variables	None
.name   .nam <i>reportname</i>	No	None
.need   .ne <i>nlines</i>   <i>expression</i>	Expression	None
.newline   .nl [ <i>nlines</i>   <i>expression</i> ]	Expression	1
.newpage   .np [[+   -] <i>pagenumber</i>   <i>expression</i> ]	Expression	Current page number + 1
.nullstring   .nullstr ' <i>null_string</i> '   <i>expression</i>	Expression	None
.output   .out <i>filename</i>	Variables	Screen
.pagelength   .pl { <i>nlines</i>   <i>expression</i> }	Expression	Files: 61 lines Screens: screen length
.pagewidth   .pw <i>width</i>	Variables	Files: 132 characters Screens: screen width
.position .pos <i>columnname</i> {, <i>columnname</i> } <i>position</i> [, <i>width</i> ]) {, <i>columnname</i> {, <i>columnname</i> } ( <i>position</i> [, <i>width</i> ])}	Variables	None
.print   .pr   .p <i>expression</i> [( <i>format</i> )] {, <i>expression</i> [( <i>format</i> )]} or .println   .prln   .pln <i>expression</i> [( <i>format</i> )] {, <i>expression</i> [( <i>format</i> )]}	Expression (Note: Formats can only take variables)	None

Statement Category	Accepts Variables or an Expression*	Default, If Not Specified
.query   .quer /* for SQL users */ select [all   distinct] <i>column_list</i> [as <i>resultcolumn_list</i> ] from [ <i>schema.</i> ] <i>table</i>  view synonym [ <i>corr_name</i> ] {, [ <i>schema.</i> ] <i>table</i>  view synonym [ <i>corr_name</i> ]} [where <i>search_condition</i> ] [group by <i>column</i> {, <i>column</i> }] [having <i>search_condition</i> ] {union select ...} [order by ...]	Variables	None
.query   .quer /* for QUEL users */ / { <i>range_statement(s)</i> } retrieve [unique] ( <i>target_list</i> ) [where <i>qual</i> ] [sort by   order by ...] <i>sort-list</i>	Variables	None
.right   .rt [[+   -] <i>n</i>   <i>columnname</i>   <i>expression</i> ]	Expression	Right margin**
.rightmargin   .rm [[+   -] <i>n</i>   <i>expression</i> ]	Expression	100 or determined by detail statement
.setup <i>SQL statement</i> ; { <i>SQL statement</i> ;}	No	None
.shortremark   .srem <i>remark-text</i>	No	None
.sort   .srt { <i>columnname</i> [: <i>sortorder</i> ] {, <i>columnname</i> [: <i>sortorder</i> ]}	Variables	None
.tab   .tb.   .t [[+   -] <i>n</i>   <i>columnname</i>   <i>expression</i> ]	Expression	None
.tformat   .tfmt <i>columnname</i> {, <i>columnname</i> } ( <i>format</i> ) {, <i>columnname</i> {, <i>columnname</i> } ( <i>format</i> )}	Variables	None
.top   .tp	No	None
.ulcharacter   .ulchar   .ulc 'c'  <i>expression</i>	Expression	Files: underscore (_) Screens: hyphen (-)
.underline   .ul   .u <i>any printing statements</i> .nunderline   .noul   .nou	No	No underline
.width   .wid <i>columnname</i> {, <i>columnname</i> } ( <i>width</i> ) {, <i>columnname</i> } {, <i>columnname</i> } ( <i>width</i> )}	Variable	Determined by format
.within   .wi <i>columnname</i> {, <i>columnname</i> }   all <i>other formatting statements</i> ..endwithin   .endwi   .end within	Variable	None

\*If an expression is used, it must evaluate to the type of object indicated in the statement; for example, a number or column name. Variable names must be preceded by a dollar sign (\$) to evaluate them. Any punctuation must be explicitly stated and cannot be a part of the expression. Statements that take variables only (not expressions) are evaluated during the load of the report specifications, while those that take expressions are evaluated each time the statement is executed.

\*\*The default value if the statement is specified with no parameter.



# Chapter 13: Using VIFRED

---

This section contains the following topics:

[What Is Visual Forms Editor \(VIFRED\)?](#) (see page 443)

[VIFRED Frames and Operations](#) (see page 444)

[Start VIFRED](#) (see page 446)

[VIFRED Forms Catalog Frame](#) (see page 447)

[Create New and Duplicate Forms](#) (see page 449)

[Form Layout Frame](#) (see page 454)

[Form Attributes](#) (see page 458)

[Form Display Style](#) (see page 459)

[Borders for Pop-up Forms](#) (see page 461)

[Form Size and Position](#) (see page 462)

[Save a Form](#) (see page 472)

[Destroy Forms](#) (see page 475)

[Edit Existing Forms](#) (see page 475)

[Rename Forms](#) (see page 475)

[Compile Forms](#) (see page 476)

[Print Forms](#) (see page 478)

[QBFFNames Operation](#) (see page 479)

[Run QBF from VIFRED](#) (see page 482)

## What Is Visual Forms Editor (VIFRED)?

The VIFRED is a visually oriented tool for creating or modifying the appearance of forms. You can use your customized forms with QBF to display, add, or modify data in your database, or in applications that you design using ABF, Vision, or an embedded query language.

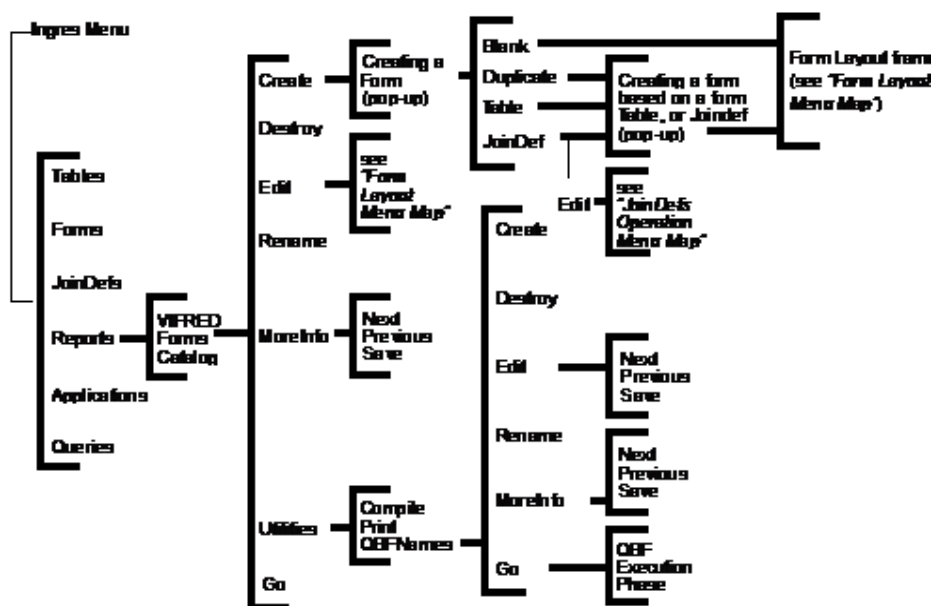
With VIFRED, you can create custom forms or edit and enhance default forms in the following ways:

- Set the default display style to full-screen or pop-up, or change the dimensions of a form, field, or data area
- Specify the order in which fields appear in the window or are accessed
- Create lines, boxes, and other trim, as well as instructions and information for the end user that make the form more attractive or easier to understand

- Edit a field's default title to more accurately describe its purpose, or specify color, underlining, blinking, or other attributes for a field
- Define fields that scroll horizontally, are display-only or query-only
- Make entry in a field mandatory, specify an automatic default value, or establish validity checks to verify that the information entered into a field meets certain criteria
- Change error messages that appear if a user enters incorrect data

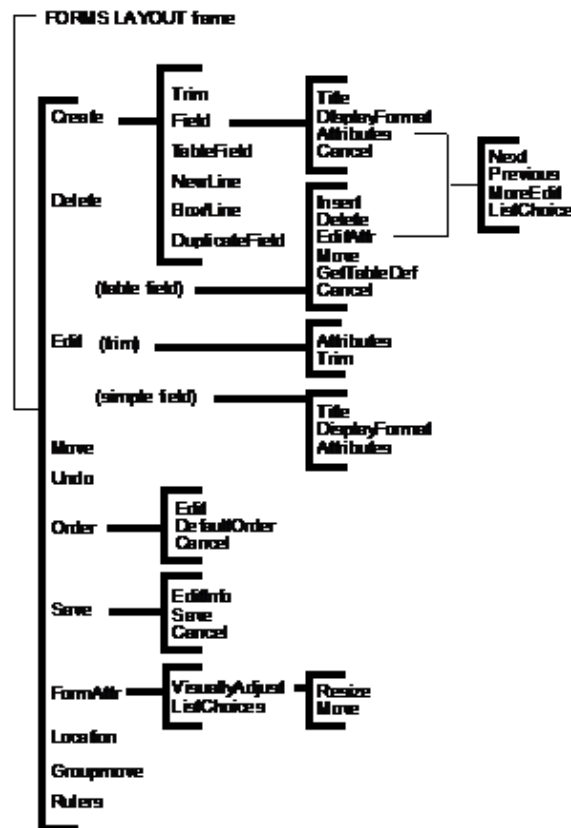
## VIFRED Frames and Operations

When you choose Forms from the Ingres Menu, the Forms Catalog frame appears. From this primary forms frame, you can access all other VIFRED frames. The following two figures contain maps of the various VIFRED frames and operations:





For a menu map of the JoinDef Edit frame's menu options, see the chapter "Using JoinDefs in QBF." The following figure is a menu map of the Forms Layout frame:



## Start VIFRED

You can start VIFRED from the operating system or from the Ingres Menu.

### To start VIFRED from the operating system

Use the `vifred` command.

For more information, see the *Command Reference Guide*.

### To start VIFRED from the Ingres Menu

To start VIFRED from the Ingres Menu, choose Forms.

VIFRED displays the Forms Catalog frame, which provides access to all other forms-editing frames in VIFRED.

## Start VIFRED in Expert Mode

*Expert mode* allows you to enter VIFRED without displaying the full list of existing forms in the Forms Catalog frame. Instead, you type the name of the form you want to work with in the Name column of a blank Catalog frame.

You can also use expert mode with pattern matching to have VIFRED retrieve a range of forms from which you can choose. For example, to retrieve and display a list of all the forms that begin with the letter `s` you would type `s*` in the Name column.

### To start VIFRED in expert mode

1. From the operating system enter the command:

```
vifred [dbname | v_node::dbname] [/server-type] -e
```

VIFRED displays an empty Forms Catalog frame.

2. In the Name column, type the name of the form you want to work with, or pattern-matching characters to select several forms at once.
3. Choose the operation that you want to perform.

If you entered a single form name in the Name column, VIFRED performs the chosen operation.

If you used pattern matching, VIFRED displays all the forms that match your specification.

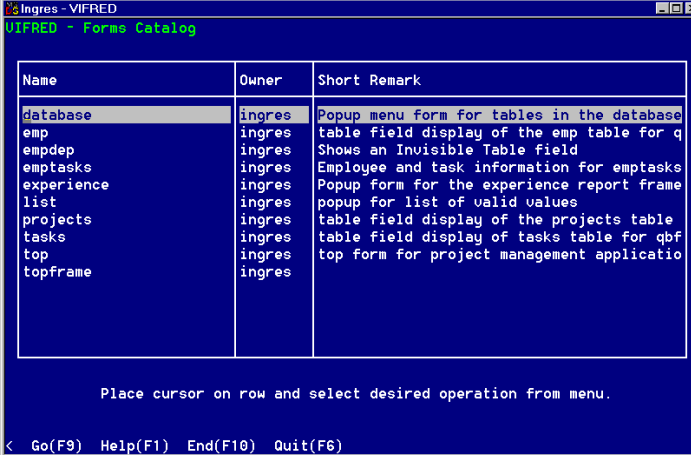
To select from the displayed list:

- a. Place the cursor on the name of the form to select it.
- b. Choose the operation to perform.

## VIFRED Forms Catalog Frame

The Forms Catalog frame lists all VIFRED forms owned by the user or the DBA and stored in the current database. It displays a menu of operations that you can perform on these forms.

The Forms Catalog frame in the following figure displays the owner (creator) and a brief description of the form beside the name of each form.



Name	Owner	Short Remark
database	ingres	Popup menu form for tables in the database
emp	ingres	table field display of the emp table for q
empdep	ingres	Shows an Invisible Table field
emptasks	ingres	Employee and task information for emptasks
experience	ingres	Popup form for the experience report frame
list	ingres	popup for list of valid values
projects	ingres	table field display of the projects table
tasks	ingres	table field display of tasks table for qbf
top	ingres	top form for project management applicatio
topframe	ingres	

Place cursor on row and select desired operation from menu.

< Go(F9) Help(F1) End(F10) Quit(F6)

To locate the name of a form, scroll through the list or use the First Letter Find function (see page 41).

The operations on the Forms Catalog frame are as follows:

### Create

Displays a pop-up form showing the sources for creating a new form.

### Destroy

Destroys the form highlighted by the cursor. You can only destroy forms that you own.

### Edit

Displays the Form Layout frame for editing or viewing the form selected by the cursor.

### **Rename**

Renames the form selected by the cursor. You can only rename forms that you own.

### **MoreInfo**

Obtain more information about the selected form. The additional information includes the time the form was created and a short and long description of the form.

### **Utilities**

Provides access to the following forms management operations:

Compile - Compiles a form definition for use with an embedded query language as a C source file (or VMS macro).

Print - Creates a picture of the form that can be printed on a line printer.

QBFNames - Creates, renames, or deletes QBFNames.

### **Go**

Starts QBF using the form highlighted by the cursor. For more information, see Run QBF from VIFRED (see page 482).

### **Help, End, Quit**

Perform standard operations.

In addition to these operations, you can also use system-level commands such as copyform, compform, delobj, and printform to manage VIFRED forms.

## Create New and Duplicate Forms

Create a new form with the Create operation on the Forms Catalog frame. The Create operation offers you the choice of duplicating an existing form (which you can then edit), beginning with a completely blank form, or creating a default form from a table or JoinDef.

### To create a form

1. Choose the Create operation on the Forms Catalog frame.

VIFRED displays a pop-up with the following choices:

#### **Duplicate**

Creates a form from an existing form.

#### **Blank**

Creates a form starting with a blank window.

#### **Table**

Creates a default form based on a table.

#### **JoinDef**

Creates a default form based on a JoinDef.

2. Choose one of the operations on the pop-up menu.

## Duplicate Forms

Create a new form by duplicating an existing form in the current database. Modify it with the Duplicate operation on the Forms Catalog frame, following the same steps used to create a default form. For details, see [Create Default Forms](#) (see page 451).

To duplicate a form from another database, use the `copyform` system-level command to copy the form into the intended database. This command also allows you to change the name and ownership of the form.

## Create Blank Forms

You can create blank forms that:

- Do not access data, but rather act as logo windows, information windows, and help windows.
- Are interactive forms linked to procedures written in a database programming language such as 4GL and embedded SQL. An example of this is an application login window in which users enter their name and password.
- Access and display database data through procedures written in programming languages.

### To create a blank form

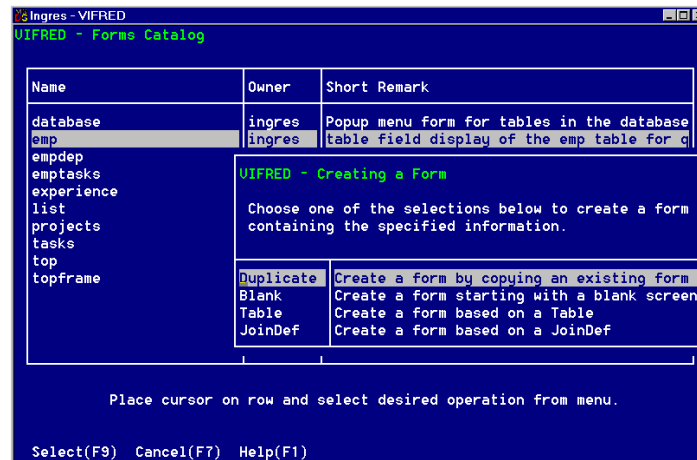
1. Choose Blank from the Creating a Form pop-up menu.
2. Choose the Select operation on the Forms Catalog frame.  
VIFRED displays a blank form in the Form Layout frame, which is described in the next section.
3. Choose the Save operation on the Form Layout frame  
You are prompted for a name.
4. Enter a name.  
The form is saved.

After creating the form, you can modify it, as discussed later in this chapter, or add components to it, as discussed in the chapter "VIFRED Form Components."

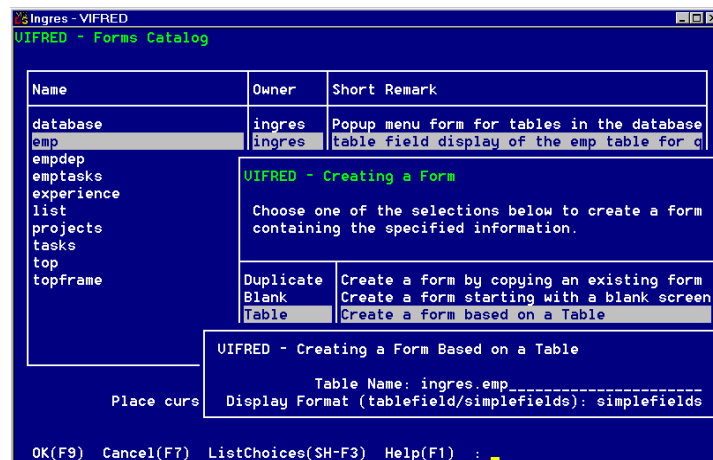
## Create Default Forms

To create a default form based on a table or JoinDef, or a form that duplicates an existing form in the current database

1. Choose Duplicate, Table, or JoinDef from the Creating a Form pop-up.



After making your selection, another pop-up similar to the one shown here appears.



2. Chose one of the following procedures:
  - If necessary, use the ListChoices operation to display a list of available choices and select an item from the list.
  - If you are basing the default form on a JoinDef, enter the JoinDef name on the appropriate pop-up. This pop-up contains the additional menu option, Edit. If you want to edit or create the JoinDef, choose the Edit operation to access QBF. VIFRED returns you to the default form creation process when done.
  - If you are basing the default form on an existing form or table, enter the form or table name on the appropriate pop-up. You can specify more than one table by separating the table names with spaces or commas.
3. In addition, if you are basing the form on a table, specify one of the following display formats :

**tablefield**

Displays several rows on a form that can be scrolled horizontally or vertically if all columns and rows do not fit in the window at one time.

**simplefields**

Displays one record at a time.

4. Choose the OK operation.

VIFRED displays the appropriate form in the Form Layout frame, which is described in the next section.

If you based the form on a table or JoinDef, VIFRED displays a default form, with one field on the form for every data column in the table or JoinDef. The titles, internal names, and data display formats of these fields are based on the underlying data columns. For special issues regarding the long varchar, byte, byte varying, and long byte data types and delimited identifiers, see the note following this procedure.

5. Choose the Save operation on the Form Layout frame to save and store the form under a name you give it.

After creating a form you can modify it, as described in this chapter, or add components to it.

**Note:** VIFRED does not create fields or table field columns on a default form for columns of data types long varchar, byte, byte varying, and long byte in the associated database table or tables.



If your table uses delimited identifiers, column-to-field name conversions in default forms can result in form field name collisions. This is because VIFRED allows delimited identifiers for column names, but not for field names. It must strip out certain characters in the database column name to make the new field name conform to regular identifier rules. For example, if the database table has separate columns named Column1 (regular identifier) and "Column 1" (delimited identifier), VIFRED converts both column names to the same default field name, Column1.

## Ways to Create Forms that Use Multiple Tables

To create a form that uses columns from multiple database tables, use one of the following techniques:

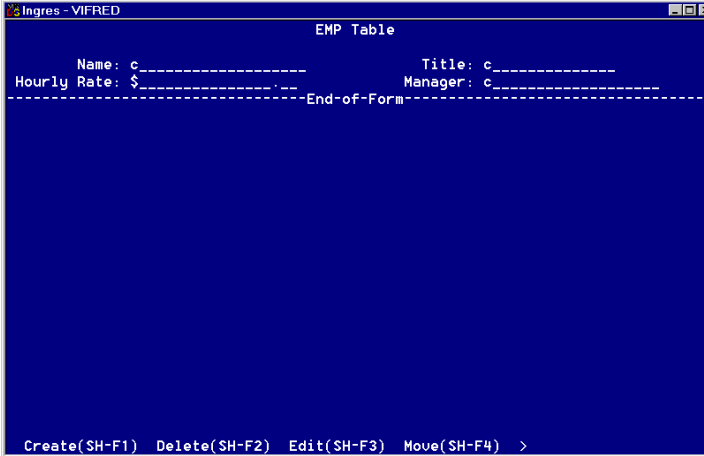
- Create a default form based on an existing table. When the Create a Form Based on a Table pop-up appears, enter more than one table name, separated by spaces or commas. VIFRED automatically creates fields with the appropriate internal names and data display formats.
- Use a query language to create a multi-table view or use QBF to create a JoinDef; then create a default form for that view or JoinDef. VIFRED automatically creates the fields with the appropriate internal names and data display formats.
- Create a blank form and create the fields you want, specifying appropriate data display formats; then use a programming language to link those fields to data columns in multiple tables by way of the fields' internal names.

You can use the GetTableDef operation to create a table field on the form that contains internal names and data display formats automatically corresponding to the columns in the database tables. You cannot use GetTableDef to create simple fields.

## Form Layout Frame

The Form Layout frame is the basic frame for modifying and enhancing an existing form.

After you create a form with the Create operation on the VIFRED Forms Catalog frame, VIFRED places you on the VIFRED Form Layout frame. You can also access the Form Layout frame by placing the cursor on the name of a form listed in the VIFRED Forms Catalog frame and choosing the Edit operation.



The screenshot shows a window titled "Ingres - VIFRED" with a sub-header "EMP Table". The form layout is displayed on a dark blue background with white text. It includes fields for "Name: c", "Title: c", "Hourly Rate: \$", and "Manager: c", each followed by a dashed line for input. A dashed line separates the form fields from the command area at the bottom. The command area contains the text "Create(SH-F1) Delete(SH-F2) Edit(SH-F3) Move(SH-F4) >".

```

Ingres - VIFRED
EMP Table
Name: c_____ Title: c_____
Hourly Rate: $_____ Manager: c_____
-----End-of-Form-----
Create(SH-F1) Delete(SH-F2) Edit(SH-F3) Move(SH-F4) >

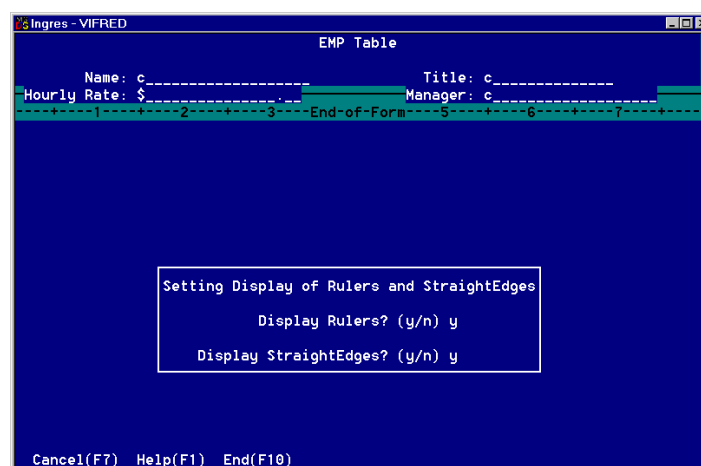
```

## Alignment Guides

You can use the following alignment guides on forms displayed in the Form Layout frame:

- Ruler marks in the form's margins
- Straight edge alignment guides
- Location operation

An example of the alignment guides are shown here.



## Margin Rulers

The right and bottom margins of your form contain optional ruler marks every fifth column or row, and a decimal digit every tenth column or row, to help you determine the coordinates of components on your form. The margins containing the rulers are initially positioned off the screen on fullscreen forms. To see them, you need to scroll the form by moving the cursor to the edge of your window, or use the Move operation to move the margins of the form (see page 465).

You can turn ruler marks on or off by choosing the Rulers operation on the Form Layout frame. On the pop-up menu, set Rulers to y (yes) or n (no). The default is n (no).

## Straight Edges

The Form Layout frame contains optional horizontal and vertical straight edge alignment guides to help you line up the components on your form. The vertical and horizontal straight edges appear initially in the last column and row of the form. If the form is larger than your window, you must scroll the form by moving the cursor off-screen to gain access to them. If a straight edge is touching a form component, such as trim or a field, the form component temporarily takes precedence and obscures that part of the straight edge.

You move straight edges with the Move operation, as you would a piece of trim or other form component. For best results, use a monitor with a line graphics character set.

You can turn straight edges on or off by choosing the Rulers operation on the Form Layout frame. On the pop-up menu, set Straight edges to y (yes) or n (no). The default is n (no).

## Location Operation

When working with forms, you sometimes need to know the precise row and column location of a place in your window. For example, to specify a fixed position for a pop-up form with the StartRow and StartColumn attributes, you must know the row and column designation of that place in your window.

The Location operation on the Form Layout frame displays the current row and column location of the cursor, relative to the boundaries of your window, regardless of the placement and orientation of the form.

To find the row and column location of any place in your window:

1. On the Form Layout frame, place the cursor at the chosen spot.
2. Choose the Location operation.

VIFRED marks the cursor location with a plus sign (+) and displays a message giving the row and column location.

## Layout Frame Menu Options

The operations that you can use on the Form Layout frame are as follows:

### **Create**

Displays a new menu that enables you to create new components on a form.

### **Delete**

Deletes the component where the cursor is positioned.

### **Edit**

Displays another menu that enables you to edit the trim or field component where the cursor is positioned.

### **Move**

Moves a component or changes the margin. This operation displays a new menu. For more information on changing the overall margins of a form, see Form Size and Position (see page 462).

### **Undo**

Reverses the results of the last operation you performed. Undo does not reverse the FormAttr operation on the Form Layout frame or the EditAttr operation on the Edit submenu.

### **Order**

Changes the order in which the form's fields are accessed when an end user presses the Tab key. This operation displays a new menu for ordering fields.

### **Save**

Displays a new menu that allows you to save your new or updated form in the database. The Save operation is independent of exiting. You can save one form and begin work on another without leaving VIFRED. For more information, see Save a Form (see page 472).

### **FormAttr**

Changes the default attributes of the form as a whole, such as fullscreen or pop-up display style. For more information, see Form Attributes (see page 458).

### **Location**

Displays a message giving the current row and column position of the cursor, relative to your window boundaries, and marks the cursor location with a plus sign (+). For details, see Location Operation (see page 456).

### **Groupmove**

Moves an entire group of components on a form at once.

### Rulers

Turns rulers (see page 455) and straight edges (see page 456) on or off.

### Help, End, Quit

Perform standard operations.

## Form Attributes

Form attributes are specifications that apply to the form as a whole (in contrast to field attributes, which are specifications that apply to individual fields). All form attributes are related to the following form characteristics:

- Display style (see page 459)
- Size and screen position

You designate a form's attributes with the Form Layout frame's FormAttr (form attribute) operation. When you choose the FormAttr operation, VIFRED displays the Form Attributes pop-up frame. Use this frame to specify attributes for the form currently displayed in the Form Layout frame.

The *display style* of your current form determines which attributes appear on the Form Attribute pop-up frame. VIFRED provides two basic display styles for forms:

- Fullscreen
- Pop-up

For more information on display styles, see Form Display Style (see page 459).

This figure shows the Form Attributes frame for a fixed-position pop-up style form.

The screenshot shows a terminal window titled 'Ingres - VIFRED'. Inside, there's a form titled 'EMP Table' with fields for 'Name: c\_...' and 'Title: c\_...'. Below these fields is a green horizontal bar with a grid of numbers 1 through 8. At the bottom of the window is a pop-up frame titled 'Form Attributes for 'emp'' containing the following text:

Style (fullscreen/popup):	popup_____	Size:	6 rows	80 cols
Position (fixed/floating):	fixed___	Border (y/n):	y	
StartRow:	6__			
StartColumn:	1__			

At the very bottom of the terminal window, there is a command line: 'VisuallyAdjust(SH-F1) ListChoices(SH-F2) Help(F1) End(F10) :'

The following table lists and summarizes all of the attributes that appear on Form Attributes frames for both fullscreen and pop-up forms. Only those attributes that apply to the style of your form appears on your Form Attributes frame.

Attribute	Description	For
Style	The display style of the form, either fullscreen or pop-up	All forms
Size	Size of the form in rows and columns	All forms (display-only)
Screen Width	Displays the screen in the terminal's current width (default), narrow width (usually 80 columns), or wide width (usually 132 columns)	Fullscreen forms
Position	How the pop-up form is positioned (either fixed position or floating position)	Pop-up display-style forms
Border	Whether the pop-up form has an automatic border	Pop-up display-style forms
StartRow	The row containing the form's upper left corner	Fixed-position pop-up display-style forms
StartColumn	The column containing the form's upper left corner	Fixed-position pop-up forms

## Form Display Style

You use the `FormAttr` operation on the Form Layout frame to choose a form's display style. Display style determines whether the form appears alone in the window or as a pop-up form, which can overlay another form.

### Fullscreen Forms

Forms in normal or *fullscreen* display style appear one at a time on your screen. Each time VIFRED displays a new form, it removes the previous form from the screen. The term *fullscreen* refers to the fact that only one form can be displayed at a time, not the size of the form. A fullscreen form can be any size you want, either larger or smaller or the same size as your screen. The default size is the width of your screen, at form-creation time.

## Pop-up Forms

Forms in *pop-up* display style can be displayed without removing previously displayed forms from the window. A pop-up style form temporarily covers over some or all of the previous form. Thus, by using pop-up style forms you can have multiple forms, or portions of forms, visible at the same time. The cursor is only active in one form at a time—always the last form activated.

You can make pop-ups any size up to the size of your window, but they are most useful when you make them small enough to be displayed without completely obscuring previous forms. When specifying the size of a pop-up form, allow space for a border, if you choose to use one. For more information, see *Form Size and Position* (see page 462).

You can specify that a pop-up style form appear at a specific window location or automatically float to a position near to but not obscuring the field in the previous form that contains the cursor.

Pop-up style forms can be used for a wide variety of purposes:

- To display information such as messages and prompts.
- To display lists of acceptable values during data entry. For example, if a user is not sure of the values that go into a particular field, a pop-up form can be invoked to show those values.
- To display lists of currently existing tables, reports, JoinDefs, QBFNames, and other objects.
- To allow users to easily interrupt one operation, perform another operation on a pop-up form, and then return to the first operation without having to load and traverse many different menus.

For example, suppose you are appending records to a table and someone asks for information contained in another table. You can code an application so that, by invoking a pop-up retrieval form for the other table, the user can perform a query (on the pop-up form) and then return to the original task by removing the pop-up form.

When using pop-up forms in an application, the application code controls the appearance and disappearance of the pop-up. When your application code displays a normal fullscreen form, it clears the previous form from the screen. When your application code displays a pop-up form, the previous form remains in your window.



## Change the Style of a Form

The display style determines whether the form appears alone in the window or as a pop-up form.

### To change the style of a form

1. Choose the FormAttr operation on the Form Layout frame.

VIFRED displays the Form Attributes pop-up frame.

2. Type an f for fullscreen or a p for pop-up in the Style field and press the Return key.

VIFRED automatically displays the attributes on the Form Attributes pop-up that are available for the new style.

If you change a fullscreen form to a pop-up form, VIFRED sets the default value of the Border attribute to Y (yes) and the Screen Width to the default setting, Current, which is the default width as set at run time.

If you started with a form that was as large or larger than your screen—for instance, a default fullscreen form—you must delete some lines from the form to make room for the pop-up form's border, which requires two lines, before changing it to a pop-up. If you change the Border attribute to N (no) for a form that occupies your entire screen, you still need to reduce its size by two lines to change it to a pop-up, in case a user turns on the borders at run time.

## Borders for Pop-up Forms

The Border attribute tells VIFRED whether to include an automatic border around a pop-up display-style form. The Border attribute is not available for fullscreen forms.

By default, a border is specified. To change the specification, enter n for no border or y for yes to include a border in the Border attribute field on the Form Attributes pop-up.

Allow space for the border when sizing your pop-up, or when creating a pop-up by changing the style of a fullscreen form that occupies the entire screen. A border requires two lines and two columns in addition to the other components on your form. A borderless pop-up requires the same allowance, in case a user turns on borders at run time.

You cannot specify display attributes such as brightness, inverse video, or color for borders created with the Border attribute. If you want to define your own border, you can use the Box/Line operation to draw a box around the inside edges of the form. This allows you to specify display attributes for the box.

## Form Size and Position

You can use the FormAttr operation on the Form Layout frame to:

- Determine the size of your form with the Size attribute.
- Change the spacing or size of characters for a fullscreen form when displayed on your screen, by setting the Screen Width attribute.
- Position a pop-up form with the Position, StartRow, and StartColumn attributes.
- Position a pop-up form with the VisuallyAdjust operation on the Form Attributes frame.

You can also use the Move operation on the Form Layout frame to change the size of your form by moving its margins.

### Set Size and Position Attributes

The FormAttr operation can be used to change the size and position of a form.

#### **To change the size of a form or the position of a pop-up form**

1. Choose the FormAttr operation on the Form Layout frame.  
VIFRED displays the Form Attribute pop-up frame.
2. Enter the change in the appropriate field on the Form Attribute frame, as described in the following subsections, and then press the End key.

In most cases, you do not need to type the entire word or words for the option you want. The first one or two letters suffice.

### Form Size

The Sizefield on the Form Attributes frame is a display-only field and reflects the current size of the form, as determined by editing on the Form Layout frame, or in the case of pop-up display-style forms, by the VisuallyAdjust operation.

When using this information to plan the location of a pop-up form, remember to account for the additional lines and columns required for a specified border. The pop-up and its border must be small enough to entirely fit within the window.

## Screen Width of a Form

You can display a fullscreen form in one of the following widths, with accordingly larger or smaller character fonts and spacing:

- Current width
- Narrow width (usually 80 columns)
- Wide width (usually 132 columns)

The default is current screen width, as specified in the termcap entry corresponding to your TERM\_INGRES setting. The narrow width is typically 80 columns, and the wide width is typically 132 columns, depending on your screen size and type.

To create forms wider than 80 columns, or to otherwise adjust the size of a form, see [Move the Margins of a Form](#) (see page 465).

## Position Mode

The Position mode attribute is only displayed for pop-up display-style forms. Use this attribute to specify whether the pop-up style form is fixed or floating when it appears in the window.

### Fixed

If you specify this mode, you also can fill in the StartRow and StartColumn attribute fields, which specify the row and column position of the pop-up form's upper left corner. The form always pops up at that location regardless of which field is current or where the cursor is.

### Floating

Whenever you invoke the pop-up display-style form, VIFRED positions the form in the window relative to the current field (the field containing the cursor). VIFRED places the pop-up form as close to the current field as possible without obscuring it.

### To change the Position mode attribute

1. Put the cursor in the Position mode field on the Form Attributes frame.
2. Type in fi for Fixed or fl for Floating and press End.

If you choose Fixed mode, you must set the size and position of the form, as discussed in the next sections.

## **StartRow and StartColumn Attributes**

The StartRow and StartColumn attributes only display for fixed-position pop-up display-style forms.

Use the StartRow and StartColumn attributes to specify the location of a fixed-position pop-up style form. You use these two attributes to specify the screen row and column of the upper left corner of the form. For example, to place the upper left corner of the form at the 10th column of the 5th row, enter 5 in the StartRow field and 10 in the StartColumn field.

Both StartRow and StartColumn are screen-relative, not form-relative. That is, they refer to locations on the user's screen regardless of the position of any underlying forms or the position of the cursor. When you use fullscreen-style forms that are larger than your screen, you must scroll around the fullscreen form. Fixed-position pop-up style forms always appear at the same place in the window no matter how the underlying fullscreen form is displayed.

By default, the initial StartRow and StartColumn specifications are set to Row 1, Column 1.

## Move the Margins of a Form

You can use the Form Layout frame Move operation to create forms wider than 80 columns or to change the boundaries or margins of the form. When you use forms that are larger than your window, the form scrolls when you move the cursor to the edge of your window.

### To use the Move operation

1. Place the cursor on the right margin marker and choose the Move operation.

A new set of menu operations appear:

#### **Place**

Moves (contracts) the margin to a specified location within the current boundaries.

#### **Expand**

Expands (increases) the current form boundaries by moving the form's margin outward to the right or further down.

#### **Help, End**

Perform standard operations.

2. Use the Place and Expand operations in conjunction with each other:
  - a. Use Expand to extend the right or bottom margin in large increments.
  - b. Use Place to fine-tune the new margin setting, positioning the margin at the specific point you want, within the confines of the expanded margins.

Each time you use Expand or Place, the Move operation is completed and VIFRED returns you to the previous menu.

You must then use the Move operation again to perform another expand or place operation.

## Expand the Right Margin

To create a form wider than the default width (determined by the column width of your screen):

1. Place the cursor on the margin of the form. The margin is a vertical broken line containing ruler marks, if rulers have been turned on.

Typically, VIFRED uses the entire column width of your screen, which places the right margin immediately off-screen to the right. If the right margin is off-screen, you can move the cursor to it with your cursor keys.

2. Choose the Move operation on the Form Layout frame.
3. Choose the Expand operation on the Move submenu.

The Expand operation adds one-fourth of your monitor's current screen width to the form. If you use Expand once on an 80-column screen, you extend the right margin to column 100. Using it a second time extends the margin to column 120. You can choose the Expand operation as often as you like.

## Expand the Bottom Margin

Follow the same procedure to expand the bottom margin as you do to expand the right margin, except that you start by placing the cursor on the bottom margin. Each time you expand the bottom margin, the margin moves down by one quarter of your monitor's screen height in lines. If you use Expand once on a 24-line screen, you extend the bottom margin down by 6 lines.

## Place the Margin of a Form

The Place operation moves the margin inward to a position within the confines of the current margins. You cannot use the Place operation to move a margin outward from its current position, nor can you use it to move the margin so far towards the center of the form that some portion of a form component extends outside of the margins. In this case, the Place operation moves the margins as close to the desired location as possible, without having any form components extending past the margins. Under no circumstances does the Place operation squeeze components closer together.

### To use the Place operation

1. Place the cursor on the margin.
2. Choose the Move operation on the Form Layout frame.
3. Relocate the cursor to the spot where you want the margin to be, within the boundaries of the current margins.
4. Choose the Place operation to move the margin to the new location.

For example, to position the margin at column 110 on an 80-column screen, use the Expand operation to extend the margin to column 100, and again to extend the margin to column 120. Then position the cursor at column 110 and choose Place.

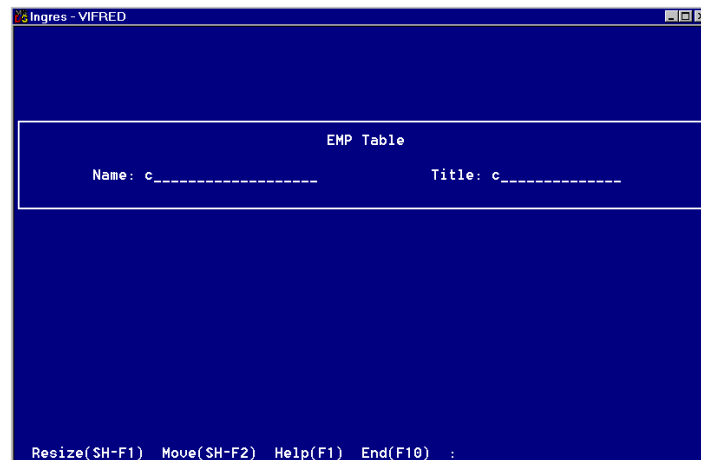
## VisuallyAdjust Operation—Visually Adjust a Form

You use the VisuallyAdjust operation on the Form Attributes frame with pop-up display-style forms only. You can:

- Preview the way a pop-up form looks in the window.
- Expand or shrink the size of a pop-up form by moving the cursor.
- Move a fixed-position pop-up to a place you designate with the cursor.

If you want, you can use this operation to change the size of a fullscreen form by changing its style to a pop-up first. You can then use the VisuallyAdjust operation to alter the form's size. When done, change the form's style back to fullscreen.

When you choose the VisuallyAdjust operation, VIFRED displays the Visual Adjust frame.





The operations on the Visual Adjust frame vary according to whether the pop-up style form is fixed-position or floating:

- **Fixed-Position** - When displayed for a fixed-position pop-up style form, such as the one shown in the above figure, the Visual Adjust frame includes the Move operation, and the pop-up form is positioned at its specified window location.
- **Floating** - When displayed for a floating pop-up style form, the Visual Adjust frame does *not* include the Move operation (because the floating pop-up has no fixed position), and VIFRED positions the pop-up style form in the center of the window.

These operations are summarized here:

#### **Resize**

Use anchor point and cursor to expand or contract the outer margins of the form.

#### **Move**

Applies to fixed-position pop-up style forms only. Moves the upper left corner of the pop-up form to the cursor location, using the Place operation on the Move submenu (same as the Move submenu on the Form Layout frame).

#### **Help, End**

Perform standard operations.

## Change Pop-up Form Size

You can use the Resize operation on the Visually Adjust frame to change the size of a pop-up style form. Using the Resize operation is easier if the form has a border, because the pop-up form margins are not displayed during the VisuallyAdjust operation. You can specify a border for the form on the Form Attributes frame, and then remove it after adjusting the size.

You can move the cursor in any direction to adjust the size of a form.

### To change the size of a pop-up style form

1. Choose the VisuallyAdjust operation on the Form Attributes frame. Choose the Resize operation on the Visual Adjust frame.

VIFRED marks one of the form corners with a plus sign. This is the anchor point. The cursor automatically shifts to the diagonal corner.

2. Move the cursor to a new location and then press the Menu key.

VIFRED redraws the form boundaries using the anchor point and the cursor's new location as the diagonal corners of the form.

If necessary, you can rotate the anchor point and cursor to different corners of the form so as to change the orientation of your modifications.

3. To shift the anchor point, press the Tab key.

Each time you press the Tab key, the anchor point and cursor rotate clockwise to a new corner of the form. By shifting the cursor and anchor point to new corners, you can expand or contract the form in any direction, provided you maintain the basic orientation of the cursor to the anchor point. VisuallyAdjust does not allow you to invert the form by moving the cursor to the opposite side of the anchor point.

4. When you have adjusted the form to the correct size, press the Menu key.

You are returned to the Visual Adjust frame menu.

5. Choose the End operation.

You are returned to the Form Attributes frame. VIFRED shows the form's new margins as dashed lines. These are the margins of the form's usable area, not counting any form border.

When you are adjusting a pop-up form's size, the form's margins must enclose all of the form's components without truncating any of them. If you attempt to reduce a form's margins in such a way that a form component would be partially or entirely outside of the form margin, VIFRED draws the new margin as close to the component as possible while still keeping the component entirely within the boundary.

## Move a Fixed-Position Pop-up Style Form

If VIFRED does not let you make a form as small as you want because a component is in the way, you must use the Move operation on the Form Layout frame to move the components of the form closer together.

### To reposition a fixed-position pop-up style form

1. Choose the VisuallyAdjust operation from the Form Attributes frame.  
The Visual Adjust frame is displayed.
2. Choose the Move operation.  
This highlights the four corners of the form and places the cursor at the upper left corner.
3. Move the cursor to the spot where you want to locate the new form's upper left corner, and then choose the Place operation to redraw the form in the new location.
4. Return to the Form Attributes frame by choosing the End operation.  
The form's new row and column location are displayed in the StartRow and StartColumn fields.

## Save a Form

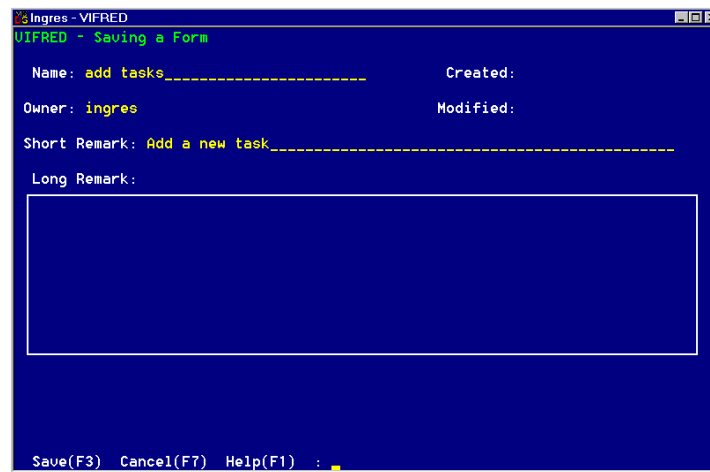
The Save operation on the Form Layout frame and other VIFRED editing frames instructs VIFRED to save a form you created or modified with VIFRED.

The Save operation is independent of exiting VIFRED. You can save a form and begin editing the same form or a different form without having to exit and reenter VIFRED.

### To save a new form

1. Choose Save on the Form Layout frame.

VIFRED displays the Save frame.



The screenshot shows a window titled "Ingres - VIFRED" with a sub-header "VIFRED - Saving a Form". The form contains the following fields:

- Name: add tasks\_\_\_\_\_
- Owner: ingres
- Short Remark: Add a new task\_\_\_\_\_
- Long Remark: [A large empty rectangular box for text entry]

At the bottom of the window, there is a status bar with the text: "Save(F3) Cancel(F7) Help(F1) : \_"

The operations available on the Save frame are as follows:

**Save**

Saves the form under the specified name with optional Short and Long Remarks.

**Cancel**

Cancels the Save operation.

**Help, End**

Perform standard operations.

2. Enter a form name in the space in the main window of the Save frame. The first character must be a letter or an underscore; case is not significant.
3. Choose the Save operation on the Save frame.  
The form is saved.

You can save the same form under different names. Each time you save a form under a different name, VIFRED creates a copy of that form and retains the original form as it was before you began to edit it. You can then edit the different copies. However, if you enter a name that is already in use, VIFRED asks for confirmation before overwriting the old form.

VIFRED stores forms in the system catalogs. VIFRED and other programs can recall forms from the data dictionary for later use.

When you save the form, VIFRED creates a QBFFName linking the form to its underlying table or JoinDef. The QBFFName is the same as the name you give the form. Thus the form, in its default state or as modified by you, is available as a query target in QBF. You can assign several different QBFFNames to the same form and use it with different tables or JoinDefs. For more information, see QBFFNames Operation (see page 479).

You can use the default QBFFName for a default form to perform QBF queries on the table or JoinDef without additional programming.

## Save Submenu

Once a form has been saved the first time, choosing the Save operation on the Form Layout frame displays a submenu rather than the Save frame. Choose one of the following operations:

### EditInfo

Displays the Save frame so you can change the form's name or Short or Long Remarks before saving the form again.

### Save

Saves the form under its current name with current Short and Long Remarks.

### Cancel

Cancels the Save operation.

### Help

Perform standard operation.

## Save Changes Pop-up

If you try to end or quit without saving a new or changed form, VIFRED displays a pop-up form like the one shown in the following figure.

The screenshot shows a terminal window titled 'Ingres - VIFRED' displaying an 'EMP Table' form. The form has fields for Name, Title, Hourly Rate, and Manager, each followed by a dashed line for input. Below the form fields is a status bar with labels 1 through 7 and 'End-of-Form'. A pop-up dialog box is centered on the screen with the title 'Save changes to form?'. It contains two options: 'Yes Save changes, then end' and 'No End without saving changes'. At the bottom of the terminal window, there is a prompt 'Select(F9) Cancel(F7) Help(F1)'.

Select Yes to save the new or changed form or No to end or quit without saving the changes.

## Destroy Forms

You can only destroy (delete) forms that you own.

### To destroy a form from within VIFRED

1. In the VIFRED Forms Catalog frame, position the cursor on the name of the form you want to delete, and choose the Destroy operation.

VIFRED prompts you for confirmation.

2. Enter **y**.

The form is deleted.

You can also use the `delobj` command on the operating system command line to delete a form or other objects.

## Edit Existing Forms

The Edit operation on the VIFRED Forms Catalog frame allows you to modify an existing VIFRED form. The Edit operation on the Form Layout frame allows you to edit the individual components, such as trim or field attributes, of the form displayed in the Form Layout frame.

### To edit an existing form

1. Access the VIFRED Forms Catalog frame.
2. Place the cursor on the name of the form you want to edit, and then choose the Edit operation.

VIFRED displays the form to edit in the Form Layout frame.

## Rename Forms

You rename forms in the VIFRED Forms Catalog frame. You can only rename forms that you own.

### To rename a form

1. Access the VIFRED Forms Catalog frame.
2. Place the cursor on the name of the form you want to rename and then choose the Rename operation.

VIFRED prompts you for the name.

3. Enter the new name for the form.

The form is renamed.

## Compile Forms

The Compile operation on the Utilities submenu generates a form definition for use with embedded SQL or other embedded query languages. Alternatively, you can use the compform system-level command to compile a form.

The Compile operation reduces start-up time when you use the form in an embedded SQL program. Consult the appropriate embedded language guide for more information.

### To use the Compile operation

1. In the Forms Catalog frame, place the cursor on the name of the form to be compiled and choose the Utilities operation.

The Utilities submenu appears.

2. Choose the Compile operation.

VIFRED prompts you for a file name under which to store the compiled form in your current directory.

3. Enter the file name and press Return.

A C language data structure describing the form is stored in the file under the name you provide.

**VMS:** This procedure produces VMS macro code. 




## Translate Compiled Form into Object Code

Before you can link the compiled form to your application, you must translate the compiled form into object code. The command for this operation depends on whether the compiled form is in C programming language format or macro format.

**Windows:** If the text file is in C language format and the file name is empform.c, the following command translates the form into object code:


```
cl -c empform
```

The compform command automatically generates the correct header file include statement for a compiled form in C language format. 


**UNIX:** If the text file is in C language format and the file name is empform.c, the following command translates the form into object code:

```
cc -c empform.c
```

If the symbol for the C language compiler at your installation is not cc, substitute the appropriate compiler symbol in place of cc.

The compform command automatically generates the correct header file include statement for a compiled form in C language format. 

**VMS:** If the compiled form is in macro format in a text file named empform.mar, the following command translates it into object code:

```
macro empform.mar 
```

## Print Forms

You can print a form by using the Print operation on the Utilities submenu, or the printscreen or printform commands at the operating system prompt.

The Print operation creates a picture of the form, which you can then print on a line printer.

### **To perform the print operation**

1. Place the cursor on the row containing the name of the form in the Form Catalog frame and choose the Utilities operation.

The Utilities submenu appears.

2. Choose the Print operation.

3. At the prompt, type the name of a file.

VIFRED places a picture of the form in that file, and appends a description of each field and the trim to the form.

4. Use your operating system print commands to print the file.

The Print operation prints all fields, including those that have been assigned the Invisible field attribute.

The FRS printscreen command sends a copy of the form currently displayed (including any character representation of data in the form) to a file or printer, depending on how the II\_PRINTSCREEN\_FILE environment variable/logical has been set. (For more information, see the *System Administrator Guide* for the system on which your database resides). Because the FRS printscreen command prints the current screen display, it does not print fields which have been assigned the Invisible attribute.

The printform command lets you print a form without entering VIFRED. It works like the Print operation on the Utilities submenu.

## QBFFNames Operation

A QBFFName is a special identifier that links a customized VIFRED data display or entry form with a particular database table or JoinDef. Choosing a QBFFName as a query target in QBF allows you to automatically display and use the customized form when accessing its associated table or JoinDef.

When you use VIFRED to create a new form based on a database table or JoinDef, VIFRED automatically assigns a default QBFFName to the new form that associates it with this table or JoinDef. The default QBFFName is the same as the name of the form. Using the QBFFNames operation on the Utilities submenu, you can change this default to another name, if you want. You can also create additional QBFFNames for this same form that link it to additional tables and JoinDefs for use in QBF. This process does not create additional *copies* of the form, but rather allows you to use *one form* with many similar tables and JoinDefs through use of its various QBFFNames.

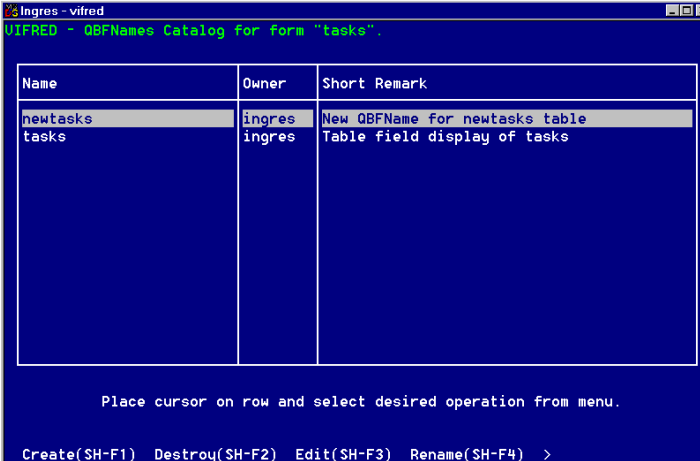
If you archive a table containing outdated information and create a new table to use instead, you can continue to use the VIFRED form created for the old table by linking it to the new table through a different QBFFName. For example, suppose a form called taskform was given the QBFFName taskform1 when it was assigned to the Tasks table. You can now give it the new QBFFName taskform2 when you assign it to the Newtasks table.

A single table or JoinDef can be associated with many different forms through the use of QBFFNames. For example, if you are using the same table or JoinDef for more than one application, you must create a separate form in each application that differs from the others in the way the fields are displayed and arranged, the assignment of mandatory fields, and the instructions displayed in the window. VIFRED assigns a different QBFFName to each separate form and links it to the common table or JoinDef.

## QBFNames Catalog Frame

The QBFNames operation applies only to the form that is currently highlighted by the cursor in the VIFRED Forms Catalog frame. To display the QBFNames Catalog for a particular form, choose the QBFNames operation after choosing Utilities on the Forms Catalog frame.

The QBFNames Catalog frame contains a list of QBFNames associated with the form identified at the top of the frame.



Name	Owner	Short Remark
newtasks	ingres	New QBFName for newtasks table
tasks	ingres	Table field display of tasks

Place cursor on row and select desired operation from menu.

Create(SH-F1) Destroy(SH-F2) Edit(SH-F3) Rename(SH-F4) >

You can use the operations available in this window to manipulate QBFNames on the list.

**Create**

Assigns an additional QBFName to the selected form.

**Destroy**

Removes the QBFName in the row containing the cursor.

**Edit**

Edits the QBFName in the row containing the cursor.

**Rename**

Renames the QBFName selected by the cursor.

**MoreInfo**

Obtains more information about the selected QBFName. The additional information includes the time the QBFName was created and a long description of the QBFName.

**Go**

Runs QBF using the QBFName.

**Help, End**

Perform standard operations.

## Assign Additional QBNames to Forms

### To assign additional QBNames to a form

1. Choose Create from the QBNames menu.  
VIFRED prompts for the new name to use with QBF.
2. Enter a name and press Return. The first character must be a letter or an underscore. For more information, see Naming and Name Use Conventions (see page 49).  
VIFRED displays a new menu.
3. Choose Table to link the form to a table, or JoinDef to link the form to a JoinDef.  
VIFRED prompts for the name of the table or JoinDef to which you want to link the form.
4. Type the name of the table or JoinDef and press Return. You can optionally enter a table as *schema.tablename*. If you omit the schema name, VIFRED assumes the owner is the same as the current user ID.  
VIFRED displays the QBFName Save frame.
5. Fill out the frame with information about the new QBFName, and then choose the Save operation.  
The new QBFName appears in the table of QBNames.

## Run QBF from VIFRED

You can test default forms by running QBF from within VIFRED.

### To run QBF from within VIFRED

1. On the VIFRED Forms Catalog frame, place the cursor on the name of the form that you want to use as a QBF query target.
2. Choose the Go operation.  
VIFRED checks to see if a QBFName with the same name as the form exists. If one already exists, it starts QBF using that QBFName as the query target.  
If VIFRED does not find a QBFName, it prompts you for the name of a table. Enter the table name and press Return to start QBF. If you press the Return key without entering a table name, VIFRED cancels the operation.

When you exit QBF you automatically return to VIFRED.

## Quit Operation—Exit VIFRED

The Quit operation exits the VIFRED. If you started VIFRED from the Ingres Menu, you return there automatically. If you started VIFRED from the operating system, you return to the operating system prompt.

You can select the Quit operation from the Form Layout frame or from the Forms Catalog frame.

If you are in the Form Layout frame, the Quit operation does not automatically save the work done in VIFRED. If you have made any changes to the form since the last save, VIFRED displays a pop-up form asking you if you want to save the changes. Select Yes to save the changes or No to exit without saving the changes.

If you are in the VIFRED Forms Catalog frame, enter Quit or End (which exits the current frame) to exit VIFRED immediately.





# Chapter 14: VIFRED Form Components

---

This section contains the following topics:

[Parts of a Form](#) (see page 485)

[Operations on the Form Layout Frame](#) (see page 487)

[Create Operation](#) (see page 488)

[Delete Form Components](#) (see page 516)

[Change the Tabbing Order of Fields on a Form](#) (see page 517)

[Move Operation—Move Components on a Form](#) (see page 518)

## Parts of a Form

VIFRED forms have two basic components:

- *Fields*—Used for entering or displaying data.
- *Trim*—Everything on the form other than fields. Trim can include instructions, general information, and lines or other decoration. Trim plays no direct part in data entry or manipulation.

## Fields on a Form

A field is a space for data entry or display on a form. It usually corresponds to a column in a database table, but also can be used for other purposes in an application, such as displaying instructions or the results of computations, or for entry of user names and passwords.

There are two kinds of fields:

- *Simple fields*—Data is displayed one item at a time.
- *Table fields*—Multiple rows and columns of data are displayed simultaneously.

If you create a form for a table, you choose one of these formats. On forms associated with JoinDefs, the JoinDef specification determines which data is displayed in simple fields and which is displayed in table fields.

Both simple fields and table fields consist of an internal name, data window, attributes, and an optional title, as described in the following sections.

## Internal Name of a Field

The internal name of a field is the name by which VIFRED identifies the field. This does not have to be the same as the field's title that users see on the form. The internal name of the field identifies the field to application procedures written in a programming language or created with Vision or Applications-By-Forms. The application code uses the field's internal name to move values to and from the field.

## Data Window

The data window is the space in a field where data is entered or displayed. The display format of the data window controls how the data appears when displayed on the form. For a data entry field, you can specify a data input template as the display format to control the way in which the user enters data in that field. If a field corresponds to a data column in a table, the display format for that field must be appropriate for the data type and length declared for the column in the database table. While data type and data display format are related, they are distinct and different entities. *Data type* identifies the kind of data. *Display format* refers to how the data is presented or entered on a form.

A single data type can have many different display formats, and some display formats can be applied to more than one type of data. For example, numeric type data can be displayed as whole integers, decimal numbers, and in scientific notation. The scientific notation display format can be used with both numeric and monetary data types.

## Field Attributes

*Attributes* are screen effects such as color, blinking, underlining, and uppercase and lowercase changes, as well as other information about each field.

## Field Title

The title of a field, which is optional, is the name of the field that appears on the form. The title provides a short description of the data that appears in the field's corresponding data window. A field's title can be different from its internal field name.

On a default form created by VIFRED from a table or JoinDef, VIFRED initially creates a field title that is similar to the field's internal name, with these exceptions: VIFRED capitalizes the first letter of every word in the title, removes any underscores to create multiple words, and adds a colon (:) at the end of the field title. You can change the field's title if you want.

## Operations on the Form Layout Frame

You create, edit, or otherwise manipulate the individual components of a form from within the Form Layout frame (discussed in detail in the chapter "Using VIFRED").

The operations on the Form Layout frame are as follows:

### **Create**

Displays a new menu that enables you to create new components on a form.

### **Delete**

Deletes the component where the cursor is positioned.

### **Edit**

Displays another menu that enables you to edit the trim or field component where the cursor is positioned.

### **Move**

Moves a component on a form. This operation displays a new menu. You can also use this operation to move a form's margins.

### **Undo**

Reverses the results of the last operation you performed. Undo does not reverse changes made with the EditAttr, FormAttr, or GroupMove operations on the Table Field frame.

### **Order**

Changes the order in which the form's fields are accessed when an end user presses the Tab key. This operation displays a new frame.

### **Save**

Displays a new menu that allows you to save your new or updated form in the database. The Save operation is independent of exiting. You can save one form and begin work on another without leaving VIFRED.

### **FormAttr**

Available only for making changes to the form as a whole.

### **Location**

Displays a message giving the current row and column position of the cursor, relative to your window boundaries, and marks the cursor location with a plus sign (+).

### **GroupMove**

Moves an entire group of components on a form at once.

**Rulers**

Turns rulers and straight edges on or off.

**Help, End, Quit**

Perform standard operations.

For detailed instructions on using these operations to manipulate the contents of forms, see the sections that follow.

## Create Operation

Choosing the Create operation while you are in the Form Layout frame displays another menu with the following choices:

**Trim**

Creates a new trim element.

**Field**

Creates a simple field.

**TableField**

Creates a table field.

**NewLine**

Inserts a blank line.

**Box/Line**

Creates a box/line feature at the cursor's location.

**DuplicateField**

Creates a field based on an existing field in the current form or in another form in the database.

**Help, End**

Perform standard operations.

## Create and Edit Trim

With the exception of boxes, each trim element is contained in a single line, which is treated as a single unit of trim. For multi-line trim elements, use the Create and Trim operations for each separate line of trim you want to create.

After creating trim, you can change the way it displays in the window by giving it attributes. For instance, you can make it blink, display in reverse video, or heighten its brightness. For instructions, see Specify Display Attributes for Trim (see page 490).

### To create new lines of trim

1. Position the cursor where you want a new trim element to be inserted.
2. Choose the Create operation on the Form Layout frame. The original cursor location appears in your window with a flashing plus sign (+).
3. Now choose the Trim operation from the Create submenu.
4. VIFRED displays the message:
5. Enter trim (press <MENU KEY> when done)
6. Type the desired trim. Use text or any other printable character.
7. The trim begins at the cursor position and continues as far as you type, up to the right margin of the form. If you need to extend the right margin, use the Move operation on the Form Layout frame.
8. Press the Menu key to complete the operation.

### To edit the text of trim

1. Position the cursor on the trim element you want to modify.
2. Choose the Edit operation on the Form Layout frame
3. Choose the Trim operation on the Create submenu.
4. Enter your corrections.

When you choose the Edit Trim operation, you enter overstrike mode. In this mode, the characters you type replace the existing character (if any) at the cursor position.

You can switch to *insert mode* by pressing the Mode key as defined for your keyboard. In insert mode, characters entered at the cursor position push existing characters to the right. Pressing the Mode key toggles you back and forth between insert and overstrike mode.

5. When you are finished editing the trim element, press the Menu key to return to the original menu.

## Specify Display Attributes for Trim

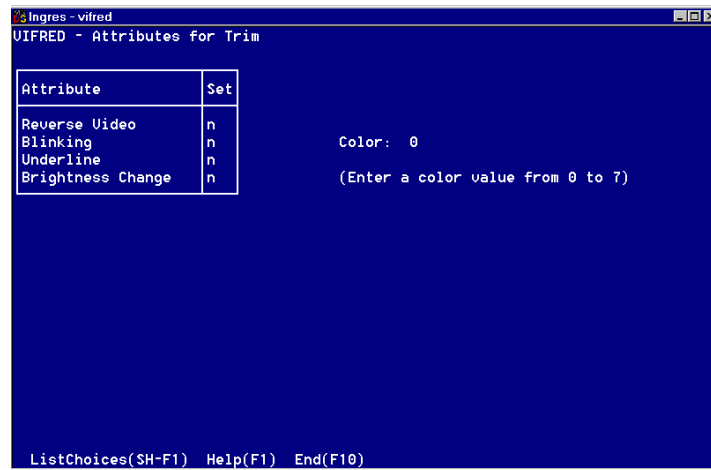
### To specify display attributes for trim

1. Position the cursor on the trim you want to modify.
2. Choose the Edit operation on the Form Layout frame.

The Edit submenu appears.

3. Choose the Attributes operation.

VIFRED displays the Attributes for Trim frame.



You can set most of the following attributes for boxes and lines, as well as for trim, unless otherwise noted:

#### Reverse Video

Reverses screen contrast from bright characters on dark background to the opposite, or vice versa.

#### Blinking

Makes the trim, box, or line blink on and off.

#### Underline

Underlines the selected trim (not for boxes or lines).

#### Brightness Change

Toggles the brightness for the trim, box, or line.

#### Color

Specifies the color of the trim, box, or line.

Except for color, these attributes are either on or off, as indicated by y or n in the column labeled Set. The default condition is n (off).

4. To change an attribute in the Set list, position the cursor on the line for that attribute and type y (yes) or n (no).
5. If you have a color display, type the appropriate color code in the Color field.

Color capabilities vary widely among monitors and can be customized by changing the color specifications in the termcap file entry for your TERM\_INGRES setting. Check the appropriate termcap file entry or your monitor documentation for color code assignment information.

6. Select End.

You are returned to the Form Layout frame.

## Create New Blank Lines on the Form

You can add new blank lines to your form to make room for additional components or to improve the form's appearance. To insert new blank lines in your form, use the Create NewLine operation.

This operation inserts new lines at the cursor location. When you create a new line, you push all components on lines below the cursor position down one line. This can have the effect of expanding the bottom margin of the form.

New blank lines are created on the Form Layout frame.

### **To create a new line**

1. Place the cursor at the location on the form where you want the new blank line inserted. You cannot create a new line if the cursor is positioned inside a multi-line field.
2. Select the Create operation.
3. Select the NewLine operation.

A new line is inserted at the cursor location and all components below the new line are pushed down by one line.

## Boxes and Lines

You can draw boxes and lines on your form for informational and aesthetic purposes. You can use boxes and lines to visually group form components, to provide borders for both normal and pop-up style forms, and to add color and graphic design elements.

You draw boxes and lines with the Create Box/Line operation. Vertical lines are actually boxes that have a vertical length of any amount and a horizontal width of only one column. Horizontal lines are boxes that have a horizontal width of any amount and a vertical length of only one column.

While you can also draw horizontal lines on your form with the Create Trim operation using keyboard keys such as the underscore character or dash, the effect is different than that achieved by using the Box/Line operation. The Box/Line operation allows you to set line attributes (described in Enhancing Boxes and Lines) and also allows you to correctly create intersecting lines. Boxes and lines have the following characteristics:

- Boxes can be of any size up to the size of your form.
- Boxes and lines are the only trim components that can occupy more than one line on your form.
- Boxes can enclose other form components.
- Boxes can be nested within each other and overlap each other.
- If a box line intersects some other form component, the other component overwrites that portion of the box or line that it overlaps. It appears as if the component is lying on top of part of the box. The form is not harmed by having a component overlap part of a box line.

### Box/Line Operation or Attribute Boxes

The Simple Fields Attributes operation allows you to automatically place a box around a simple field. Similarly, the FormAttr operation allows you to automatically place a box around a pop-up style form. These automatic box operations have no connection to creating a box with the Create Box/Line operation.

You can use the Create Box/Line operation instead of the automatic operations to place boxes around simple fields or pop-up style forms if you want. By using the Box/Line operation, you are able to specify the display attributes of the box; the automatic operations do not have that capability.

You can also use the Create Box/Line operation to place a box around a field that has been automatically enclosed by a box already, or to place a box immediately inside the border of a pop-up style form. In this case the field or form would then be displayed with two boxes, one nested within the other.



## Create a Box or Line

### To create a box or line

1. Place the cursor where you want one of the corners of the box to be and choose the Create operation.

The original cursor location that marks one corner of the box is displayed with a flashing plus sign (+).

2. Choose the Box/Line operation.

The following message displays:

Move cursor to position the opposite  
corner (press <MENU KEY> when done)

The NewLine operation inserts a new blank line into the form at the cursor location.

To create a visible line, choose the Box/Line operation.

3. Move the cursor to where you want the diagonally opposite corner of the box to be.

For example, if the plus sign marks the upper left corner of the box, move the cursor to where you want the lower right corner to be. The locations of other form components do not matter, with one exception. If you position the diagonal corner so that part of the box's line passes through another component, that portion of the box line is hidden behind the other component.

To form a *vertical line*, place the cursor directly above or below the original cursor mark. This creates a box that has vertical length and a horizontal width of 1. To create a *horizontal line*, place the cursor to the right or left of the original cursor mark. This creates a box that has horizontal width and a vertical height of 1.

4. Press the Menu key.

Lines connecting the two corners of the box appear on your form.

To set display attributes of the box, or to change its size, see the following section.

To move a box or line, use the Move operation, described in Move Operation. To delete a box or line, use the Delete operation, as described in Delete Form Components.

## Enhance Boxes and Lines

Use the Edit operation to change the size and shape of a box or line, or to specify display attributes.

### To edit a box or line

1. Place the cursor anywhere on the box or line. The cursor must be on one of the lines that forms the box, not inside the box.

On terminals that use an underscore character as a cursor, you cannot get the cursor exactly on a horizontal line or horizontal side of a box because the line is drawn in the center of that row's character cell and the underscore is at the bottom of the cell. In this case, make sure that the cursor is in the same character cell (that is, the proper row).

Place the cursor at a point on the box or line that is unique to that box or line to clearly indicate which component you want to change. For example, if you place the cursor at the point where two boxes intersect, the Edit operation cannot tell which box you must change, and arbitrarily chooses one.

2. Choose the Edit operation.

The four corners of the box (at each end of a line) display in reverse video and the Edit menu for boxes displays the following choices:

#### Resize

Expands or contracts one or more sides of the box, or horizontal or vertical line.

#### Attributes

Specifies display attributes for the box, or horizontal or vertical line.

#### Help, End

Perform standard operations.

## Resize a Box or Line

### To change the size of a box

1. Choose the Resize operation on the Edit submenu.  
The upper left corner is marked with a flashing plus sign (+). This is the anchor point. The cursor automatically shifts to the diagonally opposite corner.
2. Move the cursor to a new location. You can move the cursor in any direction.
3. Press the Menu key to redraw the box, using the anchor point and the cursor's new location as the diagonally opposite corners of the box.
4. When finished resizing the box, press the End key to return to the Form Layout frame.
5. To save your changes, choose the Save operation.

There are no restrictions on where you can move the cursor in the form. For example, you can flip a box by moving the cursor from the lower right corner (relative to the anchor point) to a new position above and left of the anchor point.

By rotating the anchor point to a different corner of the box you can change the orientation of your changes. To shift the anchor point, press the Tab key. Each time you press Tab, the anchor point and cursor rotate clockwise to a new corner of the box. By shifting the cursor and anchor point to new corners, you can expand or contract the box in any direction.

## Specify Display Attributes for Boxes and Lines

You specify display attributes for a box or line the same way you Specify Display Attributes for Trim (see page 490).

The same attribute choices are available.

When boxes with different display attributes overlap each other, the most recently created attribute overwrites the older attribute at the intersection.

## Creation of Simple Fields

If you are working with a default form, VIFRED automatically places a field on the form for every data column in the associated table or JoinDef and creates the appropriate link between the field and its data column.

If you prefer to create your own simple fields, you can do so with the Create Field operation. However, the only way you can link such a user-created field to a data column in a table is to write application code that links the field's internal name to a data column in the table. If you are not familiar with this type of programming, we recommend that you work with default forms created from JoinDefs or tables, rather than creating your own fields.

## Create a New Simple Field

To create a simple field, you must create the following components:

- Internal name of the field (see page 485)
- Display or input format of the field
- Attributes of the field (or leave them in their default state)
- Optional field title

If you create a field title, VIFRED automatically creates a default internal name (see page 499) for the field. If you do not create a field title, you can use the Attributes operation to create an internal field name.

### To create a simple field on a form

1. Position the cursor where you want to insert a new field, and then choose Create from the Form Layout frame.

The Create submenu appears.

2. Choose Field.

VIFRED displays another menu with the following operations:

#### **Title**

Creates a title for the field (optional).

#### **DisplayFormat**

Creates the data window and data display or input format for the field.

#### **Attributes**

Sets the attributes of the field.

#### **Cancel**

Cancels field creation and returns to the Form Layout frame.

#### **Help, End**

Perform standard operations.

3. Choose Title (if you are creating a field title), DisplayFormat, and Attributes in sequence to create the components of the field.

For specific instructions, see the following sections.

4. Select End.

VIFRED displays the field on the form with the specified components.

5. Use the Save operation to permanently keep a form definition. If you exit without saving the form, all your changes are lost.

## Create Your Own Internal Field Names

You must not attempt to create your own internal field name unless you are a programmer familiar with the ramifications of doing so. An alternative would be to create a field title with the Title operation and let VIFRED create a default internal field name for you. For more information, see Create Field Titles and Default Internal Names (see page 499).

### To create your own internal field name

1. Choose the Attributes operation on the Create Field submenu of the Form Layout frame.

The Attributes for Field frame appears.

2. Enter the internal name in the Internal Name field.

The name you enter must comply with VIFRED naming conventions. For more information, see Naming and Name Use Conventions.

## Create Field Titles and Default Internal Names

When you create a title for a simple field, VIFRED automatically creates an internal name for the field. The *field title* on the form has no significance other than display. The *internal field name* is the code name for the field that you use when writing application code in a programming language or with Applications-By-Forms or Vision to manipulate the form. For more information on internal field names, see Parts of a Form (see page 485).

If you do not want to create a field title, you can use the Attributes operation to create only an internal name for a field, as described in Creating Your Own Internal Field Names.

The title and internal name of a field are initially similar (up to the legal limits of internal names), but can be changed separately. This enables you to change the name of the field on a form without changing its underlying internal name or the application code that controls activity in the field. It is common practice in VIFRED to change a form's field titles, but not its internal field names. If you need to do so, you can change the internal name with the Attributes operation as described in Creating Your Own Internal Field Names or in the chapter "VIFRED Field Specifications."

### To create a title and default internal name for a new simple field

1. Choose Create on the Form Layout frame; then choose Field on the Create submenu.
2. Place the cursor where you want the first character of the field's title to appear on the form.
3. Choose the Title operation.

VIFRED displays the message:

Enter title (press <MENU KEY> when done)

4. Enter the title, and then press the Menu key.

You are returned to the previous menu.

Created are a field title, which is displayed on the form, and a field internal name (with non-legal characters automatically removed) that is not visible on the form.

## Create Data Windows and Display Formats

The data window is the part of the field where data is displayed or entered. For a more detailed explanation, see *Parts of a Form* (see page 485).

### To create a data window when creating a new field

1. Position the cursor at the point on the form in the Form Layout frame where you want the new window to be.
2. Choose the DisplayFormat operation on the Create Field submenu.

VIFRED displays the message:

Enter format (press <MENU KEY> when done)

3. Enter a display or input format.

For example, type in c10 to create a character field that can display a maximum of 10 characters, or type in a format template. For more information, see the discussion following this procedure.

4. Press the Menu key when done.

If the field is to correspond to a data column in a table, make sure that the data window display format you specify is appropriate for the data type and length declared for that column when the table was created.

**Note:** You cannot create a field with a display format that can display data from a column of long varchar, byte, byte varying, and long byte data types.

If the field is to be a data entry field, you can specify a data input template for the display format. The data input template indicates to the user how the data must be entered and causes VIFRED to check the input, character by character, to see if it matches the template. If the user's entry does not match the template format, VIFRED disallows the entry, and requires the user to either re-enter the character in that position or re-enter the data from the beginning of the field, depending on the particular input template. For more information, see *Input Masking in Data Entry Fields* (see page 501).

The following table gives some simple examples of data display and input formats. Keep in mind that some of the display format symbols are the same as data type symbols, yet they perform different functions.

Display	Description
i8	Numeric data, whole numbers only. Up to 8 digits can be entered or displayed.
f10.2	Numeric data with decimal numbers. Field can display 10 characters including the decimal point, with two places shown to the right of the decimal point.



Display	Description
i8	Numeric data, whole numbers only. Up to 8 digits can be entered or displayed.
f10.2	Numeric data with decimal numbers. Field can display 10 characters including the decimal point, with two places shown to the right of the decimal point.
c10	Character data. Field is 10 characters wide.
s'aa\ -zzzzzz'	Data input template for a part number, where a indicates an alphabetic character, z represents a numeric character, and \ - is a dereferenced hyphen (-).

## Input Masking in Data Entry Fields

VIFRED provides optional input masking for data entry fields. Input masking allows VIFRED to check a user's entry, character by character as it is being entered, against a specified data input format template. If any character does not match the template requirements, VIFRED beeps and does not put that character into the field. The user must type a valid character to continue.

To specify input masking, you need to use one of the following format templates:

- Absolute date and time templates
- Numeric templates
- String templates

You must also ensure that input masking has been turned on for that field. Do this by setting the Input Masking attribute to y (yes) in the Attributes for Field frame. Turning on input masking affects how the field interacts with the user when the application is executing and the kind of data the user can enter. If input masking is off, VIFRED uses the template to check a user's entry only upon exiting the field. Turning on input masking has no effect on fields that do not contain input templates.

## How You Create Multi-line Character Fields

You can create character fields that contain more than one line. When you enter data or display it in a multi-line field, it wraps around to the next line each time a line is filled.

To specify a multi-line character field, enter the display format as follows:

- The letter **c** for character data
- The total number of characters that can be entered in the field, followed by a period (.)
- The maximum number of characters that can be entered on any given line

VIFRED automatically figures the number of lines needed to meet your specification. For example, the display format, `c100.20`, creates a field of five lines, each of which is 20 characters long.

When creating *display-only* fields, you can use the following parameters with the `c` format to specify different types of justification:

- The `f` parameter wraps text to the next line without breaking the line in the middle of a word (it breaks *between* words instead). For example, `cf100.25` creates a field of four 25-character lines with text wrapped to the next line, with breaks between words.
- The `j` parameter right justifies text. For example, `cj100.25` creates a field of four 25-character lines with spaces entered between the words to right justify each line.

When used with either the `cf` or `cj` format, the `e` parameter preserves trailing blanks in multi-line fields; for example, `cje100.25` is the same as `cj100.25`, except that trailing spaces are preserved.

**Note:** The `f` and `j` parameters are primarily intended for use with *display-only* fields. If you use them for data entry fields, VIFRED justifies the text only after the user exits the field. Therefore, spaces between words are lost or added after the user has finished typing the entry and exits the field, which can cause some text to be truncated.

## Attributes Operation—Specify Simple Field Attributes

Attributes control the features of a field, such as:

- How the field looks on the form
- How, and if, data must be entered in the field
- Default values
- Whether data shown in the field can be changed
- Validation checks for new data
- Field scrolling
- Internal field name

While creating a new simple field, specify field attributes by choosing the Attributes operation on the Create Field submenu of the Form Layout frame. You can also specify attributes for a simple field at a later time by placing the cursor in the field and choosing the Edit operation on the Form Layout frame, followed by the Attributes operation on the Edit submenu. VIFRED then displays the Attributes menu.

For instructions on setting the Name attribute for a new field's internal name, see *Create Your Own Internal Field Names* (see page 498). For more information on changing an existing internal field name or specifying other field attributes, see *Field Attribute Specifications*.

## Edit Simple Fields

### To edit a simple field from within the VIFRED Layout frame

Position the cursor on the field you want to edit and choose the Edit operation.

The menu contains the following choices, corresponding to the elements of the field:

#### **Title**

Changes the title of the field

#### **DisplayFormat**

Changes the data window and data display or input format of the field

#### **Attributes**

Sets or changes the attributes of the field

#### **Help, End**

Perform standard operations

## Edit Field Titles

If you choose Title while editing a simple field, the cursor moves to the first character of the title. Type the new title over the existing text in the window.

By default, you are in *overstrike* mode so that the characters you type replace the characters at the cursor position. To insert characters without overstriking, press the key defined as the Mode key on your keyboard to change to insert mode.

When in *insert mode*, move the cursor with an arrow key to where you want to insert characters, and type the insertions. Increasing the width of a component shifts existing components to the right.

## Edit the Data Window

Basic data window display formats are indicated on a form by a single letter followed by underscore (\_) characters showing the width of the field. The data window of each field on a form begins with the first letter of the display format, as follows:

- **c** = character format
- **d** = date template
- **f** = floating-point format
- **g** = floating-point format
- **n** = floating-point format
- **i** = integer format
- **e** = scientific notation format

The underscore line following these letters represents the number of characters allowed in the field. For example, c\_\_\_\_\_ denotes a character display type that is 10 characters wide. For long character fields, wraparound lines can be shown by multiple lines of underscoring.

For floating-point numbers, the underscores also indicate how many decimal places to display. For example, f\_\_\_\_, denotes a floating-point display type with a maximum of four digits to the left of the decimal point and two digits to the right of the decimal point.

For date, numeric, and string input templates, the entire template appears in the field.

**To edit the data display format in the data window**

1. Place the cursor on the field and choose the Edit operation; then, choose the DisplayFormat operation.

Any single-letter display format identifier changes to the more specific letter-number-parameter display format identifier. For example, if the cursor is on a field shown as c\_\_\_\_\_, the display format identifier can change to +c10\_\_\_\_\_.

2. Type the new format over the existing format.

For example, to change a display format from 10 alphanumeric characters to 15, type c15 over c10.

3. Press the Menu key when finished.

If you change a field's data display format to a type that is not compatible with the field's data type, VIFRED automatically changes the field's data type to match the display format. This can create problems if you use the field to access data in tables. If you do not also change the data type of the underlying table, a mismatch results between the field data type and the table data type.

**Edit the Attributes of a Field****To view or change attributes in one field on an existing form**

1. Position the cursor on the field and choose the Edit operation.

A new menu appears.

2. Choose Attributes.

VIFRED displays the Attribute menu.

For more information on attributes, see Field Attribute Specifications.

## Table Fields

You can create a table field on your form that contains:

- Default columns corresponding to the columns in an existing database table or tables
- Default columns that you modify
- Columns that you specify from scratch

To create a table field, start with the instructions in [Creating a New Table Field](#), which follows. The remaining topics in this section include information on creating and editing the columns for the table field.

### Create a Table Field

#### To create a new table field on a form

1. In the Form Layout frame, place the cursor where you want the upper left corner of the table field to be, and then choose the Location operation.

The location of the cursor is marked with a plus sign (+).

2. Choose the Create operation.

The Create submenu appears.

3. Choose the TableField operation.

VIFRED displays the Table Field frame.

The screenshot shows a window titled "Ingres - VIFRED" with a sub-header "VIFRED - Table Field". Below the header, there are configuration options for the table field:

- Name of Table Field: (empty)
- Display Lines?(y/n): y
- Display Column Titles?(y/n): y
- Number of Rows to Display: 4
- Highlight Current Row?(y/n): n
- Invisible Field?(y/n): n

Below the configuration options is a table with three columns: "Title of a Column", "Column Internal Name", and "Display Format". The table is currently empty.

At the bottom of the window, there is a menu bar with the following options: "Insert(SH-F1)", "Delete(SH-F2)", "EditAttr(SH-F3)", "Move(SH-F4)", and a right arrow ">".

You can enter the following information on the Table Field form:

**Name of Table Field**

The internal name of the table field.

**Display Lines? (y/n)**

Displays or suppresses lines between rows of the table field. The default is to display lines.

**Number of Rows to Display**

The number of rows to display in the table field. The default is 4.

**Highlight Current Row? (y/n)**

Displays or suppresses highlighting of current row when cursor is in a table field. The default is no highlighting.

**Display Column Titles? (y/n)**

Displays or suppresses appearance of column titles. The default is to display column titles.

**Invisible Field? (y/n)**

Prevents on-screen display of the entire table field. You cannot place the cursor in this field with the Tab or Return keys.

**Title of a Column**

The title of the column that the user sees when the form displays.

**Column Internal Name**

The internal name by which VIFRED identifies the column. By default, the internal name is the same as the column title (with non-legal characters removed).

**Display Format**

The display format for the column's data.

In the field, Name of Table Field, type a name that complies with valid object naming rules. For more information, see Naming and Name Use Conventions (see page 49).

4. In the field, Number of Rows to Display, type the number of rows that you want displayed in the new table field or press Tab to keep the default value of 4. A table field can display up to 99 rows.
5. In the Display Lines field, choose whether to display lines between the rows of the table field. Type n (no) if you do not want to display lines or press Tab to keep the default value of y. Displaying separation lines leaves less room for data rows.

6. In the Highlight Current Row field, choose whether to highlight the current row the cursor is on whenever the cursor is in the table field. The highlighted row moves with the cursor. This Highlight Current Row function is only available through VIFRED; it cannot be turned on and off dynamically from an application.
7. In the Display Column Titles field, choose whether to display column titles. Choosing n turns off the display of column titles. If you turn off the display of column titles, you can create column titles as trim and use the Box/Line capability to create lines on the form, or you can omit column titles on your form.
8. In the field, Invisible Field, choose whether you want to prevent the on-screen display of the entire table field. Invisibility applies to the entire table field. You cannot place the cursor on an invisible field with Tab or Return.
9. To create the columns in your table field, you can either use the GetTableDef operation to create default columns, or create your own columns, as follows:
  - If you are creating a table field that corresponds to one or more tables in the database, you can choose the GetTableDef operation to create default columns based on the database tables, as discussed in Creating Default Table Field Columns with GetTableDef.
  - When you have created the columns for your Table Field, use the Insert, Delete, EditAttr, and Move operations on the Table Field frame to edit them, as you want. For more information, see the discussions following this procedure.
10. When you finish entering specifications for the new table field:
  - Select the End operation to return to the Form Layout frame. Your new table field appears.
  - To cancel the creation of a new table field, choose the Cancel operation instead of the End operation.
11. Use the Save operation to permanently retain the form definition. If you exit without saving the form, all your changes are lost.



## Create Default Columns with GetTableDef

If the table field you are creating corresponds to an existing table or tables in the database, you can use the GetTableDef operation to create default columns. VIFRED automatically creates internal names and data display types that match the columns in the corresponding table or tables.

You can then use the default columns as is, or as a starting template, rather than individually entering column titles, display formats, and internal names on the Table Field form.

### To use the GetTableDef operation

1. Place the cursor in the table field on the Table Field frame.
2. From the Table Field frame, choose GetTableDef from the menu.

VIFRED prompts you for a table name.

3. Enter the name of the table or tables, including the schema name, if appropriate.

To enter multiple table names, separate the names by a comma; for example, staff, managers, tasks. VIFRED lists all columns from each table in the table field in the order that they appear in the table, and in the order of table names that you specify.

The specifications for the table appear on the Table Field frame.

4. If you want to, edit the specifications with the Insert, Delete, EditAttr, and Move operations on the Table Field frame. For more information, see the discussions following this procedure.

## Create Columns

If you do not want to use the GetTableDef operation to create default columns for your table field, as explained in the previous discussion, you can create your own columns by entering an optional title and a mandatory internal name and display format for each column to be included in the table field.

The order in which you list columns in the Table Field frame is the order in which the columns appear across the form's table field from left to right when you are finished creating it. To change the order, see [Change the Sequence of Columns](#) (see page 514).

You must enter values for the Column Internal Name and Display Format columns on the Table Field frame. Titles of columns are optional.

### To create your own columns for the table field

1. If you want to enter an optional title, position the cursor at the first blank line in the Title of a Column section of the Table Field frame and type a title for the first column.
2. Use the Tab key to move the cursor to Column Internal Name. If you leave this column blank, VIFRED uses the column title you entered in step 1 as the internal name (excluding non-legal characters). For more information on the distinction between the internal name and the field title, see [Parts of a Form](#) (see page 485).
3. Press Tab to move the cursor to Display Format.  
The display format controls how data is displayed or entered in the field. Specify the column display or input format by typing in the appropriate format code and optional parameters.
4. Press the Menu key when done.

If the field is to correspond to a data column in a table, the display format you specify must be appropriate for the data type and length declared for that column when the table was created. For more information on how display formats and data types are related, see [Parts of a Form](#) (see page 485).

If the field is to be a data entry field, you can specify a data input template as the display format. The data input template indicates to the user how the data must be entered, and causes VIFRED to check the input, character by character as it is entered, to see if it matches the template. If the user's entry does not match the template format, VIFRED disallows the entry, and the user must either re-enter the character in that position or re-enter the data from the beginning of the field, depending on the particular input template. For more information, see [Input Masking in Data Entry Fields](#) (see page 501).

The following table gives some simple examples of data display and input formats. Keep in mind that some of the display format symbols are the same as data type symbols, yet they perform different functions.

Display	Description
i8	Numeric data, whole numbers only. Up to 8 digits can be entered or displayed.
f10.2	Numeric data with decimal numbers. Field can display 10 characters including the decimal point, with two places shown to the right of the decimal point.
c10	Character data. Field is 10 characters wide.
s'aa\~zzzzzz'	Data input template for a part number, where a indicates an alphabetic character, z represents a numeric character, and \~ is a dereferenced hyphen (-).

## Edit a Table Field

### To edit a table field

1. Position the cursor anywhere in the table field and choose the Edit operation.  
VIFRED displays the Table Field frame.
2. Type new information over the existing field contents.  
(For specific information on the components of table fields, see the table, Table Field Form Options.)
3. Select the End operation.  
You are returned to the Form Layout frame. VIFRED displays the changed table field.

To cancel your changes, choose the Cancel operation. VIFRED returns you to the Form Layout frame without recording your changes. The only exception to this is if you have used the EditAttr operation to make changes in a table field. In this case, because VIFRED has already recorded the attribute changes, you must re-enter the original values and choose End to return to the Form Layout frame with the equivalent of the original table field displayed.

## How You Add Columns

There are two ways to add columns to a table field on the Table Field frame:

- Use the GetTableDef operation to append columns from a table; then use the Delete operation to remove any unwanted columns.
- Use the Insert operation to insert or append columns.

When you choose the Insert operation on the Table Field frame, VIFRED inserts a new row before the row on which the cursor is currently resting. VIFRED then fills this row with the following default column specifications:

- NEW1 for column title, c1 for display format
- new1 for column internal name

Type the new title, column internal name, and display format over these default values.

Additional rows added using Insert are given successively numbered titles and internal names. NEW1/new1 are followed by NEW2/new2, NEW3/new3, and so on.

New data columns created with the Insert operation are not linked to and do not affect the data columns in the underlying table.

You can only link new data columns on the VIFRED form to data columns in tables by writing application code in a programming language, or with Vision or Applications-By-Forms, that links the field's internal name to the database table column. If you are not familiar with this sort of programming, we recommend that you work with default forms created from JoinDefs or tables, or add columns with the GetTableDef operation. With these methods VIFRED automatically creates the appropriate links between field and data column.

## Delete Columns

### **To delete columns from a table field in the Table Field frame**

Position the cursor on the row for the column you want to delete and choose the Delete operation.

VIFRED only removes the deleted column from the form, not from the database table.

## Edit Column Titles and Internal Name

### To edit column titles and internal names

Position the cursor on the title or internal name in the table field on the Table Field frame. Then type over any current entries you want to change.

Changing a field's internal name affects data transfer to and from the field. We recommend that you change an internal field name only if you are an experienced programmer familiar with the ramifications of such a change.

## Change Column Attributes

You can change any column attribute for one column at a time.

### To change the attributes of one column

1. In the Table Field frame, position the cursor on the name of the column whose attributes you want to change, and then choose EditAttr.

VIFRED displays the Attributes for Field frame.

2. Change the desired attributes.

Display attributes for a table-field column apply to the entire column. For example, if you set the Blinking attribute, the entire column blinks.

VIFRED changes the attributes for the column immediately.

3. Choose the End operation.

You are returned to the previous frame.

Once you have used the EditAttr operation, you cannot use the Cancel operation to cancel the changes, because VIFRED has already made them permanent. To change an attribute, choose the Edit operation on the Form Layout frame and repeat this procedure.

## Change the Sequence of Columns

You can change the sequence of columns in a table field by using the Move operation on the Table Field frame to change the order in which columns are listed in the form.

### **To change the sequence of columns in a table field**

1. In the Table Field frame, position the cursor on the row containing the specifications for the column you want to move, and then choose Move.  
A new menu appears.
2. Position the cursor to where you want to move the column specification.  
The column specification currently at that location is pushed down one row when the operation is completed.
3. Choose Place.  
VIFRED positions the column specification in the new location.

## Create Duplicate Fields

You can create a simple field or table field that is based on an existing field. The existing field can be from:

- The form being edited
- Another form in the database

### To create a duplicate field

1. On the Form Layout frame, place the cursor in the location where you want to create the duplicate field, and select the Create operation from the menu.

A blinking plus sign (+) appears where you want to create the duplicate field. The Create submenu appears.

2. Select the DuplicateField operation.

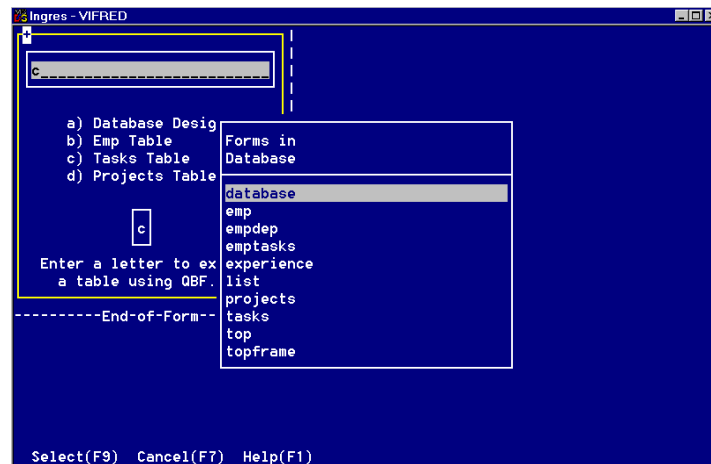
If the current form has fields, VIFRED displays a pop-up containing a list of the fields. If the current form does not have fields, VIFRED displays a list of the forms to which you have access.

The pop-up displays a maximum of 10 fields at a time. To see additional field names, scroll through the list.

3. To select a field from the current form, proceed to Step 5.

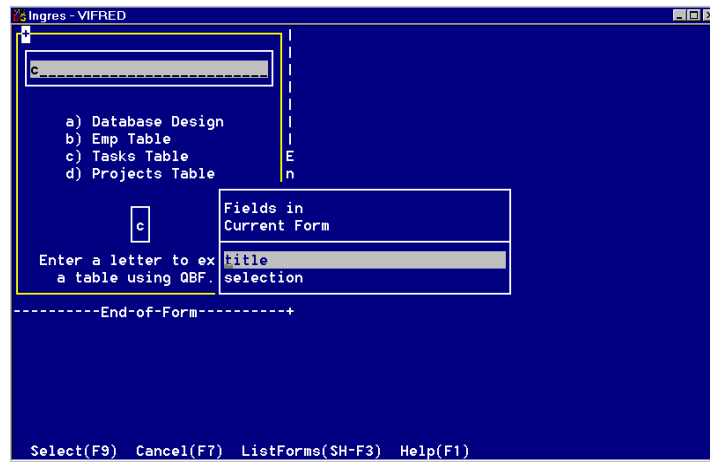
To select a field from another form, select the ListForms operation (if the pop-up listing the forms to which you have access is not displayed).

VIFRED displays the Forms in the Database pop-up.



Place the cursor on the desired form and choose the Select operation.

VIFRED displays a pop-up listing the fields on the selected form.



4. Place the cursor on the desired field and choose the Select operation.

If the internal name of the selected field is the same as the existing field, VIFRED prompts you for a new name. Type an internal name for the new field and press Return.

## Delete Form Components

Use the Delete operation to delete form components.

### To delete any component

Move the cursor to the component and choose the Delete operation.

To delete a box created with the Box/Line operation, place the cursor on one of the box's borders, not inside the box.

**Note:** You cannot delete a *line* if it contains any components.

You can immediately use the Undo operation to reverse the Delete operation if you change your mind about deleting a component.



## Change the Tabbing Order of Fields on a Form

The Layout frame editing operations include the Order operation. Use the Order operation to specify the tabbing order on the form. Tabbing order is the order in which the cursor moves from field to field on your form when someone is using it in QBF or in an application. Default tabbing order is left to right, top to bottom. However, you can use the Order operation to specify any tabbing order you want.

When you choose the Order operation, VIFRED displays the fields with their current tabbing order.

Following are the Order menu choices:

**Edit**

Enables you to type a new sequence number over the current one.

**DefaultOrder**

Restores the default order.

**Cancel**

Cancels any changes you have entered and returns to the Layout menu.

**Help, End**

Perform standard operations.

**To change the tabbing order**

1. Choose the Order operation on the Form Layout frame.  
A new menu appears. A sequence number at each field indicates its position in the tabbing order.
2. Move the cursor to the field you want to change, and then choose the Edit operation.

VIFRED displays the message:

Change sequence number (press <MENU KEY> when done)

3. Type a new, unique sequence number over the current one and press the Menu key.
4. Choose the End operation to return to editing.

If you attempt to exit the Order menu while two fields have the same sequence number assigned, VIFRED issues the error message:

Non-unique field order number(s) found

If a field has no field order number, VIFRED assigns it the next available number. VIFRED searches for fields with no field order number in the default tabbing order, left to right, top to bottom.

To restore the default order, choose the DefaultOrder operation. To cancel changes made since entering the Order menu, choose the Cancel operation.

## Move Operation—Move Components on a Form

The Form Layout frame menu provides the following two operations that move components on your form:

- Move
- GroupMove

The Move operation enables you to create forms wider than 80 columns, change the margins of the form, or reposition trim elements, boxes, vertical lines, fields, table fields, or field names on the form. This Move operation affects entire components as a whole. For example, the Move operation moves an entire table field as a unit. As an exception, you can specify that you only want to move a simple-field title or simple-field data window independent of the other part by using the Move submenu line.

Depending on whether you choose to move trim, a simple field, a table field, or the margin, various menus appear, as shown in the following table:

Operation	Menu Options
Trim	Place Left Center Right Shift
Box/Line	Place
Simple Fields	Place Left Center Right Shift Title Format
Table Field	Place Left Center Right Shift
Margins	Place Expand

Using the GroupMove operation, you can move several unrelated form components at once.

## Move a Single Component

### To move a form component

1. Position the cursor on the component to be moved and choose Move from the Form Layout menu.

The menu of Move operations appropriate for the selected component appears with the component highlighted in reverse video:

- If the cursor is on a trim component, the entire component appears in inverse video.
  - If the cursor is on a table field or box, the corners of the field or box appears in inverse video.
  - If the cursor is on either a title or data field or a simple field, both parts appears in inverse video.
2. Relocate the cursor to the place where you want the component moved and choose the Place or Shift operation, as described in the following sections.

To center or justify a component in relation to the form margin, select the Center, Left, or Right operations, as described in Centering and Justifying Components.

If you move a component to the bottom of the form and there is not enough vertical space in the form to accommodate it, VIFRED automatically moves the bottom margin of the form down enough rows to allow the component to fit. The right margin, however, does not automatically move to the right to accommodate a moved component.

If you are not satisfied with the results of any Move operation, choose the Undo operation. Because Undo reverses the last operation performed, you must use the Undo operation before performing any other operation.

## Use the Place Operation

### To use the Place operation

1. Position the cursor on the component you want to move and choose Move.
2. Position the cursor where you want to place the beginning character of the component and choose Place.

If you are moving a table field or a box, position the cursor where you want to place the upper left corner of the component.

Except for boxes and lines, if the selected position for the component would cause it to overlap another component on the form, VIFRED adjusts the position of the other component, moving it to the right if possible, and moving it down if necessary. Otherwise, the Place operation does not preserve spacing between components.

The Place operation allows boxes and lines to overlap each other. However, boxes that you create around fields as field attributes, rather than as part of the Box/Line operation, cannot overlap.

## Shift Operation

The Shift operation works the same way as the Place operation, except that it preserves spacing between components to the right.

If you shift a component to a spot where it overlaps some other component, you push the other component and all components to the right of it the same amount of space to the right. If there is not enough space to push the components to the right, VIFRED pushes one or more of them down.

## Center and Justify Components

Use the Left, Center, and Right operations to position certain components to the far left, the center, or the far right of the form, respectively.

### To use the Left, Center, or Right operation

1. Position the cursor on the component to be moved and choose the Move operation.
2. Choose the Left, Center, or Right operation. If you do not see the Left, Center, and Right operations when you choose the Move operation, they are not available for the component on which the cursor rests.

## Move Titles and Display Windows

Normally a field's title and display window move as a unit. However, you can move them separately, if you want.

When you place the cursor on a simple field and choose the Move operation, the menu displayed at the bottom of your window contains two operations not found on Move menus for other components:

### **Title**

Moves only the title.

### **Format**

Moves only the format of the data window.

### **To move the field's components separately**

- Choose the Title operation to move only the title of a simple field.
- Choose the Format operation to move only the format of the data display window of a simple field.

Once you choose either Title or Format, the Place and Shift operations apply only to the part of the field you have specified.

## Move a Group of Components at Once

You can use the GroupMove operation on the Form Layout frame to define a *bounding box* around a group of unrelated form components and move them as a unit to a new location on the form. The operation preserves relative spacing between components and from each component to the upper left corner of the defined bounding box.

Once completed, you cannot use the Undo operation to reverse a group move. During the process, you can abandon the move as described within the following procedure.

### To move an entire group of form components at once

1. Position the cursor where it defines the upper left corner of the group of components you want to move, and then choose the GroupMove operation on the Form Layout frame menu.

VIFRED marks the current cursor position at the upper left corner of the group with a blinking plus sign (+) and displays the following message:

Move cursor to position the opposite corner  
press <MENU KEY> when done)

2. Position the cursor diagonally opposite the marked corner so that it defines the lower right corner of the group of components to be moved. The components to be moved must be contained within the area defined by the upper left and lower right corners.

If you want to cancel the operation at this point, either:

- Move the cursor to its previously marked upper left corner, so that no bounding box is defined
- Define a bounding box that encompasses none of the components completely.

VIFRED highlights the group of components and displays this submenu:

Place   Help   End

3. Position the cursor at the upper left corner of the place to which you want to move the components and select the Place operation on the submenu.

Otherwise, if you want to cancel the operation, select End.

If the location you specified is the same as the starting position, or if the move would overlap other form components, VIFRED cancels the operation.

Otherwise, VIFRED immediately moves the group of components. If necessary, VIFRED expands the form to accommodate the move.

4. Choose the Save operation on the Form Layout frame.

Your changes are preserved.

# Chapter 15: VIFRED Field Specifications

---

This section contains the following topics:

[Field Attribute Specifications](#) (see page 523)

[Attributes in the Set List](#) (see page 528)

[Required and Other Attributes](#) (see page 531)

[Default Values for a Field](#) (see page 535)

[Specify a Validation Check](#) (see page 536)

[Derived Fields](#) (see page 544)

## Field Attribute Specifications

You can use the VIFRED to specify field attributes for simple fields and table field columns on a form. *Attributes* are features that can be applied to a field's data window. Attributes can affect both the visual aspects of a field and the behavior of a field.

When you create a default form or individual fields on a form, VIFRED assigns certain default attributes to those fields. You can view or change these attributes with the Attributes operation on the Edit menu.

## Default Attributes

Fields on VIFRED forms assume these default attributes:

- Fields on default forms assume the data type and internal field name attributes from the underlying column in the database table.
- Defaults for required attributes in fields on custom-created (non-default) forms are set initially by VIFRED or by the user during form creation.
- Attributes in the Set list of the Attributes for Field frame are initially set to n (no) for fields on custom-created (non-default) forms.

A field created with VIFRED on a default form assumes default attributes according to the corresponding attributes of the underlying data column. For example, if the underlying data column accepts nulls, the default attribute of the field is set to accept nulls. By default, field attributes that have no corresponding data column attribute are set to off, designated by n in the Set list on the Attributes for Field form. The default for color is zero (0).

When you create a field with the Create Field operation, the default for each of the attributes in the Set list is n (no). The default for color is zero (0). The user sets other required attributes while creating the field.

When applied to a column in a table field, the Reverse Video, Blinking, Underline, Brightness Change, and Invisible attributes affect the entire column. For example, if a column has the Underline attribute set, every data value displayed in the column is underlined.



## Set Attributes for a Field or Column

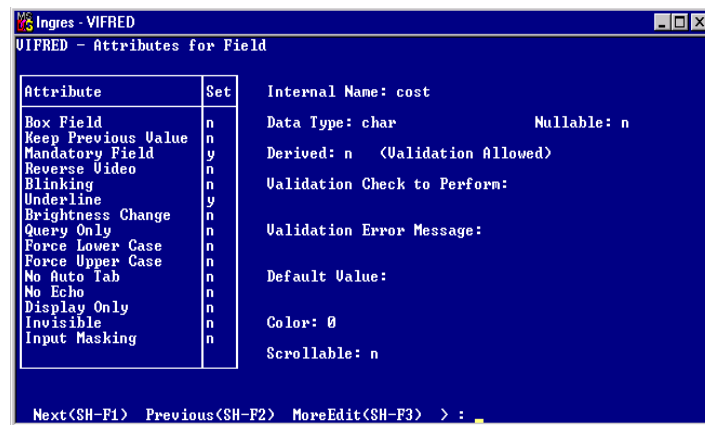
You can view or change the default attributes assigned by VIFRED.

### To view or change attributes for a particular field

1. Use one of the following, as appropriate:
  - For a simple field, place the cursor on the name of the field in the Form Layout frame and choose the Edit operation, and then choose the Attributes operation on the Edit submenu.
  - For a table field column, place the cursor on the row in the Form Layout frame that identifies the particular column for which you want to set attributes. Choose the Edit operation; then choose the EditAttr operation on the Table Field frame.

The Attributes for Field frame displays the current attributes of the selected field. The frame for a simple field varies slightly from the frame for a table field.

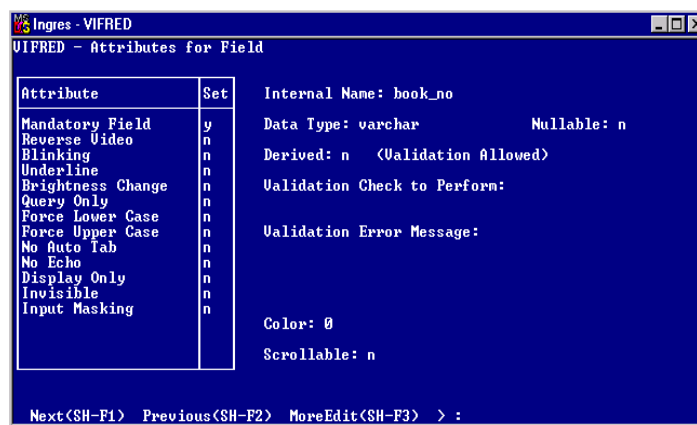
The following figure shows the Attributes for Field frame for a simple field:



The Attributes for Field frame includes an Attribute Set list on the left side of the frame. These attributes are either on or off, as indicated by y or n in the column labeled Set. The default condition is n (off). The attributes in this list vary slightly between simple fields and table fields.

This frame also includes some additional attributes that you set by typing in values, or that apply only to certain types of fields. These attributes appear on the right half of the frame. In this particular example, which shows the attributes for a text field of data type varchar, the Scrollable?(y/n) field appears near the bottom right half of the window. This field applies only to fields created for character data, and does not appear when you are setting the attributes of a numeric, date, or money data type field.

The following figure shows the Attributes for Field frame for a column in a table field:



2. To change an attribute in the Set list (see page 528), position the cursor on the line for that attribute, and type y (yes) or n (no) in the Set column.
3. To change an attribute that requires a text entry or that appears on the right side of the Attributes frame, press the Tab key to reach the appropriate field and type in the new value. For a description of each attribute specification field, see Attributes in the Set List (see page 528).
4. Choose the End operation when done.  
You exit the Attributes for Field frame and return to the Form Layout frame.
5. Save your form with the Save operation before exiting the Form Layout frame; otherwise the attributes you specified are lost.

Following are the menu choices you can use to change the attributes of additional fields without leaving the Attributes for Field frame:

**Next**

Displays the attributes for the next field.

**Previous**

Displays the attributes for the previous field.

**MoreEdit**

Displays a pop-up form that allows you to edit validation information or a derivation formula.

**ListChoices**

Displays a pop-up form with the choices available for the selected field. To make a selection, click on the choice or place the cursor on your choice and choose the Select operation.

**Help, End**

Perform standard operations.

## Attributes in the Set List

The following summarizes the attributes in the Set list on the Attributes for Field frame for both simple fields and table field columns. Some of the attributes apply only to simple fields or only to table field columns.

### **Box Field**

Automatically encloses the field in a box (simple fields only).

### **Keep Previous Value**

The last value entered in this field appears as a default value (simple fields only).

### **Mandatory Field**

Data entry is required for this field.

### **Reverse Video**

Reverses screen contrast from bright characters on dark background to the opposite, or vice versa.

### **Blinking**

Makes the field blink on and off. Microsoft Windows does not support blinking. Instead, the background color changes in intensity to represent the blinking attribute.

### **Underline**

Underlines the characters in the field.

### **Brightness Change**

Toggles the brightness of a field.

### **QueryOnly**

Can enter data in this field only if form is in Query or Fill mode.

### **ForceLower Case**

Changes any uppercase letters entered to lowercase.

### **ForceUpper Case**

Changes any lowercase letters entered to uppercase.

### **No Auto Tab**

Turns off automatic tabbing when the end of a field is reached to prevent the user from accidentally typing over the next field.

### **NoEcho**

Prevents on-screen display of data typed into this field.

**DisplayOnly**

Prevents entry of data into this field; allows only display of data. You cannot place cursor in this field with Tab or Return keys.

**Invisible**

Prevents on-screen display of a simple field or table field column. You cannot place cursor in this field with the Tab or Return keys.

**Inputmasking**

Turns input masking on or off for use with input templates in data entry field display formats. Input masking checks user input against a format template, character by character, as the user enters the data.

## An Alternative to the BoxField Attribute

Instead of using the BoxField attribute on the Attributes for Field frame to automatically enclose a simple field in a box, you can use the Box/Line operation on the Create submenu of the Form Layout frame to manually draw a box around the field. This permits you to specify display attributes for the box you draw.

## The Invisible Attribute

You can use the Invisible attribute to prevent a simple field or table field column and the data in it from being displayed in the window.

There is a difference between making an entire table field invisible and making a table field column invisible:

- Making a *table field* invisible prevents the entire table field from appearing in the window.

To make an *entire* table field invisible, enter y in the Invisible Field?(y/n) field on the Table Field frame, which is described in Creating and Editing Table Fields.

- Making a table field *column* invisible prevents only the column from appearing in the window.

To make a table field column invisible, type y next to the Invisible attribute in the Set Attributes list of the Attributes for Field frame.

When you use a form with invisible table field columns or invisible simple fields, the invisible field does not appear in the window. However, the invisible field or column *does* appear on the Form Layout frame in VIFRED and in printform output.

As with the DisplayOnly attribute, you cannot place the cursor in invisible fields with Tab or Return. The cursor skips over these fields to the next field.

## Input Masking

Input masking enables VIFRED to check a user's input in data entry fields on a character-by-character basis, as the user enters the data. If input masking is set to off, VIFRED checks the entry only when the user attempts to exit the field.

To turn on input masking, set the Inputmasking attribute to y (yes) in the Set list on the Attributes for Field frame.

In addition to turning on input masking, you must also specify an allowable input display format for the field or tablefield column. You can use only the following display formats with input masking:

- Date template
- Numeric template
- String template

For more information on display formats and input masking, see the chapter "Working with Data Types and Data Display Formats."

## Required and Other Attributes

In addition to those in the Set list, other field attributes appear on the Attributes for Field frame, as summarized here.

### **Internal Name**

(Required) Specifies the name by which a field is known to VIFRED. On a default form, the default internal name is the table column name. On a form created by the user, the default internal name is the title entered by the user (with any non-legal characters removed). The character limit is 32 (for databases compliant with ANSI/ISO Entry SQL-92 standards, see Naming and Name Use Conventions). A specification in this field is required. You initially enter this specification during the Create Field or Create Table Field procedure.

### **Data Type**

(Required) Specifies the data type of the column in the database table that corresponds to the field on the form. A specification in this field is required. VIFRED automatically supplies an initial entry when the field is created, for fields on a default form or table field columns created with the GetTableDef operation.

### **Nullable**

(Required) Indicates a NULL value is valid, if set to y. An n indicates that the NULL value is invalid. A specification in this field is required.

### **Default Value**

Enters a specified value in a field automatically when a form is used. The character limit is 50. This attribute applies to simple fields only, and is required. Not displayed for derived fields.

### **Validation Check to Perform**

Checks entered data to see if it meets the criteria established for the field. The default is no validation. The character limit is 240. Not displayed for derived fields.

### **Validation Error Message**

Error message to be displayed when data entered is not valid. The default is none and the character limit is 100. Not displayed for derived fields.

### **Derived**

If set to y, makes the field a derived field, whose value is calculated from a derivation formula that you specify.

### **Derivation Formula**

Contains the formula used to calculate the value of a derived field. Displayed only if the Derived field is set to y. This is a scrollable field, which can contain up to 240 characters and displays 50 characters at a time.

**Color**

(Required) Changes current color (on color terminals and monitors), using a numeric code. The default is zero (0) or default color. A specification in this field is required.

**Scrollable**

(Required if showing) Defines a text field as scrollable, if set t y. The default is n (non-scrollable). Displayed only for a character data type (single line, non-justified format). A specification in this field is required.

**Scroll size**

Sets the length of the scrollable field. Displayed only when the Scrollable field is set to y.

## Internal Name of a Field

The field's internal name is the name by which it is known to VIFRED.

Changing a field's internal name affects data transfer to and from the field. We recommend that you change an internal field name only if you are an experienced programmer familiar with the ramifications of such a change.

To change the name, move the cursor to the space beside Internal Name for Field, type the new name over the current name, and press Return.

## Data Type of a Field

You can change a field's data type by entering a new data type in the Data Type field of the Attributes form.

Fields on your form that correspond to data columns in tables must have the same data type as the data column in the table.

When you specify any of the character data types in the Attributes for Field frame, you cannot specify the length (number of characters) of the field, because VIFRED obtains the size of character fields from the field's display format. In this way VIFRED ensures that the data type size and format size are always the same.

Keep in mind that while VIFRED changes a field's data type to match a new display format, the reverse is not true. Thus, if you change a field's data type, you must make sure that the display format matches. When you select the Attributes for Field frame's End operation, VIFRED checks the compatibility of field data types and field display formats. VIFRED displays an error message if it finds an incompatibility. You must correct the problem before you can leave the Attributes for Field frame or edit the attributes of other fields or columns.



## Nullable Data Types

You can specify that a data type is nullable or non-nullable by entering y for nullable or n for non-nullable in the Nullable field. A nullable data type accepts a null value.

If the form field for which you are specifying attributes is to be linked to a data column in a table, the null specification on the Attributes for Field frame must match the null specification of the data column in the table.

## Color of a Field

You can change the color of a field for displays that support color and have been defined to Ingres as such in the termcap file entry designated by TERM\_INGRES. To change the color of a field, move the cursor beside Color on the Attributes form and type the code number of your choice over the current number. Color capabilities vary widely among monitors and can be customized by changing your termcap file entry. To determine the color codes for your monitor, check the termcap file entry or see your monitor documentation.

## Scrollable Field

In a *scrollable field*, only part of the data in a text field is displayed on a form. The visible display window of a scrollable field is narrower than the underlying field. VIFRED truncates the display to show only as many characters that fit in the window. Access the entire field by scrolling horizontally with the left and right arrow keys, or, if you are entering data, by continuing to type in data beyond the end of the field.

Scrollable fields allow large amounts of data to be displayed in a small area of a form. Only fields that have a data type of varchar, char, text, or c with single-row, non-justified display formats can be designated as scrollable fields.

You can use the Attributes operation to make a field scrollable. You must create the form and fields first with the Create operation, which establishes the size of the display window. And then, access the Edit Attributes frame to assign the width of the underlying field.

### To create a scrollable field on a form

1. Use the Create operation to create a form with a text field, single line, with no justification. On the field you want to make scrollable, enter the text data type and character size; for example, 5.

2. Select End.

You have established that this is a text field and the size of the display window of the scrollable field.

3. Select the Edit and then the Attributes operation, for a simple field, or the EditAttr operation for a table field in a column.

The Edit Attributes frame appears. For more information on these frames, see the preceding two figures.

The field, Scrollable?(y/n), appears at the bottom.

4. Enter y (yes) to indicate that this is a scrollable field.

The field, Scroll size, appears.

5. Type in the number of characters in the scrolling size or underlying width of the field—for example, 20.

The display window of the resulting scrollable field shows five characters, while the underlying field contains 20 characters.

You can change an existing scrollable field to a non-scrollable field at this point by changing the value of the Scrollable(y/n) field to n.

You can also change a scrollable field to a non-scrollable field at this point by changing the data type to a non-text type, as scrollable fields must be text fields.

You can apply other field attributes at any point in this process. A scrollable field can have any other attribute except NoEcho.

It is not recommend that you make a scrollable field DisplayOnly, because the end user can only read the first section of the data that is displayed in the window, and cannot access the rest of the information in the field. The only exception to this is when all of the columns in a table field are unreachable due to being DisplayOnly or QueryOnly. The end user can then place the cursor on the fields in the first column, whether it is scrollable, and scroll up and down, thus accessing all the rows in this column.

## Default Values for a Field

If you want a value to appear automatically in a field when a user sees a form, you can set a default value. Default values can be character strings or simple numeric values. For example, a point-of-sale application might contain a field for the name of the store or branch office. You can set a default to automatically enter the name of the store in that field at each location.

The default value must match the data type of the field. Default values can contain up to a maximum of 50 characters.

You can enter a default value that enters the current date in a field with a date data type when the end user tabs out of the field. There are two default date values that can be used for this purpose:

- Today enters the current date when the end user tabs out of the field.
- Now enters both the current date and the current time when the end user tabs out of the field.

## Specify a Validation Check

A validation check is set by the designer of a form. It instructs VIFRED to evaluate the data entered in the field. If the data passes the validation check, it is accepted; if it does not pass the validation check, VIFRED displays an error message and asks the user to enter the correct data. You cannot specify a validation check for derived fields.

If you leave the validation field empty (the default), no validation is performed.

**Note:** If you are writing an application that is used with a mouse, keep in mind that users can click randomly around the fields on the form. Set up field validations accordingly.

Validation checks can contain a maximum of 240 characters.

You can specify a validation check in either of these two main styles for any valid data type:

- Comparison operator validation checks
- Comparison to a set of values

### To enter or change the validation criteria

Move the cursor to the space beside Validation Check to Perform on Field on the Attribute menu and type the new string over the current one.

This field displays 50 characters at a time, but is scrollable. Use the MoreEdit operation to view or edit the entire validation string.

VIFRED does not check the syntax of a validation check until you attempt to save the form with the VIFRED Save operation. At that time, if VIFRED finds that the validation check you specified is illegal, you must correct it before you can save the form.

Note also that you can use Vision, Application-By-Forms, or an embedded query language to create validations that are more sophisticated than those you can specify with VIFRED.

## Comparison Operator Validation Checks

The format for a comparison operator validation check is a string containing one or more comparison operators. Comparison operator validation checks have the following conventional syntax:

```
fieldname comparisonoperator constant
fieldname comparisonoperator otherfieldname
fieldname IS NULL
fieldname IS NOT NULL
```

The fieldname is the internal name of the field being validated; otherfieldname is the internal name of some other field containing a value to be compared against. In a validation string for a table-field column, you use tablefieldname instead of fieldname, as follows:

*tablefieldname.internalcolumnname comparisonoperator constant*

The following table lists valid comparison operators for SQL.

Operator	Description
=	Equal to
!=	Not equal to
<>	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
LIKE	SQL string equal to

## Character String Comparisons

You can enter character strings as validation checks for fields designated as character data types. Always enclose character string validation checks in double quotation marks, except for the LIKE operator, which accepts strings with single quotation marks.

Validation checks on character strings can include pattern-matching characters. The following pattern-matching characters are valid for all operators except the LIKE operator:

Character	Description
*	(Asterisk) Matches zero or more undefined characters. For example, S* equals any character string beginning with the letter S.
?	(Question mark) Matches exactly one undefined character. For example, T?P equals TAP, TIP, TOP, etc.
[...]	Matches any characters between the brackets including ranges. For example, [ACL]* equals any string that begins with A, C, or L. ?[N-X] equals any two-character string that begins with any letter and has any letter between N and X as the second character.
%	(Percent sign) Matches zero or more undefined characters (similar to the * character described above).
_	(Underscore) Matches one undefined character (similar to the ? character described above).

Example character string validations:

```
dept = "sales"  
emptable.fname = "*son"
```

## Date and Money Comparisons

Enter comparison constants for abstract data types such as date and money in double quotation marks, as if they were character strings. For example:

```
date = "1-Jan-1988"  
projecttable.budget > "$1000000"
```

The money data type is not supported by OpenSQL. For SQL, OpenSQL and QUEL equivalents, see the appendix "Data Types."

## NULL Value Comparisons

To compare for the null value, use IS NULL. To compare for any value but the null value, use IS NOT NULL. For example:

```
salary IS NOT NULL
```

The following rules govern null values in an expression:

- A NULL value in an expression with relational or arithmetic operators causes the expression to evaluate to NULL.
- If you use the logical AND operator and either expression is FALSE, then the result is FALSE. If one expression is TRUE and the other is NULL, then the result is NULL. The result is only TRUE if both expressions are TRUE.
- If you use the logical OR operator and either expression is TRUE, then the result is TRUE. If one expression is FALSE and the other is NULL, the expression is NULL. The result is FALSE only if both expressions are FALSE.

If you use the logical NOT operator with an expression that evaluates to a null value, the result is still null.

## Numeric Comparisons

Numeric constants do not have to be enclosed in quotes. The following example requires that the number entered in the Salary field be larger than zero:

```
salary > 0
```

This example requires that the number entered in the Age field be larger than or equal to 18 and less than or equal to 70:

```
age >= 18 AND age <= 70
```

A complex numeric expression must be enclosed in parentheses to clarify the way the expression must be evaluated.

The following example requires that the number entered in the Hourly\_rate field either equal 30 or fall within the range of 40 to 50:

```
hourly_rate = 30 OR (hourly_rate >= 40 AND  
hourly_rate <= 50)
```

## Comparisons Against Other Fields

You can compare a value against whatever value is currently contained in another field. The following example requires that the name entered in the Lastname field not be the same as the name entered in the Firstname field:

```
lastname!= firstname
```

This example requires that the date entered in the Duedate field be equal to or greater than the date entered in the Orderdate field:

```
duedate => orderdate
```

## Comparison to a List of Values

By using the keyword `in`, you can compare a value against an arbitrary list of values using the following format:

```
fieldname in [list]
```

The *fieldname* parameter is the internal name of the field and *list* is the list of valid values. The values in *list* must be of the same data type as values admissible in the field. They must be separated by commas and enclosed in brackets ([ ]).

If the field has a character data type, you must enclose the character string values in double quotation marks and you can use valid pattern-matching characters. Leading blanks are not significant.

The following example requires that the name entered in the Lname field be either Jones, or Ortega, or Bridges:

```
lname in ["Jones", "Ortega", "Bridges"]
```

This example requires that the character string entered in the Address field contain the characters Bl, or Av, or end in R plus one other letter, or end with St:

```
address in ["*Bl*", "*Av*", "*R?", "*St"]
```

This example requires that the amount entered in the Salary column of the Emptable table field be either 1000, 1100, 1200, 1300, 1400, or 1500:

```
emptable.salary in [1000, 1100, 1200, 1300, 1400, 1500]
```



## Comparison to a Lookup Table

The keyword `in` can also be used to compare data entered in a field to a set of stored values in a database table. This is the easiest way to compare against a large number of values or against values that can be changed from time to time.

When your form is initialized in QBF or your own application, the current set of data from the lookup table is read into main memory. The column entries in the lookup table at the time of form initialization represent the data values used for comparison. Values added to, or deleted from, the lookup table column after the form is initialized are not reflected in the set of data used for comparison.

To compare against values in a lookup table, use the following format:

```
fieldname\tablefield.column in  
[schema.]tablename.columnname
```

If the *schema*, *tablename*, or *columnname* after the `in` keyword is a delimited identifier, you must enclose it within double quotes:

```
fieldname\tablefield.column in  
["schema name"]. "table name". "column name"
```

where:

**fieldname**

Identifies the internal name of the field being validated

**tablefield.column**

Identifies the internal name of the tablefield and column being validated

**schema**

Identifies the schema to which the table belongs and its implied owner

**tablename**

Identifies the name of a table in the current database

**columnname**

Identifies the name of a column in the specified lookup table

The following example requires that the value in the `Zipprefix` field match the values found in the `Prefix` column of the `Zip` table:

```
zipprefix in zip.prefix
```

This example requires that the names entered in the Manager column of the Emptable tablefield match names contained in the Name column of the Employee table:

```
emptable.manager in employee.name
```

The user can place the cursor in a field that is validated by a lookup table or by a *fieldname in list* validation check, as described in Comparison to a List of Values. In either case, choosing the Help Field operation at this point displays the acceptable values from the lookup table.

### Boolean Operators in Validation Checks

This chapter has discussed the following validation comparison types:

- Simple relational operator comparisons
- Comparison to a list of values
- Comparison to a lookup table

You can create more complex validation checks by using Boolean operators to connect any of these validating comparison types. The syntax for using the Boolean operators is:

```
expression OR expression  
expression AND expression  
NOT expression
```

The parameter *expression* is any of the previously discussed validation comparison types. For example, the following validation check forces a user to enter a number into the Code field that is either less than 21 or is 25, 30, or 35:

```
code < 21 or code in [25, 30, 35]
```

To allow additional flexibility, you can use parentheses to group Boolean expressions to achieve the desired semantics in the validation check. For instance, the following example forces a user to enter a number into the code field that is less than or equal to 20000 into the Salary field unless the Grade field is greater than or equal to 7. In this case, the user can enter a number up to 30000:

```
salary <= 20000 or (grade >= 7 and salary <= 30000)
```

Operator	Description
AND	Boolean conjunction
OR	Boolean disjunction
NOT	Boolean negation

Negation (NOT) has precedence over conjunction (AND) and disjunction (OR); AND and OR have equal precedence.

Some example character strings with Boolean operators are:

```
manager = "Jones" OR manager = "Ortega" AND NOT manager = "Fisher"
```

## Validation Error Message

You can specify the error message that a user sees when the user enters a value in the field that does not pass the validation check.

To specify an error message for this field's validation check, enter the text of the message in the Validation Error Message field. A validation error message can contain a maximum of 100 characters. The Validation Error Message field, which is scrollable, displays 50 characters at a time.

You can specify different validation error messages for each field that contains a validation check. If no error message is specified for a field with a validation check, VIFRED supplies a generic default message.

## Derived Fields

A *derived field* is a field whose value is based on the value of another field, called the *source field*, or constants. For example, your form could contain the following fields:

- Price (price of an item)
- Quantity Sold (total number of items sold)
- Total Amount Sold (value in the Price field multiplied by the value in the Quantity Sold field)

The Total Amount Sold field is a derived field, and the Price and Quantity Sold fields are its source fields.

Both simple fields and table field columns can be derived fields. VIFRED calculates the value of a derived field from a derivation formula, which you specify.

The derived field must be nullable when:

- One of its source fields is nullable
- The derivation formula contains an aggregate

Derived fields cannot have the following attributes:

- Default values
- Mandatory
- Force upper or lower case
- Display only
- Scrollable

## How Forms are Used with Derived Fields

The derived fields on a form are active when you view, add, or modify rows. You cannot place values directly into a derived field on a form. However, if the value of a source field changes, VIFRED recalculates the value of the derived field.

When you create a new row by moving the cursor down, the values for derived columns automatically display, if they can be calculated.

## Specify a Derived Field

The value of a field can be derived from the value of another field.

### To specify a derived field

1. On the Attributes for Field form, tab to the Derived field and type y (yes).

This figure shows the Attributes for Field frame for a simple field:

The screenshot shows the 'VIFRED - Attributes for Field' window. It contains a table of attributes and their settings, along with field metadata.

Attribute	Set
Box Field	n
Keep Previous Value	n
Mandatory Field	y
Reverse Video	n
Blinking	n
Underline	y
Brightness Change	n
Query Only	n
Force Lower Case	n
Force Upper Case	n
No Auto Tab	n
No Echo	n
Display Only	n
Invisible	n
Input Masking	n

Internal Name: cost  
 Data Type: char      Nullable: n  
 Derived: n (Validation Allowed)  
 Validation Check to Perform:  
 Validation Error Message:  
 Default Value:  
 Color: 0  
 Scrollable: n

Next(SH-F1) Previous(SH-F2) MoreEdit(SH-F3) > :

VIFRED displays the Derivation Formula field (as shown in the following figure) and removes the Validation Check to Perform field and the Validation Error Message field, because you cannot enter a validation check and message for derived fields:

The screenshot shows the 'VIFRED - Attributes for Field' window for a derived field. The 'Derived' attribute is now 'y' and 'Validation Not Allowed'. A 'Derivation Formula' field is present.

Attribute	Set
Box Field	n
Keep Previous Value	n
Mandatory Field	n
Reverse Video	n
Blinking	n
Underline	n
Brightness Change	n
Query Only	n
Force Lower Case	n
Force Upper Case	n
No Auto Tab	n
No Echo	n
Display Only	n
Invisible	n
Input Masking	n

Internal Name: cost  
 Data Type: varchar      Nullable: y  
 Derived: y (Validation Not Allowed)  
 Derivation Formula:  
 sum(Hours) \* Rate  
 Color: 0

Next(SH-F1) Previous(SH-F2) MoreEdit(SH-F3) >

2. Enter a derivation formula up to 240 characters long. Follow the Guidelines for Specifying Derivation Formulas (see page 546).

The Derivation Formula field, which is scrollable, displays 50 characters at a time. Use the MoreEdit operation to view or edit the entire derivation formula.

When you save the form, VIFRED checks for data type compatibility, including nullability, between the various sources, operators, and the derived field itself. If it finds incompatibilities, VIFRED displays an error message and does not save the form. At this time, VIFRED also checks for circular references, as described in Circular References in Derivation Formulas.

## Guidelines for Specifying Derivation Formulas

The derivation formula provides the calculation for determining the value of a derived field. When specifying a derivation formula, follow these guidelines.

A simple field can be derived from:

- Other simple fields, including other derived fields
- Aggregates of table field columns (cannot be derived directly from a table field column)
- A table field column can be derived from other columns in the same table field. A table field column cannot be derived from a simple field or aggregate value.
- You can use arithmetic operators, aggregates, and constants in derivation formulas, as explained in the following sections.

## Arithmetic Operators in Derivation Formulas

You can use the following arithmetic operators in derivation formulas (in descending order of precedence) for both simple fields and table field columns:

- (minus sign)
- \*\* (exponentiation)
- \*, / (multiplication, division)
- +, - (addition, subtraction)

You can use parentheses ( ) to change the order of evaluation.

## Aggregates in Derivation Formulas

You can use aggregate functions in derivation formulas for simple fields, but not in derivation formulas for table field columns. To derive a value for a simple field based on an aggregate of a table-field column, use the following syntax:

*aggfunction (tablefieldname[\*].columnname)*

This table lists the aggregates that can be used in derivation formulas, the data types that can be used with each aggregate, and the data type of the derived field:

Aggregate	Data type of source field:	Data type of derived field:
count	any	integer
sum	integer float money date (intervals only)	same as source
avg	integer float money date (intervals only)	same as source except for integer
max	any	same as source
min	any	same as source

If a derivation formula contains a reference to an aggregate of a table field column, and the table field contains invalid values or is empty (has no values), the value of the aggregate is:

- Zero (0), if the aggregate is count
- Null, if the aggregate is avg, sum, max, or min

If the formula contains references to other fields that contain invalid values, VIFRED blanks out the derived field.

## Constants in Derivation Formulas

You can use constants in derivation formulas for simple fields and for table field columns. For example, you can specify the derivation formula for Field\_A as:

```
2 * field_b
```

Supported data types for constants in derivation formulas are char, varchar, c, text, integer, floating point, date, and money. Specify dates, money, and non-numeric strings in single quotation marks, as:

```
'3-6-90' + field_b
```

## Dates in Derivation Formulas

You can perform the following arithmetic operations on date data types in derivation formulas:

- interval + interval = interval
- interval + absolute = absolute
- interval - interval = interval
- absolute - absolute = interval
- absolute - interval = absolute

## Circular References in Derivation Formulas

A circular reference occurs when Field\_A depends on Field\_B which, in turn, depends on Field\_A. VIFRED does not allow circular references in derivation formulas. VIFRED checks for circular references at form initialization and at form save times. If it finds a circular reference, VIFRED displays an error message. You must correct the problem before VIFRED can save the form.

## Examples of Derivation Formulas

The following formulas are valid derivation formulas for simple fields:

```
Field_3 - Field_2  
(Field_3 - Field_1) * (Field_5 - Field_4)  
Field_2 + sum(TableField1[*].Column2)  
Lastname + ',' + Firstname  
'today' + '30 days'  
'today' - '1 yrs 2 mos 3 days 12 hrs 24 mins 14 secs'
```

The following formulas are valid derivation formulas for table fields:

```
(TableField_1.Column1 + TableField_1.Column2) /2  
Order_items.price * Order_items.quantity  
Order_items.price *.90
```



## Performance Recommendations for Derived Fields

To improve performance in applications that use derived fields, follow these recommendations:

- Use aggregates primarily to derive table field aggregates of approximately 50 rows or less.
- Avoid deep nesting of dependencies; that is, do not create a derivation formula where Field\_A depends on Field\_B, which depends on Field\_C, and so forth.



# Chapter 16: Interactive Query Language Terminal Monitor

---

This section contains the following topics:

[What Is the Interactive Terminal Monitor?](#) (see page 551)

[Capabilities of the Interactive Terminal Monitor](#) (see page 553)

[Start the Interactive Terminal Monitor](#) (see page 554)

[Interactive Terminal Monitor Frame—Enter Query Language Statements](#) (see page 554)

[Query Language Statement Execution](#) (see page 558)

[Print or File Output](#) (see page 561)

[How Error Messages Are Handled](#) (see page 561)

## What Is the Interactive Terminal Monitor?

The Interactive Terminal Monitor lets you enter query language statements on a blank form in a window and then press a function key or make a choice from a menu to execute the query.

The Interactive Terminal Monitor includes a full-screen editor for entering and editing interactive query language statements. When you execute a statement, the Interactive Terminal Monitor immediately displays the result in the window. If the statement cannot be executed, a detailed error message appears. Like other Ingres tools, the Interactive Terminal Monitor also includes context-sensitive Help windows.

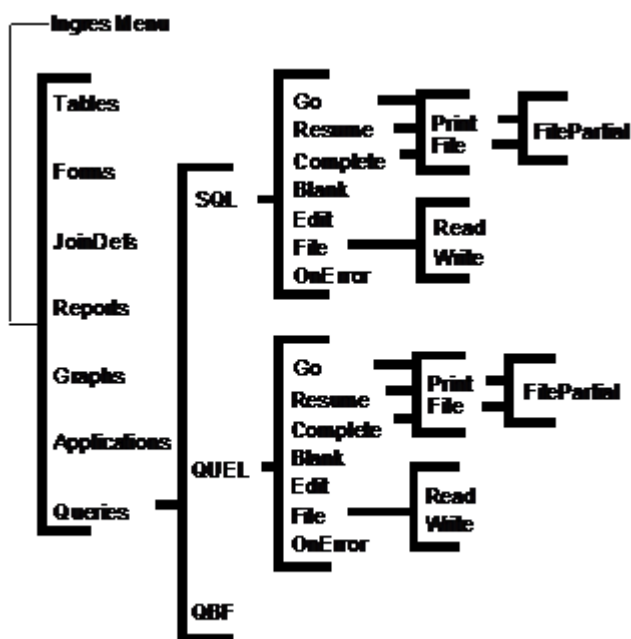
The Interactive Terminal Monitor gives you more direct and more extensive control over data management functions than do other Ingres forms-based tools, such as QBF or the Tables utility.

The Interactive Terminal Monitor supports interactive forms-based releases of the query languages, Interactive SQL (ISQL) and Interactive QUEL (IQUEL). Choosing one of the query languages from within Ingres Menu automatically starts the Interactive Terminal Monitor.

### Notes:

- On Windows, Ingres character-based utilities like the Terminal Monitor will display characters correctly only if run under the supplied Ingres command prompt, which has the correct code page and font settings.
- On UNIX, Ingres character-based utilities like the Terminal Monitor will display characters correctly only if the console window on which they are run has the correct code page set, which must match the character set value set in II\_CHARSETxx for the database.

The map in the following figure illustrates the access path and the available Terminal Monitor operations:



## Capabilities of the Interactive Terminal Monitor

Using a query language in the Interactive Terminal Monitor, you can:

- Define data structures in your database
- Manipulate data in your database

Data definition includes the creation of tables, views, and indexes. *Views* are virtual tables that provide alternative ways to access the data in database tables. *Indexes* contain keys to speed access to data in other tables. Data definition statements define the structure of data. For example, you can use query language statements in the Interactive Terminal Monitor to create a new table, naming its columns, and specifying the data type and length of each field. Or you can create a table based on an existing table, copying certain data from the old table into the new.

You can use the Interactive Terminal Monitor to manipulate data in the following ways:

- Insert (add) data
- Update data
- Delete data
- Retrieve data

These activities can be performed on one or more tables with a single statement. You can also perform global operations on a table. For example, you can add several rows of new data to a table at once, or you can update the value of a particular column in all the existing rows. In addition, you can perform computations on existing data. For example, you can add 15 percent to every employee's hourly rate.

You can narrow the scope of an update or retrieval, or any other function, to rows containing specific values in a column. You can also qualify the scope of a function with various operators and set functions. For example, you can retrieve records for employees whose names begin with S, display an average hourly rate for the employees of a certain manager, or add 15 percent to the hourly rates of everyone except managers.

For complete query language statement syntax and details of using a query language for data manipulation, see your query language reference guide.

## Start the Interactive Terminal Monitor

You can start the Interactive Terminal Monitor from the operating system or from the Ingres Menu.

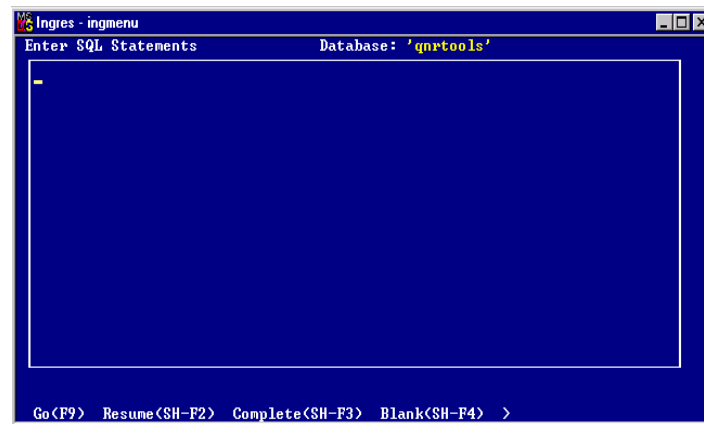
### To start the Interactive Terminal Monitor from the Ingres Menu

1. Choose the Queries operation from Ingres Menu.
2. Choose SQL or QUEL (if your system supports Interactive QUEL).

The Interactive Terminal Monitor frame (see the following figure) displays in the window.

## Interactive Terminal Monitor Frame—Enter Query Language Statements

The Interactive Terminal Monitor frame's input screen (in the following figure) consists of a blank window and a menu of operations. Enter and edit your query language statements in the frame's window, which is your workspace. The cursor initially appears in the upper left corner of the window.



The workspace is essentially a single-column table field, within which you have access to all cursor movement keystrokes, forms-based operations, and forms-based functions. You can also use the default editor on your computer system to edit your work, by selecting the Edit operation from the main menu. The default editor is determined by the ING\_EDIT environment variable/logical, as discussed in the *System Administrator Guide* for the system on which your database resides.

The input window retains all the statements you enter in the workspace unless you explicitly edit or clear them.

## Menu Operations on the Terminal Monitor Frame

The following operations are available on the Interactive Terminal Monitor frame:

### **Go**

Executes the query language statements and immediately begins displaying the results.

### **Resume**

Displays the results of the last query language statements you executed (shows the previous output window, as it appeared when you exited the output window).

### **Complete**

Executes the statements in the workspace, but does not display the results until all processing is completed. This command displays the end of the query's output.

### **Blank**

Clears the workspace of any statements you have entered.

### **Edit**

Edits the statements in the workspace with the standard system editor.

### **File**

Calls the submenu of file operations to write the information in the window into a file or read information from an existing file.

### **OnError**

Indicates whether errors terminate or continue the processing of SQL statements, and lets you change the setting.

### **LineEdit**

Displays a submenu to insert, delete, split, or join lines in the input window.

### **Help**

Gets help about this frame, including help about the syntax and usage of the interactive query language.

### **Quit**

Leaves the Interactive Terminal Monitor.

The Edit operation writes the workspace contents to a temporary file and invokes your default system text editor on that temporary file. When you finish editing the temporary file and exit from the editor, the newly edited text in the workspace on the Interactive Terminal Monitor frame displays. You can then execute the statements or continue editing or adding text in the input window.

The LineEdit operation enables you to insert, delete, split, or join lines in the input window. When you select the LineEdit operation, you access a submenu that displays the following line editing functions:

**InsertLine**

Inserts a blank line above the line on which the cursor is positioned.

**DeleteLine**

Deletes the line on which the cursor is positioned.

**SplitLine**

Divides a line into two lines at the point where the cursor is positioned.

**JoinLines**

Moves the next line to the end of the line on which the cursor is positioned.

**Help, End**

Perform standard operations.

The Help operation includes complete summaries of how to use statements in the particular database language you have chosen. If available for your terminal, you can use the key mapped to the string-search function (the Windows Find operation) to search for database language keywords to find helpful hints on syntax and usage.

## Enter Statements from a File

In addition to statements that you type directly into the workspace on the Interactive Terminal Monitor frame, you can enter database language statements into the frame from existing text files.

**To enter query language statements from a file into the workspace**

1. Choose the File operation on the Interactive Terminal Monitor input frame.

The File submenu appears.

2. Choose the Read operation.

The following prompt displays:

Enter name of file to read:

3. Type the file name. If the file to be read is not in the current (working) directory, you must include the full file name specification.

The contents of the file you named displays.

If you have already entered database language statements into the workspace, the file contents are added above the current row, as designated by the cursor location.



## Write Statements to a File

You can preserve a script you have written in the workspace by saving it to a file.

### **To write the contents of your workspace into a file**

1. Choose the File operation on the Interactive Terminal Monitor input frame.  
The File submenu appears.

2. Choose the Write operation.  
The following prompt displays:  
Enter name of file to be written:

3. Type a file name.  
The workspace contents are written to that file and the workspace contents are preserved for further editing.

## Query Language Statement Execution

After specifying query language statements in the workspace, execute them to receive output.

For example, suppose you entered the following ISQL statement in your workspace:

```
select * from staff
```

When you choose the Go or Complete operation, the following message displays while retrieving the data:

Run the request

The resulting data displays in the output window for the Interactive Terminal Monitor frame, as shown in the following figure. In this example, all rows from the Staff table in the current database have been retrieved, but not all of the data can fit in the window at the same time.

name	title	hourly_rate	manager
Alcott, Scott	Sr Programmer	\$50.00	Wolfe, Neal
Applegate, Donald	Analyst	\$51.00	Wolfe, Neal
Bee, Charles	Sr Programmer	\$43.00	Fielding, Wallace
Belter, Kris	Programmer	\$33.00	Alcott, Scott
Beringer, Tom	Programmer	\$41.00	King, Richard
Beveridge, Fern	Project Leader	\$57.00	Wolfe, Neal
Bluff, Clarence	Programmer	\$24.00	Jones, Ashley
Bridges, Debra	Sr Programmer	\$48.00	Parsons, Carol
Chung, Arthur	Programmer	\$21.00	Ortega, Julio
Downing, Susan	Programmer	\$29.00	Bee, Charles
Fielding, Wallace	Project Leader	\$47.00	Jones, Betty
Fine, Laurence	Sr Programmer	\$42.00	Jones, Betty
Hilton, Connie	Programmer	\$37.00	Bridges, Debra
Jones, Ashley	Sr Programmer	\$49.00	Turner, Russell
Jones, Betty	Project Leader	\$66.00	Turner, Russell

A Start of Output banner appears at the top of the output window if the data extends beyond the first window:

Start of Output                      Column 1/80   Line 1

The Column and Line indicators in the Start of Output banner indicate the portion of your output that is currently visible in the window, as determined by a theoretical column and row grid. Each grid column is one character wide. The first number in the Column indicator shows which output column currently appears at the left margin of your window. The number after the slash indicates the total width of the output. In this case, Column 1/80 indicates that the first column of the output appears at the left margin and the total output is 80 columns wide. The Line indicator shows the current line, at the top of the output area.

The database language statements you entered in the input window appear on the output window after the banner. Each statement line is preceded on the same line by a number and a greater than ( > ) character. In the preceding figure, the ISQL statement is preceded by 1>.

The rest of the output window contains the data and messages returned by the request. You can scroll the rows of output up and down or left and right in the output window, using your mouse or the keys mapped to the scrolling functions. To go directly to the end of the output, use the key mapped to the Bottom FRS key operation; to go to the top of the output, use the key mapped to the Top FRS key operation.

This figure shows the frame contents at the end of the output.

End of Output				Column 1/80 Line 21/42
Jones, Ashley	Sr Programmer	\$49.00	Turner, Russell	
Jones, Betty	Project Leader	\$66.00	Turner, Russell	
King, Richard	Sr Programmer	\$39.00	Beveridge, Fernce	
Lorenzo, Sue	Consultant	\$52.00	Parsons, Carol	
Moore, Holly	Programmer	\$36.00	Thompson, Howard	
Noonan, Brad	Programmer	\$25.00	Jones, Ashley	
O'Foote, Suzanne	Programmer	\$40.00	Bridges, Debra	
Ortega, Julio	Sr Programmer	\$50.00	Wolfe, Neal	
Parsons, Carol	Project Leader	\$55.00	Wolfe, Neal	
Peterson, Jean	Analyst	\$32.00	Alcott, Scott	
Randall, David	Programmer	\$34.00	Alcott, Scott	
Rolls, Richard	Programmer	\$28.00	King, Richard	
Smith, Chester	Programmer	\$22.00	Bee, Charles	
Smith, Peggy	Consultant	\$32.00	Thompson, Howard	
Stein, Frank	Programmer	\$27.00	Thompson, Howard	
Thompson, Howard	Sr Programmer	\$45.00	Jones, Betty	
Turner, Russell	Project Leader	\$53.00	Jones, Betty	
Walters, Lindsay	Analyst	\$44.00	Fine, Laurence	
Wolfe, Neal	Project Leader	\$65.00		

<32 rows>  
End of Request

Print(SH-F1) File(SH-F2) Help(F1) End(F10) :

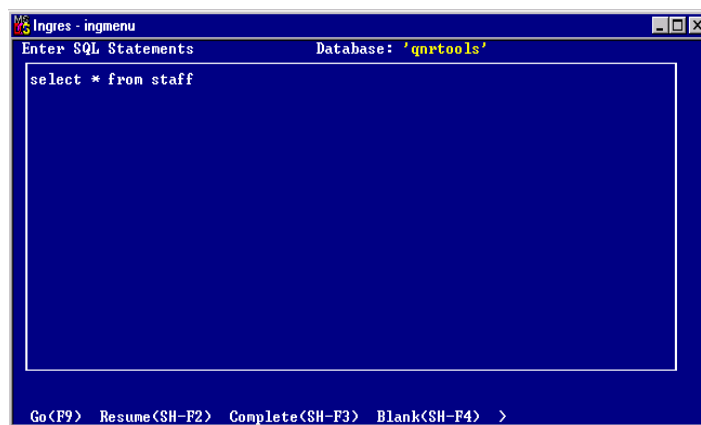
If the query runs to completion, the Line indicator in the End of Output banner at the top of the output shows the current line at the top of the output area (in this case, Line 24), as well as the total number of lines in the output, including trim, blank lines, and explanatory text (in this case, 42 lines).

An End of Request banner also appears if you:

- Execute the Go operation and all output from the request displays in one window.
- Execute the Complete operation, which runs the request to completion and displays the last portion of the output.
- Scroll the cursor to the last window of data in the output.

The File operation in the output window differs from the File operation in the input window. In the output window, the File operation sends the query *output results* to a file; in the input window it sends the *query language statements* to a file.

When you return to the input window, your original query reappears, as shown in this figure.



To edit the current request or execute it again, use the Go or Complete operations. To return to the point at which you last inspected the current output, use the Resume operation. Or, clear the workspace with the Blank operation and enter a new request.

## Print or File Output

To print or file the currently displayed window, use the printscreen function key.

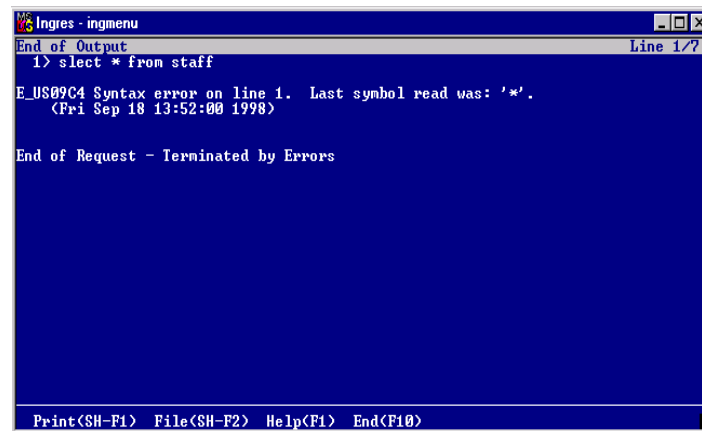
To print the results of your query or to store the results in a file, use the Print or File operations in the output window of the Interactive Terminal Monitor frame.

These are the same operations you use with RBF. See the following sections in the chapter "Producing a RBF Report:"

- Send Reports from a Screen to a File (see page 231)
- Send Reports from a Screen to a Printer (see page 232)

## How Error Messages Are Handled

If a query language statement contains errors, an error message is displayed. The error message includes information on statement syntax. The erroneous part of the statement can be pointed out, as shown in the following figure. The error message indicates that the statement select in line one was typed incorrectly.



If the OnError setting is set to CONTINUE, the query continues after the error message appears. If set to TERMINATE, then the remaining portion of the query is terminated after the error message appears. Choose End to return to the input window and edit your erroneous entry, or clear the frame with the Blank operation and correctly retype your request.

### **To change the OnError setting**

1. Choose the OnError operation on the Interactive Terminal Monitor input frame.

A pop-up window appears, offering choices of CONTINUE and TERMINATE.

2. Choose CONTINUE if you want queries to continue after an error message appears. Choose TERMINATE if you want the remaining portion of a query to terminate when an error message appear.

# Chapter 17: Working with Data Types and Data Display Formats

---

This section contains the following topics:

[Data Types](#) (see page 563)

[Data Display and Input Formats](#) (see page 572)

[Numeric Templates](#) (see page 583)

[Date and Time Templates](#) (see page 587)

[String Input Templates](#) (see page 594)

## Data Types

*Data types* control the type of information a column can contain and determine how the Data Manager carries out particular operations. By assigning the correct data type to each column in a table, you can implement and execute many operations more easily.

Data types are significant in working with a relational database because when you add or change data in a column, the new data must be of the type specified for that column when the table was created. For example, you cannot accidentally store a date in a salary field designated for the money data type.

There are four fundamental data types:

- Character (or text)
- Numeric
- Date
- Money

These data types may not all be compatible with Enterprise Access products. The character and numeric data types are broken into subsets. For example, integers can be stored as 1-byte, 2-byte, and 4-byte integers. Data types are named differently in SQL and QUEL. For more information, see the appendix "Data Types" for OpenSQL and QUEL data types.

### **c(*n*)**

Fixed-length string of up to *n* printable ASCII characters, with non-printable characters converted to blank; *n* represents the lesser of the maximum configured row size and 32,000.

### **char(*n*)**

Fixed-length string of up to *n* ASCII characters, including any non-printable characters; *n* represents the lesser of the maximum configured row size and 32,000.

### **varchar(*n*)**

Variable-length ASCII character string of up to *n* characters; *n* represents the lesser of the maximum configured row size and 32,000.

### **text(*n*)**

Variable-length string of up to *n* ASCII characters; *n* represents the lesser of the maximum configured row size and 32,000.

### **float(*n*)**

*n*-byte floating point; converted to a 4-byte or 8-byte floating point data type.

### **float4**

4-byte floating point; for numbers including decimal fractions, from  $0.29 \times 10^{-38}$  to  $1.7 \times 10^{38}$  with 7 digit precision.

### **float8**

8-byte floating point; for numbers including decimal fractions, from  $0.29 \times 10^{-38}$  to  $1.7 \times 10^{38}$  with 16 digit precision.

### **decimal**

Exact numeric data type defined by its precision (total number of digits) and scale (number of digits to the right of the decimal point). Precision must be between 1 and 31. Scale can be zero (0) up to the maximum scale.

### **integer1**

1-byte integer; for whole numbers ranging from -128 to +127.

### **integer2 or smallint**

2-byte integer; for whole numbers ranging from -32,768 to +32,767.



**integer4 or integer**

4-byte integer; for whole numbers ranging from -2,147,483,648 to +2,147,483,647.

**money**

8 byte monetary data; from -\$999999999999.99 to +\$999999999999.99.

**date**

12 bytes; dates ranging from 1/1/1582 to 12/31/2382 for absolute dates and -800 years to +800 years for time intervals.

A user-defined data type (UDT) is perceived and treated as a character string.

Some forms utilities do not support the long varchar, byte, byte varying, and long byte data types. For more information, see each specific product for a discussion of how long varchar, byte, byte varying, and long byte is handled.

## Character Data Types

Four different data types define text:

- `c`
- `char`
- `text`
- `varchar`

You enter character data as alphanumeric strings. You must enclose the string in single quotes ('). To include an explicit single quote (or apostrophe) within a character string, dereference it by preceding it with another single quote. For example:

```
'Enter your supervisor''s initials'
```

You can include double quotes within a single-quoted character string without dereferencing them:

```
'according to the "experts"'
```

In displaying a query result, SQL displays character data as alphanumeric strings without the surrounding single quotes. SQL differentiates between uppercase and lowercase data in character strings. For example, suppose you use this string in an SQL statement:

```
'Bee, Charles'
```

It is evaluated differently from any of the strings below:

```
'BEE, CHARLES'  
'bee, charles'  
'Bee, charles'
```

## C Data Type

The **C** data type consists of a string of up to *n* printable ASCII characters. It converts non-printable characters to blanks; *n* represents the lesser of the maximum configured row size and 32,000. Blanks are ignored when comparing strings of the **C** data type. For example, it treats the following two examples identically:

```
newtable and oldtable  
newtableandoldtable
```

### Char Data Type

The char data type consists of a string of up to  $n$  printing ASCII characters, including any non-printable characters;  $n$  represents the lesser of the maximum configured row size and 32,000. Between c and char, the char data type is preferred, as it is compatible with ANSI SQL.

### Text Data Type

The text data type consists of up to  $n$  characters in the ASCII extended set or blanks;  $n$  represents the lesser of the maximum configured row size and 32,000. All ASCII characters except the ASCII null character (null) or the hex \0 are allowed.

### Varchar Data Type

The varchar data type consists of up to  $n$  characters in the ASCII extended set or blanks;  $n$  represents the lesser of the maximum configured row size and 32,000. All ASCII characters are permitted, including non-printable control characters, such as the NULL character. A varchar column is defined as varchar( $n$ ) where  $n$  is the maximum number of characters stored in the column. Between text and varchar, the varchar data type is preferred as it is compatible with ANSI SQL.

## How Character Strings Are Compared

When comparing two character strings, it is important to consider the range of ASCII characters permitted by a given data type. Including non-printable characters in varchar or char comparison strings affects the outcome of the comparison.

Blanks are not ignored in comparisons by either text or varchar. However, the way blanks are handled by the two data types differs. In comparing strings of unequal length, varchar effectively adds blanks at the end of the shorter string to bring it to the same length as the longer string. The text data type does not add blanks; it considers a shorter string as less than a longer string if all characters up to the length of the shorter string are equal.

As an example of the way this affects comparisons, consider the following two strings:

```
abcd\001
abcd
```

Assume that \001 represents one ASCII character, Control-A. If these are compared as text, the first string is greater than the second. However, if they are compared as varchar, the first string is less than the second, because the blanks added by varchar to the shorter string have a higher ASCII value than 001.

The following table summarizes the various character data types, where *n* represents the lesser of the maximum configured row size and 32,000:

Aspect	c	char	text	varchar
Length	1-n	1-n	1-n	1-n
Nullable	Yes	Yes	Yes	Yes
Legal characters	Printable characters only	All ASCII characters	All ASCII characters except nul and \0	All ASCII characters
Storage	Fixed length	Fixed length	Variable length	Variable length
Blanks significant?	No	Yes	Yes	Yes
Comparison of short to long	Blanks ignored	Blank padding of short length to long	No padding; blanks included	Blank padding of short length to long

## Date Data Type

Date data type columns hold absolute dates, absolute times, or time intervals. Date data types can contain any valid date between January 1, 1582 and December 31, 2382. When used in database language commands, date data types are enclosed in quotation marks as are text types.

An absolute date is a time, such as 12:30 on November 13. There are many ways of expressing absolute dates and time.

For example, you can express the date December 25, 1998, in any of the following ways:

mm/dd/yy	12/25/98
dd-mmm-yyyy	25-Dec-1998
mm-dd-yyyy	12-25-1998
yyyy.mm.dd	1998.12.25
mm/dd	12/25
mm-dd	12-25
mmdyy	122598
mm/dd/yy hh:mm:ss ampm	12/25/98 10:30:30 am

You can express the time 10:30 a.m. as:

hh:mm:ss ampm tz	10:30:00 am pst
------------------	-----------------

You can also use `now` or `today` to mean the current date and time (`now`), or current date only (`today`).

Various conventions for date and time are recognized. For example, it assumes a 24-hour clock unless you specify `p` or `p.m.` The program adjusts dates and times to the designated time zone, such as Pacific Standard Time.

For information on how to specify various date display formats, see [Data Display and Input Formats](#) (see page 572).

## International Conventions for Date and Time

International date and time conventions can also be adapted. For example, dates in Sweden and Finland are expressed as a year, followed by the month, followed by the day. The standard German date format can be expressed as *ddmmyy*, *ddmmyyy*, *dd.mm.yy*, and *dd.mm.yyyy*. The date format is set by the system variable `II_DATE_FORMAT`.

The time zone is set with the system variable, `II_TIMEZONE_NAME`. These dates are stored in Greenwich mean time. The conversion between Greenwich and your local time is a value set with `II_TIMEZONE_NAME`. For more information, see the *System Administrator Guide* for the system on which your database resides.

## Relative Times and Dates

Relative time and date intervals are also handled, which are units of time not fixed as absolutes. An example of a relative time interval is:

5 years 8 months 14 days

You can use the interval and date functions to carry out operations on dates in SQL or in the Interactive Terminal Monitor. The interval function calculates the difference between two dates. The date conversion function transforms strings into dates.

## Floating Point Data Type

The floating point data type consists of numeric data with decimal points and/or exponents.

Floating point data can contain an integer portion, a decimal point, and a fractional portion and/or scientific notation, in this format:

[+|-]{digit}[.digit{digit}][e|E[+|-]{digit}]

For example:

2.3e-02

You can change the default decimal point indicator, the period (.), to another character, such as the comma (,), used in European decimal format, by using the `II_DECIMAL` system variable. For more information, see the *System Administrator Guide* and the `set decimal` command in your query language reference guide.

On output, a mantissa of 1 is supplied when an exponent is missing a mantissa.

The floating point range and precision vary greatly among the various operating systems and hardware.

## Decimal Data Type

A numeric constant is considered to be a decimal data type if it has a decimal point but no exponent, and you are working with Ingres.

Under the following circumstances, a decimal data type is treated as type float if:

- The total number of digits exceeds 31
- You have set the `II_NUMERIC_LITERAL` environment variable/logical to float

## Integer Data Types

Integer data types are numbers containing no fractional part. For example, ages can be designated as integer data types. Integers are classified as 1-byte, 2-byte, and 4-byte, depending on the range of numbers to be stored within a given column.

If you enter integer data outside the range of an integer4 data type, the integer is converted to floating point. For instance, an entry of 5000000000 exceeds the range of an integer4, and so is stored as float.

## Money Data Type

The money data type contains decimal currency data. Great flexibility is provided with regard to the money data type. Using system variables, you can adopt the conventions of your local currency. These include:

- Currency symbol
- Decimal separator
- Number of decimal places

For more information, see the system administrator guide for the system on which your database resides.

On input and output, money data is rounded to dollars and cents, pounds and pence, or other currency units. Arithmetic operations on money values retain precision as established by the `II_MONEY_PREC` environment variable/logical.

You can enter up to 14 digits into a column specified for the money data type. When displayed, money data can appear as up to 20 characters, allowing room for commas, currency signs, and place holder symbols.

## Nulls

Nullability is an attribute of stored data. A null represents inapplicable or missing data. It is most useful when working with numeric data used in aggregate computations such as totals and averages. An example of nullable text is an unfilled Home Address field on a personnel form for a new employee when not all information is known.

You can determine which of your tables' columns are to be nullable. In this way, missing data, default values, or empty strings cannot be assigned a value of zero.

It is important to distinguish between ASCII NULL character and column nullability. The NULL character, ASCII octal \000, is a value permitted in certain character data types. Nullability is a condition or status that is available to all the data types. A user assigns nullability to a specific column in a table while creating the table.

## Data Display and Input Formats

Data display formats offer alternative ways to display or enter data that is stored in the database as a particular data type. The display format controls the width of the data display window on a form or the width of a field or column in a report.

In RBF, you specify the data display format for each column or field in a report on the Report Layout Frame. In VIFRED, you specify the data display format when you create the data window for a simple field or when you specify columns on the Table Field frame. This section explains the relationship between data types and data display formats, and describes data display and input formats in detail.

## Data Types and Display Formats

When you create a column in a database table, you assign a *data type* to control the way data is stored in that column. Sample data types for Ingres databases include char, integer, date, and money. When you create a report or form, you can specify a *display format* to determine the way that data appears on a form or in a report.

You can assign a display format for data stored in a database table as a given data type. For example, a number stored as an integer data type can be displayed as a whole number, in decimal notation, in scientific notation, or as a right-justified number. The data display format must be of a type that is compatible with the data type for the corresponding column in the database table.



Even though some data types and display formats have the same name and abbreviated symbol, they perform different functions. For example, there is a floating point (f) data type and a floating point (f) display format. You can display a floating point data type in formats other than the floating point format. You can also use the floating point display format for numeric data stored as data types other than floating point.

Specify any display format that is appropriate for the data type in both display-only and data entry fields. After the user enters data in a data entry field as instructed by the application, the data is redisplayed in the specified format. An *input masking format*, or *input template*, is a special display format you can use in data entry fields only. It controls how a user enters data on a form, character by character. Date, numeric, and string templates are allowed as input masking formats in data entry fields.

The following table lists the valid display and input formats for any data type in each data type category. The letter in parentheses is the code or symbol by which you specify the display format.

<b>Data Type Category</b>	<b>Valid Display and Input Formats</b>	
	<b>Display Formats</b>	<b>Input Masking Formats</b>
Character	Character (c)	String template
Numeric	Integer (i) Floating Point (f) Scientific (exponential) Notation (e) Decimal or Scientific (g) Numeric template	Numeric template
Money	Right-justified decimal or scientific (n) Numeric template	Numeric template
Date	Character (c) Date (d)	Date template (d), with some restrictions

## Display Format Syntax

The display format symbols are often combined with numbers designating the character width of the field. For example, the display format f6 denotes a floating point display format containing up to six digits. Similarly, f6.2 denotes a floating point display format of up to six whole numbers and a maximum of two decimal digits.

The following table shows the format symbol, compatible data types, and syntax for each display format, and briefly describes the function of each:

Format Symbol	Data Type	Display Syntax	Display Format Description
c	character date	[+ * -]c[f j] [e]n[.w]	Character format, which determines the size and shape of character fields.
d	date	[+ * -]d 'template'	Date template, specifying an absolute date and time or time interval.
e	decimal float integer money	[+ * -]ew[.d]	Scientific (exponential) notation format, which displays numbers as [-]m.dddddE[e[+ -]ppp] in which <i>m</i> represents the mantissa, <i>d</i> represents a digit in the fraction, and <i>p</i> is an exponential digit. When specifying the width of the field ( <i>w</i> ), be sure to include five spaces for E+ppp.
f	decimal float integer money	[+ * -]fw[.d]	Floating point format, which displays a number in standard decimal format.
g	decimal float integer money	[+ * -]gw[.d]	Displays the number in floating point format (f) if there is room; otherwise, displays it in scientific notation (e). Data is first formatted so that the decimal point will align in both f and e format. (Note that this reduces the space available for display in floating point format compared to n

Format Symbol	Data Type	Display Syntax	Display Format Description
			format). The data is then justified as specified (default is right justified).
i	decimal float integer money	[+ * -]iw	Integer format, which displays the number as an integer.
n	decimal float integer money	[+ * -]nw[.d]	Displays the number in floating point format (f) if there is room; otherwise, displays it in scientific notation (e). Unlike g format, data is not first formatted so that the decimal point will align in both f and e format. The data is justified as specified (default is right justified).
numeric template	decimal float integer money	[+ * -]['{c}']	Numeric template containing character codes indicating the characters allowed in each position.
string template	c char text varchar		String character template containing character codes indicating the characters allowed in each position.

RBF and VIFRED can use the same data display formats as Report-Writer, with two exceptions:

- B format is not allowed
- Variable length format c(0) is not allowed

You can use either uppercase or lowercase letters for symbols in most data display formats. However, case can be significant in date templates (see page 587), numeric templates (see page 583), and text templates (see page 594).

The following are the optional parameters you can use with the data display formats shown in the preceding table:

**+|\*|-**

Precedes the format letter to indicate right (+), center (\*), or left (-) justification. If no sign is used, the default is left justification for both character and numeric columns.

**c**

Any one of the special numeric or character template code characters allowed in that type of template. For definitions of these codes, see the sections on character and numeric templates later in this chapter.

**d**

Specifies the precision (the number of digits to display after the decimal point) in any of the numeric formats. To allow room in the display field for the decimal point and a plus (+) or minus (-) sign, this number must be at least 2 less than the value of w (maximum column width).

**e**

When used with the wrapping character options (cfe and cje), preserves trailing white space on every line of a multi-line text string, which is otherwise trimmed by default. If you use it with the cj format, the margin justification includes the significant white space. In some instances, this makes the text appear unjustified. For more information, see *How You Create Multi-line Character Fields*.

**f**

When used with the c format for multi-line text strings, wraps text to the next line, with breaks occurring between words. Most useful in display-only fields. Results in data entry fields can be unpredictable.

**j**

When used with the c format for multi-line text strings, right-justifies the text, with breaks between words, and pads the line with blanks, so that all lines end evenly at the right margin. Most useful in display-only fields. Results in data entry fields can be unpredictable.

**n**

The maximum number of characters allowed in the data window.

**w**

When used in a character format (c) designation, specifies the number of characters displayed on each line. If n is greater than w, the data window occupies more than one line. The default of w is the value of n.

When used in a numeric format (e, f, g, i, or n), specifies the maximum column width, including all punctuation, symbols, decimal points, and decimal digits. For example, to display \$15.44 the w value must be at least 6 (for example, f6.2). For columns in e, g or n format, be sure that w is wide enough for representations in scientific notation.

For example, i9 specifies an integer format nine characters wide, and +i9 specifies a right-justified integer format nine characters wide.

To specify a floating point format for a column that is a total of nine characters wide, and for which you want to show two decimal places, enter f9.2. Allowing for the decimal point and a plus or minus sign, a display format of f9.2 permits a maximum of five digits to the left of the decimal point and two to the right of it.

## Default Data Display Formats

The following table lists default display formats for each SQL data type. For more information, see the appendix "Data Types" for OpenSQL and QUEL data type equivalents.

SQL Data Type	For Forms and Most Reports	For Block Style Reports
c1 - c35	c1 - c35	c1 - c35
c36 - cx	cj0.35	cj0.35
char(1) - char(35)	c1 - c35	c1 - c35
char(36) - char(x)	cj0.35	cj0.35
text(1) - text(35)	c1 - c35	c1 - c35
text(36) - text(x)	cj0.35	cj0.35

SQL Data Type	For Forms and Most Reports	For Block Style Reports
varchar(1) - varchar(35)	c1 - c35	c1 - c35
varchar(36) - varchar(x)	cj0.35	cj0.35
integer1	f6	f10
smallint (integer2)	f6	f10
integer (integer4)	f13	f10
decimal (31.31 - 31.0)	Based on size. For example, decimal (5.1) defaults to f7.1, allowing additional digits for the decimal point and an optional plus (+) or minus (-) sign.	
float4	n10.3	f10.3
float (float8)	n10.3	f10.3
date	c25	c25
money	\$-----.nn	\$-----.nn

**Note:** 'x' represents the lesser of the maximum configured row size and 32,000.

**Note:** All character data types are fully supported in non-Ingres databases accessed through Enterprise Access products, in which case the column size limit can be greater than 2000 bytes. Other data types may not be supported by all Enterprise Access products.

If your computer supports the IEEE standard for floating point numbers, the range and accuracy of floating point numbers reflect that standard.

## How Character Data Is Displayed

Character data includes any of the character data types (`varchar`, `char`, `c`, `text`). You can display character data types only in:

- Character (c) display format (in display-only and data entry fields)
- String template display format (in data entry fields only)

For the character (c) data display format, the default is left justification. To justify the contents of a single-line field to the right, left, or center, precede the data display symbol with a plus sign (+), asterisk (\*), or minus sign (-), respectively. For example, entering `+c5` specifies a right-justified text field of five or fewer characters. You can use justification symbols with any data type.

If you specify a value for *w* as well as *n*, you can display text in column format. When character fields contain more than one line, the first line of the field is filled with characters and the second line is filled. This can produce line breaks in the middle of a word. For display-only fields, you can use the *f* or *j* option to specify that lines must break at the end of a word instead of in the middle. For example, `cf80.20` specifies a text field containing a total of 80 characters with no line containing more than 20 characters and with breaks between words.

To right justify the contents of a display-only multi-line character field, use the *j* flag. The *j* flag performs the same function as the *f* flag, except that it pads the line with blanks between words to make the right margin of each line come out even, like a column of text in a newspaper.

The *f* and *j* flags are most useful in *display-only* fields. If used in a *data entry* field, the word wrap or justification operation is performed only after the user has exited the field. This can cause truncated text or other unpredictable results, because a word can wrap to the next line or the line is padded with extra blanks after the user has already exited the field.

## How Numeric and Money Data Is Displayed

For the money and various numeric data types, a default display format and field width is set that is appropriate for the data type and width of the corresponding database column. If these defaults do not meet your needs, you can edit the data display format accordingly.

However, if you make the width of a column on a report or a data window in a form too small for the value that is to be printed or displayed there, the field is filled with asterisks. For example, if you attempt to print or display the value 667298 in a field with a data display format set at i5, five asterisks are printed or displayed in the field instead.

When you use Ingres programs to enter data into a table from a field that has a numeric display format, such as integer (i), the value entered is checked to make sure it can be stored correctly in the corresponding database column. For example, the value for an integer1 column must be between -127 and +127. In customized applications, this checking is not always automatic; you may need to use your application code to perform the check.

For floating point and decimal numbers, which have decimal fractions, you can specify the number of places to the right of the decimal point. When the data is displayed, it rounds the value to fit the number of specified places, if necessary. To specify scientific notation, use the format symbols e, g, or n.

Make sure that the display format is wide enough to include symbols such as plus (+) and minus (-) if your data uses them. When using a form for queries, you can make the display window wider than the underlying column to allow two extra spaces for comparison operators.

You can also use a numeric template for the specialized display of numeric data. Specify characters in the template that tell Ingres what to do with each digit in the data that is being displayed or entered. When using a numeric template, the width of the field on a form or in a report is determined by the number of characters in the template. For example, the template \$\$\$,\$\$.nnDb allows for a maximum of 6 digits to the left of the decimal point plus a comma, and 2 digits to the right of the decimal point plus a two-character Credit/Debit symbol. For more information, see Numeric Templates (see page 583).



## How Date Data Is Displayed

A *complete absolute date* includes day, month, year, and time according to a 24-hour clock. The default display format for a date data type is c25, which displays absolute dates in a field 25 characters wide, in the format:

Mon Apr 01 1988 22:50:43

If the date to be displayed does not include the time of day, you can specify a correspondingly smaller display width for the field.

By using a date template such as d'SUN Feb 3 1901' to specify the display format, you can display dates and times in a wide variety of formats. For more information, see *Date and Time Templates* (see page 587).

## Format Templates

A format template is a picture of how you want the data to look when it is displayed or entered in a single-line field.

For numeric and string data types, a format template consists of character set definitions and special characters. These represent the type and order of the characters that are displayed, or the valid characters that the user can enter in that field. To the system, each character set definition in the template represents one or more characters to be displayed or validated, depending on whether the field is a display-only or data entry field. To the user, the template provides visual clues about the type and format of the data that must be entered in a data entry field.

For date data types, a format template provides an actual date in a variety of formats. In data entry fields, the user follows the form of the example date in the field to enter the actual date that he or she wants to enter.

## Input Masking with Format Templates

In data entry fields, you can optionally use input masking in conjunction with format templates to ensure that the user always enters valid characters in the correct order and position. The format template determines whether a character is valid; what is valid in one template may be invalid in another.

Without input masking, the validity of the user's entry is checked when the user attempts to leave the field. However, if input masking has been turned on for that field and its display format is a format template, the validity of the input is checked as the user enters the data, character by character. The input template is made up of special character codes that indicate what kind of input is allowed for that position in the template. If the user enters an invalid character for that position, the system beeps and requires the user to make a correct entry before continuing. For input templates, data entry mode is overstrike only; insert mode is not allowed. The user can correct entries with the Backspace and Delete keys.

To turn on input masking, you specify a numeric, date, or string template as the display format and set the Input Masking attribute for the field to y (yes) on the VIFRED Attributes for Field frame.

When specifying format templates, you must enter an appropriate template for the data type of the corresponding column in the database table. For example, a numeric template is appropriate for a field corresponding to a column of data type smallint, integer, float, or money, but is not appropriate for a field corresponding to a column of date or character data. For date fields that represent a fixed date value, you must select an absolute date/time template. The following sections tell you how to specify a format template for numeric, date, and string fields, including input masking for data entry fields.

## Numeric Templates

In addition to other display formats for numeric columns, numeric templates for specialized needs are provided.

The general syntax of a numeric template is:

```
'{c}'
```

The character *c* is one of the special template characters. Use one special character for each space of the column width. For example, '\$\$\$\$' specifies that the column is four digits wide and a dollar sign must be printed to the left of whatever number is displayed in the field. You must delimit the template with single (') or double (") quotation marks; single quotes are recommended for consistency with requirements for quoting string constants.

You can include any printable character explicitly in a numeric template by preceding it with a backslash (\). For example, to include the percent symbol in a template, enter \%. The backslash is ignored and the character immediately following the backslash is read or displayed.

When input masking is off, checking is only performed when you exit Ingres.

## Input Masking with Numeric Templates

When input masking is on, 0.00 is displayed in the entry field for any numeric template, including at least one zero to the left and two zeros to the right of the decimal point. It displays as many zeros to the right of the decimal point as are needed to match the template. The cursor is positioned at the decimal point for input. Enter the number as you do on a calculator, with each digit pushing previously entered digits to the left. Each digit is checked against its corresponding position in the template as you enter it. After the user enters a decimal point, digits are checked and displayed as entered, from left to right, on the right side of the decimal point. Left-justification for numeric templates is not supported with input masking.

**Note:** Decimal data types are not supported with input masking.

When input masking is on, the user cannot enter a negative number unless the format template includes a negative indicator.

## Special Numeric Template Characters

The special template characters and the actions they specify are as follows:

### **n**

If a digit in the number corresponds to the position of *n*, displays or accepts the digit. If no digits remain, displays or enters zero (0) to the left of the preceding character. If a field is specified *without* *n* in the numeric positions and a value of zero (0) is encountered, enters blanks in the field.

### **z**

If a digit in the number corresponds to the position of *z*, displays or accepts the digit. If no digits remain, displays or enters a space to the left of the preceding character. This is used for standard blank-padded numeric fields.

### **\$**

(Dollar sign) If a digit in the number corresponds to the position of a dollar sign, displays or accepts the digit. If no digits remain, displays a floating dollar sign immediately to the left of the last evaluated digit. A dollar sign displays only once in the output field. If a dollar sign has been displayed in a previous position, a space is displayed in this position. This can be used either to place a dollar sign directly to the left of the number, or to place a dollar sign in a fixed position in the field when used with other template characters.

### **-**

(Minus sign—preceding or trailing)

For preceding: If a digit in the number corresponds to the position of a minus sign, displays or accepts the digit. If no digits remain and the number is negative, a floating minus sign displays immediately to the left of the last evaluated digit. A minus sign displays only once in the output field. If a minus sign has already been displayed in a previous position, or if the number is positive and no digits remain, a space displays in that position.

For trailing: Displays a minus sign in this position if the number is negative; displays a space if the number is positive.

### **+**

(Plus sign—preceding or trailing)

For preceding: If a digit in the number corresponds to the position of a plus sign, displays or accepts the digit. If no digits remain, displays a floating sign (+ or -) to the left of the last evaluated digit. A plus or minus sign displays only once in the output field. If a sign has already been displayed in a previous position, a space displays in that position.

For trailing: Displays a plus sign in this position if the number is negative; displays a space if the number is positive.

,

(Comma) Displays a comma if the number contains any digits to the left of this position. If no digits remain, displays a space. This is used for inserting commas to break up large numbers.

.

(Decimal Point) Displays or accepts a decimal point in this position. The template can contain only one decimal point.

\*

(Asterisk) If no digits remain, displays an asterisk. This is useful to fill a number on the left with asterisks (for example, for checks).

### **space**

Displays a blank space in this position. This is identical to specifying a backslash followed by a space, and is provided for convenience only. Do not use spaces as thousands separators in place of commas and a decimal point if your template contains floating characters (+ - \$ [ ] ( ) < > { }). Floating characters work correctly only with commas and the decimal point as separators.

\c

(Backslash) Displays in the specified position any character c preceded by a backslash. This allows you to insert hyphens, slashes, or other characters into the number. (The backslash itself is not displayed.)

### **CR**

(Two characters) If the number is negative, inserts the characters CR (for credit). If positive, displays two blanks. You can specify the characters in either uppercase or lowercase (or one of each).

### **DB**

(Two characters) If the number is negative, inserts the characters DB (for debit). If positive, displays two blanks. You can specify the characters in either uppercase or lowercase (or one of each).

( ), [ ], < >

(Parentheses, brackets, or angle brackets) If the number is negative, displays it within the specified symbol.

## Numeric Template Examples

The following examples demonstrate the use of numeric templates. This table uses a caret (^) in the output column to represent a space, so that you can better interpret the results:

Format	Example Data	Report Output (^ indicates a blank space)
'zzzzzzzz'	123	^^^123
'zZzZz.Zz'	0	^^^^^^
'zzzzzzzz.nn'	0	^^^^^^.00
'+++,+++,+++'	23456	^^^+23,456
'---,---,---.NN'	23456.789	^^^23,456.79
'---,---,---.zz'	-3142.666	^^^3,142.67
'\$\$\$\$,\$\$\$,\$\$\$.\$nnCr'	235122.21	^^^\$235,122.21Cr
'\$\$\$\$,\$\$\$,\$\$\$.\$nnDb'	-235122.21	^^^\$235,122.21Db
'\$zz,zzz,zzn.nn'	1234.56	\$^^^^1,234.56
'\$**,**,**.\$nn'	12345	\$*****12,345.00
'+\$,\$,\$,\$,\$.\$nn'	54321	^^^+\$54,321.00
'nnn\ -nn\ -nnnn'	023243567	023-24-3567
'(zzzzz)'	-123	(^^123)

## Date and Time Templates

You can specify a date (d) format template for a field or column with a date data type. The syntax of a date template is:

```
[+|-]d 'template'
```

The *d* indicates this is a date template, and the *template* parameter indicates exactly how a specific date is displayed or entered. You must delimit the template with single (') or double (") quotation marks; single quotes are recommended for consistency with requirements for quoting string constants.

The date is left-justified by default. You can right justify it by specifying the optional plus (+) sign.

When input masking is off, the *template* can be a string of characters specifying either the absolute date and time format or specifying a time interval format, for either a display-only or data entry field.

### Absolute Date and Time Templates

You can specify absolute date and time templates by entering an example date that indicates exactly how you want each date and time element to be displayed or entered. You can specify any one of many variations of the following representative date and time for your date template:

Sunday, 1901 February 3 at 4:05:06 p.m.

You can use any or all parts of the representative date in your date template, as in the following examples:

```
d'Sun, Feb 3, 1901'  
d'02/03/01 16:05:06'  
d'04:05:06p'  
d'3/01
```

An absolute date and time template overrides any date format specified by the `II_DATE_FORMAT` environment variable/logical. For more information on `II_DATE_FORMAT`, see the *System Administrator Guide* for the system on which your database resides.

## Absolute Date and Time Template Specification

You can use the following representations of components in a date and time template, with the exceptions noted in the table that follows:

- Sunday, Sun, or Su represents a day of the week
- 1, 01, or 1901 represents the year
- 2, 02, February, Feb, or Fe, represents the month
- 3 or 03 represent the day of the month
- 4, 04, or 16 (24-hour time) represent the hour
- 5 or 05 represent minutes
- 6 or 06 represent seconds
- p or p.m. represent the designations am or pm

This is easy to remember because Sunday is the first day of the week, and arguments 1, 2, 3, 4, 5 and 6 are the year, month, day, hour, minute, and second, respectively.

The following rules apply when using the representative date components to indicate position and style:

### Day

You can specify the day as an ordinal number by suffixing it with the appropriate abbreviation: st, nd, rd, or th. For example, the template 3rd day of February 1901 produces a date such as 15th day of January 1998.

If input masking is on, you cannot use the full name of the day, Sunday, or ordinal numbers and their abbreviations. If you use the single digit 3 for the day, you must precede it with a space. For more information, see Input Masking for Absolute Dates.

### Month

You can use either a single or double digit for the month. Use a 2 (single digit) for the month to display the months of January through September as a single digit and the months of October through December as a double digit. For example, if you specify 2 for the month, July appears as month 7 and November appears as month 11. Use 02 for the month to display or enter all months as two-digit numbers. In this case, July appears or is entered as 07.

If input masking is on, you must precede the single digit 2 with a space in your template. You cannot use the full name of the month, February. For more information, see Input Masking for Absolute Dates (see page 591).



**Year**

If you use the single digit 1 in the template to indicate the year, the years zero through nine are displayed or entered with only one digit; double-digit years are displayed as two digits. For example, specifying 1 for the year displays 2004 as 4 and 2014 as 14. If you use the four digits 1901, all four digits of the year are displayed. Thus, 2004 is displayed as 2004.

If input masking is on, you must precede the single digit 1 with a space in your template. For more information, see *Input Masking for Absolute Dates* (see page 591).

**Time**

You can specify 24-hour (military) time by using 16 instead of 4 for the hour. You cannot use p or p.m. with 24-hour time.

**Order**

You can arrange the arguments in various combinations in any order. For example, 03/02/01 displays first the day (3), the month (2), and the year (1), with each element separated by slashes. The date January 15, 1998 appears as 15/01/98. If your template is Sun 03/02/01, the date January 15, 1998 appears as Saturday 15/01/98. Similarly, 1901.02.03 first displays the year, the month, and the day, separated with periods, resulting in 1998.01.15.

**Constant characters**

Separators and other constant characters in your template, such as forward slashes (/) and periods (.), are displayed exactly as indicated in the template.

**Reserved characters**

If you want to use one of the special reserved template characters or symbols as an explicit character in your template, you must precede it with a backslash (\). For example, if you want to use the numeral 2 in the date template as a constant rather than as the template symbol for the month number, you must enter it as \2.

If input masking is on, you cannot use the vertical bar (|) as a special alignment character.

Ingres displays as a string constant any word you include in a date template other than the month name of February (or its allowed abbreviations), the weekday of Sunday (or its allowed abbreviations), and the time designation p or p.m.

You can combine any of the values given to specify your template, but be aware that some combinations of those values can lead to incorrect outputs. For example, you can display the numeric day of the year by specifying the day and year but leaving out the month. Therefore, 3/1901 in the template results in dates like 9/1998 for January 9, 1998, and 121/1998 for April 30, 1998. Also note that you can display just the numeric representation for month and year by leaving out the day-year and day-month. This results in outputs such as 3 for March 6, 1998, when specifying the template 02, and 98 for April 4, 1998, when specifying the template 01. However, when you use templates with 03, leaving out the month and the year, the template *does not* return the day of the month; instead, it returns the number of days since January 1, 1582 (Gregorian calendar), which leads to outputs like 151943 for February 1, 1998.

Numbers requiring more than one digit use up preceding blanks or zeroes. If there are no preceding blanks or zeroes left, the number expands to the right. A succeeding number does not use up a single blank immediately following a letter, word, or number. Columns of numbers can be lined up by preceding them with an appropriate number of blanks or zeroes.

Because full month and weekday names (as well as numbers without preceding blanks or zeros) are of differing lengths, date columns using these components in the format do not usually line up when displayed in columns. Follow February or Sunday with a vertical bar (|) to specify that for shorter month names or weekdays, an appropriate number of blanks is substituted for the vertical bar to line up the components. When you follow a single digit in your template with a vertical bar, each single-digit entry is automatically preceded with a space, to line up single and double-digit entries.

For example, the template Sunday,| February | 3,| 1901 produces dates like:

```
Friday,   January 15, 1998
Wednesday, May    4, 1998
Saturday, November 20, 1998
```

Any character preceded by a backslash (\) is printed as it appears.

## Absolute Date and Time Examples

The following table demonstrates the use of absolute date and time templates:

Format Template	Example Data	Output
d'2/3/1'	25-oct-1998	10/25/98
D'2/3/1'	5-jun-1909	6/5/9
d' 2/ 3/01'	25-oct-1998	10/25/98
d' 2/ 3/01'	5-jun-1909	6/ 5/09

Format Template	Example Data	Output
d'03-02-01'	5-oct-1998 07:24:12	05-10-98
d'010203'	5-oct-1997	971005
d'1\ 2\ 3'	5-oct-1997	97 10 5
d'FEBRUARY, 1901'	1-sep-2134 09:13:02	SEPTEMBER 2134
d'FEBRUARY, 1901'	7-may-1962 13:08:42	MAY 1962
d'Sunday'	5-oct-1998	Wednesday
d'SUN Feb 3 16:05 1901'	13-oct-1998 07:24:03	THU Oct 13 07:24 1998
d'FEB 03 4:05:06 p.m.'	12-dec-1998 22:13:03	DEC 12 10:13:03 p.m.
d'04:05:06 PM'	5-oct-1998 14:08:45	02:08:45 PM
d'04:05:06 PM'	5-oct-1998 07:29:12	07:29:12 AM
d 16:05 pst	5-oct-1998 14:08:45	14:08 pst
d'3/01'	5-oct-1998	278/98
d'February 3rd'	29-jul-1954	July 29th
d'3rd day of 1901'	11-may-1999	131st day of 1999

### Input Masking for Absolute Dates

With input masking turned on, the only valid date template is the absolute date and time template. Input masking does not allow you to use date interval templates. Other restrictions apply, as discussed in Absolute Date and Time Templates (see page 587).

There must be adequate room in the date input template for all the digits required by the template. For example, you must provide enough space to enter both single-digit and double-digit days of the month.

To use single digits for the day, month, or year in your input template, you must do *either* of the following:

- Precede the digit with a space to allow space for possible double-digit entries
- Be sure that the data consists only of a single digit for that component

For example, assume that you specify a format representing the month/day/year with no leading blanks or zeros in front of the digits:

```
d'2/3/1'
```

If you turn input masking on, this template is unacceptable because all of the entries can be at least two digits (00/00/00 or 00/00/0000) and the preceding format only allows for the one digit in each part. To specify a format in this manner that is acceptable when input masking is on, you can precede each single digit with a space (d' 2/ 3/ 1') or with a zero (d'02/03/01'). Specifying d' 2/ 3/ 1' causes a blank-filled date to display in the field prior to user entry; specifying '02/03/01' displays a zero-filled date prior to user entry.

If you specify a template that is incorrect for use with input masking and turn on input masking, the template is used, if it is otherwise correct, without activating input masking. It does *not* generate an error message. However, it can compare the entered data to the template only when the user attempts to exit the field. If you correct the template, and input masking is still on, input masking is automatically activated or the next data entered.

You must enter data from left to right in a date field that has input masking turned on. Only numbers in the numeric portions of the template and alphabetic characters in the non-numeric portions (month and day names) are accepted. If you try to enter an incorrect keystroke in a position, you hear a beep and the keystroke is not entered. The entire date and time string is validated only when you exit the field.

## Time Interval Templates

Time interval templates show the amount of elapsed time rather than an absolute date or time.

**Note:** You cannot use input masking with time interval templates.

As in the absolute date format, you specify a time interval with a quoted string containing one of many possible representations of a sample time interval, such as 1 year or 1 yr 3 day. The selection and arrangement of the time interval elements within the template indicate the way you want time intervals to be displayed or printed. You must use the following representative time interval as the basis for your template:

1 year 2 months 3 days 4 hours 5 minutes 6 seconds

You can use all or only some of these units in your template, and arrange them in any order. You can use the plural or singular form of any unit, as well as the singular or plural form of the abbreviations, yr, mo, hr, min, and sec.

To specify a time interval, use the `d'template'`, where *template* contains one or more time interval keywords (for example, minutes) preceded by the appropriate digit from the representative time interval string, as in:

```
d'5 minutes'
```

This format displays results followed by the keyword; for example:

```
9 minutes
```

Format	Example Data	Report Output
d'1 year'	3 years 5 mos 16 days	3 years
d'2 MONTHS, 3 DAYS'	3 years 5 mos 1 days	41 MONTHS, 1 DAY
d'1 yr 3 day'	1 yrs 5 mos 16 days	1 yr 168 days
d'4 hours 6 seconds'	23 hrs 8 mins 53 secs	23 hours 533 seconds
d'04:05 \hours'	23 hrs 0 mins 53 secs	23:01 hours
d'3 days 4 hours'	23 hrs 8 mins 53 secs	0 days 23 hours
d' 1 yr 2 mos 3 days'	200 yrs 11 mos 28 days	200 yrs 11 mos 28 days
d' 1 yr 2 mos 3 days'	5 yrs 1 mos 3 days	5 yrs 1 mo 3 days

There are 30.4376875 days in a month and 365.2425 days in a year. When calculating a date interval, the smallest unit of time is rounded up.

Numbers requiring more than one digit use up preceding blanks or zeroes. If there are no preceding blanks or zeroes to the left, the number expands to the right. A succeeding number does not use up a single blank immediately following a letter, word, or number. You can line up columns of numbers by preceding them with an appropriate number of blanks or zeroes (note the last two examples).

The word immediately following a number is made singular if the number is one, or plural if the number is zero or greater than one. You can prevent this by preceding the word with a backslash (\). Any character preceded by a backslash is printed as you enter it.

## String Input Templates

String input templates allow you to specify the type, order, and number of characters a user can enter in a data entry field. You can define string input templates only on single-line character data entry fields. This template is not available for display-only fields, multi-line character entry fields, or entry fields with non-character data types.

The general syntax of a string input template is:

`s' {c} '`

The `s` indicates this is a string template and the `c` represents any of the special template characters. You use one special character for each space in the field or column width, up to a possible width of 100. For example, `s'naaannn'` specifies that the field or column is 7 characters wide and that numbers (`n`) must be entered in the first and fifth through seventh positions, with alphabetic characters (`a`) in the second, third, and fourth positions.

You must delimit the template with single (`'`) or double (`"`) quotation marks; single quotes are recommended for consistency with requirements for quoting string constants.

## String Template Creation

To create a string template that suits your needs, you can use any combination of special characters that are pre-defined by Ingres, constant characters, and user-defined custom character sets. A custom character set is a definition of allowable user input for one of the five special characters, `i`, `j`, `k`, `l`, and `m`. You can define up to 12 different custom character sets in a single template.

## Special Characters

There are a number of special characters that you can use in a string template. Each special character represents a set of characters from which the user can choose, to enter at the specified position.

For example, the special characters **a** and **z** represent alphabetic characters and digits, respectively. Assume that you enter a template that looks like this:

```
aaazz
```

With this template, users can make entries such as `Afm35` or `pRt44` but not entries such as `2acm5` or `C342d`. They must enter alphabetic characters in the first three places and numeric characters in the last two places.

You can use the following special characters in a string template:

**a**

Represents any alphabetic character.

**h**

Represents any hexadecimal digit.

**n**

Represents any digit (the default is 0).

**o**

Represents any printable character (only 7 bit).

**p**

Represents any printable character.

**q**

Represents any character that can be the first character in an Ingres name.

**r**

Represents any character that can be the second or subsequent character in an Ingres name.

**s**

Represents any character (only 7 bit).

**t**

Represents any character.

**x**

Represents both alphabetic and numeric characters.

**z**

Represents digits (the default is a space).

**i**

Represents a user-defined character.

**j**

Represents a user-defined character.

**k**

Represents a user-defined character.

**l**

Represents a user-defined character.

**m**

Represents a user-defined character.

Using templates, you can enforce mandatory entry (the user must make an entry), force the case of an entry, and insert a default entry if the user fails to make an entry.

All of the special characters support mandatory entry (see page 601). To force the case of a user's entry or insert a default if the user does not make an entry, use a user-defined special character. The pre-defined special characters do not support forced case or default entries.

## Escape Character

In addition to the special characters, a template can also include constant (literal) characters. To indicate that a character is to be read as a literal in a template, precede the character with the escape character, a backslash (\).



## Custom Character Sets

In some instances, you may want to limit a user's entries to some set of characters that are broader or narrower than those available using the pre-defined special characters. To accommodate these needs, you can define up to 12 custom character sets per template. If the set you want is small—for example, one or two characters—you can use the following syntax:

```
[c{c}]
```

The *c* represents the valid character or characters. You can include any number of valid characters. Simply use this syntax in the template in the positions where you want the specified characters. You must include the brackets. For example, the following template allows the user to enter only the letters a, b, or c in the first position:

```
[abc]nnn
```

If you want to specify a range of characters, use the following syntax:

```
[c-c2]
```

The *c* represents the beginning character in the range and *c2* represents the ending character.

For either syntax, the user must enter one of the specified characters in the specified position.

In either syntax version, the case of the characters is unimportant for alphabetic characters. You can use an uppercase or lowercase character to represent the character. To force the user's entry to either case, use the special formatting character that specifies forced case. For more information, see [How to Specify Uppercase or Lowercase](#) (see page 601) .

When defining a character set for a single position in this manner, you cannot enforce mandatory entry in that position. To enforce mandatory entry, you must define the character set with a user-defined special character.

If you are defining a large character set or are using the set in several places in your custom template, it is probably more convenient to define a special character to represent the set and use that character in the template. For more information, see [User-defined Characters](#) (see page 598).

## User-defined Characters

In addition to the predefined special characters for string templates, you are given five special characters whose meaning you can define. This is done by associating one of the user-defined special characters with a custom character set. This feature makes it easy to define a custom character set and specify that set in several positions in your custom template.

To use a user-defined special character, you define the special character at the first instance of its use in the template and put the special character in the template for subsequent occurrences.

The special characters reserved for user definition are *i*, *j*, *k*, *l*, and *m*.

In the same way that each of the predefined characters are defined to represent some set of characters, you can define each of these special characters to represent some set of one or more characters. To do this, use the following syntax:

```
[c=c1{c2}]
```

The *c* is one of the five special characters (*i*, *j*, *k*, *l*, or *m*) and *c1* and *c2* represent the characters that you want to include in the set of characters defined for the special character. You can include any number of characters in the set.

You can also specify a range of characters for your special character:

```
[c=c1-c2]
```

In this case, *c* represents the special character, *c1* represents the beginning of the range, and *c2* represents its end.

When you include a user-defined special character in a template, the first instance of the character in the template must be its definition. For subsequent occurrences, you just use the special character. For example, assume that you want to build a template in which the special character *j* represents the numbers 4 and 6. An example of such a template is:

```
[j=46]aazaj
```

The above template describes a field of six characters, of which the first and last must be either a 4 or a 6. Another example is:

```
nn[j=46]nj
```

This example describes a field of six characters also, in which the third, fifth, and sixth must be either a 4 or a 6.

## Predefined Characters for Custom Character Sets

You are also given a set of predefined special characters that you can use in the definition of a custom character set. These characters represent a pre-defined set of characters and can appear only in a custom character set definition. You cannot place them directly in a template definition. These special characters are useful because you can force the case or apply a default to any of the characters represented by these built-in special characters. (The pre-defined special characters described in The Special Characters do not allow you to apply forced case or a default to any of the characters that they represent.)

The predefined special characters that you can use in a custom character set definition are as follows:

**#**

The pound sign represents any digit.

**@**

The at sign represents any alphabetic character.

**\***

The asterisk represents any printable character.

**+**

The plus sign represents any printable character (only 7 bits).

**&**

The ampersand represents any character that can be the first character in a name.

**%**

The percent sign represents the second and subsequent characters in a name.

**:**

The colon represents any hexadecimal digit.

Use these characters when you want to force the case of an entry or provide a default for the entry. For example, to specify a four-letter entry, use a template that looks like this:

```
aaaa
```

However, assume that you want the first position to be an uppercase letter. Because you cannot force case on the special character `a`, you must use the special character `@` instead:

```
[@/u]aaa
```

If you want to include any of these special characters as literals in the template, use the escape character with the character. For example, you want the user to enter a two-figure percentage value. A template looks like this:

```
nn\%
```

### How to Specify a Default Character

To specify a default character for a position, include the following syntax in your character set definition:

```
//c
```

The *c* is the default character. For example, assume that you want the user to enter a 4-digit number. They must enter at least two numbers, but if they fail to enter the final two numbers, you want the number 1 in each of those positions. Your template looks like this:

```
nn[##/1][#/1]
```

The *n* forces entry of a digit in those positions. The other two positions allow entry of a digit, but insert the specified default if the user does not make an entry.

When you specify a default character for a position, with one exception, the default character is inserted whenever the user inserts a space in that position or fails to make an entry into that position. The exception occurs when the space is a valid entry for that position. In such instances, the space is placed in the position. If a space is not a valid entry, the default character is placed in the position.

The specified default character is always considered to be a valid character for the position, even if it is not part of the character set defined for that position by the template.

## How to Specify Uppercase or Lowercase

To force the case of the user's entry, use the following in the special character definition:

- For uppercase:  
/U or /u
- For lowercase:  
/L or /l

You can only include one case specification in each special character definition. For example, assume that you want the user to enter a part number that consists of an uppercase letter, from A - D, followed by three numbers.

Your template looks like this:

```
[a-d/U]nnn
```

or

```
[a-d/u]nnn
```

The user can make entries in lowercase or uppercase, and the data appears in the field in uppercase.

## How to Force Mandatory Entry

Mandatory entry means that the user cannot leave the field without supplying a valid entry for that position. If they attempt to leave the field without providing a valid entry, they receive an error message and the cursor is positioned at the beginning of the template.

To specify mandatory entry, you use an uppercase special character, either built-in or user-defined. For example, assume that you want the user to enter a part number that has a two-letter code followed by two numbers and one final letter, and that the first two letters are mandatory. Your template looks like this:

```
AAAnna
```

Perhaps the part number uses a restricted letter set, for example A-D. Your template looks like this:

```
[K=a-d]Knnk
```

The user must enter one of the letters from A to D in the first two positions; this is enforced by the uppercase K. Making an entry in the final position is optional, but it must be a letter from A to D also.

## Examples of User-Defined Character Sets

The following table shows some examples of user-defined character sets:

Example	Description
<code>[\#-\*]</code>	All of the characters found between # and * in the ASCII collation sequence.
<code>[A0\[ \]/u]</code>	The letter a, forced to uppercase, the digit 0 and brackets.
<code>[#abc]</code>	Any character acceptable in your installation as a digit and the letters a, b, and c
<code>[&amp;\*\\$]</code>	Any character acceptable in your system as the first character in a name and the characters * and \$.
<code>[abc//a]</code>	The characters a, b, and c, with a default of a if the user fails to make an entry or enters a space.

## Examples of String Templates

The following table shows some examples of string templates:

Example	Description
<code>nnn\ -nn\ -nnnn</code>	This template, with embedded dashes, lets the user enter 9 digits.
<code>AA\ -nnnn</code>	This template specifies two alphabetic characters and four digits, a dash separating the alphabetic and numeric characters. The alphabetic characters are mandatory.
<code>nn\%</code>	This template specifies two digits and places a percent following the digits.
<code>[e/u][x/u]nnnn \-[#//0][#//0]</code>	This template specifies that the user enter an e and x in the first and second positions, respectively, followed by four digits and an additional two more digits. The template forces uppercase on the first two positions, inserts zeros by default in the last two positions if the user does not make entries into those positions, and puts a dash before the final two positions.
<code>N'[j=a-m/l]jjj\'</code>	This template specifies a five-character entry. The first position is a digit and is a mandatory entry. The final four positions can be any character between a and m, inclusive. The final four positions are set off by single quotes and are forced to lower case.







# Appendix A: Defining Your Terminal

---

This section contains the following topics:

[The Termcap File](#) (see page 605)

[How to Define Your Terminal](#) (see page 605)

## The Termcap File

Defining your terminal allows you to use the forms-based utilities and the many features of Ingres. Your computer system can support a wide variety of terminals, each with its own particular characteristics.

The termcap file contains a description of all terminals supported by Ingres, including their color capabilities and available function, control, and arrow keys. Each supported terminal has a termcap description that is based on the vendor's specifications for that device. This appendix lists supported terminals. For unsupported terminals, you must write your own termcap descriptions. For more information, see the appendix "Writing Termcap Descriptions."

When you start the Forms Run-time System, it uses the TERM\_INGRES environment variable/logical to determine the user's terminal type and verifies basic terminal attributes. The terminal type defined by TERM\_INGRES tells the Forms Run-time System, which terminal description to read from the Ingres termcap file. The type is checked only once for each session, so the user must exit the current session to reset TERM\_INGRES to change terminal types.

Forms Run-time System key definitions can be changed dynamically by the installation, terminal type, user, and application key mapping files. For more information on key mapping, see the appendix "Defining Function and Control Keys."

## How to Define Your Terminal

To define your terminal to Ingres, you enter the setenv command at the operating system prompt to define the TERM\_INGRES environment variable/logical. Once the forms system has started, TERM\_INGRES cannot be reset until the session has ended. It is a good practice to include the command that defines TERM\_INGRES in the automatic login procedures on your individual account on the computer.

## Define Your Terminal (Windows)

To define your terminal for Windows, use the following command:

```
ingsetenv TERM_INGRES termname
```

where *termname* is the designation for your terminal type, as listed in Terminal Names (see page 608).

For example, if you are working on an IBM compatible PC, the list under Terminal Names shows you that "IBMPC" is the correct *termname* designation.

## Define Your Terminal (UNIX)

To define your terminal in UNIX, select the appropriate command for your shell.

For the C shell:

```
setenv TERM_INGRES termname
```

For the Bourne shell:

```
TERM_INGRES=termname  
export TERM_INGRES
```

where *termname* is the designation for your terminal type, as listed in Terminal Names (see page 608).

For example, if you have a VT100 terminal, and you want to be able to use the arrow keys as cursor movement keys and the keypad keys as definable function keys, the list under Terminal Names tells you that vt100i is the proper designation. To define your terminal accordingly to Ingres, enter one of the following commands:

C shell:

```
setenv TERM_INGRES vt100i
```

Bourne shell:

```
TERM_INGRES=vt100i  
export TERM_INGRES
```

Thereafter, you can use the VT100 default assignment of keys for cursor movement and executing forms commands in Ingres.

The vt100nk is another terminal designation available for VT100 terminals. This terminal designation is particularly suited to applications that require use of the keypad for numeric input. This designation gives the user access to the arrow keys and the top four function keys on the numeric keypad. The other keys on the keypad are available for numeric input.

If your UNIX system provides full support for VT220 terminals, you can define a VT220 terminal as vt220 and use the VT220 keystrokes. Some UNIX systems do not provide full support for VT220 terminals. In this case, you must set a VT220 terminal to emulate a VT100, using the VT100 keystrokes, as discussed previously.

## Define Your Terminal (VMS)

To define your terminal in VMS, use the following command:

```
define TERM_INGRES termname
```

where *termname* is the designation for your terminal type, as listed in Terminal Names (see page 608).

For example, if you have a VT100 terminal, and you want to be able to use the arrow keys as cursor movement keys and the keypad keys as definable function keys, the list under Terminal Names tells you that vt100i is the proper designation. To define your terminal accordingly to Ingres, execute the following command:

```
define term_ingres vt100i
```

In most VAX/VMS installations, VT100 and VT220 terminals are prevalent. We recommend the vt100i designation for VT100 terminals and the vt220i designation for VT220 terminals.

## Terminal Names

The Forms Run-time System must function correctly on the terminals listed here. However, not all of these terminals have been tested to determine the functionality of the Forms Run-time System. If you have any problems with the terminal designations, notify technical support.

If you cannot locate your terminal on this list, look through the entries in the termcap file to find an entry that meets your needs:

Terminal Type	Menu Key	Name
ADDRINFO	ESC	addrinfo
ADDS CONSUL 980	ESC	a980
ADDS REGENT 100	ESC	regent100
ADDS REGENT 20	ESC	regent20
ADDS REGENT 25	ESC	regent25
ADDS REGENT 40	ESC	regent40
ADDS REGENT 60	ESC	regent60
REGENT 60 w/no arrow keys	ESC	regent60na
ADDS REGENT SERIES	ESC	regent

Terminal Type	Menu Key	Name
AMPEX DIALOGUE 80	ESC	ampex
ANN ARBOR	ESC	aa
ANN ARBOR AMBASSADOR 48 with destructive backspace	ESC	aaadb
ANN ARBOR AMBASSADOR/48 lines	ESC	aaa
ANSI PC console with function keys	PF1	ansinf
ANSI PC console with no function keys	PF1	ansif
BDS1 in Wyse-50 mode	ESC	bds1
BDS1 in Wyse-50 mode with function keys	PF1	bds1f
BEEHIVE SUPER BEE	ESC	sb1
FIXED SUPERBEE	ESC	sb2
BEEHIVE III <sub>m</sub>	ESC	bh3m
BULL QUESTAR 210	ESC	Q210
BULL QUESTAR 210 with function keys	F1	Q210f
BULL QUESTAR 305/310	ESC	Q310
BULL QUESTAR 305/310 in 132-column mode	ESC	Q310-w
BULL QUESTAR 305/310 with function keys	PF1	Q310f
BULL QUESTAR 305/310 in 132-column mode with function keys	PF1	Q310-wf
BULL X-TERMINAL running WindowView with function keys	F1	bwvf
CONCEPT 100	ESC	c100
CONCEPT 100 slow	ESC	c100s
CONCEPT 100 slow reverse video	ESC	c100rvs
C100 reverse video	ESC	c100rv
C100 with 4 pages	ESC	c1004p
C100 reverse video with 4 pages	ESC	c100rv4p
C100 with no arrows, reverse video, 4 pages	ESC	c100rv4pna
C100 with printer port, reverse video, 4 pages	ESC	c100rv4ppp
CDC	ESC	cdc456
CDC456tst	ESC	cdc456tst
CDI1203	ESC	cdi
COMPUCOLORII	ESC	compucolor

Terminal Type	Menu Key	Name
CYBERNEX mdl-110	ESC	mdl110
DATAMEDIA 1520	ESC	dm1520
DATAMEDIA 1521	ESC	dm1521
DATAMEDIA 2500	ESC	dm2500
DATAMEDIA 3025a	ESC	dm3025
DATAMEDIA 3045a	ESC	dm3045
DATAMEDIA dt80/1	ESC	dt80
DATAMEDIA dt80/1 in 132 character mode	ESC	dt80132
DATAPOINT 3360	ESC	datapoint
DEC VT100 with function keys activated	PF1	vt100f
DEC VT100 with function keys activated (3.0 release)	PF1	vt100k
DEC VT100 with numeric keypad	PF1	vt100nk
DEC VT100 without function keys activated	ESC	vt100
DEC VT100 in 132-column mode with function keys activated	PF1	vt100fw
DEC VT100 in 132-column mode with function keys activated (3.0 release)	PF1	vt100kw
DEC VT100 in 132-column mode with numeric keypad	PF1	vt100nkw
DEC VT100 with function keys activated and Return key mapped to Nextitem instead of Clearrest (works like vt100f, but Return does not clear to end of field)	PF1	vt100i
DEC VT100 in 132-column mode with function keys activated and Return key mapped to Nextitem instead of Clearrest (works like vt100fw, but Return does not clear to end of field)	PF1	vt100iw
DEC VT100 in 132-column mode without function keys activated	PF1	vt100w
DEC VT100 with no initialization	ESC	vt100n
DEC VT125	ESC	vt125
DEC VT220	PF1	vt220
DEC VT220 with Return key mapped to Nextitem instead of Clearrest	PF1	vt220i
DEC VT220 in 132-column mode	PF1	vt200w
DECVT220 in 132-column mode with Return mapped to Nextitem instead of Clearrest	PF1	vt220iw
DEC VT241	PF1	vt241
DEC VT50	ESC	vt50

Terminal Type	Menu Key	Name
DEC VT50h	ESC	vt50h
DEC VT52	ESC	vt52
DEC VT132	ESC	vt132
DECTERM EMULATOR with function keys and mouse support	PF1	decterm
DELTA DATA 5000	ESC	delta
DIGILOG 333	ESC	digilog
ENVISION	PF1	envision
ENVISION with color	PF1	envisionc
GENERAL TERMINAL 100A (formerly INFOTON 100)	ESC	i100
HAZELTINE 1500	ESC	h1500
HAZELTINE 1510	ESC	h1510
HAZELTINE 1520	ESC	h1520
HAZELTINE 1552	ESC	h1552
HAZELTINE 1552 reverse video	ESC	h1552rv
HAZELTINE 2000	ESC	h2000
HEATHKIT h19	ESC	h19
HEATHKIT h19 ansi mode	ESC	h19A
HEATHKIT with keypad shifted	ESC	h19bs
HEATHKIT with keypad shifted, underscore cursor	ESC	h19us
HEATHKIT with underscore cursor	ESC	h19u
HEWLETT PACKARD 2392 in hp mode, 80 columns	PF1	hp2392
HEWLETT PACKARD 2393 in hp mode, 80 columns	PF1	hp2393
HEWLETT PACKARD 2394 in hp mode, 80 columns	PF1	hp2394
HEWLETT PACKARD 700/92 in hp mode, 80 columns	PF1	hp70092
HEWLETT PACKARD 700/94 in hp mode, 80 columns	PF1	hp70094
HEWLETT PACKARD 700/96 in hp mode, 80 columns	PF1	hp70096
HEWLETT PACKARD 700/98 in hp mode, 80 columns	PF1	hp70098
HEWLETT PACKARD 2621	ESC	2621
HEWLETT PACKARD 2621 with 45 keyboard	ESC	2621k45
HEWLETT PACKARD 2621 with labels	ESC	2621wl

Terminal Type	Menu Key	Name
HEWLETT PACKARD 2621 with no labels	ESC	2621nl
HEWLETT PACKARD 2621 48 lines	ESC	big2621
HEWLETT PACKARD 2626	ESC	hp2626
HEWLETT PACKARD 2640a	ESC	2640
HEWLETT PACKARD 2648a graphics terminal	ESC	hp2648
HEWLETT PACKARD 264x series	ESC	2640b
HEWLETT PACKARD 264x series	ESC	hp
IBM 3101-10	ESC	ibm
IBM PC	ESC	ibmpc
INFOTON 400	ESC	i400
INFOTON KAS	ESC	infotonKAS
ISC modified owl 1200	ESC	intext
ISC8001	ESC	8001
LSI adm2	ESC	adm2
LSI adm3	ESC	adm3
LSI adm3a+	ESC	adm3a+
LSI adm31	ESC	adm31
LSI adm3a	ESC	adm3a
LSI adm42	ESC	adm42
MICRO BEE SERIES	ESC	microb
MICROTERM ACT IV	ESC	microterm
MICROTERM ACT V	ESC	microterm5
MICROTERM MIME1	ESC	mime
FULL BRIGHT MIME1	ESC	mimefb
HALF BRIGHT MIME1	ESC	mimehb
MICROTERM MIME2A (emulating an enhanced SOROC iq120)	ESC	mime2as
MICROTERM MIME2A (emulating an enhanced VT52)	ESC	mime2a
MIME1 emulating 3A	ESC	mime3a
MIME1 emulating enhanced 3A	ESC	mime3ax
NETRONICS	ESC	netx



Terminal Type	Menu Key	Name
PERKIN ELMER 1100	ESC	fox
PERKIN ELMER 1200	ESC	owl
SOL	ESC	sol
SOROC 120	ESC	soroc
SOUTHWEST TECHNICAL PRODUCTS CT82	ESC	swtp
SUN CMDTOOL with function keys	PF1	sun-cmdf
SUN CMDTOOL no function keys	ESC	sun-cmd
SUN CONSOLE	ESC	sun
SUN CONSOLE with function keys	F2	sunf
SUN MICROSYSTEMS SUN TYPE5 keyboard for shelltool/commandtool	PF1	suntype5
SUN TERMINAL EMULATOR	PF1	vt100te
SUNTOOLS CONSOLE emulator	R11	sunm
SUPER BEE with insert character	ESC	superbeeic
TEKTRONIX 4105	PF1	tk4105
TEKTRONIX 4105 with color	PF1	tk4105c
TEKTRONIX 4105 with 24 lines and color and function keys	PF1	vt4105c
TEKTRONIX 4106 with function keys	PF1	tk4106
TEKTRONIX 4107 with function keys	PF1	tk4107
TEKTRONIX 4115 with function keys	PF1	tk4115
TELERAY 1061	ESC	t1061
TELERAY 1061 with fast PROMs	ESC	t1061f
DUMB TELERAY 3700	ESC	t3700
TELERAY 3800 series	ESC	t3800
TELETEC DATASCREEN	ESC	teletec
NEW TELEVIDEO 912	ESC	912b
NEW TELEVIDEO 920	ESC	920b
OLD TELEVIDEO 912	ESC	tvi912
OLD TELEVIDEO 920	ESC	tvi920
PuTTY	F1	putty
VISUAL 200 no function keys	ESC	vi200f

<b>Terminal Type</b>	<b>Menu Key</b>	<b>Name</b>
VISUAL 200 reverse video	ESC	vi200rv
VISUAL 200 reverse video using insert character	ESC	vi200rvic
VISUAL 200 using insert character	ESC	vi200ic
VISUAL 200 with function keys	ESC	vi200
VT100f in 132 mode that supports 80/132 size change	PF1	vt100fwc
VT100f that supports 80/132 size change	PF1	vt100fc
VT100f in Xterm with sizing on startup and mouse support	PF1	vt100fx
VT100f with nextitem mapped and 80/132 size change	PF1	vt100ic
VT100f in 132 mode with nextitem and 80/132 size change	PF1	vt100iwc
VT200 in 7 bit with nextitem and reset to 8 bit	PF1	vt200-8i
VT200 in 7 bit, 132 cols, nextitem and reset to 8 bit	PF1	vt200-8iw
VT200 in 80 mode with nextitem and 80/132 support	PF1	vt200ic
VT200 in 132 columns and reset to 8 bits on exit	PF1	vt200-8w
VT200 in 132 mode with nextitem and 80/132 support	PF1	vt200iwc
VT220 that resets to 8 bits on exit	PF1	vt200-8
VT220 in 7 bit that supports 80/132 size change	PF1	vt200c
VT220 in 7 bit 132 mode and supports 80/132 switching	PF1	vt200wc
VT240 with function keys	PF1	vt240
VT241 terminal in 80 mode	PF1	vt241
VT241 terminal in 132 mode	PF1	vt241w
VT3XX running as a vt200 in 132 mode	PF1	vt300w
VTU 0010	ESC	hn10
VTU 0010 with function keys	F1	hn10f
VTU 0040	ESC	hn40
VTU 0040 with function keys	F1	hn40f
VTU 0050	ESC	hn50
VTU 0050 with function keys	F1	hn50f
XEROX 1720	ESC	x1720
XITEX sct-100	ESC	xitex
ZENTEC 30	ESC	zen30

# Appendix B: Defining Function and Control Keys

---

This section contains the following topics:

[Why Key Mapping?](#) (see page 615)

[Key Mapping Overview \(Windows Environment\)](#) (see page 616)

[Key Mapping Overview \(UNIX and VMS Environments\)](#) (see page 626)

[Obtaining Information on Mappings](#) (see page 643)

[FRS Mapping Objects](#) (see page 643)

[Mapping File Syntax](#) (see page 654)

[Restrictions and Limitations \(Windows Environment\)](#) (see page 661)

[Restrictions and Limitations \(UNIX and VMS Environments\)](#) (see page 663)

## Why Key Mapping?

You may want to define function and control keys in Microsoft Windows (function, control, and arrow keys in UNIX and VMS) to the FRS for use in user-designed applications or to customize your keyboard for use with the Ingres forms-based tools. FRS is a built-in screen management system for all forms-based tools, as well as for custom applications that use forms.

When designing custom applications, the actual keystrokes that you employ to perform various functions can differ from those used with the Ingres forms-based tools. If you want to maintain consistency with key definitions in these forms-based tools, you can use Ingres conventions when programming menu items, cursor movement, and other operations performed by users.

You also may want to define keys to the FRS to customize the end user's environment while the user is running either forms-based tools or specialized Ingres applications. Ordinarily, such customizing is implemented by the application developer, rather than by end users of applications.

Use *mapping* files to *map* menu item operations, cursor movement, and all other operations, to function or control keys, and (in UNIX and VMS) arrow keys on your keyboard. Once the mapping has been specified, you or the end user can execute the operation by pressing the specified key. If the keyboards at your installation do not support function keys, you can still map operations to control keys or (in UNIX and VMS) arrow keys. The end user can enter a control character to execute the operation.

## Key Mapping Overview (Windows Environment)

These files define keyboard key usage in the FRS for Ingres on a PC running Windows:

- Termcap file
- FRS default key-mapping file (or personal variation of this file)
- Application key-mapping files

The first two files define and map the overall use of keys in the FRS for a standard PC. Application key-mapping files define and map the use of keys within a specific application that an Ingres user can design.

### Termcap File (Windows)

The Ingres termcap file for the PC environment contains:

- Statements that define the basic capabilities of your PC to the FRS. For more information, see the appendix "Writing Termcap Descriptions."
- Escape sequences that associate keys on your keyboard with internal program function, arrow, and control keys

Ingres defines physical keys and key combinations in the termcap file by associating their escape sequences with a set of 40 internal program function keys (pf1 through pf40), the arrow keys (up, down, right, and left), and some ASCII control keys. For example, F1 on an IBM PC AT keyboard is assigned to the internal function key, pf1, the Shift+F1 key combination is assigned to internal function key, pf11, and the Insert key is assigned to the ASCII control key, controlE. The internal function and control keys are then mapped to Ingres operations in a mapping file. This makes it possible to map predefined Ingres operations to many more physical keys than the standard set of function keys on your PC keyboard.

ASCII control keys that have been mapped to operations but have *not* been defined in the termcap file can be invoked by pressing the appropriate Ctrl+key combination. However, the following control keys are reserved by the operating system for its own use and must not be mapped to any operations in any key mapping file:

- Ctrl+C
- Ctrl+P
- Ctrl+S

The termcap file supports the enhanced keyboard function keys, F11, F12, Shift+F11, Shift+F12, Ctrl+F11, and Ctrl+F12. Depending on the key-mapping file you use, these keys can or cannot be assigned to Ingres functions.

The following table shows the key assignments for your PC in the termcap file:

<b>Internal Ingres Key Name</b>	<b>PC Key</b>
controlE	Insert
controlH	Backspace
controlY	Home
controlZ	End
Up arrow	Up arrow
Down arrow	Down arrow
Right arrow	Right arrow
Left arrow	Left arrow
pf1	F1
pf2	F2
pf3	F3
pf4	F4
pf5	F5
pf6	F6
pf7	F7
pf8	F8
pf9	F9
pf10	F10
pf11	Shift+F1
pf12	Shift+F2
pf13	Shift+F3
pf14	Shift+F4
pf15	Shift+F5
pf16	Shift+F6
pf17	Shift+F7
pf18	Shift+F8

Internal Ingres Key Name	PC Key
pf19	Shift+F9
pf20	Shift+F10
pf21	Alt+F1
pf22	Alt+F2
pf23	Alt+F3
pf24	Alt+F4
pf25	Alt+F5
pf26	Alt+F6
pf27	Alt+F7
pf28	Alt+F8
pf29	Alt+F9
pf30	Alt+F10
pf31	F11 and Ctrl+F1
pf32	F12 and Ctrl+F2
pf33	Shift+F11 and Ctrl+F3
pf34	Shift+F12 and Ctrl+F4
pf35	Alt+F11, Ctrl+F5, and Ctrl+F11
pf36	Alt+F12, Ctrl+F6, and Ctrl+F12
pf37	PgUp
pf38	PgDn
pf39	Delete
pf40	Escape

## FRS Mapping File (Windows)

A mapping file defines the use of the program function and control keys in the FRS. The mapping file associates a pf or control key with a particular Ingres function by mapping it to a FRS object. There are different types of FRS mapping objects. A FRS object can be an:

- FRS command
- Menu item
- FRS Key

An example of a FRS object is the FRS command, menu. FRS commands are built-in functions of the FRS, such as moving to the menu line or moving to the next field, that enable the user to view and edit the data on a form. The FRS defines the behavior of each FRS command. For instance, the menu command moves the cursor to the menu line.

The FRS mapping file maps these functions to the pf keys and other internal keys associated with the physical keys on your keyboard. The entire mapping sequence in the FRS mapping file and the termcap file can be depicted as:

*FRS mapping object = pf or control key = keyboard key*

For example, the key mapping sequence for the menu command can be represented as:

**menu FRS command = pf2 program function key = F2 key**

The actual mapping statement in the FRS mapping file would be:

```
menu = pf2 (F2)    /* Menu (menu key) */
```

The key name shown in parentheses is a *label* that indicates to the user which physical key to press. The functionality of each key mapping is shown as a comment statement between the characters */\** and *\*/*.

This mapping scheme provides flexibility in assigning functions to keyboard keys, without requiring editing of the termcap file. You can easily assign any FRS mapping object to a different key by changing the mapping statement in the FRS mapping file, or by overriding it with a statement in a mapping file of higher precedence. For more information, see Application Mapping Files (Windows) (see page 626).

You can also substitute your own personal key mapping file for the FRS mapping file. To do so, set the Ingres\_KEYS environment variable to the substitute file in a batch file, such as your autoexec.bat file, using the following syntax:

```
set INGRES_KEYS=fullpathname\filename
```

This can be useful if multiple users share the same PC and want to customize their key map assignments individually.

## Mapping File Example (Windows)

The following is an example of an FRS mapping file:

```
/* This is an example of a mapping file */
menuitem2 = pf3 (F3)
  menuitem3 = controlE (^E)
  frskey7 = pf8
previousfield = controlP (Sh-Tab)
  rubout = controlDEL (Backspace)
  controlA = off
/* this turns control-A off */
pf7 = off
```

The first line of this sample file is a *comment*. Comments can appear anywhere in a mapping file. Their purpose is to provide information to someone looking at the file; they are ignored by the FRS.

The next five lines are examples of *mapping statements*. All mapping statements follow the same basic syntax. To the left of the equal sign is a *mapping object*, which specifies the operation or function to which the key is being mapped. To the right of the equal sign is the internal program function key or control key that maps to, and activates, the mapping object. For instance, the following statement specifies that the second item on each menu line (menuitem2) map to program function key pf3:

```
menuitem2 = pf3 (F3)
```



The parentheses to the right of the internal function or control key contain a *label*. The label tells the user what key to press to activate the menu selection or operation. The label appears on the menu line, if appropriate, or in the Help Keys operation display. If no label is specified in the mapping statement, Ingres displays an appropriate default label. Although you can put any text you want here, you must make sure the label indicates the physical key to which the internal function key is mapped in the termcap file, so that the end user presses the correct key to invoke the operation. In the preceding example, a user invokes the operation specified by the second item on any menu by pressing F3.

The following statement enables the user to move the cursor to the previous field on a form by pressing Sh-Tab, which is mapped internally to the controlP function:

```
previousfield = controlP (Sh-Tab)
```

The Ingres controlP function has been mapped internally to Sh-Tab, rather than to the Ctrl+p key combination, because Ctrl+p is trapped by the operating system. For more information, see The Termcap File.

The last two statements in the example are known as *disabling statements*. A disabling statement is used to disable a control or function key. For example, the following statement turns off the key mapped to the pf7 program function key (which is associated with function key F7 in the termcap file):

```
pf7 = off
```

This means that the function key F7 has no effect in a forms program governed by this map file, and merely beeps when pressed.

## Standard FRS Mapping Files (Windows)

Depending on your PC keyboard and environment, you can use the following standard FRS mapping files provided with the forms-based tools:

- frs.map, for a PC without function keys
- ibmpc.map, for an ANSI standard PC

Ingres always reads and uses the frs.map file as the basic key mapping file. You cannot change this by setting an environment variable. However, you can point to an additional mapping file from within the termcap file by setting the map= entry to the following file:

ibmpc.map

Mappings in the termcap-specified key map file take precedence over mappings in the frs.map file. All other key mappings in the frs.map file remain active.

The ibmpc.map mapping file is designed to accommodate PC computers with older keyboards having only ten function keys. This mapping file assigns functions to only ten PC function keys.

You can edit the mapping files or create an alternate mapping file. However, keep in mind the physical key assignments in the termcap file, because the label assigned to a pf key must match the physical key associated with that pf key.

The following table depicts the FRS object-to-key mappings in the standard FRS mapping files. For reference purposes, the table displays the mapping file settings in tabular form and alphabetical order, within the three groups of FRS objects (FRS keys, menu items, and FRS operations).

FRS Object & Function	frs.map	ibmpc.map
frskey1 Help (invoke help facility)	N/A	pf1 (F1)
frskey2 Quit (exit the system)	N/A	pf6 (F6)
frskey3 End (return to previous frame)	N/A	pf10 (F10)
frskey4 Go (execute function)	N/A	pf9 (F9)
frskey5	N/A	pf37

<b>FRS Object &amp; Function</b>	<b>frs.map</b>	<b>ibmpc.map</b>
Top (move to top of table field)		(Home)
frskey6	N/A	pf38
Bottom (move to bottom of table field)		(End)
frskey7	N/A	pf5
Find (search table field for specified string)		(F5)
frskey8	N/A	pf3
Save object in database		(F3)
frskey9	N/A	pf7
Undo or Forget (undo last action or forget changes made in frame)		(F7)
frskey10	N/A	pf8
List the available choices		(F8)
menuitem1	N/A	pf11
Select first menu item		(Sh+F1)
menuitem2	N/A	pf12
Select second menu item		(Sh+F2)
menuitem3	N/A	pf4
Select third menu item - often edit		(F4)
menuitem4	N/A	pf14
Select fourth menu item		(Sh+F4)
menuitem5	N/A	pf15
Select fifth menu item		(Sh+F5)
menuitem6	N/A	pf16
Select sixth menu item		(Sh+F6)
menuitem7	N/A	pf17
Select seventh menu item		(Sh+F7)
menuitem8	N/A	pf18
Select eighth menu item		(Sh+F8)
menuitem9	N/A	pf19
Select ninth menu item		(Sh+F9)
menuitem10	N/A	pf20

<b>FRS Object &amp; Function</b>	<b>frs.map</b>	<b>ibmpc.map</b>
Select tenth menu item		(Sh+F10)
clear Clear remainder of field	controlX (Ctrl-X)	controlX (Ctrl+X)
clearrest Clear remainder of field (and move to next field?)	N/A	
deletechar Delete the character under the cursor	N/A	pf36 (Delete)
downline Move down one line	controlG (down arrow)	
duplicate Auto-duplicate value while in fill mode	controlA (Ctrl-A)	controlA (Ctrl+A)
editor Start default text editor on field	controlV (Ctrl-V)	controlV (Ctrl+V)
leftchar Move left one space within a field		controlR (left arrow)
menu Moves cursor to the menu (ring menus require the Esc key)		pf34 (Esc)
mode Switch between insert and overstrike mode	controlE (Ctrl+E)	pf35 (Ins)
newrow Move to first column of next row in table field	controlN (Ctrl+N)	controlN (Ctrl+N)
nextfield Move cursor to next field	controlI (Tab)	controlI (Ctrl+I)
nextitem Move cursor to next field without clearing this field	N/A	
nextword Move to next word in this field	controlB (Ctrl+right arrow)	

<b>FRS Object &amp; Function</b>	<b>frs.map</b>	<b>ibmpc.map</b>
previousfield Move cursor to previous field	controlP (Sh+Tab)	controlP (Sh+Tab)
previousword Move to previous word in this field	controlR (Ctrl+left arrow)	
printscreen Sends a copy of the currently displayed form to a file or the printer, depending on how the II_PRINTSCREEN_FILE environment variable/logical has been set. For more information, see the <i>System Administrator Guide</i> for the system on which your database resides.	controlF (Ctrl+F)	
redraw Redraw the screen	controlW (Ctrl+W)	ControlW (Ctrl+W)
rightchar Move right one space within a field	controlL (right arrow)	controlL (right arrow)
rubout Delete character immediately to left of the cursor	controlH (Backspace)	controlH (Backspace)
scrolldown Previous screen or set of rows in table field		pf39 (PgDn)
scrollleft Scroll left on a form	controlO (Ctrl+PgDn)	controlO (Ctrl+O)
scrollright Scroll right on a form	c	ControlU (Ctrl+U)
scrollup Next screen or set of rows in table field		pf40 (PgUp)
shell Spawns a DOS shell		
upline Move up one line	controlK (up arrow)	

## Application Mapping Files (Windows)

An application created with 4GL or one of the embedded query languages can use its own set of key mappings, invoked by the `set_frs` or `set_forms frs` (for 4GL) statements. Applications developers can use these statements in an application to read in a key-mapping file that is specific to that application or to map a FRS key to a function or control key explicitly in the application code.

An application mapping file takes precedence over the FRS mapping file. When an application starts up, the FRS merges all mapping files and resolves any conflicts between mapping statements by giving precedence to the statement in the mapping file of highest precedence. The FRS always honors the most *recent* reference to any mappable key in the file with the higher level of precedence.

For more information on application key mapping, see the *Forms-based Application Development Tools User Guide*.

## Key Mapping Overview (UNIX and VMS Environments)

Before you can take advantage of the function key feature, the terminal must be defined to Ingres so that *physical* keys on the terminal become associated with *logical* functions. Through this mapping, the physical keys can execute various operations that are either built into the forms system or programmed inside the forms-based tools and custom applications.

These files are involved in defining keyboard key usage in the FRS:

- Termcap file
- Key mapping files

The locations (escape sequences) and availability of function (PF), control, and arrow keys are unique to the type of terminal you are using. Control keys are available on all ASCII terminals, but only certain types of terminals also support function keys.

The termcap file contains a description of all terminals supported by Ingres, including their available function, control, and arrow keys. When you start the FRS, it reads the appropriate terminal type description from the termcap file. Before using Ingres, make sure your terminal has been defined to the FRS in the termcap file.

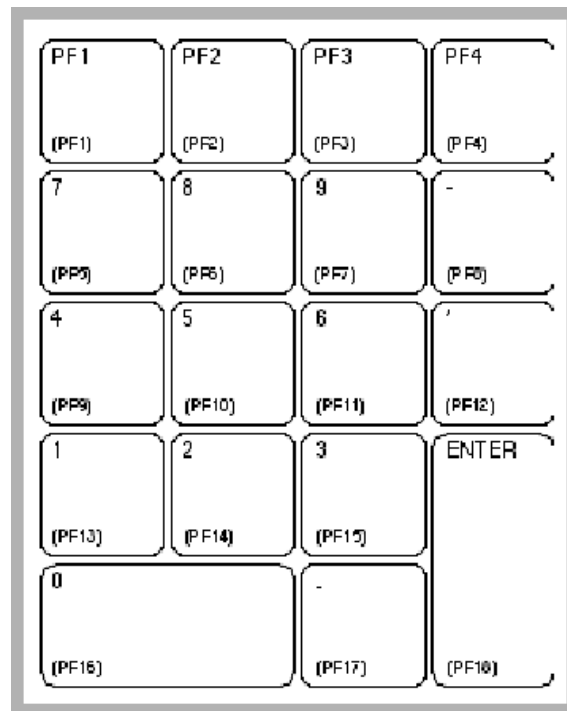
Once the terminal has been defined in the termcap file, you can change FRS key definitions dynamically in the key mapping files at the installation, terminal type, user, and application mapping levels. This appendix describes the key mapping process in detail.

## Role of the Termcap File (UNIX and VMS)

Fundamental changes to the FRS's interpretation of keystrokes for a particular terminal require modification of that terminal's termcap description. For example, if you want the FRS to permanently interpret a Down Arrow key as nextfield for the VT100i, you must change the termcap description for VT100i. Key map files cannot accomplish such changes directly.

Certain termcap files allow you to define the terminal to recognize function keys. For example, to use function keys with a VT100 or VT100-like terminal, the terminal must be defined as vt100i.

The vt100i definition turns the terminal's keypad into a set of 18 function keys, as shown in the following figure.



In the figure, the numbers correspond to the numbers that appear on the keys, while those in parentheses correspond to which function key that key is (for example, key #7 is the fifth function key (PF5)).

**VMS:** Function keys do not work as expected if the user has issued the following VMS command:

```
set term/uppercase
```

This is because this VMS set term command translates all characters received from the terminal into uppercase characters. The escape sequences for function keys are defined with lowercase characters in the termcap description. To correct this situation, create a new termcap description to define function key escape sequences with uppercase characters. ■

By default, a VT220 terminal uses function keys; therefore this terminal must be defined as vt220i. A VT220 can also emulate a VT100 terminal. If the terminal itself is set this way, it is appropriately defined to Ingres as vt100i. Other types of terminals with function keys can be defined to accept function key mappings by editing the termcap file descriptions for the terminal types. For more information, see the appendix "Writing Termcap Descriptions."

Users of terminals defined as vt100i or vt220i can also use arrow keys to move the cursor. For these terminals, the arrow keys are defined by default in the Ingres code to perform standard arrow key actions; for example, uparrow = upline and downarrow = downline. The use of arrow keys on any other terminal type can require the editing of its termcap file description.

No special terminal definition is required for control keys. Certain control keys, however, are reserved by the operating system for its own use and must not be mapped to any operations. These can include, but are not limited to:

- Control-C
- Control-O
- Control-Q
- Control-S
- Control-T
- Control-X
- Control-Y

Also, if your terminal uses escape sequences to define function keys (the case for terminals defined as vt100i and vt220i and most other terminals), you must also consider Escape as reserved.



## Mapping Files (UNIX and VMS)

Typically, entries within *mapping* files define a connection between logical FRS mapping objects, which can be accessed by the forms system, and program function (pf) keys, control keys, and arrow keys on a keyboard.

There are different types of FRS mapping objects. A FRS object can be any one of the following:

- FRS command
- Menu item
- FRS Key

An example of an FRS object is the FRS command menu. FRS commands are built-in functions of the FRS, such as moving to the menu line or moving to the next field, that enable the user to view and edit the data on a form. The FRS defines the behavior of each FRS command. For instance, the menu command moves the cursor to the menu line.

A mapping file maps these functions to the function, control, and arrow keys that are defined for your terminal in the termcap file. The key mapping sequence in the mapping file and the termcap file can be depicted as follows:

In the mapping file as:

FRS mapping object = pf, control, or arrow key

In the termcap file as:

pf, control, or arrow key = escape sequence

For example, the key mapping sequence for the previous field command can be represented as follows:

In the mapping file as:

**menu** FRS command = **pf1**

In the termcap file as:

**pf1** = escape sequence for pf1

The actual mapping statement in the FRS mapping file is:

```
menu = pf1 (PF1) /* Menu (menu key) */
```

The key name shown in parentheses is a *label* that indicates to the user which physical key to press. The functionality of each key mapping is shown as a comment statement between the characters */\** and *\*/*.

This mapping scheme provides flexibility in assigning functions to keyboard keys, without requiring editing of the termcap file. You can easily assign any FRS mapping object to a different key by changing the mapping statement in the mapping file, or by overriding it with a statement in a mapping file of higher precedence. For more information, see Levels of Mapping (UNIX and VMS) (see page 632).

## Mapping File Example (UNIX and VMS)

The following example of a mapping file illustrates the full range of statements available to specify any sort of mapping:

```
/* This is an example of a mapping file */
menuitem2 = pf3 (Key 3)
  menuitem3 = controlE (^E)
  frskey7 = pf8
previousfield = controlP
nextfield = rightarrow
rubout = controlDEL
controlA = off
/* this turns control-A off */
pf7 = off
```

The first line in the mapping file sample is a *comment*. Comments can appear anywhere in a mapping file. Their purpose is to provide information to someone looking at the file; they are ignored by the FRS.

The next five lines are examples of *mapping statements*. All mapping statements follow the same basic syntax. To the left of the equal sign is a *mapping object*, which specifies the operation or function to which the key is being mapped. To the right of the equal sign is the internal designation for the function, control, or arrow key that maps to, and activates, the mapping object. For instance, the following statement specifies that the second item on each menu line (menuitem2) maps to program function key pf3:

```
menuitem2 = pf3 (Key 3)
```

This mapping statement contains a *label* within parentheses to the right of the internal program function key. The specified label appears next to the second menu item on the user's screen to indicate what key to press to activate the menu selection (in this case, key 3). The label also appears in the Help Keys operation display. If no label is specified in the mapping statement, an appropriate default label is displayed on the menu line and in the Help Keys display. If you specify the label explicitly, it must indicate the physical key on the keyboard that generates the correct escape sequence, as defined in the termcap file, for the specified program function key, control key, or arrow key.

The following statement enables the user to move the cursor to the previous field on a form by pressing Control-P:

```
previousfield = controlP
```

This mapping statement does not contain a label.

The last two statements in the example are known as *disabling statements*. A disabling statement is used to disable a function, control, or arrow key. For example, the following statement turns off the key mapped to the pf7 program function key:

```
pf7 = off
```

This means that the key has no effect in a forms program governed by this map file, and merely beeps when pressed.

## Levels of Mapping (UNIX and VMS)

The FRS allows you to define key mappings for function, control, and arrow keys on four separate levels:

- Installation
- Terminal type
- User (environment)
- Application

This allows you to tailor key definitions to the specific requirements of the environment, the terminal type, the user, and the application. The installation-level mapping has the lowest precedence, and is overridden by all other mappings. Through terminal type-level mapping, a default can exist for all terminals of a given type, such as VT100s or VT220s. This default is overridden by any conflicting user or application mappings. The user-level mapping is next in precedence, allowing each individual user a good degree of latitude in the use of function, control, and arrow keys. Application-specific mappings have the highest precedence to provide the same environment to all users of an application and to ensure data integrity both in the form and the database.

When the FRS starts up, the four levels of mapping are merged and conflicts are resolved based on the precedence outlined previously. Any additional mapping specified by the program during the running of an application is merged in the same manner and takes precedence over conflicting mappings. While the four levels of mappings can coexist, any of the levels can be omitted. Because it is possible that a function, control, or arrow key is defined at more than one of the four levels, the FRS always honors the most *recent* reference to any mappable key from a file of higher-level precedence.

The following sections discuss each of the various levels at which mapping occurs and how to specify the mapping file for each level.

## Installation Level Mapping (UNIX and VMS)

As mentioned previously, installation-level mapping has the lowest precedence. It provides an underlying default, common to all terminal types, which can be overlaid with mappings for specific terminal types, as well as individual users and applications.

A default installation-level mapping file is shipped with Ingres. Because this default file references only control keys and not function keys, it must be usable for all Ingres terminal types.

The complete file specification for this mapping file is:

### **UNIX:**

`$II_SYSTEM/ingres/files/frs.map`

### **VMS:**

`II_SYSTEM:[ingres.files]frs.maps`

It contains the following statements:

```
/* Move cursor to next field */
nextfield = controlI (Tab)
/* Move cursor to previous field */
previousfield = controlP (^P)
/* Move up one word within field */
nextword = controlB (^B)
/* Move back one word within field */
previousword = controlR (^R)
/* Switch between insert and */
/* overstrike mode*/
mode = controlE (^E)
/* Redraw the screen */
redraw = controlW (^W)
/* Delete character under the cursor */
deletechar = controlD (^D)
/* Delete character to left of cursor */
rubout = controlDEL (Delete)
/* Start default text editor on field */
editor = controlV (^V)
/* Move left one space within a field */
leftchar = controlH (^H)
/* Move right one space within a field */
rightchar = controlL (^L)
/* Move down one line */
downline = controlJ (^J)
/* Move up one line */
upline = controlK (^K)
/* Move to first column of next row */
/* in table field */
newrow = controlN (^N)
/* Clear the field */
clear = controlX (^X)
/* Clear out rest of field */
/* and move to next field */
clearrest = controlM (Return)
/* Scroll up on the form */
scrollup = controlF (^F)
/* Scroll down on the form */
scrolldown = controlG (^G)
/* Scroll left on a form */
scrollleft = controlO (^O)
/* Scroll right on a form */
scrollright = controlU (^U)
/* Auto-duplicate value while in fill mode*/
duplicate = controlA (^A)
```

While you can edit this file to customize your installation's default mappings, you must *not* change the name of this file. The FRS automatically looks for the file when starting up. If you do modify this file, be sure to map only those keys that are available for all terminal types at your installation.

Notice that the file does not specify a menu key. This is because the FRS command menu automatically defaults to Escape.

## Terminal Type Level Mapping

The next higher level of mapping is terminal type. Each terminal type used at your installation needs its own mapping file, because function key support varies from terminal to terminal. Combined with the installation mapping, this file provides a common terminal default for the function and control keys, which can be altered by mappings at higher levels to fit the needs of an application or an individual user. Arrow keys are defined by default in the termcap file.

The terminal-type mapping file must be placed in the following directory:

### **UNIX:**

`$II_SYSTEM/ingres/files/`

### **VMS:**

`II_SYSTEM:[ingres.files]`

You can specify the terminal-type mapping file name with the `mf` capability in the termcap file description for each terminal type in use at an installation. You can also point to a termcap file by using the `II_TERMCAP_FILE` environment variable/logical.

Default mapping files for VT100 and VT220 terminals are shipped with Ingres. You can edit these files if desired. (The termcap file descriptions for the `vt100i` and `vt220i` terminal definitions already have the names of their mapping files specified; therefore, there is no need to edit those termcap descriptions.)

### Terminal Type vt100i Map File

For VT100 terminals defined as the vt100i terminal type, the mapping file location is:

**UNIX:**

`$II_SYSTEM/ingres/files/vt100i.map`

**VMS:**

`II_SYSTEM:[ingres.files]vt100i.map`

It contains the following statements:



```

/* Menu Key */
menu = pf1 (PF1)
/* Help facility */
frskey1 = pf2 (PF2)
/* Quit from program */
frskey2 = pf4 (PF4)
/* End current screen and return */
/* to previous screen */
frskey3 = pf3 (PF3)
/* Go or execute function */
frskey4 = pf18 (Enter)
/* Put cursor on top of form or */
/* table field*/
frskey5 = controlK (^K)
/* Put cursor on bottom of form or */
/* table field*/
frskey6 = controlJ (^J)
/* Find next occurrence of string*/
/* in this column of table field      */
frskey7 = controlF (^F)
/* Save object in database */
frskey8 = pf16 (0)
/* Cancel and undo */
frskey9 = pf17 (.)
/* ListChoices */
frskey10 = pf7 (9)
/* Scroll page or form left */
scrollleft = controlL (^L)
/* Scroll page or form right */
scrollright = controlH (^H)
/* Previous screen or set of rows */
/* in table field */
scrolldown = pf8 (-)
/* Next screen or set of rows in */
/* table field */
scrollup = pf12 (,)
/* Print contents of current screen */
/* to file or printer */
printscreen = controlG (^G)
/* Select first menu item */
menuitem1 = pf13 (1)
/* Select second menu item */
menuitem2 = pf14 (2)
/* Select third menu item */
menuitem3 = pf15 (3)
/* Select fourth menu item */
menuitem4 = pf9 (4)
/* Select fifth menu item */
menuitem5 = pf10 (5)
/* Select sixth menu item */
menuitem6 = pf11 (6)
/* Select seventh menu item */
menuitem7 = pf5 (7)
/* Select eighth menu item */
menuitem8 = pf6 (8)
/* Move cursor to next field */
/* defined to controlI in frs.map */
/* Move cursor to previous field */
/* defined to controlP in frs.map */

```

```
/* Move up one word within field */
nextword = controlU (^U)
/* Move back one word within field */
/* defined to controlR in frs.map */
/* Switch between insert and overstrike */
/* mode defined to controlE in frs.map */
/* Redraw the screen defined to controlW */ /* in frs.map */
/* Delete the character under the cursor */
/* defined to controlD in frs.map */
/* Delete character immediately to left */
/* of cursor--defined to controlDEL */
/* in frs.map */
/* Start default text editor on field */
/* defined to controlV in frs.map */
/* Move to first column of next row */
/* in table field defined to controlN */
/* in frs.map */
```

#### **UNIX:**

```
/* Clear the field */
```

```
clear = controlX (^X) 
```

#### **VMS:**

```
/* Clear the field */
```

```
clear = controlB (^B) 
```

```
/* Move to nextitem in form. If on */
/* regular field, move to next field. */
/* If in table field, move to next column */
/* if NOT in last accessible column.*/
/* Move to next row if in last accessible */
/* column of table field. */
nextitem = controlM (Return)
/* Auto-duplicate value while in */
/* fill mode defined to controlA */
/* in frs.map */
```

## Terminal Type vt220i Map File

The default file for VT220 terminals is located in:

### **UNIX:**

`$II_SYSTEM/ingres/files/vt220i.map`

### **VMS:**

`II_SYSTEM:[ingres.files]vt220i.map`


The VT220 mapping file contains these statements:

```
/* Menu Key */
menu = pf1 (PF1)
/* Help facility */
frskey1 = pf15 (Help)
/* Quit from program */
frskey2 = pf4 (PF4)
/* End current screen and */
/* return to previous screen */
frskey3 = pf3 (PF3)
/* Go or execute function */
frskey4 = pf16 (Do)
/* Put cursor on top of form or */
/* table field */
frskey5 = controlK (^K)
/* Put cursor on bottom of form or */
/* table field */
frskey6 = controlJ (^J)
/* Find next occurrence of string */
/* in this column of table field */
frskey7 = pf21 (Find)
/* Save function */
frskey8 = pf10 (PF10)
/* Undo and cancel */
frskey9 = pf2 (PF2)
/* Bring Up ListChoices */
frskey10 = pf20 (PF20)
/* Scroll page or form left */
scrollleft = controlL (^L)
/* Scroll page or form right */
scrollright = controlH (^H)
/* Previous screen or set of rows */
/* in table field */
scrolldown = pf25 (Prev Screen)
/* Next screen or set of rows */
/* in table field */
scrollup = pf26 (Next Screen)
/* Print contents of current screen */
/* to file or printer */
printscreen = pf8 (PF8)
/* Select first menu item */
menuitem1 = pf11 (PF11)
/* Select second menu item */
menuitem2 = pf12 (PF12)
/* Select third menu item */
menuitem3 = pf13 (PF13)
/* Select fourth menu item */
menuitem4 = pf14 (PF14)
/* Select fifth menu item */
menuitem5 = pf17 (PF17)
/* Select sixth menu item */
menuitem6 = pf18 (PF18)
/* Select seventh menu item */
menuitem7 = pf19 (PF19)
/* Remove character under cursor */
deletechar = pf23 (Remove)
/* Switch between insert and overstrike */
mode = pf22 (Insert Here)
/* Move cursor to next field defined */
/* to controlI in frs.map */
```

```
/* Move cursor to previous field */
/* defined to controlP in frs.map */
/* Move up one word within field */
  nextword = controlU (^U)
/* Move back one word within field */
/* defined to controlR in frs.map */
/* Redraw the screen defined to */
/* controlW in frs.map */
/* Delete character immediately to left */
/* of cursor defined to controlDEL */
/* in frs.map */
/* Start default text editor on field */
/* defined to controlV in frs.map */
/* Move to first column of next row in */
/* table field defined to controlN */
/* in frs.map */
```


**UNIX:**

```
/* Clear the field */
```

```
clear = controlX (^X) 
```

**VMS:**

```
/* Clear the field */
```

```
clear = controlB (^B) 
```

```
/* Move to nextitem in form. If on */
/* regular field, move to next field. */
/* If in table field, move to next column */
/* if NOT in last accessible column.*/
/* Move to next row if in last accessible */
/* column of table field. */
  nextitem = controlM (Return)
/* Auto-duplicate value while in fill */
/* mode defined to controlA in frs.map */
```

## User Level Mapping (UNIX and VMS)

Next higher in precedence are the individual user's mappings. Of the four levels, user-level mapping is probably the least frequently used; the combination of mappings at the other three levels suffices for most users.

To make the user-level mapping file known to the FRS, the user must execute the following command at the operating system level:

### UNIX:

For the C shell:

```
setenv INGRES_KEYS full_pathname/file_name
```

For the Bourne shell:

```
INGRES_KEYS=full_pathname/file_name  
export INGRES_KEYS
```

The parameter *full-pathname/file\_name* is the full pathname and file name for the mapping file. To eliminate the need to invoke this command for each terminal session, you can include this command in the file `.login` (C shell) or `.profile` (Bourne shell). ■

### VMS:

```
define INGRES_KEYS file_specification
```

The *file\_specification* parameter is the full specification for the mapping file. To eliminate the need to invoke this command for each terminal session, you can include this command in the `login.com` file.

## Application Level Mapping (UNIX and VMS)


An application created with 4GL or one of the embedded query languages can use its own set of mappings, invoked by the `set_frs frs` or `set_forms frs` (for 4GL) statements. Applications developers can use these statements in an application to read in a key-mapping file that is specific to that application or to map an FRS key to a function, control, or arrow key explicitly in the application code.

Application-level mappings take precedence over all other mappings. For more information on application key mapping, see the *Forms-based Application Development Tools User Guide*. ■

## Obtaining Information on Mappings

Within a forms-based system, such as QBF, a user can invoke the Help menu operation to find out the current settings for function and control keys, including (if UNIX or VMS) arrow keys. For more information, see online help. Within an embedded query language or 4GL application, you can get information about function, control, and arrow key mappings and other information about the FRS by using the `inquire_frs` command (`inquire_forms` in 4GL).

You can use this command to find out which key is mapped to a specific mapping object, the name of the application mapping file, or the current setting (on or off) of the menu map (the display of key labels on the menu line).

**Windows:** The button bar menu does not display associated function or control keys. Use the Help Keys operation to see these key mappings, or use the Menu and arrow keys to move the input focus bar to a menu item; the associated key appears in the message window on the bottom line of the main window. 

You can also obtain information about the user's terminal type so that application mapping can be coded conditionally, according to the particular user's terminal. (For complete information on `inquire_frs` and `inquire_forms`, see the *Forms-based Application Development Tools User Guide*.)

## FRS Mapping Objects

You can map function and control keys (and arrow keys in UNIX or VMS) to several types of objects. These mappings allow the keys to be used to perform menu item operations, cursor movement, and virtually any other functions available in a forms application.

The three types of mapping objects are:

- FRS commands
- Menu items
- FRS keys

The following sections describe each type of mapping object in detail.

## FRS Commands

FRS commands are built-in functions of the forms system that enable the user to view and edit the data on a form. Because they are actually built into FRS, they are available both in the Ingres forms-based tools, such as QBF, and in customized forms applications built with Vision, ABF, or the embedded query languages.

Through key mapping, you can link such capabilities as deleting a character, moving to the menu line, or scrolling within a table field, to program function and control keys (and arrow keys in UNIX or VMS), which are defined to Ingres as specific keyboard keys. For example, the following statement maps the redraw FRS command to controlW, which is defined to Ingres on most keyboards as the physical key, Control-W:

```
redraw = controlW
```

FRS commands allow you to write terminal-independent application programs while taking advantage of available mapping keys. The application does not specify which control or escape sequences the user must generate at the keyboard to run a block of code; it just specifies which FRS command runs that block of code. FRS commands cannot be executed directly in program code; they are only available to the user through key mapping. You can map any function or control key to any FRS command. You can also map any arrow keys to any FRS command.



## FRS Command Definitions

The FRS commands and their definitions are as follows. When used as a mapping object, an FRS command name must be typed exactly as shown:

**clear**

Clears field or menu input.

**clearrest**

Clears rest of field, beginning from current cursor position. Then moves to next field if cursor is not in a table field, to next column if not in last column of table field, to first column of next row if in last column of table field.

**deletechar**

Deletes character under cursor.

**downline**

Moves down one line in field or next row in table field.

**duplicate**

Enters a value in a simple field that duplicates a value in the previous row of a table. For more information, see Duplicate Previous Entries.

**editor**

Starts text editor on field.

**leftchar**

Moves left one character within field.

**menu**

Goes to the menu line (the Menu key).

**mode**

Switches between insert and overstrike editing mode.

**newrow**

Moves to first column of next row in table field.

**nextfield**

Goes to the next field on the form.

**nextitem**

Moves to next field if cursor is not in a table field, to next column if not in last column of table field, to first column of next row if in last column of table field. Unlike clearrest, does not clear rest of current field.

**nextword**

Moves forward one word within a field.

**previousfield**

Goes to the previous field on the form.

**previousword**

Moves backward one word within a field.

**printscreen**

Sends a copy of the currently displayed form to a file or the printer, depending on how the II\_PRINTSCREEN\_FILE environment variable/logical has been set. For more information, see the *System Administrator Guide* for the system on which your database resides.

**redraw**

Redraws the frame.

**rightchar**

Moves right one character within field.

**rubout**

Deletes character immediately to left of cursor.

**scrolldown**

Scrolls down in current table field or form, leaving cursor on same field.

**scrollleft**

Scrolls form left, leaving cursor on same field.


**scrollright**

Scrolls form right, leaving cursor on same field.

**scrollup**

Scrolls up in current table field or form, leaving cursor on same field.

**Upline (Windows only)**

Move up one line in field or previous row in table field. 

**Shell (UNIX only)**

In UNIX and VMS, spawns (creates) a shell.

In UNIX, spawns a Bourne shell only. 

**upline (VMS only)**

In VMS, creates a DCL process.

In UNIX and VMS, move up one line in field or previous row in table field.



Default mapping files are provided that assign keys to some or all of the FRS commands.

## Default Key Mapping Assignments (UNIX and VMS)

Ingres provides several default mapping files that contain key assignments for the FRS commands. The first is an installation-level mapping file, valid for all terminal types. The other two are default-mapping files for the VT100 and VT220 terminals. These mapping files assign default control or function keys to the FRS commands. The installation file assigns control keys to most of the FRS commands. The terminal-type files expand and, in certain instances, override the installation mappings.

The following table lists the FRS commands and their default installation-level and terminal-type level assignments for the vt100i and vt220i terminal types. Consult your system administrator to determine whether the defaults listed in the table are valid for your installation and terminal type. Check the files directory for other mapping files supplied with Ingres.

**Note:** The system administrator can modify the default mapping files provided with Ingres. If the files have been modified, the mappings for the FRS commands can differ from the following table. The system administrator can also create mapping files for other terminal types besides vt100i and vt220i. These terminal-type files would then override the installation-level file.

Mapping files for VT100 and VT220 terminals have been optimized to avoid unnecessary remapping functions that are already designated in the installation-level frs.map file. If you modify the frs.map file, be careful to ensure the reliability of the mapping files for the VT100 and VT220 terminals.

FRS Command	Installation (UNIX and VMS)	VT100i (VMS)	VT220i (VMS)
Clear	CONTROL-X	CONTROL-B	CONTROL-B
Cleararrest	RETURN	N/A	N/A
Deletechar	CONTROL-D	CONTROL-D	REMOVE
Downline	CONTROL-J	down_arrow	down_arrow
duplicate	CONTROL-A	CONTROL-A	CONTROL-A
editor	CONTROL-V	CONTROL-V	CONTROL-V
leftchar	CONTROL-H	left_arrow	left_arrow
menu	ESC	PF1	PF1
mode	CONTROL-E	CONTROL-E	INSERT HERE
newrow	CONTROL-N	CONTROL-N	CONTROL-N
nextfield	TAB	TAB	TAB
nextitem	no default	RETURN	RETURN

<b>FRS Command</b>	<b>Installation (UNIX and VMS)</b>	<b>VT100i (VMS)</b>	<b>VT220i (VMS)</b>
nextword	CONTROL-B	CONTROL-U	CONTROL-U
previousfield	CONTROL-P	CONTROL-P	CONTROL-P
previousword	CONTROL-R	CONTROL-R	CONTROL-R
printscreen	(no default)	CONTROL-G	PF8
redraw	CONTROL-W	CONTROL-W	CONTROL-W
rightchar	CONTROL-L	right_arrow	right_arrow
rubout	DEL	DEL	DEL
scrolldown	CONTROL-G	PF8	PREV SCR
scrollleft	CONTROL-O	CONTROL-L	CONTROL-L
scrollright	CONTROL-U	CONTROL-H	CONTROL-H
scrollup	CONTROL-F	PF12	NEXT SCR
shell	no default	no default	CONTROL-B
upline	CONTROL-K	up_arrow	up_arrow

## Menu Items

You can also map a function or control key (or arrow key in UNIX or VMS) to any of the items appearing on a menu line. This mapping occurs by position within the menu line.

The syntax for specifying a menu item is:

```
menuitemN = internal_key_name (key_label)
```

The *N* indicates the position of the menu item in the line and is in the range 1 to 25. Alternatively, you can also designate menuitem*N* as menu*N* in a mapping statement.

For example, the following statement maps the second item on the menu line to function key 3:

### **Windows:**

```
menuitem2 = F3
```

### **UNIX and VMS:**

```
menuitem2 = pf3
```

This statement causes F3 (Windows) or PF3 (UNIX or VMS) to perform the operation indicated by the second menu item. As the end user moves to a new frame and the menu changes, F3 or PF3 continues to correspond to the item in the second position on the new menu line.

By default, the menu line or Help Keys operation automatically displays the current mappings between the menu items and their associated function and control keys (and arrow keys in UNIX or VMS). It uses either the label provided in the mapping file, or a default label if none has been specified in the mapping file.

For example, assuming such mappings have been specified, a menu can appear like this:

```
Help(PF2) Add(PF3) Editor(control-E) End(PF4)
```

or

```
Help(F1) Add(F3) Editor(Ctrl-e) End(F4)
```

The button bar menu does not display associated function or control keys. Use the Help Keys operation to see these key mappings.

In this example, pressing PF2 or F1 is equivalent to moving to the menu line and typing Help. Similarly, PF3 or F3 substitutes for Add; Ctrl-E substitutes for Editor; and PF4 or F4 substitutes for End.

The text in parentheses in the mapping statement and on the menu line shows the corresponding function or arrow or control key and is known as a *label*. The label is specified in the mapping file. For more information, see Mapping File Syntax (see page 654). An application can specify that the entire *menu map*, as the collective set of labels is known, be turned off so that no labels appear. For example, if the menu map for the preceding example were turned off, the menu line appears like this:

```
Help  Add  Editor  End
```

However, the function and control keys still retain their correspondence with the menu items, even though that correspondence is no longer displayed.

For more information on how to turn off the menu map, see the *Forms-based Application Development Tools User Guide*. The relevant statements are `set_forms` in 4GL and `set_frs` in the embedded query languages.

## FRS Keys

FRS keys enable you to map function and control keys (including arrow keys) directly to a program operation. While a function, control, or arrow key can always map to a menu item operation by *position*, you can map the key directly to the operation itself.

For instance, you might not want every operation to appear as a menu item, particularly if the operation is available across all the frames in an application. By requiring that certain repeated operations be invoked only through function or control or arrow keys, you can shorten long menu lines and use the menu primarily for operations that are unique to each frame.

Because not all keyboards have the same set of function keys, it makes little sense to assign an operation directly to a particular key inside an application. Instead, you equate operations with logical FRS keys. The FRS keys can then be mapped to function, control, or arrow keys at any of the four mapping levels.

FRS keys invoke operations only when specified in the application. A mapping file, by itself, merely assigns the FRS key to a function or control or arrow key; it cannot define the meaning of the FRS key. A FRS key must be equated with an operation when you write the application. For instructions on how to do this, see the *Using Forms-based Application Development Tools User Guide*. The Ingres forms-based tools also use FRS keys extensively. For more information, see Predefined FRS Keys (see page 653).

You can define the same operation for both a menu item and a FRS key. The end user can then activate the operation either with the key that is mapped to the menu item's position or with the key that is mapped to the FRS key. This means the user is always be able to invoke the menu item with the same function key, even if the item's position on the menu line changes from frame to frame. Whenever an operation has been associated with both a menu item and an FRS key, the label for the menu item indicates the function or control or arrow key that maps to the corresponding FRS key, *not* the function or control or arrow key that maps to the item's position on the line.

## Map a FRS Key

For the end user to invoke an operation through an FRS key, two conditions must be met:

- The application program must provide the operation code and specify that it be invoked or activated by a FRS key.
- The FRS key must be mapped to a physical key on the user's keyboard.

Mapping the FRS key to a function or control key is handled identically to mapping a menu item or FRS command. If an FRS key defined in an application is *not* mapped to a function or control or arrow key, the end user has no way of accessing that FRS key's operation, unless the operation itself has also been defined for a menu item.

### To map a FRS key

An FRS key is designated by the keyword `frskey`, followed by an integer in the range 1 to 40.

For example, the following statement maps FRS key 7 to function key 2:

```
frskey7 = f2
```

By pressing PF2 or F2, the user invokes the operation to which FRS key 7 has been defined, within the current frame.



## Predefined FRS Keys

The Ingres forms-based tools consistently equate certain FRS keys with standard menu item operations. Because of this, the end user can always use the same function or control or arrow key to perform that operation, no matter what its position on the menu line. Because the end user does not need to refer to the menu item's label, these operations are usually located at the end of the menu line, allowing those operations that are unique to a particular frame to appear first. For example, if frskey1 is mapped to control-H, the end user can always get help by pressing Ctrl+H.

FRS key operations are predefined in the applications code for the Ingres forms-based tools as shown in the following table. The terminal level key-mapping files contain default key assignments for these FRS keys. For more information, see FRS Mapping File (Windows) (see page 619) and Terminal Type Level Mapping (see page 635) (for UNIX and VMS environments).

FRS Key	Menu Item	Meaning
frskey1	Help	Accesses as help facility.
frskey2	Quit	Exits the system.
frskey3	End	Exits the frame, returning to previous frame.
frskey4	Go/Next	Executes function.
frskey5	Top	Moves to top of table field.
frskey6	Bottom	Moves to bottom of table field.
frskey7	Find	Searches table field for specified string.
frskey8	Save	Saves object in database.
frskey9	Undo/Cancel	Undoes last action or cancels changes made in frame.
frskey10	ListChoices	Lists the available choices for the selected field.

An application developed with an embedded query language or 4GL cannot automatically have the same FRS key-menu item linkage. By using this table as a model for associating FRS keys with program operations, an application developer can help provide a consistent end-user interface for all custom applications that is also consistent with key usage in the Ingres forms-based tools.

Specifications written using the function key facility provided in earlier releases of Ingres continues to operate correctly. If desired, such specifications can be placed in a mapping file containing statements using the new function/control/arrow key mapping facility. In a future release, however, the function key facility is phased out; therefore, it is not documented here.

## Mapping File Syntax

Mapping files are the main method for mapping function, control, and arrow keys to menu items, FRS commands, and FRS keys. Mapping files can exist at each level of mapping in your environment. For example, you can have an installation-wide mapping file, terminal-type mapping files for every terminal type in your organization, user mapping files for each user, and application mapping files for every Ingres application.

In Windows, you can have several applications mapping files in addition to the FRS mapping file.

The syntax for the mapping files is the same across all levels of mapping. A mapping file can consist of three components:

- Mapping statements
- Disabling statements
- Comments

Mapping statements designate the actual mappings. These are the most commonly used. The mapping statements can also designate labels for menu items, if you do not use the default labels.

*Disabling statements* disable the use of function or control keys (and arrow keys in UNIX and VMS).

*Comments* provide explanatory text.

The statements in a mapping file can appear in any order, except for the placement of disabling statements (see page 659). Each statement must fit entirely on one line. Alphabetic characters within a mapping file can appear in either uppercase or lowercase, with no difference in meaning. Blank lines are ignored.

## Mapping Statements

A mapping statement has the following syntax:

```
mapping_object = pfM|controlX|uparrow|downarrow|leftarrow|rightarrow [(label)]
```

### ***mapping\_object***

Specifies a menu item, FRS command, or FRS key, designated through mapping.

### ***pfN***

Designates a function key. *N* must be in the range 1 to 40. (Note that the maximum number of definable keys set in the termcap file for your terminal can be less than 40. You must raise this limit to set additional keys.)

### ***controlX***

Designates a control key. *X* can be any single letter of the alphabet, or the designations *del* to indicate the Delete key as in *controldel* or *esc* to indicate the Escape key as in *controlesc* (if the Escape key is not reserved on your terminal). This guide designates control keys in user instructions as Control-*X*, where *X* is the key used in combination with the Control key.

On most keyboards, pressing Control-I is equivalent to pressing the Tab key, and pressing Control-M is equivalent to pressing the Enter key in Windows or the Return key in UNIX or VMS.

### ***Uparrow, Downarrow, Leftarrow, Rightarrow***

Designates the up arrow, down arrow, left arrow, and right arrow keys.

### ***label***

Designates any alphanumeric string. It appears in place of the default label for a menu item. It also appears in the Keys operation of the Help facility.

A single mapping object cannot be mapped to be invoked by pressing either two different function, control, or arrow keys on the same screen. Each function or control key (and each arrow key in UNIX or VMS) can correspond to only a single mapping object at a time. If you map one mapping object to more than one physical key, or multiple mapping objects to the same key, one mapping overrides the others.

If the conflicting mappings occur within the same mapping file, only the first mapping within the file is considered to be valid. If the conflict occurs between different mapping files, the key mapping in the file of higher precedence overrides the other mappings. For a discussion of precedence in key mapping, see the Key Mapping Overview section specific to your environment.

The appearance of an exception to this rule occurs when both of the following conditions exist:

- A menu item is mapped to a function or control key (or arrow key in UNIX or VMS) by the menu item's position
- A FRS key that has the same function as the menu item is mapped to a *different* function or control key (or arrow key in UNIX or VMS)

This only appears to be an exception, because the two mapping objects (the menu item and the FRS key) perform the same function although they are actually separate FRS objects that are mapped to two different function, control, or arrow keys.

## Example of Mapping Statements (Windows)

```
frskey8 = pf16 (Sh+F6)
menuitem1 = pf13 (Sh+F3)
menuitem2 = pf3 (F3)
menuitem3 = controlE (^E)
previousfield = controlP (Sh+Tab)
nextfield = controlI (Tab)
menuitem4 = pf9 (F9)
rubout = controlDEL
```

In this example, Sh+F6 on your PC keyboard maps to the operation associated with frskey8, Sh+F3 activates the first item on the menu line, F3 activates the second, Ctrl+E activates the third menu item, Sh+Tab moves the cursor to the previous field on the form, and Delete deletes the character immediately to the left of the cursor. Any previous mappings that do not conflict with these statements remain in effect. (When mapping *menu items*, the action to be taken after the key is pressed is determined by the application and can differ between applications.)

Notice the effect of including an explicit label in a mapping statement. Assume a frame's menu includes the following operations:

```
Help Add Editor End
```

Assume, also, that the frame containing these menu items also specifies that the Help operation be invoked either by selecting the Help menu item (in this case, menuitem1) or by pressing the key mapped to frskey8. The preceding mapping file example, with its labels, would result in the following key associations:

```
Help(Sh-F6) Add(F3) Editor(^E) End(F9)
```

Note the case of the Help menu item, which the application also allows to be invoked by frskey8. As shown in the mapping file example, two different keys could be used to invoke this item—Sh+F3 (the key associated with the first menu item) and Sh+F6 (the key associated with frskey8). In a case like this, the label for the FRS key always takes precedence over the label for the menu item for key assignments displayed on the menu line or in the Help Keys operation. All other labels in the example are those associated with the menu item's position on the menu line. ■

### Example of Mapping Statements (UNIX and VMS)

```
frskey8 = pf16 (0)
menuitem1 = pf13 (1)
menuitem2 = pf3 (PF3)
menuitem3 = controlE (^E)
previousfield = controlP
nextfield = rightarrow
menuitem4 = pf9 (4)
rubout = controlDEL
```

In this example, PF16 (the 0 on a VT100 keypad) maps to the operation associated with `frskey8`, PF13 (the 1 on a VT100 keypad) activates the first item on the menu line, PF3 activates the second menu item, Control-E activates the third menu item, Control-P moves the cursor to the previous field on the form, Right Arrow moves the cursor to the next field, and Delete deletes the character immediately to the left of the cursor. Any previous mappings that do not conflict with these statements remain in effect. (When mapping *menu items*, the action to be taken after the key is pressed is determined by the application and can differ between applications.)

Notice the effect that including an explicit label has on the appearance of a menu line. Assume a frame's menu includes the following operations:

```
Help Add Editor End
```

Assume, also, that the frame containing these menu items also specifies that the Help operation be invoked either by selecting the Help menu item or by pressing the key mapped to `frskey8`. The previous mapping file example, with its labels, would cause the menu to appear as follows:

```
Help(0) Add(PF3) Editor(^E) End(4)
```

Note the case of the Help menu item, which the application also allows to be invoked by `frskey8`. Two different labels from the map file could be used for this item—1 (the label for the first menu item), or 0 (the label for `frskey8`). In a case like this, the label for the FRS key always takes precedence over the label for the menu item. All other labels on the menu are those associated with the menu item's position in the menu line. ■

## How to Disable Statements

Turn off any of the function or control keys (or arrow keys in UNIX or VMS) by using one of the following methods:

- Replace the original mapping statement with a disabling statement in the same mapping file
- Comment out the original mapping statement and place a disabling statement in the same mapping file
- Place a disabling statement in a key map file of higher precedence than the original mapping statement

If you do not want to edit or comment out the original mapping statement, leave the original mapping statement as is and place the disabling statement in a mapping file of higher precedence. For example, to disable a key mapped to an object in an installation-level or terminal-type mapping file, put a disabling statement in a user or application-level mapping file. To disable a key mapped to an object in a user-level mapping file, put a disabling statement in an application-level mapping file.

Once disabled, a key remains disabled until used in a mapping statement within a mapping file of higher precedence. For a discussion of precedence in key mapping, see the Key Mapping Overview section specific to your environment.

A disabling statement has the following syntax:

```
pfN|controlX|uparrow|downarrow|rightarrow|leftarrow = off
```

### **Windows:**

The following statements disable Control-A and PF7 when placed in a mapping file or position of higher precedence than any assignment statement for these keys:

```
controlA = off
```

```
pf7 = off
```

### **UNIX and VMS:**

The following statements disable Control-A, PF7, and the Right Arrow key when placed in a mapping file or position of higher precedence than any assignment statement for these keys:

```
controlA = off
```

```
pf7 = off
```

```
rightarrow = off
```

While these statements are in effect, using these disabled keys produces a beep from the terminal.

To disable a FRS command in a UNIX or VMS environment, map the FRS command to a control or function key (or to an arrow key), and then disable that key.

## How to Use Comments

Comments are delimited by `/*` and `*/`. They can appear anywhere, including on the same line as a statement. The whole comment must appear within a single line, as shown:

```
/* This is a comment */  
controlA = off /*this turns Control-A off*/
```

## Mapping File Errors

When the FRS starts up, the mapping files for the various levels of mapping are merged and conflicts between files are resolved based on a specific precedence. When the FRS detects errors in a mapping file, you can find detailed error messages in the `ingkey.err` file in the user's current directory. After exiting the application, the user can look at the error file to determine the nature of the errors.



## Restrictions and Limitations (Windows Environment)

When defining function and control keys, beware of the following restrictions and limitations.

- The FRS has the following internal limitations: 40 program function (pf) keys, 40 FRS keys, and 25 menu items.
- The following control keys are reserved by the operating system for its own use and must not be mapped to any other operations: Ctrl+C, Ctrl+Q, Ctrl+S, and Ctrl+P.

Although Ctrl+P is trapped by the operating system before it reaches FRS, you *can* assign FRS objects to its *internal* designation, controlP, because controlP is mapped to the Sh+Tab key combination in the termcap file.

- FRS commands cannot be mapped to an FRS key. This is syntactically illegal because FRS commands and FRS keys both appear on the left side of the equals sign in the mapping statement.
- Positional menu item mapping cannot be turned off. You can see this if you look at the mapping file, for example:

```
menuitem1 = pf13 (Sh-F3)
```

If you assign a FRS key to an activation block, that key's label appears on the menu line or in the Help Keys operation in place of the default key's label. The default key, however, still works.

```
"Go", key frskey5 =
begin
...      [first menu item]
end;
```

In the previous example, both Sh+F3 and whatever is mapped to frskey5 triggers the Go operation, even though you only see the frskey5's label on the menu or in the Help Keys display. To deactivate this, you must assign another FRS function to the F key now assigned to menuitem1:

```
frskey10 = pf13 (Sh-F3)
```

Normally, you must use an unused FRS key for this function.

- Currently, map files are the only way to turn off a function or control key (for example, controlV = off). Keys cannot be turned off by the Ingres/4GL or embedded query language set command.

## Troubleshooting Checklist (Windows)

When troubleshooting your function and control key definitions, consider the following:

1. Have you checked the contents of the map error files in the working directory (ingkey.err or app\_ingkey.err)?

The error messages written to this file indicate map file problems. For example, if the same key is referenced twice, the warning error message is written to this file. If this file is empty, an error occurred before the map file was parsed, and an error message was sent to your screen.

For example, no forms statements can be executed prior to starting the FRS. If a set mapfile statement preceded ## forms or exec frs forms in the embedded query language program, the error message is written to the screen.

2. Have any of the key map file path names become invalid?

For example, if the file system was moved to a new device, the file pathnames referred to by INGRES\_KEYS or set mapfile are now invalid or perhaps file permissions were changed and the FRS cannot open the map files.

3. Are lower-level key definitions showing through on the user's menu line?

This is the result of the key map merging by the FRS. This most often occurs when you forget to remap or turn off an intended key in the application map file or do not realize that INGRES\_KEYS is also pointing to a map file. 🗑

## Restrictions and Limitations (UNIX and VMS Environments)

When defining function, control, and arrow keys, beware of the following restrictions and limitations:

- The FRS has the following internal limitations: 40 function keys (PFn or Fn), 40 FRS keys, and 25 menu items.
- The FRS imposes no restrictions on which function, control, or arrow keys can be mapped or remapped; however, certain control keys can be captured by the operating system before they reach the FRS. For example, the operating system reserves: Control-C, Control-O, Control-Q, Control-S, Control-T, Control-X, Control-Y.
- FRS commands cannot be mapped to a FRS key. This is syntactically illegal because FRS commands and FRS keys both appear on the left side of the equals sign in the mapping statement.
- TERM\_INGRES cannot be reset dynamically by the application once the Forms System has been initialized.
- Positional menu item mapping cannot be turned off. In the vt100i termcap description, the application key pad is assigned by default to positional menu items. 1 on the key pad is assigned to the first menu item. You can see this if you look at the mapping file, for example:

```
menuitem1 = pf13
```

If you assign a FRS key to an activation block, that key's label appears on the menu line in place of the default key's label. The default key, however, still works.

```
"Go", key frskey5 =  
begin  
...      [first menu item]  
end;
```

In the above example, both **1** and whatever is mapped to frskey5 triggers the Go operation, even though you only see the frskey5's label. To deactivate this, you must assign another FRS function to the PF key now assigned to menuitem1:

```
frskey10 = pf13
```

Normally, you must use an unused FRS key for this function.

- Currently, map files are the only way to turn off a function, control, or arrow key (for example, controlV = off). Keys cannot be turned off by the 4GL or embedded query language set command.

- Keyboard review for the DEC VT series:

<b>TERM_INGRES</b>	<b>Top Row Keys</b>	<b>Alternate Keypad</b>	<b>Menu Key</b>
vt100	disabled	disabled	Escape
vt100k	disabled	disabled	PF1
vt100nk	disabled	numerics	PF1
vt100i	disabled	Functions (PFn)	PF1
vt200i	Functions (Fn)	numerics	PF1

- On a VT220, F1 through F6 are reserved by the terminal. These cannot be mapped in map files.
- Escape is considered reserved for vt100, vt220, and all other terminals that use Escape sequences to define function keys.

## Troubleshooting Checklist (UNIX and VMS)

When troubleshooting your function and control key definitions, consider the following:

1. Have you checked the contents of the map error files in the working directory (ingkey.err or app\_ingkey.err)?

The error messages written to this file indicate map file problems. For example, if the same key is referenced twice, the warning error message is written to this file. If this file is empty, an error occurred before the map file was parsed, and an error message was sent to the terminal screen.

For example, no forms statements can be executed prior to starting the FRS. If a set mapfile statement preceded ## forms or exec frs forms in the embedded query language program, the error message is written to the terminal.

2. Are the terminal's physical setup or emulation characteristics compatible with the current TERM\_INGRES and key map definitions?

For example if you are using a terminal, a problem occurs if a VT220 terminal is setup as a VT220 but TERM\_INGRES is set to vt100i whose map file references PF keys.

3. Is the user's TERM\_INGRES terminal type compatible with the active key definitions?

If not, mapping can seem to be broken when an OpenIngres tool or application starts up; for example, if TERM\_INGRES is set to vt100nk but INGRES\_KEYS points to vt200.map.

4. Is the INGRES\_KEYS environment variable/logical set unintentionally?

This often happens when applications are moved to a new machine where INGRES\_KEYS is defined, unlike the previous environment.

5. Have any of the key map file path names become invalid?

For example, if the file system was moved to a new device, the file pathnames referred to by INGRES\_KEYS or set mapfile are now invalid or perhaps file permissions were changed and the FRS cannot open the map files.

6. Are lower-level key definitions showing through on the user's menu line?

This is the result of the key map merging by the FRS. This most often occurs when you forget to remap or turn off an intended key in the application map file or do not realize that INGRES\_KEYS is also pointing to a map file. ■



# Appendix C: Writing Termcap Descriptions

---

This section contains the following topics:

[Why Modify the Termcap File?](#) (see page 667)

[Supplied Termcap File](#) (see page 668)

[How to Start Writing New Termcap Descriptions](#) (see page 674)

[Eleven Basic Commands](#) (see page 676)

[Commands for Advanced Features \(Windows Environment\)](#) (see page 681)

[Commands for Advanced Features \(UNIX and VMS Environments\)](#) (see page 686)

[Commands for Special Situations](#) (see page 693)

[Examples of Termcap Descriptions](#) (see page 695)

## Why Modify the Termcap File?

For various reasons, you must modify your termcap file. If you are using Ingres in a UNIX or VMS environment, and your terminal is not defined in the standard termcap file, you must provide information about the terminal's characteristics to use the forms system.

Whatever your environment, you can modify the termcap file to optionally take advantage of your terminal's additional capabilities.

You can:

- Edit an existing termcap description to define advanced features of your terminal that can make basic functions easier to use
- Write a new termcap description for a terminal not currently described in the termcap file

## Supplied Termcap File

The following read-only termcap file, which contains information about your terminal's characteristics, is supplied with Ingres:

**Windows:** %II\_SYSTEM%\ingres\files\termcap

**UNIX:** \$II\_SYSTEM/ingres/files/termcap

**VMS:** II\_SYSTEM:[INGRES.FILES]TERMCAP

The termcap file is based closely on the standard UNIX termcap file, except that it contains extra commands that allow the terminal to work with the forms system. A correctly written termcap description allows the forms programs to work properly, although it does not support all of the advanced features that the terminal provides, such as certain video attributes.

You can use terminals that are in the currently supported termcap file for UNIX and VMS environments (listed in an appendix in this guide) without having to do any of the special programming described here.

### **Windows:**

In a PC environment, the termcap file contains nine primary records, as follows:

#### **pccolor**

Color monitor

#### **pcwin**

Microsoft Windows colors

#### **pcfont**

Color monitor; similar to pccolor, but with multiple fonts

#### **pcfonts**

Color monitor; same as pcfont, but with different fonts

#### **pcgaudy**

Color monitor; similar to pccolor, but with gaudier colors

#### **pc43**

Color monitor with 43-line windows; same as pcgaudy, but for 43-line windows

#### **pcmono**

Monochrome monitor



**pcansic**

Not for use with Microsoft Windows

**pcansim**

Not for use with Microsoft Windows

Each record in the PC termcap file contains a number of command strings that define terminal behavior. The only modifications you make to these records are to define some of the advanced features of your PC or to define the size of your monitor. ■

## How to Set the **II\_TERMCAP\_FILE** Variable

Whether editing an existing termcap description or writing a new one, you must edit a copy of the termcap file that is not currently in use. You can set the **II\_TERMCAP\_FILE** environment variable/logical to point to a working copy of the termcap file. This allows you to edit a new version of the termcap file in any directory that you want, without interfering with other users or the original distribution copy of the termcap file.

Use the command appropriate to your system to tell Ingres to use an alternate termcap file:

**Windows:**

```
set II_TERMCAP_FILE=fullpathname\filename ■
```

**UNIX:**

C shell:

```
setenv II_TERMCAP_FILE = fullpathname/ filename
```

Bourne shell:

```
II_TERMCAP_FILE = fullpathname/filename  
export II_TERMCAP_FILE ■
```

**VMS:**

```
define II_TERMCAP_FILE=file_specification ■
```

If this name is defined, any forms program starts up with that file instead of the standard distribution file.

## Format of a Termcap Description

Each termcap entry can be up to 2 KB long, including comment lines.

Consider the following sample termcap description for a fictitious terminal called rti100:

```
R1|rti100|rti fictitious terminal:\
:co#132:li#25:\
:am:bs:\
:is=\E[0m:cm=\E[%2;%2:
```

It has an abbreviated name R1 and a long name rti fictitious terminal. (To use this with the forms system, set the TERM\_INGRES environment variable/logical to rti100.) The rti100 screen is 132 columns wide and 25 lines high. It has automatic margins (am) and uses Control-H for the backspace character (bs). The description contains an initialization string (is) and a cursor positioning string (cm).

As shown in this example, the first line of a termcap description is the list of names. All names must be separated by a vertical bar (|). There must be a colon (:) between the last name and the first capability.

If the termcap description is more than one line (as it is for clarity), then each line except the last must end with a backslash (\) to signify continuation. Capabilities, which can be presented in any order, must be separated by colons (:). The last line must end with a colon (:) to signify that it is the end of the description. You can place tabs at the beginning of lines for readability.

## Special Characters Used in a Termcap Description

The special symbols used in the termcap description are as follows:

**:**

Separates capabilities.

**|**

Separates names.

**\**

Continues to next line (when at the end of line).

**\E**

Is the Escape character.

**100**

Pauses for 100 milliseconds (when before a command).

**#**

Indicates that capability is a number that immediately follows.

**=**

Sets capability to a string that follows

**^X**

Is Control-X for any X; thus, ^g is Control-G, ^h is Control-H, etc.

**\n**

Is the Newline character.

**\r**

Is the Return character.

**\t**

Is the Tab.

**\b**

Is the Backspace.

**\f**

Produces a Formfeed.

### **\072**

Is a colon (072 is the octal value for :). In general, any character can be specified as a three-digit octal value of the ASCII character by preceding it with a backslash.

### **@**

When placed after a command, it means do *not* apply this command. It is used in descriptions that have the tc command. For more information, see Eleven Basic Commands (see page 676).

## **Termcap Description Names**

Names have a special format that you must follow. The first name must be two letters long. The second name is the common name to which you must set TERM\_INGRES. The last name can be a concise description of the terminal's brand and model number. The last name can contain blanks, though the other names can not have blanks.

All names must be separated by a vertical bar (|). Additional names can be placed between the second name and the last name. The additional names can be used as alternative names for TERM\_INGRES.

**Note:** Names must always be checked for uniqueness. You must check through the termcap file before writing a new description to make sure that the names you want to use have not already been selected. A duplicated name is not recognized; only the first one is used.

## How Capabilities of the Terminal Are Specified in a Termcap Description

Capabilities are designated by commands, which must be separated by colons. All commands are two letters long. String and numeric commands are followed by additional information that is read by the forms system. The three types of commands are:

- String
- Numeric
- Boolean

*Strings* contain sequences of characters. The command must be followed by an equal sign. For example, **up=\EA** indicates that the command up (which stands for the sequence to move the cursor up) is set to the sequence Escape-A.

Some string commands can be preceded by a time delay, which is referred to as *padding*. Padding may be required by older, slower terminals (such as the HDS AVT), which require a delay to execute some screen commands, such as clearing the screen. Padding ensures that the terminal has time to execute those commands without losing characters. Modern terminals and terminal emulators will almost never require padding.

The two types of padding are *nonproportional* (or *straight*) padding, and *proportional* padding (to the number of lines affected). To specify straight padding, put the needed time of delay (in milliseconds) before the command. To specify proportional padding, place an asterisk (\*) after the amount of time. For example, on the concept 100 terminal, the ta command (tab character) takes a straight time delay of 8 milliseconds, and the cd command (clear display) takes a proportional delay of 16 milliseconds. The termcap entries look like this:

Straight padding:

```
ta=8\t
```

Proportional padding:

```
cd=16*\EarC
```

*Numeric* commands are followed by a number sign (#). For example, co#80 indicates that there are 80 columns on the screen. (co is the command which specifies the number of columns on the screen.)

*Boolean* commands signify the existence of a capability by their presence. They are not followed by any sequence or other symbols.

## How to Start Writing New Termcap Descriptions

To write new termcap descriptions, we recommend that you start by reading the terminal manufacturer's technical guide. That guide provides information for the eleven basic capabilities described in Eleven Basic Commands (see page 676) in this chapter. You must also have the terminal in front of you so that you can check the operation of the terminal as you are working on the termcap description. When you have included the eleven basic capabilities in the description, try the terminal to see if it works. Once you get it working, you can try adding additional features listed under Commands for Advanced Features.

**Note:** Users not familiar with programming or lacking general knowledge about terminals can find creating new terminal descriptions difficult. Terminal programmer's guides can be difficult to understand. Ask for help from an experienced programmer to create new termcap entries. Our technical support personnel can give only general guidance on creating termcap descriptions, because they cannot have access to the terminal for checking out your problems.

If you have problems, first check to make sure that you entered the sequences from the guide correctly and that the format of the termcap description is correct. If it still does not work, check to see if there are some additional capabilities that need to be added to make it work. Also, certain terminals require special initialization commands. Check the technical guide to see which additional sequences you must add to the initialization string.

One way of preparing termcap descriptions is to examine the termcap entries for similar terminals. If you are trying to write a description for a terminal that is similar to one in the termcap file, you can use the `tc` command to indicate that all attributes for the new terminal are to be taken from the description of a terminal already in the termcap file. Then, you only need to specify the few differences.

Alternatively, you can manually copy the capabilities from a similar terminal and see if it works. Most terminals conform to a system of specifying escape sequences called the ANSI standard. Thus, if you have an ANSI standard terminal, you must be able to get about 90 percent of the capabilities by copying them from another ANSI terminal. The VT100 is an example of an ANSI terminal that has capabilities similar to many different terminals. For this reason, this document contains numerous references to the VT100 sequences in its examples.

Finally, if your terminal has a VT100 emulation mode, you can save time by jumping to VT100 emulation mode and using the VT100 termcap description. In most cases, it is necessary to make a termcap description. In other cases, the terminal works with a termcap description that is identical to the VT100 except that it contains the VT100 emulation sequence in its initialization string. If your terminal has VT100 emulation mode, we recommend that you try it, as a VT100 supports the most advanced features of the forms system.

## Eleven Basic Commands

To work properly, all terminals must have the following eleven commands. Termcap descriptions that have only these eleven basic descriptions usually work, although they lack extra features such as function keys and video attributes. These eleven descriptions form the core of the termcap description.

**Windows:** Although the Ingres forms-based tools are generally run on an 80 x 25 monitor, Ingres runs on a monitor larger than 80 x 25 (to a maximum of 200 x 100). Applications can be created to use the entire larger-size screen. To enable use of a larger screen, change the `co` and `li` entries in the termcap file to the correct values for your screen. ■

The eleven commands are as follows:

**co**

The number of columns on the screen. Without this command there is no way to know how wide to make a form. This command is numeric and must always be followed by a number.

VT100 Example: `co#80`

**li**

The number of rows down the screen. This command is numeric and must always be followed by a number.

VT100 Example: `li#24`

**bs**

This terminal can backspace using Control-H. This is a Boolean command. Include it if your terminal can backspace using Control-H.

**bc**

Sequence to use if the terminal does not use Control-H. You cannot use `bc` and `bs` together.

**cd**

Clears to the end of the display. Sequences to clear everything from the cursor down.

VT100 Example: `cd=\E[J`

**ce**

Clears to the end of the line. Sequences to clear everything from the cursor to the end of the line. (Not used in Windows environment.)

VT100 Example: `ce=\E[K`

**cl**

Clears the entire screen. (Not used in Windows environment.)



VT100 Example: `cl=\E[;H\E[2J`

**cm**

Moves the cursor to an Ingres-specified location. For more information and examples of this command, see the discussion following this table.

**nd**

Nondestructive space. This string specifies the command for moving the cursor right one space without overwriting the contents of the screen at that point.

VT100 Example: `nd=\E[C`

**is**

Terminal initialization string. This includes any sequences needed to set up the terminal prior to running a forms program. (Not used in Windows environment.)

VT100 Example:

`is=\E\E[?3I\E[?4I\E[?7I\E[?8h`

This terminal initialization string example does the following:

- `\E` puts the terminal in keypad numeric mode.
- `\E[?3I` puts terminal into 80-column mode.
- `\E[?4I` puts terminal into jump mode.
- `\E[?7I` turns wraparound off.
- `\E[?8h` turns on autorepeat.

The command is not always needed, but for most terminals it is needed for tailoring the setup to your needs.

**tc**

Entry of a similar terminal. It allows you to use all the capabilities listed for another terminal without rewriting them. This command is actually optional, but is so useful that it has been listed as one of the basic commands. This capability must always be the *last* capability in the description. This rule, which exists so that duplicated commands can be unambiguously defined, is the only exception to the general rule that commands can be presented in any order.

Example:

`(tek4115): dk|tk4115|tek-4115|tektronix4115:\:ld@:tc=vt100f:`

In this example, the `tek4115` is given all the commands from the `vt100f` description, except the `ld` command to initialize the boxing characters. Proper use of this command can make writing termcap descriptions much easier.

## Cursor Motion Command

The cursor motion (cm) command sends the cursor motion string (called the cursor position string in some guides) to the terminal when the cursor is moved from one location to another. As such, the string must accept two parameters: an x-coordinate and an y-coordinate, whose values are obtained by counting the number of rows/spaces from the top-left corner of the screen. Because these values must be sent along with the string at run time, special place markers must be left in the string to tell the forms system where to place the x and y coordinates.

To implement this, find the cursor-addressing scheme described in the guide for your terminal. Then substitute the special place marker characters (described below) in the spot where numbers are expected. Also, be sure to include any special modifiers (described below) in the description if they are needed.

Listed here are place markers and their cursor-addressing options (modifiers):

**%2**

Place marker for a decimal integer of two places.

**%3**

Place marker for a decimal integer of three places.

**%.**

Place marker for a binary value character.

**%+N**

Place marker for a binary value character, with the value of the character *N* added to it.

**%%**

Produces a single %.

**%r**

Reverses the order of the coordinates. Normally the column marker is substituted into the first place marker. If your terminal cursor positioning string expects the row first, include a %r before the first place marker in the cursor motion sequence.

**%B**

BCD ( $16 \cdot (x/10) + (x \bmod 10)$ ) Parameter values are transformed according to this formula.

**%n**

Perform an exclusive OR on the row and column values with the octal value 0140 before generating the string for cursor motion. (This is used only for the DM2500 terminal.)

**%D**

Reverse coding ( $x - 2 \cdot (x \bmod 16)$ ) Parameter values are transformed according to this formula. (This is used only for Delta Data terminals.)

### Example 1: rti100, a Fictitious Terminal

The cursor motion string listed in the user's guide for the rti100 (a fictitious terminal) is ESC|x;y, where x and y are two-digit integers specifying the column and the row, respectively. The rti100 is an example of a terminal that uses a (0,0) origin, so x and y must be one less than the whole number that represents the position on the screen. Thus, to move the cursor to column 8 row 17 on the rti100, you would enter ESC|07;16. The termcap entry is:

```
cm=\E| %2;%2
```

The \E maps to ESC and %2 is the place marker that maps to a decimal integer of two places.

### Example 2: vt100

The cursor motion string for the VT100 is ESC[x;yH, where x and y are two-digit integers specifying the column and row, respectively. The VT100, as opposed to Example 1 above, positions the cursor relative to a origin of (1,1). Thus, to move the cursor to column 8 row 17 on the VT100, enter ESC[08;17H. The termcap entry is:

```
cm=\E[%i%2;%2H
```

The \E maps to ESC, %2 maps to a decimal integer of two places, and the %i signifies that the VT100 uses a (1,1) origin. The default setting for the cm string is for a (0,0) origin. If your terminal uses a (1,1), origin you must explicitly state that by placing a %i somewhere inside the cm string.

### Example 3: Datamedia 3045

The cursor motion string for Datamedia 3045 is ESC Y y x; where y and x are characters whose binary values are offset by 20 hex. (For this terminal, the row must be given before the column.)

To move the cursor to the position (19,11) on this terminal, you must include the sequence ESCY2\*. The ESCY is the first part of the sequence. The 2 is the ASCII character with hexadecimal value 32, which is the same as 12 hex plus the 20 hex offset. Note that 12 hex corresponds to column 19 on the screen. The \* is the ASCII character with a hexadecimal value of 2A, which equals 0A hex plus the 20 hex offset. Again, 0A hex corresponds to row 11 on the screen.

The cm string for this terminal is:

```
cm=\EY%r%+%+
```

The \E maps to ESC, %r is a modifier that tells the forms system that the row and column parameters are reversed, and %+ is the place marker for a character offset by a blank (which has the ASCII value of 20 hex).

**Example 4: Delta Data 5000**

The cursor motion string for the delta data 5000 is Control-Oxy, where x and y are characters whose binary values must be offset by 3A hex, and converted according to the reverse coding formula:

$$(x - 2 * (x \bmod 16))$$

The cm string for this terminal is:

```
cm=^O%D%+9%D%+9
```

The ^O stands for Control-O, %D indicates that the parameters must be transformed according to the reverse coding formula, and %+9 is the place marker for a character offset by 3A hex (ASCII character 9).

Terminal	Entry Listed in Guide	Example Usage on Terminal Mode	Example of the Termcap Description
rti100	ESC x;y	ESC 02;07	cm=\E %2;%2
vt100	ESC[x;yH	ESC[03;08H	cm=5\E[%i%2;%2H
dm3045	ESCYyx	ESCY2*	cm=\EY%r%+%+
delta	Control-Oxy	Control-ORS	cm=^O%D%+9%D%+9

## Commands for Advanced Features (Windows Environment)

The Ingres termcap commands included in this section are used to define some of your PC's advanced features to Ingres. These are features that make your PC easier to use and more attractive, but they are not essential for the basic functions of Ingres.

The commands listed here include:

- Video attributes such as inverse video and blinking
- Color attributes

## Commands Used to Program Video Attributes (Windows)

The four basic modes are: underscore, blinking, reverse video, and high intensity (bold).

All the commands below are combinations of the four basic modes:

Command	Description	Windows Display Code
rv	Turns on reverse video.	rv=71
bl	Turns on blinking mode.	bl=9F
bo	Turns on high intensity (bold) mode.	bo=1E
us	Turns on underscore mode.	us=1F
za	Turns on reverse video, blinking, high intensity, and underscore modes.	za=F0
zb	Turns on high intensity and underscore modes.	zb=1E
zc	Turns on high intensity and blinking modes.	zc=9E
zd	Turns on high intensity and reverse video modes.	zd=70
ze	Turns on underscore and blinking modes.	ze=9F
zf	Turns on underscore and reverse video modes.	zf=71
zg	Turns on blinking and reverse video modes.	zg=F1
zh	Turns on high intensity, blinking, and underscore modes.	zh=9E
zi	Turns on blinking, underscore, and reverse video modes.	zi=F1
zj	Turns on high intensity, blinking, and reverse video modes.	zj=F0
zk	Turns on high intensity, underscore, and reverse video modes.	zk=70

## Commands Used for Color (Windows)

The following commands are used to define color in monitors that support color. They are mapped to the color codes, from 0 to 7, as defined for the VIFRED form.

The `ya` command specifies the default colors for background and foreground objects. The other color-related commands indicate colors for fields as specified in the VIFRED attributes form. The following is an example of the format in which you specify the background and foreground colors:

```
:ya=1F
```

In this example, the 1 specifies a background color of blue and the F specifies a foreground color of white.

Alternatively, you can express these color specifications in the embedded query language with the `set_frs` statement and in 4GL with the `set_forms` statement. For more information, see your query language reference guide or the *Forms-based Application Development Tools User Guide*.

For each color specification, you can also specify a default font. For more information, see [Font Specifications \(Windows\)](#) (see page 685).

All the commands are strings and can have optional padding. The commands are described in the following table:

Command	Description	Windows Display Code
<code>ya</code>	Default color (0)	<code>ya=1F</code>
<code>yb</code>	Alternate color #1	<code>yb=1F</code>
<code>yc</code>	Alternate color #2	<code>yc=1A</code>
<code>yd</code>	Alternate color #3	<code>yd=1E</code>
<code>ye</code>	Alternate color #4	<code>ye=30</code>
<code>yf</code>	Alternate color #5	
<code>yg</code>	Alternate color #6	<code>yg=0F:\</code>
<code>yh</code>	Alternate color #7	<code>yh=2F:\</code>

The following chart shows codes for color displays:

Codes for Background	Codes for Foreground
0 = black	0 = black

<b>Codes for Background</b>	<b>Codes for Foreground</b>
1 = blue	1 = blue
2 = green	2 = green
3 = cyan	3 = cyan
4 = red	4 = red
5 = magenta	5 = magenta
6 = yellow	6 = brown
7 = white	7 = light grey
8 = blinking black	8 = dark grey
9 = blinking blue	9 = light blue
A = blinking green	A = light green
B = blinking cyan	B = light cyan
C = blinking red	C = light red
D = blinking magenta	D = light magenta
E = blinking yellow	E = yellow
F = blinking white	F = white

For all systems, the `ya` entry is used for the normal attribute (that is, no underscore, reverse video, high intensity, or blinking).

For color systems, the `us`, `rv`, `bo`, `bl` and the `za` through `zk` entries are used only with color 0 (entry `ya`). These attributes work for other colors as you would expect if the flags are set for a field. For example, white on red becomes red on white if reverse video is set, and light green on black becomes green on black if change intensity is set.

Some of the termcap color commands that are used by Ingres forms are shown in the following table:

<b>Command</b>	<b>Forms and Menus</b>
<code>ya</code>	Background for Ingres forms and the ring menu line when the menu is active
<code>yb</code>	Tablefields in Ingres forms
<code>yc</code>	Some simple fields in certain Ingres forms
<code>yd</code>	Some simple fields in certain Ingres forms
<code>us</code>	Some simple fields in certain Ingres forms



Command	Forms and Menus
bo	Some simple fields in certain Ingres forms

## Font Specifications (Windows)

For each of the ya through yh termcap strings, Ingres allows you to set both a color and a font. Wherever the specified color is used, the specified font is used as well. You can specify a different font for each entry if you want. The following fixed fonts are available:

- OEM (Windows OEM\_FIXED\_FONT)
- ANSI (Windows ANSI\_FIXED\_FONT)
- SYSTEM (Windows SYSTEM\_FIXED\_FONT)

You must use one of these three fixed fonts for both the ya and the yb entry. For the other entries, you can use any fixed or proportional font loaded in your Windows system. However, for data entry fields, it is best to use one of the three fixed fonts provided, because the cursor placement is based on these fixed fonts.

**Note:** Underlining in stock fixed fonts (OEM, ANSI, and SYSTEM) is not possible on a PC monitor, due to a Windows font limitation. On form fields for which the Underline attribute has been set, leading and trailing blanks display underscore characters to indicate underlining for these fonts. Text appears with full underlining, however, in fields with fonts that support it.

Specify fonts by adding a comma after the background/ foreground color information in the ya through yh entries, followed by the font name. OEM is the default if you do not specify a font. You can specify the point size for the font by placing a comma after the font name, followed by the point size. Following is a sample from the termcap file:

```
:ya=1F,ANSI:\
:yb=1F,OEM:\
:yc=1F,MS Serif,14:\
:yd=1F,Times New Roman:\
:ye=1F,Courier:\
:yf=1F,Courier new:\
:yg=1F,MS Sans Serif:\
:yh=1F,SYSTEM,15:\
```

The default point size for a proportional font is the largest size that fits on a line, where the line size is determined by the font for the ya entry. The point size specified for ya determines the spacing for all lines on a form. You must increase the line size if you are using a taller font for an entry other than ya.

To do so, use a large enough point size for ya, as in the following example:

```
:ya=1F,SYSTEM,16:\
:yb=1F,OEM:\
:yc=1F,MS Serif,16:\
:yd=1F,Times New Roman:\
:ye=2F,Courier:\
:yf=4F,Courier new:\
:yg=0B,MS Sans Serif:\
:yh=1F,Courier,18:\
```

The default pixel and point sizes for the fixed fonts on a SVGA 800 x 600, 20-inch monitor are shown in the following table:

Font	Pixels	Points
OEM	12	9
ANSI	13	10
SYSTEM	15	11



## Commands for Advanced Features (UNIX and VMS Environments)

The termcap commands included in this section are used to define some of the advanced features of your terminal. These are features that make the terminal look good and make it easier to use but are not essential for the basic functions.

The advanced features commands include:

- Video attributes such as inverse video and blinking
- Color attributes
- Boxing characters
- Commands to set up function and arrow keys
- Command to specify a default FRS key mapping file
- Commands to optimize cursor movement

## Commands Used to Program Video Attributes (UNIX and VMS)

The four basic modes are: underscore, blinking, reverse video, and high intensity. All the commands in the following table are combinations of the four basic modes:

Command	Description	VT100 Example
rv	Turns on reverse video.	rv=1\E[7m
bl	Turns on blinking mode.	bl=1\E[5m
bo	Turns on high intensity mode.	bo=1\E[1m
us	Turns on underscore mode.	us=2\E[4m
ea	Turns off all special display characteristics.	ea=1\E[m
za	Turns on reverse video, blinking high intensity, and underscore modes.	za=1\E[1;4;5;7m
zb	Turns on high intensity and underscore modes.	zb=1\E[1;4m
zc	Turns on high intensity and blinking modes.	zc=1\E[1;5m
zd	Turns on high intensity and reverse video modes.	zd=1\E[1;7m
ze	Turns on underscore and blinking modes.	ze=1\E[4;5m
zf	Turns on underscore and reverse video modes.	zf=1\E[4;7m
zg	Turns on blinking and reverse video modes.	zg=1\E[5;7m
zh	Turns on high intensity, blinking, and underscore modes.	zh=1\E[1;4;5m
zi	Turns on blinking, underscore, and reverse video modes.	zi=1\E[4;5;7m
zj	Turns on high intensity, blinking, and reverse video modes.	zj=1\E[1;5;7m
zk	Turns on high intensity, underscore, and reverse video modes.	zk=1\E[1;4;7m

## Commands Needed for Boxing Characters (UNIX and VMS)

Not all terminals have special boxing characters. If your terminal does not have them, hyphens (-) and vertical bars (|) are used instead. Using boxing characters, however, greatly improves the appearance of forms using table fields.

Command	Description	VT100 Example
ld	Initializes terminal to draw solid lines.	ld=\E)0
ls	Interprets subsequent characters for drawing solid lines.	ls=\016
le	Interprets subsequent characters as regular characters.	le=\017
qa through qk	Boxing characters; see the next table.	

The table below describes the boxing characters qa through qk mentioned in the previous table:

Command	Description	VT100 Example
qa	Lower right corner of a box	qa=j
qb	Upper right corner of a box	qb=k
qc	Upper left corner of a box	qc=l
qd	Lower left corner of a box	qd=m
qe	Crossing lines	qe=n
qf	Horizontal line	qf=q
qg	Left T (stem points right)	qg=t
qh	Right T (stem points left)	qh=u
qi	Bottom T (like upside down T)	qi=v
qj	Top T (like a regular T)	qj=w
qk	Vertical line	qk=x

## Commands Needed for Function Keys (UNIX and VMS)

The following table describes the commands the termcap files use to activate function keys:

Command	Description
ke	Takes the terminal out of keypad transmit mode.
ks	Puts the terminal in keypad transmit mode.
kn	The number of function keys available; for example, this is specified as kn#18 on the VT100. You can have a maximum value of 40 for the number of function keys.
ky	This terminal has cursor and function keys. This is a Boolean command. It must be present if you want to use function keys. Using it disables the ESC key and sets the first function key, PF1, to the Menu function.
k0 through KD	Strings sent by function keys; see the next table for the specifics on each command.

The following table describes the commands that you can set to send strings by using function keys. These commands allow the Forms Runtime to recognize when a function key is pressed. You must still to map the function key to a forms action in your FRS key mapping file. For example, the key mapping file would use "pf3" to reference function key 3, the k2 command.

Command	Function Key Number	VT100 Example
k0	function key 1	k0=\EOP
k1	function key 2	k1=\EOQ
k2	function key 3	k2=\EOR
k3	function key 4	k3=\EOS
k4	function key 5	k4=\EOW
k5	function key 6	k5=\EOx
k6	function key 7	k6=\EOy
k7	function key 8	k7=\EOM
k8	function key 9	k8=\EOt
k9	function key 10	k9=\EOu
kA	function key 11	kA=\EOv

Command	Function Key Number	VT100 Example
kB	function key 12	kB=\EOI
kC	function key 13	kC=\EOq
kD	function key 14	kD=\EOr
kE	function key 15	kE=\EOs
kF	function key 16	kF=\EOp
kG	function key 17	kG=\EOn
kH	function key 18	kH=\EOM
kI	function key 19	none
kJ	function key 20	none
kK	function key 21	none
kL	function key 22	none
kM	function key 23	none
kN	function key 24	none
kO	function key 25	none
kP	function key 26	none
kQ	function key 27	none
kR	function key 28	none
kS	function key 29	none
kT	function key 30	none
kU	function key 31	none
kV	function key 32	none
kW	function key 33	none
kX	function key 34	none
kY	function key 35	none
kZ	function key 36	none
KA	function key 37	none
KB	function key 38	none
KC	function key 39	none
KD	function key 40	none

## Commands Needed for Arrow Keys (UNIX and VMS)

The arrow key commands are all strings:

Command	Description	VT100 Example
ku	Sent by terminal up arrow key	ku=\EOA
kd	Sent by terminal down arrow key	kd=\EOB
kr	Sent by terminal right arrow key	kr=\EOC
kl	Sent by terminal left arrow key	kl=\EOD

## Commands Used for Color (UNIX and VMS)

You can use these commands to turn on color in terminals or monitors that support color. These are mapped to the color codes, from 0 to 7, as defined for the VIFRED form. Some terminals, such as the VT2410, are capable of only four colors.

The ya command specifies the default color for foreground objects, and the other commands indicate colors for fields as specified in the VIFRED attributes form. Alternatively, you can express these color specifications in the embedded query language with the set\_frs statement and in 4GL with the set\_forms statement. For more information, see your query language reference guide or the *Forms-based Application Development Tools User Guide*.

All the commands are strings and can have optional padding. The commands are described in the following table:

Command	Description	Envision Example
ya	Default foreground color (0)	2\Ea7
yb	Alternate foreground color #1	2\Ea1
yc	Alternate foreground color #2	2\Ea2
yd	Alternate foreground color #3	2\Ea3
ye	Alternate foreground color #4	2\Ea4
yf	Alternate foreground color #5	2\Ea5
yg	Alternate foreground color #6	2\Ea6
yh	Alternate foreground color #7	2\Ea7

#### Example: Envision

```
E3|envisionc|envision230|this has the color definitions:\

:ya=2\Ea7:yb=2\Ea1:yc=2\Ea2:yd=2\Ea3:\

:ye=2\Ea4:\:yf=2\Ea5:yg=2\Ea6:\

:yh=2\Ea7:tc=vt100k:
```

## Commands to Specify Display Width (UNIX and VMS)

The following commands define terminal display width:

Command	Description	VT100 Example
yn	Number specifying narrow width of terminal	yn#80
yo	Number specifying wide width of terminal	yo#132
yp	Narrow cm string	yp=\E[%i%2;%2H
yq	Wide cm string	yq=\E[%i%2;%3H
yr	String to switch to narrow width	yr=\E[?3l
ys	String to switch to wide width	ys=\E[?3h

## Command to Specify FRS Mapping File for Terminal (UNIX and VMS)

You can use the mf command to specify a default FRS key mapping file for a terminal. You must include the name of a file in the OpenIngres files directory (*not* the full directory specification or path name of the file). This file contains the default FRS key mapping for the terminal.

#### Example: vt100

```
:mf=vt100i.map:
```



## Commands to Optimize Cursor Movement (UNIX and VMS)

These commands generally improve the way the forms system moves the cursor around the form. They are usually optional.

Command	Description
am	This terminal has automatic margins. This Boolean command is important on forms that run to the edge of the screen.
do	Down one line. Inclusion of this command helps the forms system move the cursor faster.
sr	Scroll reverse. This command makes the form scroll backwards instead of jumping if you are moving up on a long form.
cs	Change scrolling region. This command improves the appearance of the cursor movements when scrolling on a long form. The forms system still works if this is not defined, but it does not look as nice. This command is very similar in form to the cm command; however, the cs command's parameters are the upper and lower limits of scrolling instead of the position on the screen. Otherwise, all the place markers and modifiers are the same. If you use the cs command, you must also include the sr command.

Example: vt100

```
cs=5\E[%2;%2r
```

## Commands for Special Situations

The termcap commands in this section are rarely required for the forms system.

## Commands from the UNIX Termcap File

Because the termcap file is based upon the UNIX release, some additional UNIX entries have been included in the termcap file, but are usually not needed by the forms system. The following table provides a list of these additional UNIX termcap entries:

Name	Type	Description
bt	str	Back tab. Padding can be required on this command.
ho	str	Sequence to move the cursor to the home position. This command is used if and only if the terminal does not possess a cursor positioning string (cm).
ll	str	Last line, first column (if no cm).
ms	bool	Safe to move while in standout and underline mode.
pc	str	Pad character (rather than null).
sf	str	Scroll forward. Padding can be required on this command.
ta	str	Tab, other than Control-I or with padding. Padding can be required on this command.
te	str	String to end programs that use cm.
ti	str	String to begin programs that use cm.
ve	str	Sequence to end open/visual mode.
vs	str	Sequence to start open/visual mode.
hz	str	Hazeltine; cannot print apostrophes.
nc	bool	No correctly working carriage return (DM2500, H2000).
xb	bool	Beehive (f1=ESC, f2=Control-C)
xn	bool	A newline is ignored after a wrap (Concept).
xr	bool	Return acts like ce \r \n (Delta Data).
xs	bool	Standout not erased by writing over it (HP 264?).
xt	bool	Tabs are destructive, magic so character (Telaray 1061).

## Examples of Termcap Descriptions

This section includes several examples of termcap entries that have been tested.

If you want to learn more about the termcap process, compare these descriptions with the guides for the particular terminals.

### DEC VT100 (All Inclusive) Termcap Description

This description contains all the features previously described. This example is longer than most termcap descriptions.

```
d7|vt100k|vt-100k|pt100k|vt100 with everything:\
:co#80:li#24:cl=20\E[;H\E[2J:bs:cm=5\E[%1%2;%2H:\
:nd=2\E[C:\
:up=2\E[A:ce=3\E[K;cd=50\E[J:us=2\E[4m:ue=2\E[m:\
:is=\E\E[?31\E[?41\E[?71\E[?8h:ks=\E[?1h\E=:\
:ke=\E[?11\E:ku=\E0A:kd=\E0B:kr=\E0C:kl=\E0D:\
:ld=\E)0:\
:qa=j:qb=k:qc=l:qd=m:qe=n:qf=q:qg=t:qh=u:qi=v:\
:qj=w:qk=x:\
:ls=\016:le=\017:\
:cs=5\E[%2;%2r:bl=1\E[5m:be=1\E[m:\
:bo=1\E[1m:eb=1\E[m:rv=1\E[7m:re=1\E[m:ea=1\E[m:\

:za=1\E[1;4;5;7m:zb=1\E[1;4m:zc=1\E[1;5m:\
:zd=1\E[1;7m:\
:ze=1\E[4;5m:zf=1\E[4;7m:zg=1\E[5;7m:\
:zh=1\E[1;4;5m:\
:zi=1\E[4;5;7m:zj=1\E[1;5;7m:zk=1\E[1;4;7m:\
:kh=\E[H:ky:k0=\EOP:k1=\EQQ:k2=\EOR:k3=\EOS:pt:\
:sr=5\EM:\
:k4=\EOW:k5=\EOx:k6=\EOy:k7=\EOM:\
:k8=\EOt:k9=\EOu:kA=\EOv:\
:kB=\EOL:kC=\EQq:kD=\EOr:kE=\E0s:kF=\E0p:\
:kG=\EOn:kH=\EOM:\
:kn#18:mf=vt100k.map:
```

### DEC VT100 (Simple) Termcap Description

Here is the VT100 using only basic features. This description lacks many of the niceties found in the previous longer description, but it illustrates that minimal descriptions that provide basic functioning are relatively easy to write.

```
d8|vt100s|simple vt100 entry:\
:co#80:li#24:cl=20\E[;H\E[2J:bs:\
:cm=5\E[%1%2;%2H:nd=2\E[C:\
:is=\E\E[?31\E[?41\E[?71\E[?8h:\
:up=2\E[A:ce=3\E[K;cd=50\E[J:
```

## Envision 230 Termcap Description

This termcap description illustrates the use of the tc command. This description contains all the features of the VT100 except that it does not employ the VT100 initialization string. Also note the large number of names; this example covers three different varieties of Envision terminal. If you need to write descriptions for terminals similar to known terminals, this example is particularly pertinent.

```
E1|envision|envision230|envision220:\
:is@:tc=vt100k:
```

## Concept 100 Termcap Description

This description was taken from the BSD UNIX termcap description. It is not programmed for Ingres function keys or boxing characters. The Concept 100 is an example of a terminal for which writing descriptions is particularly difficult. It has several nonstandard features. Note the use of the xn command, a command written specifically for terminals like the Concept 100.

```
c1|c100|concept100:\
:is=\EU\Ef\E7\E5\E8\E1\ENH\EK\E\200\Eo&\200:\
:al=3*\EarR:am:bs:cd=16*\arC:ce=16\EarS:\
:cl=2*arL:\
:cm=\Ea%+:%+:co#80:dc=16\EarA:dl=3*\EarB:\
:ei=\E\200:\eo:im=\EarP:in:ip=16*:li#24:mi:\
:nd=\E=:se=\Ed\Ee:so=\ED\EE:ta=8\t:ul:\
:up=\E;:\
:vb=\Ek\EK:xn:
```

## Datamedia 3045 Termcap Description

This description, which was also taken from the BSD UNIX termcap description, contains the rather tricky cm string discussed in The Eleven Basic Commands section. This description has not been programmed to use Ingres function keys or boxing characters.

```
D4|3045|dm3045|datamedia 3045a:\
:is=\EU\EV:\
:am:bs:cd=2\EJ:ce=\EK:cl=2\EM:\
:cm=\EY%r%+%+:co#80:\
:dc=6\EB:dm=:ed=:ei=\EP:ho=\EH:ic=:\
:im=\EP:ip=6:\
:k0=\Ey\r:k1=\Ep\r:k2=\Eq\r:k3=\Er\r:\
:k4=\Es\r:\
:k5=\Et\r:k6=\Eu\r:k7=\Ev\r:k8=\Ew\r:\
:k9=\Ex\r:\
:kh=\EH:ku=\EA:kr=\EC:li#24:nd=\EC:\
:pc=\177:pt:\
:eo:ul:up=\EA:xn:
```

# Appendix D: Data Types

This section contains the following topics:

[Data Types in SQL, OpenSQL, and QUEL](#) (see page 697)

## Data Types in SQL, OpenSQL, and QUEL

The following table gives equivalents for SQL, OpenSQL, and QUEL data types:

SQL	OpenSQL	QUEL	Description
<code>c(n)</code>	not used	<code>c(n)</code>	A fixed-length string of up to <i>n</i> printable ASCII characters, with nonprintable characters converted to blank; <i>n</i> represents the lesser of the maximum configured row size and 32,000.
<code>char(n)</code>	<code>char(n)</code>	<code>char(n)</code>	A fixed-length string of up to <i>n</i> ASCII characters, including any nonprintable characters; <i>n</i> represents the lesser of the maximum configured row size and 32,000.
<code>varchar(n)</code>	<code>varchar(n)</code>	<code>varchar(n)</code>	A variable-length string of up to <i>n</i> ASCII characters; <i>n</i> represents the lesser of the maximum configured row size and 32,000.
<code>text(n)</code>	not used	<code>text(n)</code>	A string of up to <i>n</i> ASCII characters (excluding nonprintable characters); <i>n</i> represents the lesser of the maximum configured row size and 32,000.
<code>float4</code>	not used	<code>f4</code>	4-byte floating point; for numbers including decimal fractions, from $0.29 \times 10^{-38}$ to $1.7 \times 10^{38}$ (7 digit precision).
<code>float8</code>	<code>float</code>	<code>f8</code>	8-byte floating point; for numbers including decimal fractions, from $0.29 \times 10^{-38}$ to $1.7 \times 10^{38}$ (16 digit precision).
<code>decimal</code>	<code>decimal</code>	not used	Exact numeric data type defined by

SQL	OpenSQL	QUEL	Description
			its precision (total number of digits) and scale (number of digits to the right of the decimal point). Precision must be between 1 and 31. Scale can be 0 up to the maximum scale.
integer1	not used	i1	1-byte integer; for whole numbers ranging from -128 to +127.
integer2 or smallint	smallint	i2	2-byte integer; for whole numbers ranging from -32,768 to +32,767.
integer4 or integer	integer	i4	4-byte integer; for whole numbers ranging from -2,147,483,648 to +2,147,483,647.
money	not used	money	8 byte monetary data from -999999999999.99 to +999999999999.99.
date	not used	date	12 bytes; dates ranging from 1-jan-1582 to 31-dec-2382 for absolute dates and -800 years to 800 years for time intervals.

An SQL user-defined data type (UDT) is perceived and treated as a character string. User-defined data types are not supported in OpenSQL or QUEL.

Some forms utilities do not support the long varchar, byte, byte varying, and long byte data types. For more information, see the documentation for each specific product for details about long varchar, byte, byte varying, and long byte data types.

# Appendix E: Calling Ingres Tools from Embedded SQL and OpenSQL

---

This section contains the following topics:

[How Ingres Tools Are Called from Embedded SQL and OpenSQL](#) (see page 699)

[Call Statement—Call an Ingres Tool or Operating System](#) (see page 700)

[Ingres Tool Parameters](#) (see page 701)

## How Ingres Tools Are Called from Embedded SQL and OpenSQL

The call statement in embedded SQL and OpenSQL lets your applications call an Ingres tool (such as QBF) or the operating system.

When an Ingres tool is called from an embedded SQL or OpenSQL program, you can specify parameters, many of which have the same effect as flags used when the tool is called from the operating system command line.

Some of the tool parameters correspond to arguments passed to the particular tool, others to specific command line flags.

For each call, the flags parameter can be used to pass the values of all flags, including those that do not have defined parameter names. Within the string containing the value for flags, each distinct flag must be separated by a blank space.

## Call Statement—Call an Ingres Tool or Operating System

The embedded SQL and OpenSQL call statement enables your applications to call an Ingres tool (such as QBF) or the operating system.

The syntax to call an Ingres tool is as follows:

```
exec sql call subsystem (database = dbname {, parameter = value});
```

The syntax to call the operating system is as follows:

```
exec sql call system (command = command_string);
```

The value for a particular parameter can be specified with or without quotes or within a string variable. If there is no value for a particular parameter name, specify an empty, quoted string.

### Examples

Following are sample calls to Ingres tools from embedded SQL and OpenSQL:

```
exec sql call qbf (database = 'empdb',  
                 table = 'employee');
```

```
exec sql call rbf (database = 'empdb',  
                 flags = '-s -mblock emptable');
```

```
exec sql call report (database = :dbvar,  
                    name = :namevar, mode = :modevar);
```

```
exec sql call system (command = 'mail');
```

In the third example, *dbvar*, *namevar*, and *modevar* are host language string variables.



## Ingres Tool Parameters

Each Ingres tool accepts parameters that are specified when the tool is called from an embedded SQL or OpenSQL program. Many of these parameters have the same effect as flags used when the tool is called from the operating system command line.

The maximum number of parameters for the call statement is 29, and the combined length of the parameters must not exceed 2043 bytes.

A parameter that does not take an argument must be set to an empty string, specified by an empty pair of quotes. For example, the silent parameter of the report call is equivalent to the -s flag on the report command line. Because -s does not take an argument, specify the silent parameter in the following way:

```
exec sql call report
      (database = 'mydb', name = 'employee', silent = ' ' );
```

For more information on each command, see the *Command Reference Guide*.

## Abf Command Parameters

The abf command accepts the following parameters:

**application**

Name of the application.

**flags**

Can be used for any flags on the command line. Flags must be separated by a blank.

## Ingmenu Command Parameters

The ingmenu command to call Ingres Menu accepts the flags parameter, which can be used for any flags on the command line. Flags must be separated by a blank.

## Isql Command Parameters

The isql command to call the interactive Terminal Monitor for SQL accepts the flags parameter, which can be used for any flags on the command line. Flags must be separated by a blank.

## Qbf Command Parameters

The qbf command accepts the following parameters:

### **qbfname**

Equivalent to the -f flag. Invoke QBF using the specified qbfname. If name is blank, start at Catalogs frame for qbfnames.

### **joindef**

Equivalent to the -j flag. Invoke QBF using the specified JoinDef. If the name is blank, start at Catalogs frame for JoinDefs.

### **tblfld**

Equivalent to the -t flag. Invoke QBF on the specified table, using a table field format to display the data. If the name is blank, start at Catalogs frame for tables.

### **lookup**

Equivalent to the -l flag. Invoke QBF using the specified name. QBF can look up the name in the following order: QB FName, JoinDef name, table name.

### **silent**

Equivalent to the -s flag. Suppresses verbose messages.

### **mode**

Equivalent to the -m flag. Enter QBF directly in the specified mode. Possible values for this parameter are retrieve, append, update, or all.

### **table**

Name of the table on which QBF is being invoked. This parameter must be omitted if one of the joindef, qbfname, tblfld, or lookup parameters has been used.

### **emptycat**

Equivalent to -e flag. If set, catalogs are displayed empty, and the user can enter names directly.

### **flags**

Can be used for any flags on the command line. Flags must be separated by a blank.

## Rbf Command Parameters

The rbf command accepts the following parameters:

**silent**

Equivalent to the -s flag. Suppresses status messages.

**report**

Equivalent to the -r flag. Indicates that a report, rather than a table, is being specified. The name of the report is the value for this parameter.

**style**

Equivalent to the -m flag. Indicates that a table, rather than a report, is being specified. Optional values for this parameter are column, wrap and block. The name of the table is given as the value for the table parameter.

**table**

The name of a table or view for which a default report is to be formatted.

**emptycat**

Equivalent to -e flag. If set, the Catalog form is displayed empty, and the user can enter names directly.

**flags**

Can be used for any flags on the command line. Flags must be separated by a blank.

## Report Command Parameters

The report command of Report-Writer accepts the following parameters:

### **file**

Equivalent to the -f flag. Directs the formatted report to the specified file for output.

### **silent**

Equivalent to the -s flag. Suppresses status messages.

### **report**

Equivalent to the -r flag. Indicates that a report, rather than a table, is being specified. The name of the report is the value for this parameter.

### **style**

Equivalent to the -m flag. Indicates that a table, rather than a report, is being specified. Optional values for this parameter are column, wrap, and block. The name of the table is given as the value for the name parameter.

### **name**

Name of a table or view in the database for which a default report is to be formatted.

### **param**

The list of parameters for the report. Each element in the list must be of this form:

*name =value*

Blanks or tabs must separate Name/value combinations. The entire list must be enclosed within quotes. In addition, if *name* is a character report parameter, then value must be enclosed in quotes. (Values of numeric report parameters cannot, however, be quoted.) The inner quotes that surround *value* must then be de-referenced according to host language rules so they can be passed through to the report command.

For example, assume that you want to call the Report-Writer from within embedded SQL or OpenSQL with the equivalent of this system-level command:

```
report newdb -r myrpt (bin='f01'  
wstation='u1' type=12 sect=11)
```

Note that bin and wstation are character parameters and that type and sect are numeric parameters.

You could call the report by placing the parameters in a program variable. For example:

```
exec sql call report (database = 'newdb', report = 'myrpt', param =  
:parmvar);
```

The variable, `parmvar`, must contain the value:

```
bin="f01" wstation="u1" type=12 sect=11
```

Double quotes must surround the constant string values within the variable. If your host language requires the de-referencing of double quotes, be sure to do so, according to the rules of your host language.

**forcerep**

Equivalent to the `-h` flag. Report-Writer can put headers and footers, even if no data is found for the report.

**formfeed**

Equivalent to the `+b` flag. Report-Writer forces formfeeds at page breaks, overriding any settings in the report formatting commands.

**noformfeed**

Equivalent to the `-b` flag. Report-Writer suppresses formfeeds, overriding any settings in the report formatting commands.

**pagelength**

Equivalent to the `-v` flag. Sets the page length, in lines, for the report, overriding any `.PL` commands in the report.

**brkfmt**

Equivalent to the `+t` flag (default). If set, breaks and calculations for dates and numbers are based on the displayed data, rather than the internal database values.

**nobrkfmt**

Equivalent to the `-t` flag. If set, breaks and calculations for dates and numbers are based on the internal database values, rather than the displayed values.

**flags**

Can be used for any flags on the command line. Flags must be separated by a blank.

## Sql Command Parameters

The `sql` command accepts the `flags` parameter, which can be used for any flags on the command line. Flags must be separated by a blank.

## Sreport Command Parameters

The sreport command of the Report-Writer accepts the following parameters:

**file**

Name of a text file containing report formatting commands for one or more reports.

**silent**

Equivalent to the -s flag. Suppresses status messages.

**flags**

Can be used for any flags on the command line. Flags must be separated by a blank.

## System Command Parameters

The system command has only one argument, *command*. It executes the operating system level command specified by *command\_string*. If *command\_string* is null, empty, or blank, then it transfers the user to the operating system.

## Vifred Command Parameters

The vifred command accepts the following parameters:

**form**

Equivalent to the -f flag. Invoke VIFRED on the specified form.

**table**

Equivalent to the -t flag. Invoke VIFRED with a default form for the specified table.

**joindef**

Equivalent to the -j flag. Invoke VIFRED with a default form for the specified JoinDef.

**emptycat**

Equivalent to -e flag. If set, an empty Catalogs form is displayed, and the user can enter names directly.

**flags**

Can be used for any flags on the command line. Flags must be separated by a blank.

# Appendix F: Report-Writer Report Examples

---

This section contains the following topics:

[Overview of Report Examples](#) (see page 707)

[POPULATION Example](#) (see page 708)

[POP2 Example](#) (see page 718)

[ACCOUNT Example](#) (see page 721)

[DICTIONARY Example](#) (see page 729)

[DICT2 Example](#) (see page 735)

[LABEL Example](#) (see page 737)

[Report Creation Using Several Tables](#) (see page 740)

## Overview of Report Examples

This section contains five complete sample reports:

- The POPULATION report demonstrates a common type of report with subtotalling. POP2 shows an alternative set of formatting statements for producing the same output.
- The ACCOUNT report demonstrates a complex report that might be used in accounting applications.
- The DICTIONARY report demonstrates the use of character printing options within Report-Writer. DICT2 shows an alternative set of formatting statements for the same output.
- The LABEL report demonstrates the formatting of mailing labels that print three across on a page, generated from a list of names and addresses.
- The BOOKS report demonstrates the use of joining tables for producing a report.

Each report example contains the following information:

- Table definition for the report
- Data for the report
- Listing of the report formatting statements used in the report code
- Sample listing of the report itself

For the sake of clarity, the formatting statements appear in the examples as uppercase letters, although they can actually be specified in either uppercase or lowercase letters.

## POPULATION Example

The POPULATION example demonstrates the use of Report-Writer in formatting a report of census data, by region and state, for the United States. The base tables for this report are as follows:

- "Region" contains region names associated with region abbreviations
- "State" contains state names, as well as state abbreviations, and associated region abbreviations
- "Pop" contains population data for each state for different census years

Additional details for each of these table layouts are provided in the tables that follow:

### Region Table Definition

Column Name	Type	Length	Nulls	Defaults
regabbrev	char	3	yes	no
region	char	20	yes	no

### Region Data for the Sample Report

regabb	region
ENC	East North Central
ESC	East South Central
M	Mountain
MA	Middle Atlantic
NE	New England
P	Pacific
SA	South Atlantic
WNC	West North Central
WSC	West South Central

### State Table Definition

Column Name	Type	Length	Nulls	Defaults
regabbrev	char	3	yes	no
stateabbrev	char	2	yes	no
state	char	20	yes	no



**State Data for the Sample Report**

<b>regabb</b>	<b>Statab</b>	<b>state</b>
ENC	IL	Illinois
ENC	IN	Indiana
ENC	MI	Michigan
ENC	OH	Ohio
ENC	WI	Wisconsin
ESC	AL	Alabama
ESC	KY	Kentucky
ESC	MS	Mississippi
ESC	TN	Tennessee
MA	NJ	New Jersey
MA	NY	New York
MA	PN	Pennsylvania
M	AZ	Arizona
M	CO	Colorado
M	ID	Idaho
M	MT	Montana
M	NV	Nevada
M	NM	New Mexico
M	UT	Utah
M	WY	Wyoming
NE	CN	Connecticut
NE	ME	Maine
NE	MA	Massachusetts
NE	NH	New Hampshire
NE	RI	Rhode Island
NE	VT	Vermont
P	AK	Alaska
P	CA	California
P	HI	Hawaii
P	OR	Oregon

<b>regabb</b>	<b>Statab</b>	<b>state</b>
P	WA	Washington
SA	DE	Delaware
SA	DC	District of Columbia
SA	FL	Florida
SA	GA	Georgia
SA	MD	Maryland
SA	NC	North Carolina
SA	SC	South Carolina
SA	VA	Virginia
SA	WV	West Virginia
WNC	IA	Iowa
WNC	KS	Kansas
WNC	MN	Minnesota
WNC	MO	Missouri
WNC	NB	Nebraska
WNC	ND	North Dakota
WNC	SD	South Dakota
WSC	AR	Arkansas
WSC	LA	Louisiana
WSC	OK	Oklahoma
WSC	TX	Texas

**Population Table Definition**

<b>Column Name</b>	<b>Type</b>	<b>Length</b>	<b>Nulls</b>	<b>Defaults</b>
year	integer	4	yes	no
stateabbrev	char	2	yes	no
tot_18to65	integer	4	yes	no
tot_under18	integer	4	yes	no
tot_over65	integer	4	yes	no

**Population Data for the Sample Report**

<b>year</b>	<b>statab</b>	<b>tot_18to65</b>	<b>tot_under18</b>	<b>tot_over65</b>
1970	IL	9600381	1425674	87921
1970	IN	4820324	357464	15881
1970	MI	7833474	991066	50543
1970	OH	9646997	970477	34543
1970	WI	4258959	128224	30548
1970	AL	2528983	908247	6935
1970	KY	2971232	241292	6182
1970	MS	1393283	815770	7859
1970	TN	3283432	631696	8559
1970	NJ	6349908	770292	47964
1970	NY	15790307	2166933	233500
1970	PN	10737732	1016514	39663
1970	AZ	1604948	53344	112608
1970	CO	2212352	66411	28496
1970	ID	698802	2130	11635
1970	MT	663043	1995	29371
1970	NV	448177	27762	12799
1970	NM	915815	19555	80630
1970	UT	1031926	6617	20730
1970	WY	323024	2568	6824
1970	CN	2835458	181177	15074
1970	ME	985276	2800	3972
1970	MA	5477624	175817	35729
1970	NH	733106	2505	2070
1970	RI	914757	25338	6630
1970	VT	442553	761	1016
1970	AK	236767	8911	54704
1970	CA	17761032	1400143	791959
1970	HI	298160	7573	462828
1970	OR	2032079	26308	32998

<b>year</b>	<b>statab</b>	<b>tot_18to65</b>	<b>tot_under18</b>	<b>tot_over65</b>
1970	WA	3251055	71308	86806
1970	DE	466459	78276	3369
1970	DC	209272	537712	9526
1970	FL	5711411	1049578	28454
1970	GA	3387516	1190779	11280
1970	MD	3193021	701341	28037
1970	NC	3891510	1137664	52885
1970	SC	1794430	789041	7045
1970	VA	3757478	865388	25628
1970	WV	1666870	73931	3436
1970	IA	2782762	32596	9018
1970	KS	2122068	106977	17533
1970	MN	3736038	34868	34065
1970	MO	4177495	480172	18834
1970	NB	1432867	39911	10715
1970	ND	599485	2494	15782
1970	SD	630333	1627	33547
1970	AR	1561108	357225	4962
1970	LA	2539547	1088734	13025
1970	OK	2275104	177910	106218
1970	TX	9696569	1419677	80484

Report formatting statements for the POPULATION report are explained here:

- The .query statement shows the database query needed to set up the data in the form required to write the report. Essentially, the query sets up a table with one row for each state, including the columns region (name of region), state (name of state), tot (the total population of the state), tot\_under18, tot\_18to65, and tot\_over65 (populations of three age groups).

- The query contains a variable, *\$year*, which is used in the where clause to select data for only one census year. In the example shown, you can select the data for 1970 by running the report with the command:

```
report rwsqldb pop (year=1970)
```

You must enclose the entire *variable=value* clause and its delimiting parentheses within double quotes to pass it through Windows NT or UNIX.

```
report rwsqldb pop "(year=1970)"
```

You can also run the report with the command:

```
report
```

In this case Report-Writer prompts you for the report name, database name, and value for *\$year*.

- The .sort statement specifies a sorting of the data by region, and within region, by state. This also defines potential break actions for changes in value of region and state.
- The .format statement sets up a default format for a set of columns in the report. These are used not only for the printing of the actual data but also for the printing of subtotals based on that data. The four numeric columns (tot, tot\_under18, tot\_18to65, and tot\_over65) have the same format specification. Actually, the .format statement is not strictly needed, but provides a convenient way to specify the same format for a number of columns.
- The .header report statement precedes a set of formatting statements that execute at the start of the report and write out the centered title at the top of the report. The dollar sign preceding *year* in the print statement for the second line of this title indicates that *year* is a variable entered at run time. The statements in this section also print underlined column headings. Report-Writer determines the locations of the headings from the positions of the column names given as variables to the .right (right justify) statements. Report-Writer determines the column positions from print locations for the associated columns in the .detail statements.
- The .header region statement precedes a set of formatting statements that execute at the start of each region. The .need statement insures that at least four lines are available on a page before printing the heading for region. This assures that Report-Writer prints the heading and the detail lines for at least two states on a page.

- The .detail statement precedes a set of formatting statements to be processed for every row created by the query. The statements in this section create rows for each state and specify printing of the actual population data. By analyzing these statements, Report-Writer determines the positions of the columns used throughout the report in the .rt statements.
- The .footer statement precedes a set of formatting statements to be processed after Report-Writer has read the last state in each region and has processed the requisite .detail formatting statements. This report specification uses a .need statement to insure that the two lines in the footer both print on the same page. The statements in this section print a region heading, followed on the same line with the values of some subtotals for the region. The formats used in printing the subtotals are those specified in the .format statement at the start of the report.
- The statements following the .footer report statement are almost identical to those following the .footer region statement, except for the heading and length of the dashed line separators. The values of the subtotals, however, are different because of the different context.
- The statements following the .header page statement specify the title at the top of the second page of the report, as well as a re-specification of the column headings.
- The .footer page statement starts the block of statements that print at the bottom of each page, including the current page number. Because the .right statement has no parameters, the text is justified to the right margin (determined as the right-most position printed in the formatting statements in the report).

```

/* POPULATION - Population Report */
.NAME pop
.OUTPUT pop.out
.LONGREMARK
The POPULATION report demonstrates a fairly common type
of report with subtotaling.
.ENDREMARK
.QUERY
    select region.region, state.state,
           pop.tot_18to65 + pop.tot_under18 + pop.tot_over65 as tot,
           pop.tot_18to65, pop.tot_under18, pop.tot_over65
    from region, state, pop
    where state.statabbrev = pop.statabbrev
          and state.regabbrev = region.regabbrev
          and pop.year = $year
.SORT region, state
.DECLARE year = varchar(4) with prompt 'Enter Year:'
.FORMAT tot, tot_18to65, tot_under18, tot_over65 (' zzz,zzz,zzz')
.HEADER report
    .NEWLINE 3
    .UL .CE .PR 'Population of the United States, by Age Group' .NOU
    .NEWLINE .CE .PR 'Data for the Year - ', $year(c4) .NL 2
    .U .RT tot .PR 'Total Pop' .RT tot_18to65 .PR '18 to 65'
    .RT tot_under18 .PR 'Under 18' .RT tot_over65 .P 'Over 65'
    .NOU .NL 2
.HEADER region
    .NEED 4 .PR 'Region: ', region .NL
.DETAIL
    .NEED 2 .T5 .PR state(c20)
    .T+11 .PR tot, tot_18to65, tot_under18, tot_over65 .NL
.FOOTER region
    .NEED 2 .RT tot .PR '-----' .RT tot_18to65 .P '-----'
    .RT tot_under18 .P '-----' .RT tot_over65 .P '-----'
    .NL .PR 'Totals: ', region (c0) .T tot
    .PR sum(tot), sum(tot_18to65), sum(tot_under18), sum(tot_over65)
    .NL 2
.FOOTER report
    .NEED 2
    .RT tot .PR '-----' .RT tot_18to65 .P '-----'
    .RT tot_under18 .P '-----'
    .RT tot_over65 .P '-----' .NL
    .PR 'USA Totals' .T tot
    .PR sum(tot), sum(tot_18to65), sum(tot_under18), sum(tot_over65)
    .NL
.HEADER page
    .NL 3 .PR 'Population by State and Region: ', $year .NL 2
    .U .RT tot .P 'Total Pop' .RT tot_18to65 .P '18 to 65'
    .RT tot_under18 .P 'Under 18' .RT tot_over65 .P 'Over 65' .NOU
    .NL 2
.FOOTER page
    .NL
    .PR 'Source: US Department of the Interior, Bureau of the
        Census.'
.RIGHT .PR 'Page', page_number('zN') .NL 4

```

Population of the United States, by Age Group  
Data for the Year - 1970

	Total Pop	18 to 65	Under 18	Over 65
Region: East North Central				
Illinois	11,113,976	9,600,381	1,425,674	87,921
Indiana	5,193,669	4,820,324	357,464	15,881
Michigan	8,875,083	7,833,474	991,066	50,543
Ohio	10,652,017	9,646,997	970,477	34,543
Wisconsin	4,417,731	4,258,959	128,224	30,548
	-----	-----	-----	-----
Totals: East North Central	40,252,476	36,160,135	3,872,905	219,436
Region: East South Central				
Alabama	3,444,165	2,528,983	908,247	6,935
Kentucky	3,218,706	2,971,232	241,292	6,182
Mississippi	2,216,912	1,393,283	815,770	7,859
Tennessee	3,923,687	3,283,432	631,696	8,559
	-----	-----	-----	-----
Totals: East South Central	12,803,470	10,176,930	2,597,005	29,535
Region: Middle Atlantic				
New Jersey	7,168,164	6,349,908	770,292	47,964
New York	18,190,740	15,790,307	2,166,933	233,500
Pennsylvania	11,793,909	10,737,732	1,016,514	39,663
	-----	-----	-----	-----
Totals: Middle Atlantic	37,152,813	32,877,947	3,953,739	321,127
Region: Mountain				
Arizona	1,770,900	1,604,948	53,344	112,608
Colorado	2,207,259	2,112,352	66,411	28,496
Idaho	712,567	698,802	2,130	11,635
Montana	694,409	663,043	1,995	29,371
Nevada	488,738	448,177	27,762	12,799
New Mexico	1,016,000	915,815	19,555	80,630
Utah	1,059,273	1,031,926	6,617	20,730
Wyoming	332,416	323,024	2,568	6,824
	-----	-----	-----	-----
Totals: Mountain	8,281,562	7,798,087	180,382	303,093
Region: New England				
Connecticut	3,031,709	2,835,458	181,177	15,074
Maine	992,048	985,276	2,800	3,972
Massachusetts	5,689,170	5,477,624	175,817	35,729
New Hampshire	737,681	733,106	2,505	2,070
Rhode Island	946,725	914,757	25,338	6,630
Vermont	444,330	442,553	761	1,016
	-----	-----	-----	-----
Totals: New England	11,841,663	11,388,774	388,398	64,491

Source: US Department of the interior, Bureau of the Census.

Page 1



## Population by State and Region: 1970

	Total Pop	18 to 65	Under 18	Over 65
Region: Pacific				
Alaska	300,382	236,767	8,911	54,704
California	19,953,134	17,761,032	1,400,143	791,959
Hawaii	768,561	298,160	7,573	462,828
Oregon	2,091,385	2,032,079	26,308	32,998
Washington	3,409,169	3,251,055	71,308	86,806
	-----	-----	-----	-----
Totals: Pacific	26,522,631	23,579,093	1,514,243	1,429,295
Region: South Atlantic				
Delaware	548,104	466,459	78,276	3,369
District of Columbia	756,510	209,272	537,712	9,526
Florida	6,789,443	5,711,411	1,049,578	28,454
Georgia	4,589,575	3,387,516	1,190,779	11,280
Maryland	3,922,399	3,193,021	701,341	28,037
North Carolina	5,082,059	3,891,510	1,137,664	52,885
South Carolina	2,590,516	1,794,430	789,041	7,045
Virginia	4,648,494	3,757,478	865,388	25,628
West Virginia	1,744,237	1,666,870	73,931	3,436
	-----	-----	-----	-----
Totals: South Atlantic	30,671,337	24,077,967	6,423,710	169,660
Region: West North Central				
Iowa	2,824,376	2,782,762	32,596	9,018
Kansas	2,246,578	2,122,068	106,977	17,533
Minnesota	3,804,971	3,736,038	34,868	34,065
Missouri	4,676,501	4,177,495	480,172	18,834
Nebraska	1,483,493	1,432,867	39,911	10,715
North Dakota	617,761	599,485	2,494	15,782
South Dakota	665,507	630,333	1,627	33,547
	-----	-----	-----	-----
Totals: West North Central	16,319,187	15,481,048	698,645	139,494
Region: West South Central				
Arkansas	1,923,295	1,561,108	357,225	4,962
Louisiana	3,641,306	2,539,547	1,088,734	13,025
Oklahoma	2,559,232	2,275,104	177,910	106,218
Texas	11,196,730	9,696,569	1,419,677	80,484
	-----	-----	-----	-----
Totals: West South Central	19,320,563	16,072,328	3,043,546	204,689
	-----	-----	-----	-----
USA Totals	203,165,702	177,612,309	22,672,573	2,880,820

Source: US Department of the interior, Bureau of the Census.

Page 2

## POP2 Example

The POP2 example shows an alternative set of formatting statements for producing the same output as POPULATION. This report makes use of the `.block` and `.endblock` statements, as well as the `.within` and `.endwithin` statements, in producing the report. These statements are useful for reports which contain several columns for which the same set of statements is repeated as is the case with the `tot`, `tot_18to65`, `tot_under18`, and `tot_over65` columns in POPULATION.

All of the statements in POP2 are identical to the statements in POPULATION with the exception of those in the `.footer` region and `.footer` report sections. In these sections, instead of spelling out the format of the subtotals line by line, the block and column formatting statements can be used to duplicate the same set of statements for each of several columns.

In detail, the statements are:

- The `.block` statement sets the Report-Writer into block mode, which allows you to write a two-dimensional block of text, in which you can write text on several lines, return to the first line in the block, and then write more text on the first lines in the block.
- The `.within` statement sets the Report-Writer into column formatting mode. Because the statement is followed by four column names (`tot`, `tot_18to65`, `tot_under18`, and `tot_over65`), Report-Writer executes all statements between the `.within` and its corresponding `.endwithin` statement four times, using the margins for each of the columns in turn.
- The string of hyphens ("-----") prints, right justified, within each of the four columns in the first line of the block. Because this is a `.printline` statement, the current output line moves down one line in the block after the string prints. A sum prints on the second line of the block, right justified within each of the columns. Because this sum uses the special name `w_column`, Report-Writer calculates and prints a separate sum for each of the columns in turn.
- The `.endwithin` statement ends the set of formatting statements to be done within each column. Because block mode is in effect, a `.top` statement automatically executes immediately before the `.endwithin` statement. This ensures that the statements for each of the columns prints across the page, rather than stair-stepping down the page.
- Immediately following the `.endwithin` statement is the `.top` statement, which moves the current output line back to the first line in the current block (which contains all of the strings of hyphens ("-----")). The `.newline` statement moves the current position to the second line in the block (as block mode is still in effect), and Report-Writer prints the Totals: region string. The `.endblock` statement causes Report-Writer to print the block, consisting of two lines, and to leave block mode. The last `.newline` statement inserts a blank line.

- The statements within the .footer section are identical, except for the string, USA Totals. Because of the context, "sum(w\_column)" refers to totals for the entire report, rather than for the region.

The statements in this example are not quite as intuitive as those in the POPULATION report specification, but they show an important capability for formatting reports with column-oriented statements.

```

/* POP2 - Population Report using .WITHIN */
.NAME pop2
.OUTPUT pop2.out
.LONGREMARK
POP2 shows an alternative set of formatting statements
(.BLOCK and .ENDBLOCK and .WITHIN and .ENDWITHIN)
for producing the same output as the POPULATION report. The same set of
statements is repeated for the columns: totpop, tot_18to65, tot_under18,
tot_over65. All statements in POP2 are the same as POPULATION with the
exception of those in the ".FOOT region" and ".FOOT report" sections.
.ENDREMARK
.QUERY
    select region.region, state.state,
           pop.tot_18to65 + pop.tot_under18 + pop.tot_over65 as tot,
           pop.tot_18to65, pop.tot_under18, pop.tot_over65
    from region, state, pop
    where state.statabbrev = pop.statabbrev
          and state.regabbrev = region.regabbrev
          and pop.year = $year
.SORT region, state
.DECLARE year = varchar(4) with prompt 'Enter Year:'
.FORMAT tot, tot_18to65, tot_under18, tot_over65 (' zzz,zzz,zzz')
.FORMFEEDS
.HEADER report
    .NEWLINE 3
    .UL .CE .PR 'Population of the United States, by Age Group' .NOU
    .NEWLINE .CE .PR 'Data for the Year - ', $year(c4) .NL 2
    .U .RT tot .PR 'Total Pop' .RT tot_18to65 .PR '18 to 65'
        .RT tot_under18 .PR 'Under 18' .RT tot_over65 .P 'Over 65'
    .NOU .NL 2
.HEADER region
    .NEED 4
    .PR 'Region: ', region .NL
.DETAIL
    .NEED 2 .T5
    .PR state(c20) .T+11 .PR tot, tot_18to65, tot_under18, tot_over65
    .NL
.FOOTER region
    .NEED2
    .BLOCK
        .WITHIN tot, tot_18to65, tot_under18, tot_over65
        .RT .PRINTLINE '-----'
        .RT .PRINTLN sum(w_column)
    .ENDWITHIN
    .TOP .NEWLINE .PR 'Totals: ', region(c0)
    .ENDBLOCK .NEWLINE


```

```
.FOOTER report
.NEED2
.BLOCK .WITHIN tot, tot_18to65, tot_under18, tot_over65
      .RT .PRINTLINE '-----'
      .RT .PRINTLN sum(w_column)
      .END WITHIN
      .TOP .NEWLINE .PR 'USA Totals'
.ENDBLOCK .NEWLINE
.HEADER page
.NL 3 .PR 'Population by State and Region: ', $year .NL 2
.U .RT tot .P 'Total Pop' .RT tot_18to65 .P '18 to 65'
.RT tot_under18 .P 'Under 18' .RT tot_over65 .P 'Over 65'
.NOI .NL 2
.FOOTER page
.NL
.PR 'Source: US Department of the Interior, Bureau of the
    Census.'
.RIGHT .PR 'Page', page_number('zN') .NL 4
```


## QUEL User Notes for POPULATION and POP2 Examples

For the `$year` variable, you can select the data for 1970 by running the report with the command:


### Windows:

```
report rwqueldb pop (year=1970) 
```

### UNIX:

```
report rwqueldb pop '(year=1970)' 
```

### VMS:

```
report rwqueldb pop (year=1970) 
```

The QUEL version of the query for this example is shown below. This query is identical for both the POPULATION and POP2 examples.

```
/* POPULATION - Population Report */
.NAME pop
.QUERY
  range of r is region
  range of s is state
  range of p is pop
  retrieve (r.region, s.state,
    tot = p.tot_18to65 + p.tot_under18 +
    p.tot_over65, p.tot_18to65, p.tot_under18,
    p.tot_over65,
  where s.statabbrev = p.statabbrev
    and s.regabbrev = r.regabbrev
    and p.year = $year
```

## ACCOUNT Example

The ACCOUNT example shows a fairly complex report that could be written from some accounting data. For each account, the report prints the name and address of a customer, followed by a listing of transactions in an account. The report lists deposits in one column, withdrawals in another, and a running balance in a third.

The following base tables are used:

- The table, "customer table," which contains the name and address of a customer
- The "account" table, which associates an account number with a customer name and address (because a customer can have more than one account)

This table also contains the balance of an account as of an arbitrary date. In actual accounting applications, this balance must be updated outside of Report-Writer.

- The "transact" table, which contains a description of all transactions for an account

This table contains columns, transnum (the transaction number), acct num (the account number), tdate (the date of the transaction), amount (the dollar amount of transaction), and type (the type of transaction: 0 for deposits, 1 for withdrawals).

The following tables provide additional details on each of the database table layouts:

### Customer Table Definition

Column Name	Type	Length	Nulls	Defaults
"c name"	char	20	yes	no
address	char	20	yes	no
city	char	20	yes	no
state	char	2	yes	no
zip	integer	4	yes	no
balance	decimal	(12,2)	yes	no

### Customer Data for the Sample Report

"c name"	Address	city	state	zip
P.J. Megabucks	1 Panorama Lane	Hilltop	CT	12345
C. Richard Runn	123 Primer Path	Reading	PA	23456

**Account Table Definition**

Column Name	Type	Length	Nulls	Defaults
name	char	20	yes	no
"acct num"	integer	4	yes	no
balance	decimal	(12,2)	yes	no

**Account Data for the Sample Report**

name	acctnum	balance
P.J. Megabucks	749025436	234657.00
C. Richard Runn	488219082	1245.00

**Transact Table Definition**

Column Name	Type	Length	Nulls	Defaults
acctnum	integer	4	yes	no
tdate	date		yes	no
transnum	integer	4	yes	no
type	integer	4	yes	no
amount	decimal	(12,2)	yes	no

**Transact Data for the Sample Report**

acctnum	Tdate	transnum	type	amount
749025436	01-jul-1998	0101	0	100000.00
749025436	01-jul-1998	0102	1	50500.00
749025436	01-jul-1998	0103	1	24.56
749025436	01-jul-1998	0104	1	10100.00
749025436	15-jul-1998	0105	0	50000.00
749025436	17-jul-1998	0106	1	10143.54
749025436	17-jul-1998	0107	1	243.56
749025436	22-jul-1998	0108	1	100.00
749025436	23-jul-1998	0109	1	25000.00
749025436	23-jul-1998	0110	0	100000.00
488219082	25-may-1998	0101	1	200.00
488219082	03-jul-1998	0102	0	250.00

<b>acctnum</b>	<b>Tdate</b>	<b>transnum</b>	<b>type</b>	<b>amount</b>
488219082	05-jul-1998	0103	1	320.34
488219082	05-jul-1998	0104	0	65.23
488219082	08-jul-1998	0105	1	100.00
488219082	10-jul-1998	0106	1	56.32
488219082	16-jul-1998	0107	1	24.71
488219082	20-jul-1998	0108	1	120.00
488219082	25-jul-1998	0109	1	31.16

The report formatting statements are described here:

- The `.delimid` statement enables recognition of delimited identifiers used as a table name (customer table) and column names (c name and acct num).
- The `.declare` section declares the variables `$min_limit` and `$max_limit`, to be given values in response to a run-time prompt, and `$final_balance`, with an initial value of 0.00.
- The query, shown after the `.longremark` section, provides data for the report. This query retrieves the `transact` table, with data from the other tables joined in. The specification also shows the calculation of the columns `amt`, `withdrawal`, and `deposit`. The `withdrawal` value is set to `amount` if `type` is 1; otherwise, it is set to zero. The `depositvalue` is set to `amount` if `type` equals 0; otherwise, it is set to zero. The value of `amt` is calculated as a signed value of `amount`, which is negative for withdrawals and positive for deposits. The `amt` value is used in calculating the running balance.
- The `.sort` statement describes the order of the data. In the example output, only one account appears for each name, although this sort order would print additional accounts for each name if they existed in the data.
- The `.formfeeds` statement inserts form feed characters at the start of the report and at the end of each page of the report. Because no `.pagelength` statement is specified, a default page size of 61 lines is assumed. The default is determined by the output for the report. The output can be to a screen, file, or printer.
- The `.format` statements provide default formats for some of the output columns. The hyphen (-) in the format for `acct num` forces hyphens in specific places in the output.
- The `.header` statement begins the set of formatting statements to be executed at the start of each new name. The `.newpage` statement tells Report-Writer to skip to the top of a new page and to set the page number to the value 1 at the start of each new name. The next statements print the address and skip some lines.

- The `.head acct num` block prints the opening balance, column headings, and sets a temporary format for `acct num` (so that it is printed for the first transaction only). Report-Writer determines the positions associated with the columns from a scan of the formatting statements in the `.detail` section, including a position for the `amt` column, which is somewhat hidden in the cumulative sum function.
- The `.head tdate` block sets a temporary format for `tdate`, so that Report-Writer prints the date only the first time it encounters that particular date value.
- The `.detail` block prints out the lines in the report. It also determines the default margins and column positions from an analysis of these statements. The formats for `tdate` and `acct num`, which specify nonprinting formats, can be overridden by the `.tformat` statements specified in the header text for `tdate` and `acct num`.

The `"cum("acct num") sum(amt,balance)"` aggregate specifies the calculation and printing of the running balance. The first part, `"cum("acct num")"` specifies that the running balance is a cumulative aggregate, which is initialized at the most recent break in `"acct num."` The rest, `"sum(amt,balance),"` specifies that the cumulative aggregate is a sum of `"amt,"` and that the cumulative is to be initialized to the value of `"balance"` when the report starts (at the most recent break in `"acct num"`). The format to be used is specified as the default for `"amt"` because the aggregate specification is not followed by a parenthesized format.

- The `.foot acct num` block prints out summations of the withdrawal and deposit columns and the closing balance of the account, as calculated in the `"sum(amt,balance)"` aggregate. Report-Writer calculates the closing balance as the sum of `amt` for a specific `acct num` (because of the context), and then initializes it to the value of `balance` at the start of `"acct num."` Remember that the figure is negative for withdrawals and positive for deposits. Because the aggregate specification is not followed by a parenthesized format specification, the `.format` statement for `amt` at the beginning of the report is used as the default format for the aggregate.

The `.if-.elseif-.endif` block prints a `"Balance below ..."` or `"Balance exceeds ..."` message if the customer's closing balance is less than the established minimum (`$min_limit`) or greater than the established maximum (`$max_limit`).

- The `.foot name` block specifies the printing of an ending statement.
- The `.head page` block describes the heading shown at the top of each page. The `.newpage` statement in the `.head name` statements forces the printing of the page header on the first page (which normally does not happen).
- The `.foot page` block tells Report-Writer to skip some lines at the end of each page.



The pages following the report specification contain sample reports generated with the following values:

- In the first example,  $\$min\_limit = 500.00$  and  $\$max\_limit = 100,000.00$
- In the second example,  $\$min\_limit = 1,000.00$  and  $\$max\_limit = 250,000.00$

```
/* ACCOUNT - example of bank statement report. */
.NAME account_delim
.OUTPUT account_delim.out
.LONGREMARK
The ACCOUNT_DELIM report shows a fairly complex report that could be
written for some accounting data. For each account, the report prints
the name and address of a customer, followed by a listing of
transactions in an account. Deposits are listed in one column,
withdrawals in another, and a running balance is listed in a third.
The report orders the transactions in LIFO date order. It also
demonstrates the use of:
    o schema.tablename
    o delimited identifiers
    o decimal datatypes
.ENDREMARK
.DELIMID
.DECLARE
    min_limit = decimal(12,2) with prompt
        'Enter minimum balance flag level: ',
    max_limit = decimal(12,2) with prompt
        'Enter maximum balance flag level: ',
    final_balance = decimal(12,2) with value '0.00'
.QUERY
SELECT  "c tbl"."c name", "c tbl".address, "c tbl".city,
        "c tbl".state, "c tbl".zip,
        a."acct num", a.balance,
        t.transnum, t.tdate,
        t.amount * t.type AS withdrawal,
        t.amount * (1 - t.type) AS deposit,
        (t.amount * (1 - t.type)) - (t.amount *
            t.type) AS amt
FROM    transact t, account a, dave."customer table" "c tbl"
WHERE   a."acct num" = t.acctnum and "c tbl"."c name" = a.name
.SORT   "c name", "acct num", tdate:d, transnum
.FORMAT "acct num" (' nnnnnnnnn '),
        tdate (d'01/02/03'),
        withdrawal, deposit, amt, balance (' $$$,$$$,$$$,zz')
.HEAD  "c name"
      .NEWPAGE
      .NL 3
      .PRINT "c name"
      .NL
      .PRINT address
      .NL
      .PRINT city (c0), ' ', state (c0), ' ', zip ('nnnnn')
      .NL 4
.FOOT  "c name"
      .NL 3
      .PRINT 'End of accounts for: ', "c name"
      .NL
.HEAD  "acct num"
      .NL 3
      .PRINT 'Account: ', "acct num"
      .RT amt
      .PRINT 'Opening balance:', balance
      .NL 2
      .UL
        .CE "acct num"
        .PRINT 'Account'
```

```

        .CE tdate
        .PRINT 'Date'
    .CE transnum
        .PRINT 'Transaction'
        .RT deposit
        .PRINT 'Deposit'
        .RT withdrawal
        .PRINT 'Withdrawal'
        .RT amt
        .PRINT 'Balance'
    .NL
    .NOU
    .TFORMAT "acct num" (' nnnnnnnnn ' )
.FOOT "acct num"
    .NL 2
    .PRINT 'Account', "acct num", 'totals.'
    .TAB deposit
    .PRINT sum(deposit)
    .TAB withdrawal
    .PRINT sum(withdrawal)
    .NL 2
    .LET final_balance = sum(amt, balance)
    .RT amt
    .PRINT 'Closing balance:', $final_balance (' $$$,$$$,$$$.$z')
    .IF $final_balance < $min_limit .THEN
        .NL
        .PRINT '*** Balance below established minimum of ',
            $min_limit,' ***'
    .ELSEIF $final_balance > $max_limit .THEN'
        .NL
        .PRINT '*** Balance exceeds established maximum of ',
            $max_limit,' ***'
    .ENDIF
    .HEAD tdate
    .TFORMAT tdate (d'01/02/03 ')
    .DETAIL
        .PRINT "acct num" (b16), tdate (b16),
        .TAB +8
        .PRINT transnum ('nnnn'), deposit, withdrawal
        .TAB +5
        .PRINT cum("acct num") sum(amt, balance)
    .NL
    .HEAD page
    .NL 2
    .PRINT 'Customer: ', "c name"
    .CE
        .PRINT 'Date: ', current_date (d'February 3, 1901'),
        .RT
        .PRINT 'Page ', page_number
    .NL 4
    .FOOT page
    .NL 3

```

```

Customer: P.J. Megabucks      Date: July 27, 2000      Page  1
P.J. Megabucks
1 Panorama Lane
Hilltop CT 12345
Account: 74-902543-6      Opening balance:  $234,657.00
Account      Date      Transaction      Deposit      Withdrawal      Balance

```

74-902543-6	93/07/23	0109	\$25,000.00	\$288,545.34
		0110	\$100,000.00	\$388,545.34
	93/07/22	0108	\$100.00	\$313,545.34
	93/07/17	0106	\$10,143.54	\$313,888.90
		0107	\$243.56	\$313,645.34
	93/07/15	0105	\$50,000.00	\$324,032.44
		0101	\$100,000.00	\$274,032.44
	93/07/01	0102	\$50,500.00	\$184,157.00
		0103	\$24.56	\$174,032.44
		0104	\$10,100.00	\$174,057.00
Account 74-902543-6 totals.			\$250,000.00	\$96,111.66
			Closing balance:	\$388,545.34
*** Balance exceeds established maximum of 250000.00 ***				
End of accounts for: P.J. Megabucks				

Customer: C. Richard Runn    Date: July 27, 2000    Page 1  
C. Richard Runn  
123 Primer Path  
Reading PA 23456

Account	Date	Transaction	Deposit	Withdrawal	Balance
Account: 48-821908-2			Opening balance: \$234,657.00		
48-821908-2	93/07/25	0109	\$31.16		\$707.70
	93/07/20	0108	\$120.00	\$738.86	
	93/07/16	0107	\$24.71	\$858.86	
	93/07/10	0106	\$56.32	\$883.57	
	93/07/08	0105	\$100.00	\$939.89	
	93/07/05	0103	\$320.34	\$974.66	
		0104	\$65.23	\$1,039.89	
	93/07/03	0102	\$250.00	\$1,295.00	
	93/05/25	0101	\$200.00	\$1,045.00	
Account 48-821908-2 totals.			\$315.23	\$852.53	
			Closing balance:	\$707.70	
End of accounts for: C. Richard Runn					

## QUEL User Notes for ACCOUNT Example

You cannot use delimited identifiers, schema or owner qualification for table names, or the decimal data type in QUEL queries. Therefore, to create a QUEL version of this report, you have to set up the tables without delimited identifiers as column or table names. The balance column is data type float8 rather than decimal (12,2). You are not able to use owner qualification for the customer table.

An equivalent QUEL query for a similar example would be:

```
/* ACCOUNT - example of bank statement report. */
.NAME account
.QUERY
    range of t is transact
    range of a is account
    range of c is customer
    retrieve (c.name, c.address, c.city, c.state, c.zip,
             a.acctnum, a.balance, t.transnum, tdate = t.date,
             withdrawal = t.amount * t.type,
             deposit = t.amount * (1 - t.type),
             amt = (t.amount * (1 - t.type)) - (t.amount * t.type))
    where a.acctnum = t.acctnum and c.name = a.name
```

## DICTIONARY Example

The DICTIONARY example shows an example of a report that lists a glossary of Ingres terms, with a listing of related keywords. This demonstrates the use of some of the word-processing functions available in Report-Writer. The following base tables are used:

- The "ddef" table, which contains names of terms and definitions of those terms
- The "dref" table, which contains a list of terms and their related keywords

Additional details on each of these table layouts are provided in the tables that follow:

**Dref Table Definition**

Column Name	Type	Length	Nulls	Defaults
word	char	20	yes	no
definition	char	250	yes	no

### Ddef Data for the Sample Report

Word	Definition
aggregate function	An aggregate operator which first groups rows on the basis of the value of a column or list of columns called the by-list, before computing the aggregate for each value of the by-list.
aggregate operator	An aggregate operator is a computation performed on a column across all rows in a table. Common aggregate operators are SUM, COUNT, and AVG. Aggregate operators can have qualifications to limit the number of rows used in the calculation.
attribute	Another term for a column in a table.
buffer	Another term for the Ingres workspace.
column	All data in Ingres is saved in the form of tables made up of rows and columns. In traditional database terminology, a column is a field in a record.
comparison operator	A symbol which specifies the kind of comparison to make in a qualification, such as > (for greater than), or = (for equality check).
compressed	Any of the Ingres internal storage structures can be compressed. Compression reduces the storage required for a table, by deleting all trailing blanks in character columns.

### Dref Table

Column Name	Type	Length	Nulls	Defaults
Word	char	20	yes	no
Ref	char	20	yes	no

### Dref Data for the Sample Report

Word	Ref
aggregate function	aggregate operator
aggregate function	aggregation
aggregate function	by list
aggregate function	computation
aggregate operator	aggregate function
aggregate operator	aggregation
aggregate operator	computation
Attribute	column

Word	Ref
Buffer	workspace
Column	attribute
Column	domain
Column	field
comparison operator	qualification
comparison operator	restriction
Compressed	character strings
Compressed	compression
Compressed	storage structures

The report formatting statements are described here:

- The query used to create the data for the report joins words and definitions with a list of the related keywords. Therefore, the data returned to the report contains one row for each related keyword. The `.detail` section statement pertains to the related keywords, and the `.head` word statement pertains to the word itself and the definition.
- Report-Writer sorts the data by word, and within word, by related keyword.
- The report specification sets the left and right margins to specific values because the default margins calculated for the report do not reflect the required margins of the report.
- The `.head` report section statement performs a page break and prints a page header at the top of the page.
- The `.head` statement for "word" prints out the underlined word and the newspaper style printing of the definition. The `cj0.50` format specifies a column format 50 spaces wide, with right justification, with printing occurring until the end of the string. The `.t80` statement then moves to position 80 (5 spaces to the right of the edge of the definition), and sets the left margin of the report to that position. This causes all printing to wrap around between the left margin (80) and the right margin (100). No `.newline` statement is given, so that the next printing occur at column 80 of the top line of the definition.
- The `.detail` section statement prints out the next related keyword for "word," until the next word is found. Because the format specified for `ref` is `c20`, it exactly fits within the temporary margins, and wrap-around format causes each keyword to appear on a separate line. Remember that the `.lm0` statement in the header text for "word" resets the left margin for printing a new word and definition.
- The `.foot` statement for "word" finishes the text for one "word" by printing out all the lines in the definition and related keyword list, and prints a blank line.
- The `.head` page section statements print out a title, page number, and column headings.



```

/* DICTIONARY - text example */
.NAME dict
.OUTPUT dict.out
.LONGREMARK
The DICTIONARY report demonstrates the use of character printing options
within the Report-Writer. It lists a glossary of Ingres terms,
  with a listing of related keywords. This demonstrates the use of some
of the word-processing functions available in the Report-Writer.
.ENDREMARK
.QUERY
    select ddef.word, ddef.definition, dref.ref
    from ddef ddef, dref dref
    where ddef.word = dref.word
.SORT word, ref
.LM 0
.RM 100
.HEAD report
  .NEWPAGE 1
.HEAD word
  .NE 3 .LM 0
  .UL .PR word(c25) .NOU
  .P definition(cj0.50)
  .T80 .LM80
.DETAIL
  .P ref(c20)
.FOOT word
  .NL 2
.HEAD page
  .NL 2
  .P 'Dictionary of Ingres Terms'
  .RT .P 'Page', page_number .NL 2
  .UL .P 'Word' .T definition .P 'Definition'
  .T80 .P 'Related Term' .NOU .NL 2
.FOOT page
  .NL 3

```

Dictionary of Ingres Terms	Page 1	Related Term
Word	Definition	
aggregate function	An aggregate operator which first groups rows on the basis of the value of a (list of) column(s) (called the "by-list"), before computing the aggregate for each value of the "by-list."	aggregate operator aggregation by list computation
aggregate operator	An aggregate operator is a computation performed on a column across all rows in a table. Common aggregate operators are SUM, COUNT, and AVG. Aggregate operators can have qualifications to limit the number of rows used in the calculation.	aggregate function aggregation computation
attribute	Another term for "column" in a table.	column

buffer	Another term for the Ingres "workspace".	workspace
column	All data in Ingres is saved in the form of tables made up of rows and columns. In traditional database terminology, a "column" is a "field" in a record.	attribute domain field
comparison operator	A symbol which specified the kind of comparison to make in a qualification, such as ">" (for greater than), or "=" (for equality check).	qualification restriction
compressed	Any of the Ingres internal storage structures can be compressed. Compression reduces the storage required for a table, by deleting all trailing blanks in character columns.	character strings compression storage structure

## DICT2 Example

The previous DICTIONARY report uses some margin tricks to accomplish what can perhaps more easily be accomplished with the block mode of Report-Writer. Instead of letting the margins and wrap-around format accomplish the task of moving down the page, you can use the more natural `.newline` statement in block mode to do this. The DICT2 report is the same as the DICTIONARY report, except for differences in the `.head` and `.foot` for "word," and a slight change in the `.detail` section.

The changed statements are:

- In the `.header` for "word," the report specification sets Report-Writer into block mode. This allows you to move down the page in a more orderly fashion than would otherwise be possible. The underlined word prints on the first line of the block. The newspaper-style printing of the definition causes some number of lines within the block to be written, depending on the length of the definition. However, when it has finished printing, the current output line is the top line in the block. You are now ready to print the detail lines, which contain the keywords for a term.
- Within the `.detail` section of the report, the specification statements cause Report-Writer to tab to column 80, and print the next value of "ref." The `.newline` statement moves the current output line down one line in preparation for the next value of "ref." Because it is in block mode, Report-Writer saves all text after the header for "word" until it encounters the `.endblock` statement.
- The `.end block` statement in the `.footer` section for "word" prints out the current block containing the word, its definition, and a list of related keywords. A `.newline` statement adds another blank line.

The DICT2 report accomplishes the same output as the DICTIONARY report, but in a somewhat more natural fashion.

```
/* DICT2 - text example, using .BLOCK */
.NAME dict2
.OUTPUT dict2.out
.LONGREMARK
The DICT2 report shows an alternative set of formatting statements for
producing the same output as the DICTIONARY report. The DICTIONARY
report uses some margin tricks to accomplish what can perhaps more
easily be accomplished with the block mode of the Report-Writer. Instead
of letting the margins and wraparound accomplish the task of moving down
the page, within block mode, you can use the more natural .NEWLINE
statement. Differences between the two reports is limited to the ".HEAD
word" and ".FOOT word" and a slight change in the ".DETAIL" section.
.ENDREMARK
.QUERY
        select ddef.word, ddef.definition, dref.ref
        from ddef ddef, dref dref
        where ddef.word = dref.word
.SORT word, ref
.LM 0
.RM 100
.HEAD report
        .NEWPAGE 1
.HEAD word
        .NEED 3
        .BLOCK
        .UL .PR word(c25) .NOU
        .PR definition(cj0.50)
.DETAIL
        .T80 .PR ref(c20) .NL
.FOOT word
        .END BLOCK
        .NL
.HEAD page
        .NL 2
        .P 'Dictionary of Ingres Terms'
        .RT .P 'Page', page_number .NL 2
        .UL .P 'Word' .T definition .P 'Definition'
        .T80 .P 'Related Term' .NOU .NL 2
.FOOT page
        .NL 3
```

## QUEL User Notes for DICTIONARY and DICT2 Examples

The QUEL version of the query for this example is shown below. This query is identical for both the DICTIONARY and DICT2 examples.

```
/* DICTIONARY - text example */
.NAME dict
.QUERY
        range of d is ddef
        range of r is dref
        retrieve (d.word, d.definition, r.ref)
        where d.word = r.word
```

## LABEL Example

The LABEL example shows a report that prints mailing labels three across the page. The base table, subscriber, is a mailing list containing the name, post office box, address, city, state and zip code for each label. If there is no post office box for the label, the field is left blank.

Additional details on the base table layout are provided in the following tables:

**Subscriber Table Definition**

Column Name	Type	Length	Nulls	Defaults
name	char	20	yes	no
po_box	char	20	yes	no
address	char	20	yes	no
city	char	20	yes	no
state	char	2	yes	no
zip	integer	4	yes	no


**Subscriber Data for the Sample Report**

name	po_box	Address	city	state	zip
Betty Clark		2556 Carey Rd	Boston	MA	01002
Ming Ho		1020 The Parkway	Mamaroneck	NY	10543
Pat McTigue		Route 146	Trumbell	CT	04239
T. Shigio	1234	201 Emperor Lane	Rye	NY	10101
Marvin Blumbert		17 Saville Row	Carmel	CA	93001
Carlos Ramos		2459 39th Ave	San Francisco	CA	94121
AnastassiosVasos		722 Fourth St.	Gualala	CA	95035
Mario Verducci	x-207	General Delivery	Middletown	WA	98112


The report formatting statements are described here:

- Report-Writer sorts the data by zip code.
- The report first begins a block so that the labels can be printed across the page.
- The labels are assumed to be *\$lbl\_width* wide and *\$lbl\_length* number of lines long. The value of *\$lbl\_width* must be greater than the actual number of characters printed and less than the page width, *\$pg\_columns*. The number of lines actually printed depends on the number of *.newline* statements in the report. Because *\$lbl\_length* is used to determine page breaks, it must be less than the number of lines per page, *\$pg\_lines*.
- Report-Writer creates a label by printing all fields of the table across four lines. If the field for the post office box is blank, the corresponding line cannot be printed.
- Some labels are best printed one set per page. This is useful if FF is used and the lines per page on the printer is set to the number of lines on the label. To print one set per page, adjust the value of *\$pg\_lines* to be equal to the *\$lbl\_lines* + 1.
- Report-Writer moves the left margin for the next label one label's width to the right of the previous left margin, if doing so does not cause the label to move beyond the right margin (for example, if only 1 or 2 labels have been formatted for a line which can fit three). When no more room exists on the line, the block of labels ends, the *.endblock* statement moves the report to the top of the next block of labels, the left margin is reset to 0 and a new block of labels begins. When the report runs out of data the block automatically ends, whether or not there is space left in the block buffer.
- This report uses declared variables without prompts. This means that the values of the variables must be passed in from the command line. The following report was run using these commands typed on a single line:

**Windows:**

```
report rwsqldb labels (pg_lines=11, pg_columns=79, lbl_width=25, lbl_length=5)

```

**UNIX:**

```
report rwsqldb labels "(pg_lines=11, pg_columns=79, lbl_width=25,
lbl_length=5)" 
```

/\*

```

** LABEL. Write out three across mailing labels
** with suppression of blank PO boxes.
*/
.NAME labels
.OUTPUT labels.out
.SHORTREMARK Prints data in mailing label format:
.LONGREMARK
Requires parameters pg_columns, pg_lines, lbl_width, and lbl_length
passed in on the command line. They will not be prompted for. The
following values will result in three labels across and two down:
  (pg_lines=11,pg_columns=79,lbl_width=25,lbl_length=5) By varying the
values you can vary the layout of the labels.
.ENDREMARK
.QUERY select name, po_box, address, city, state, zip
        from subscriber
.SORT zip
.DECLARE
        pg_columns = int, /* number of characters across the page */
        pg_lines = int, /* number of lines per page */
        lbl_width = int, /* the number of characters across one label*/
        lbl_length = int /* the number of lines in one label */
.PL $pg_lines
.PW $pg_columns
.RM $pg_columns
.LM 0
.HEADER report
        .NEED $lbl_length
        .BLOCK
.HEADER page
        .NEED $lbl_length
        .BLOCK
.DETAIL
        .TOP
        .LINESTART
        .PRINTLN name
        .IF po_box != ' ' .THEN
                .PRINTLN 'PO Box ', po_box
        .ENDIF
        .PRINTLN address(cf0.30)
        .PRINTLN city (c0),',',state(c0),' ',zip('nnnnn')
        .IF left_margin + ($lbl_width * 2) < right_margin .THEN
                .LM + $lbl_width
        .ELSE
                .ENDBLOCK
                .NL
                .NEED $lbl_length
                .BLOCK
                .LM 0
        .ENDIF
.HEADER page
        .ENDBLOCK
.HEADER report
        .ENDBLOCK

```

### Completed Report

Betty Clark 2556 Carey Rd. Boston, MA 01002	Pat McTigue Route 146 Trumbell, CT 04239	Ming Ho 1020 The Parkway Mamaroneck, NY 10012
T. Shigio PO Box 1234 201 Emperor Lane Rye, NY 10101	Marvin Blumbert 17 Saville Row Carmel, CA 93001	Carlos Ramos 2459 39th Ave San Francisco, CA 94121
Anastassios Vasos 722 Fourth St. Gualala, CA 95035	Mario Verducci PO Box X-207 General Delivery Middletown, WA 98112	

### QUEL User Notes for LABEL Example

The QUEL version of the query for this example is:

```
/*
** LABEL. Write out three across mailing labels
** with suppression of blank PO boxes.
*/
.NAME label
.QUERY
    range of s is subscriber
    retrieve (s.name, s.po_box, s.address, s.city,
             s.state, s.zip)
```

## Report Creation Using Several Tables

There are times when you want to use Report-Writer to produce a report from related information scattered across several tables that share one or more column definitions. You can do this by creating a temporary table or view based on multiple tables in the .setup section, using SQL statements. You can then drop the table or view, or update a status, in the .cleanup section of the same report.



## How to Join Tables for a Report

Suppose you want to assemble a report from a database of the books in your personal library. You decide upon a report design to present title, author, and subject information like this:

```
TITLE OF BOOK
Author1      Subject1
Author2      Subject2
              Subject3
```

In your database, you have designated three separate tables to hold this information—one for titles (title), one for authors (name) and one for subject information (subject):

### Book Table Definition

Column Name	Type	Length	Nulls	Defaults
id	integer	4	yes	no
title	varchar	20	yes	no

### Book Data for the Sample Report

id	title
1001	The C Programming Language
1002	Computer Programming Arch.
1003	The INGRES Papers
1004	Database Systems
1005	The Quiet American

### Author Table Definition

Column Name	Type	Length	Nulls	Default
id	integer	4	yes	no
name	varchar	15	yes	no

### Author Data for the Sample Report

Id	title
1001	Ritchie
1001	Kernighan
1002	Eckhouse
1002	Levy

<b>Id</b>	<b>title</b>
1003	Stonebraker
1004	Ullman
1005	Greene

**Subject Table Definition**

<b>Column Name</b>	<b>Type</b>	<b>Length</b>	<b>Nulls</b>	<b>Default</b>
id	integer	4	yes	no
subject	varchar	15	yes	no

**Subject Data for the Sample Report**

<b>id</b>	<b>Subject</b>
1001	C
1001	Programming
1001	Language
1002	Architecture
1002	Assembler
1002	Computer
1002	Programming
1003	Database
1003	Ingres
1003	Computer
1004	Database
1004	Management
1005	Vietnam

Now you must combine these tables to produce the data shown in the following table:

<b>id</b>	<b>title</b>	<b>name</b>	<b>subject</b>	<b>code</b>
1001	The C Programming Language	Kernighan		1
1001	The C Programming Language	Ritchie		1

<b>id</b>	<b>title</b>	<b>name</b>	<b>subject</b>	<b>code</b>
1001	The C Programming Language		C	2
1001	The C Programming Language		language	2
1001	The C Programming Language		programming	2

The easiest method is to use a UNION clause, as is shown in the .query section of the following report example:

```
.NAME books1
.OUTPUT books1.out
.LONGREMARK
    The BOOKS report demonstrates the use of joining tables
    for producing a report.
.ENDREMARK
.QUERY
    select b.id, b.title, a.name, '' as subject, 1 as code
    from book b, author a
    where b.id = a.id
    union
    select b.id, b.title, '' as name, s.subject, 2 as code
    from book b, subject s
    where b.id = s.id
.SORT title, code
.BREAK title, code /* title and code will be break columns */
.RIGHTMARGIN 80 /* Initialize right margin */
.DECLARE
    authors_column = integer,
    subject_column = integer,
    title_string = varchar(8)
.HEADER report
    /* Initialize variables */
    .LET authors_column = 5
    .LET subject_column = 20
.HEADER title
    /* Reset margin and print title of book */
    /* start a block after printing the master info */
    .LEFTMARGIN 0
    .ULCHARACTER '='
    .UNDERLINE
    .PRINT title .NEWLINE
    .NOUNDERLINE
    .BLOCK
.HEADER code
    /* goto the top of the block each time code changes */
    /* set margin to the correct column for the code type */
    .TOP
    .ULCHARACTER '-'
    .IF code = 1 .THEN
        .LEFTMARGIN $authors_column
        .LET title_string = 'Authors'
    .ELSEIF code = 2 .THEN
        .LEFTMARGIN $subject_column
        .LET title_string = 'Subject'
    .ENDIF
    .UNDERLINE
    .PRINT $title_string .NEWLINE
    .NOUNDERLINE
```

```
.DETAIL
/* test the value of code to see which column to print */
.IF code = 1 .THEN
.PRINT name .NEWLINE
.ELSEIF code = 2 .THEN
.PRINT subject .NEWLINE
.ENDIF
.FOOTER title
.ENDBLOCK /* end the block at the end of the master info */
.NEWLINE
```

### Completed Report

Computer Programming and Arch.

```
=====
Authors      Subjects
-----
Eckhouse     architecture
Levy         assembler
              Computer
              programming
```

### QUEL User Notes for a Join

In QUEL, you specify a join of several tables with a retrieve statement. In QUEL, you would use these queries to join the tables:

```
destroy tempbooksq\p\g
range of b is book\p\g
\range of a is author\p\g
range of s is subject\p\g
create tempbooksq (
  id      = i4,
  title   = varchar(30),
  name    = varchar(15) not null with default,
  subject = varchar(15) not null with default,
  code    = i1
)\p\g
append tempbooksq (b.all, a.name, code=1)
  where b.id = a.id\p\g
append tempbooksq (b.all, s.subject, code=2)
  where b.id = s.id\p\g
```

## How to Avoid Awkward Page Breaks

Assume you have invested in a new bookcase and have expanded the size of your personal library by many volumes. Now when you combine your three tables, you create a much larger union than before:

id	Title	name	subject	code
1001	The C Programming Language	Kernighan		1
1001	The C Programming Language	Ritchie		1
1001	The C Programming Language		C	2
1001	The C Programming Language		language	2
1001	The C Programming Language		programming	2

id	title	name	subject	code
1001	The C Programming Language	Kernighan		1
1001	The C Programming Language	Ritchie		1
1002	Computer Programming and Arch.	Eckhouse		1
1002	Computer Programming and Arch.	Levy		1
1003	The INGRES Papers	Stonebraker		1
1004	Database Systems	Ullman		1
1005	The Quiet American	Greene		1
1001	The C Programming Language		C	2
1001	The C Programming Language		language	2
1001	The C Programming Language		programming	2
1002	Computer Programming and Arch.		architecture	2
1002	Computer Programming and Arch.		assembler	2
1002	Computer Programming and Arch.		computer	2
1002	Computer Programming and Arch.		programming	2
1003	The INGRES Papers		Database	2
1003	The INGRES Papers		INGRES	2
1003	The INGRES Papers		computer	2
1004	Database Systems		Database	2
1004	Database Systems		management	2
1005	The Quiet American		Vietnam	2

If you create a report from such variable blocks of data, you must issue specific instructions to Report-Writer about where and where not to place page breaks; otherwise, you find that some of your data has been incongruously parceled across two pages. In cases where you use Report-Writer to generate a report from a single, unjoined table, you would use a simple .need statement to establish proper page breaks..

In this case, when generating a report from a joined table, you must use variable parameters to the `.need` statement to assure proper page breaks, as follows:

1. Change the selection statement in your `.query` section to a creation of a view in your `.setup` section.
2. Create two new tables in your `.setup` section based on that view which contain information about the number of authors or subjects per book.
3. Join the two new tables with the view in the `.query` section.
4. Alter the report specifications to use the new information for paging.

**Note:** The `.setup` and `.cleanup` sections can only use the SQL language. For the QUEL implementation of this example, see the QUEL Users Notes section below.

The reason for the three-step process is that SQL requires a special method for the calculation of *num\_sub* and *num\_auth*. In SQL, when you perform a set function on a set of data and group rows together, you cannot place in the select clause any column not also listed in the group by clause, except as an argument to a set function. When a select statement includes a group by clause, any columns listed in the select clause must be single-valued per group.

Here is the revised specification file for the report, with the new `.setup`, `.cleanup` and `.query` sections:



```

.NAME books2
.OUTPUT books2.out
.LONGREMARK
    The BOOKS2 report demonstrates using setup and cleanup to
    produce temporary tables.
.ENDREMARK
.SETUP
    create view tempbooks as
        select b.id, b.title, a.name, '' as subject, 1 as code
            from book b, author a
            where b.id = a.id
        union
        select b.id, b.title, '' as name, s.subject, 2 as code
            from book b, subject s
            where b.id = s.id;
    create table subj_count as
        select id, num_sub=count(subject)
            from tempbooks
            where code = 2
            group by id;
    create table auth_count as
        select id, num_auths=count(name)
            from tempbooks
            where code = 1
            group by id;
.CLEANUP
    drop tempbooks;
    drop subj_count;
    drop auth_count;
.QUERY
    select      b.id, b.title, b.name, b.subject, b.code,
               a.num_auths, s.num_sub
    from tempbooks b, subj_count s, auth_count a
    where b.id = a.id and b.id = s.id
.PAGELength 20
.SORT title, code
.BREAK title, code /* title and code will be break columns */
.RIGHTMARGIN 80 /* it is important to set the right margin here */
.DECLARE
    lines_in_title = integer,
    authors_column = integer,
    subject_column = integer,
    title_string = varchar(8)
.HEADER report
    /* Initialize variables */
    .LET lines_in_title = 4
    .LET authors_column = 5
    .LET subject_column = 20
.HEADER title
    /* Request the maximum number of lines needed to print all */
    /* book information on one page. */
    .IF num_sub num_auths .THEN
        .need num_sub + $lines_in_title
    .ELSE
        .need num_sub + $lines_in_title
    .ENDIF
    /* Reset margin and print title of book */
    /* start a block after printing the master info */
    .LEFTMARGIN 0

```

```
.ULCHARACTER '='
.UNDERLINE
.PRINT title .NEWLINE
.NOUNDERLINE
.BLOCK
.HEADER code
/* goto the top of the block each time code changes */
/* set the margin to the correct column for the code type */
.TOP
.ULCHARACTER '-'
.IF code = 1 .THEN
    .LEFTMARGIN $authors_column
    .LET title_string = 'Authors'
.ELSEIF code = 2 .THEN
    .LEFTMARGIN $subject_column
    .LET title_string = 'Subject'
.ENDIF
.UNDERLINE
.PRINT $title_string .NEWLINE
.NOUNDERLINE
.DETAIL
/* test the value of code to see which column to print */
.IF code = 1 .THEN
    .PRINT name .NEWLINE
.ELSEIF code = 2 .THEN
    .PRINT subject .NEWLINE
.ENDIF
.FOOTER title
.ENDBLOCK /* end the block at the end of the master info */
.NEWLINE
```

## Computer Programming and Arch.

=====

Authors      Subject

-----

Eckhouse    architecture

Levy        assembler

Computer

Programming

## Database Systems

=====

Authors      Subject

-----

Ullman      Database

Management

## The C Programming Language

=== = =====

Authors      Subject

-----

Kernighan   C

Ritchie     language

Programming

## The INGRES Papers

=== =====

Authors      Subject

-----

Stonebraker Database

INGRES

Computer

## The Quiet American

=== =====

Authors      Subject

-----

Greene      Vietnam

## QUEL User Notes for a Join

When you generate a report from a joined table, you must simulate the .need statement to assure proper page breaks. In QUEL, this involves making the same alterations to the report specifications file as shown in the text, but no additional tables need to be constructed.

Here is the query statement for the final report specification:

```
.NAME booksq
.OUTPUT booksq.out
.LONGREMARK
    The BOOKSQ report uses previously executed QUEL
    statements to create a temporary table,tempbooksq
    which is the join of books, authors and subject.
.ENDREMARK
.QUERY
    range of b is tempbooksq
    retrieve (b.all,
        num_auths=count(b.subject by b.id where
            b.code=1),
        num_sub=count(b.subject by b.id where b.code=2)
    )
...
```

# Appendix G: Troubleshooting Report-Writer

---

This section contains the following topics:

[Parameter Substitution Troubleshooting](#) (see page 753)

[Query Troubleshooting](#) (see page 755)

[Comment Troubleshooting](#) (see page 756)

[Default Print Position Troubleshooting](#) (see page 757)

[Formfeed Troubleshooting](#) (see page 757)

[Performance Problem Troubleshooting](#) (see page 759)

## Parameter Substitution Troubleshooting

Report-Writer parameter substitution problems commonly fall into the following areas:

- Failure to declare variables
- Failure to always precede variable references with a dollar sign (\$)
- Special characters in the variable string
- Embedded quotes in the variable string

Always declare all variables. Although Report-Writer recognizes any name preceded by a dollar sign (\$) as a variable, undeclared variables assume default types and characteristics that are often incompatible with their intended use. If a variable has *not* been declared, you can assign it a value only by using a command line parameter or by entering a value in response to a run-time prompt. You cannot specify a value for an undeclared variable with a .let statement. Also, unless the variable has been declared, attempting to pass a parameter with a null value to Report-Writer can produce incorrect results.

Always specify variable instances by including the leading dollar sign (\$). If the dollar sign is not present, and the variable has the same name as an identifier, such as a column or table name, Report-Writer assumes the variable is an identifier.

Check to make sure you have dereferenced any special characters in the variable's value string, such as embedded dollar sign (\$), SQL or QUEL wild cards, and so forth. Special characters in a variable's value string can cause unexpected results. They can be stripped and/or processed by your system's command line processor, or misinterpreted by Report-Writer. Some characters require multiple dereferencing, if the character is meaningful to both the native system's command line processor and Report-Writer.

Check to make sure you have correctly handled quotes in the variable string. Embedded quotes in the variable's value string present problems similar to those of special characters, and you must handle them in the same manner. Because of the potential confusion with delimited identifiers, we strongly advise you to always use single rather than double quotes to surround a variable value string. This greatly reduces the number of double quotes that must be escaped and standardizes the dereferencing of quotes, because string constants must always be delimited by single quotes.

To determine a variable's actual substituted value, use a test report such as this:

```
.NAME check_variable
.DECLARE string_variable = varchar(32) with value 'Hi!'
.QUERY
        SELECT count(object_id)
              FROM ii_reports
.DETAIL
        .PRINTLN $string_variable
```

This report prints the value of the variable as it can be used by Report-Writer. Date and format template variables behave similarly to string constants. Numeric variables behave in a much simpler manner, because they cannot contain characters that can be confused with comment or string delimiters. The above example works equally well for parameters specified on the command line and variables for which the user is prompted.

An alternate method of determining if the parameter value string is the cause of the failure is to save the report specification with the sreport command. If this fails, then the problem lies in the existing syntax. If this succeeds, then the problem most likely results from the effects of substituting the parameter value string.

The .setup, .cleanup, and .query statements differ from other statements in that parameter substitution can occur within quoted strings. For example, suppose your query contained the following where clause:

```
WHERE object_name = '$match_name'
```

If you substitute the value "ABC" for *\$match\_name*, the resulting where clause would be:

```
WHERE object_name='ABC'
```

Other statements, such as .println, treat a variable within a quoted string as a constant, as shown by the following results:

Statement	Printed Result
.PRINTLN '\$match_name'	\$match_name

Statement	Printed Result
.PRINTLN \$match_name	ABC

## Query Troubleshooting

Problems related to query processing usually result from misstating the query or from syntax errors or ambiguities as a consequence of parameter substitution.

You can easily test the form and structure of the query by using a terminal monitor utility. Its diagnostics are sometimes more detailed than those of Report-Writer, particularly where syntax is concerned.

**Note:** If your Report-Writer code includes a .sort statement, add an order by clause to the select statement when testing the query in a terminal monitor.

The single most useful means for determining query behavior is the II\_EMBED\_SET environment variable/logical. If you include the value printqry in its value string, execution of the report produces an "iiprtqry.log" file in the current directory. This log contains the text of all queries sent to the database, as formulated after all variable substitutions have occurred. The actual report query appears towards the end of the log file (after all queries related to Report-Writer's access of the Ingres catalogs). For information on setting this environment variable, see the *System Administrator Guide*.

## Comment Troubleshooting

In general, you must format comments as follows:

- Avoid comment nesting
- Separate comment delimiters from the comment text and from each other by white space, as in the following example:

```
/* Comment Text /* Unavoidable nested comment */ */
```

*Do not* use comment delimiters without intervening white space:

```
/*CommentText/*Unavoidable nested comment*/*/
```

- Within a .query statement, place comments *before* any from clause:

```
.QUERY
/*
** explanation of query
*/
SELECT col1
      FROM table
```

- Avoid placing comments *after* the from clause in a .query statement:

```
.QUERY
      SELECT col1
            FROM table
/*
** explanation of query
*/
```

Reports created in RBF can contain RBF-generated comments after the from clause in the .query statement that are preserved and used by Report-Writer. Placing your comments *prior* to the from clause avoids any potential confusion with the RBF-generated comments, because Report-Writer removes all such comments before saving the report specification to the database. This also saves room in the database. Report-Writer assumes that comments occurring *after* the from clause are RBF-generated comments and saves them along with the report specification.



## Default Print Position Troubleshooting

For other than very simple, default style reports, you must always specify values for `.pagewidth`, `.pagelength`, `.leftmargin`, and `.rightmargin`. If you do not specify these and other print positions explicitly, Report-Writer determines default positions from an analysis of your other report formatting statements. The results can or cannot compare favorably to your expectations. For an explanation of how Report-Writer determines default positioning values, see Automatic Determination of Default Settings (see page 282).

Within a single line, you must specify `.position` and non-relative `.tab` statements in ascending order, from left to right:

```
.print 'A' .tab 10 .print 'B' .tab 20 .print 'C' .tab 50
```

Specifying these statements in a different order can cause Report-Writer to calculate an incorrect right margin, which could result in unexpected wrapping or truncation of text.

## Formfeed Troubleshooting

The default for formfeeds in Report-Writer reports is:

### Windows and UNIX:

`.formfeeds` 

### VMS:

`.noformfeeds` 

The following Report-Writer statements and command line flags for the report command turn formfeeds on or off in your report, overriding the default behavior for Report-Writer in your environment:

- `.formfeeds` (`.ffs`, `.ff`) statement
- `.noformfeeds` (`.noffs`, `.noff`) statement
- `+b` | `-b` command line flags

Additionally, if formfeeds are enabled, you can specify *no initial formfeed* prior to the first page of your report with the following Report-Writer statement or report command line flag:

- `.nofirstff` statement
- `-nofirstff` command line flag

The following report command line flag overrides the `.nofirstff` statement if formfeeds are enabled, but has no effect otherwise:

- `-firstff`

If you are having trouble with formfeeds, use the following tables to determine the combined result of the system default, the formfeed statements in your report specification, and the formfeed run time command line flags.

The following table indicates the resulting behavior when formfeeds are enabled or disabled:

	No Flag Set		+b Flag	-b Flag
	UNIX Windows	VMS		
No formfeed statement specified	On	Off	On	Off
<code>.formfeeds</code> specified	On	On	On	Off
<code>.noformfeeds</code> specified	Off	Off	On	Off

The following table indicates whether an initial formfeed can occur when formfeeds are enabled:

	No Flag Set	-nofirstff	-firstff
No initial formfeed statement specified	Yes	No	Yes
<code>.nofirstff</code> specified	No	No	Yes

The following table indicates whether an initial formfeed can occur when formfeeds are disabled:

	No Flag Set	-nofirstff	-firstff
No initial formfeed statement specified	No	No	No
<code>.nofirstff</code> specified	No	No	No

## Performance Problem Troubleshooting

Approaches to solving performance problems are inherently implementation-dependent. The following observations and recommendations are specific to the release of Report-Writer associated with this manual. They can have little or no effect on other releases, and can in some cases degrade performance.

Performance problems can be related to the manner in which the query is stated, or to extensive expression usage and/or the use of functions in the body of the report specification.

### Query Performance

The performance of the query itself can easily be tested using a terminal monitor utility, and must never differ from its performance within Report-Writer.

### Performance and Conversion Function Usage

Conversion functions that occur as part of query processing can execute faster by up to two orders of magnitude compared to conversion function execution within the body of the report specification. Of the following two sample reports, the first report handles the conversion function most efficiently, assuming the function is being applied unconditionally to all rows retrieved.

Example 1:

```
.NAME query_convert
.QUERY
    SELECT uppercase(col1) AS uc_col1
        FROM test_table
.DETAIL
    .PRINT uc_col1
    .NL
```

Example 2:

```
.NAME rw_body_convert
.QUERY
    SELECT col1
        FROM test_table
.DETAIL
    .PRINT uppercase(col1)
    .NL
```

## Performance and Memory Usage

Report-Writer must allocate memory to control execution of each .print statement. Limiting the number of .print statements, as shown in Example 1, can result in significant run-time memory savings compared to Example 2.

Example 1:

```
.NAME small_memory
.QUERY
      SELECT col1, col2, col3, col4
      FROM test_table
.DETAIL
      .PRINT col1, col2, col3, col4
      .NL
```

Example 2:

```
.NAME large_memory
.QUERY
      SELECT col1, col2, col3, col4
      FROM test_table
.DETAIL
      .PRINT col1
      .PRINT col2
      .PRINT col3
      .PRINT col4
      .NL
      .noformfeeds
```

# Index

---

-

- (hyphen) • 99, 433
  - .ulcharacter • 433
  - pattern matching • 99
  - underlining character • 433

- (minus sign) • 296, 310, 316, 336, 382, 386, 392, 415, 418, 424, 579, 584
  - arithmetic • 310
  - exponent • 296
  - justification with • 316, 382, 386, 392, 415, 418, 424, 579
  - numeric templates • 336, 584
  - subtraction • 310
  - unary • 296, 310

'

- ' (single quotation mark) • 236, 291, 293, 566, 753
  - constants and • 291, 293
  - parameter passing • 236, 753
  - string literals and • 566
  - string variables and • 236

!

- != (not equal to) • 311
  - comparison operator • 311

#

- # (number sign) • 671
  - termcap descriptions • 671

\$

- \$ (dollar sign) • 270, 301, 336, 353, 364, 584, 753
  - numeric templates • 336, 584
  - report formats • 270, 364
  - variable names • 301, 353, 753

%

- % (percent sign) • 99
  - pattern match character • 99

(

- ( ) (parentheses) • 235, 236, 240, 242, 301, 336, 364, 369, 584
  - numeric templates • 336, 584
  - parameter passing • 235
  - system-level commands • 236, 240, 242, 364, 369
  - variable names • 301
- (colon) • 301, 349, 374, 378
  - .sort (Report-Writer statement) • 378
  - report cleanup and setup sections • 349, 374
  - variable names • 301

\*

- \* (asterisk) • 310, 312, 316, 336, 351, 538, 579, 580, 584
  - centering • 316
  - comment indicator • 351
  - displayed in a field • 580
  - exponentiation • 310
  - integer data display • 580
  - justification with • 579
  - multiplication • 310
  - numeric • 336
  - numeric templates • 584
  - wild card character • 312, 538

,

- , (comma) • 263, 301, 336, 584
  - numeric templates • 336, 584
  - value separator • 263
  - variable names • 301

.

- . (period) • 37, 296, 336, 570, 584
  - decimal indicator • 296, 336, 570, 584

- 
- numeric templates • 336, 584
  - owner qualification • 37
  - .block (Report-Writer statement) • 259, 398, 399, 411, 718
    - .blk abbreviation • 399
    - .within/.endwithin sections • 411
    - described • 399
    - examples • 399, 718
    - overview • 259, 398
  - .bottom (Report-Writer statement) • 259, 399, 401
    - .bot abbreviation • 401
    - described • 259, 399, 401
    - examples • 401
  - .break (Report-Writer statement) • 257, 268, 347, 378
    - .brk abbreviation • 347
    - .sort (Report-Writer statement) • 347, 378
    - described • 347
    - footers • 347
    - headers • 347
    - overview • 257, 268
  - .center (Report-Writer statement) • 258, 260, 415
    - .ce abbreviation • 415
    - .cen abbreviation • 415
    - described • 415
    - examples • 415
    - overview • 258, 260
  - .cleanup (Report-Writer statement) • 257, 268, 349, 740
    - autocommit • 349
    - described • 349
    - examples • 349, 740
    - overview • 257, 268
  - .data (Report-Writer statement) • 257, 268, 352, 369
    - .dat abbreviation • 352
    - .table synonym • 352
    - .view synonym • 352
    - described • 257, 268, 352
    - examples • 352
    - QUEL and • 369
  - .declare (Report-Writer statement) • 257, 268, 270, 353, 369, 721
    - .let (Report-Writer statement) and • 369
    - described • 353
    - examples • 353, 721
    - overview • 257, 268, 369
    - using with variables • 270
  - .delimid (Report-Writer statement) • 357, 358, 721
    - described • 357
    - example • 721
    - QUEL and • 358
  - .detail (Report-Writer statement) • 258, 283, 393, 708, 721, 729, 735
    - .det abbreviation • 393
    - described • 393
    - examples • 708, 721, 729, 735
    - overview • 258, 283
  - .else (clause) • 436
  - .elseif (clause) • 436
  - .elseif (Report-Writer statement) • 721
    - example • 721
  - .endblock (Report-Writer statement) • 259, 399, 718, 735, 737
    - .end block synonym • 399
    - .endblk abbreviation • 399
    - described • 399
    - examples • 399, 718, 735, 737
    - overview • 259
  - .endif (Report-Writer statement) • 436, 721
  - .endremark (Report-Writer statement) • 257, 361
    - .endrem abbreviation • 361
    - described • 361
    - examples • 361
    - overview • 257
  - .endwithin (Report-Writer statement) • 259, 411, 718
    - .end within synonym • 411
    - .endwi abbreviation • 411
    - described • 411
    - examples • 411, 718
    - overview • 259
  - .footer (Report-Writer statement) • 258, 347, 395, 708, 721, 729
    - .foot abbreviation • 395
    - .footing synonym • 395
    - breaks • 347
    - described • 395
    - examples • 395, 708, 721, 729
    - overview • 258
  - .format (Report-Writer statement) • 259, 284, 402, 708, 721
    - described • 402
    - examples • 402, 708, 721

---

- overview • 259
- using with columns • 284
- .formfeeds (Report-Writer statement) • 258, 274, 380, 721, 757
  - .ff abbreviation • 380
  - .ffs abbreviation • 380
  - described • 380
  - examples • 380, 721
  - overview • 258
  - troubleshooting • 757
  - using with pagination • 274
- .header (Report-Writer statement) • 258, 397, 708, 721, 729, 735
  - .head abbreviation • 397
  - .heading synonym • 397
  - described • 397
  - example • 397, 708, 721, 729, 735
  - overview • 258
- .if (Report-Writer statement) • 261, 281, 436, 721
  - described • 436
  - example • 436, 721
  - overview • 261
  - using with conditions • 281
- .include (Report-Writer statement) • 257, 268, 359
  - described • 359
  - example • 359
  - overview • 257, 268
- .left (Report-Writer statement) • 260, 418
  - .lft abbreviation • 418
  - .tab (Report-Writer statement) versus • 418
  - described • 418
  - example • 418
  - overview • 260
- .leftmargin (Report-Writer statement) • 258, 382
  - .lm abbreviation • 382
  - described • 382
  - example • 382
  - overview • 258
- .let (Report-Writer statement) • 261, 281, 438
  - described • 438
  - overview • 261
  - using with variables • 281
- .lineend (Report-Writer statement) • 260, 420
  - .lnend abbreviation • 420
  - described • 420
  - example • 420

- overview • 260
- .linestart (Report-Writer statement) • 260, 421
  - .linebegin synonym • 421
  - .lstart abbreviation • 421
  - described • 421
  - examples • 421
  - overview • 260
- .longremark (Report-Writer statement) • 257, 268, 361, 721
  - .lrem abbreviation • 361
  - described • 361
  - example • 361, 721
  - overview • 257, 268
- .name (Report-Writer statement) • 257, 268, 362
  - .nam abbreviation • 362
  - described • 362
  - example • 362
  - overview • 257, 268
- .need (Report-Writer statement) • 258, 274, 384, 746
  - .ne abbreviation • 384
  - described • 384
  - example • 384, 746
  - overview • 258
  - using with pagination • 274
- .newline (Report-Writer statement) • 260, 399, 422, 718, 735, 737
  - .nl abbreviation • 422
  - default limit • 399
  - described • 422
  - example • 422, 718, 735, 737
  - overview • 260
- .newpage (Report-Writer statement) • 258, 274, 386, 721
  - .np abbreviation • 386
  - described • 386
  - example • 386, 721
  - overview • 258
  - using with pagination • 274
- .nofirstff (Report-Writer statement) • 387, 757
  - described • 387
  - troubleshooting • 757
- .noformfeeds (Report-Writer statement) • 258, 380, 757
  - .noff abbreviation • 380
  - .noffs abbreviation • 380
  - described • 380
  - example • 380

- 
- overview • 258
  - troubleshooting • 757
  - .nounderline (Report-Writer statement) • 261, 435
    - .nou abbreviation • 435
    - described • 435
    - example • 435
    - overview • 261
  - .nullstring (Report-Writer statement) • 261, 429
    - .nullstr abbreviation • 429
    - described • 429
    - examples • 429
    - overview • 261
  - .output (Report-Writer statement) • 257, 268, 363
    - described • 363
    - example • 363
    - overview • 257, 268
  - .pagelength (Report-Writer statement) • 258, 274, 388, 721
    - .pl abbreviation • 388
    - described • 388
    - example • 388, 721
    - overview • 258, 274
  - .pagewidth (Report-Writer statement) • 258, 274, 283, 390
    - defaults • 283
    - described • 390
    - example • 390
    - overview • 258, 274
  - .position (Report-Writer statement) • 259, 277, 398, 404
    - .pos abbreviation • 404
    - described • 404
    - example • 404
    - overview • 259, 277, 398
  - .print (Report-Writer statement) • 261, 430
    - described • 261, 430
    - example • 430
  - .println (Report-Writer statement) • 430, 718
    - .pln abbreviation • 430
    - .prln abbreviation • 430
    - described • 430
    - example • 430, 718
  - .query (Report-Writer statement) • 257, 268, 347, 364, 369, 708
    - .quel synonym • 364
    - column breaks • 347
    - described • 364
    - example • 364, 708
    - overview • 257, 268
    - QUEL queries • 369
  - .right (Report-Writer statement) • 258, 259, 260, 398, 424, 708
    - .rt abbreviation • 424
    - described • 424
    - example • 424, 708
    - overview • 258, 259, 260, 398
  - .rightmargin (Report-Writer statement) • 258, 392
    - described • 392
    - example • 392
    - overview • 258
  - .rw filename extension • 359
    - .include (Report-Writer statement) • 359
  - .setup (Report-Writer statement) • 257, 268, 374, 740
    - described • 374
    - example • 374, 740
    - overview • 257, 268
  - .shortremark (Report-Writer statement) • 257, 268, 377
    - .srem abbreviation • 377
    - described • 377
    - example • 377
    - overview • 257, 268
  - .sort (Report-Writer statement) • 257, 268, 378, 708, 721
    - .srt abbreviation • 378
    - described • 378
    - example • 378, 708, 721
    - overview • 257, 268
  - .tab (Report-Writer statement) • 259, 260, 283, 398, 418, 427
    - .left (Report-Writer statement) versus • 418
    - described • 427
    - example • 427
    - overview • 259, 260, 283, 398
  - .tformat (Report-Writer statement) • 259, 406, 721
    - described • 406
    - example • 406, 721
    - overview • 259
  - .then (clause) • 436
  - .top (Report-Writer statement) • 259, 399, 408, 718
    - .tp abbreviation • 408
-



---

- described • 408
- examples • 408, 718
- overview • 259, 399
- .ulcharacter (Report-Writer statement) • 261, 433
  - .ulc abbreviation • 433
  - described • 433
  - example • 433
  - overview • 261
- .underline (Report-Writer statement) • 261, 435
  - .u abbreviation • 435
  - .ul abbreviation • 435
  - described • 435
  - example • 435
  - overview • 261
- .width (Report-Writer statement) • 259, 277, 398, 409
  - described • 409
  - example • 409
  - overview • 259, 277, 398
- .within (Report-Writer statement) • 259, 411, 718
  - .wi abbreviation • 411
  - described • 411
  - examples • 411, 718
  - overview • 259

## /

- / (slash) • 34, 257, 351, 660
  - comment indicator (with asterisk) • 257, 351, 660
  - server type syntax • 34
- /\* (comment indicator) • 268
- report specification • 268

## ;

- ; (semicolon) • 349, 374
  - report cleanup and setup sections • 349, 374
  - statement separator • 349, 374

## ?

- ? (question mark) • 312, 538
  - wild card character • 312, 538

## [

- [ ] (square brackets) • 99, 312, 336, 538, 540, 584
  - comparisons in validation checks • 540
  - numeric templates • 336, 584
  - pattern matching • 99, 312, 538

## \

- \ (backslash) • 97, 240, 293, 328, 336, 364, 583, 584, 670
  - dereference character • 240
  - numeric templates • 336, 583, 584
  - string continuation character • 670
  - string literals • 293
  - text match indicator • 97, 364
  - time interval templates • 328
  - wild card characters • 97

## \_

- \_ (underscore) • 99, 212, 433, 435
  - .ulcharacter • 433
  - pattern matching • 99
  - reports • 212
  - underlining character • 435
- \_date (function) • 313
- \_time (function) • 313

## |

- | (vertical bar) • 325, 670
  - date templates • 325
  - separators • 670

## +

- + (plus sign) • 296, 310, 316, 336, 382, 386, 392, 415, 418, 424, 579, 584
  - addition • 310
  - arithmetic • 310
  - exponent • 296
  - justification with • 316, 382, 386, 392, 415, 418, 424, 579
  - numeric templates • 336, 584
  - unary • 296, 310
- ++b flag • 757
- report (command) • 757

---

## <

- < (less than) • 311
  - comparison operator • 311
- < > (angle brackets) • 336, 584
  - numeric templates • 336, 584
- <= (less than or equal to) • 311
  - comparison operator • 311

## =

- = (equal to) • 311
  - comparison operator • 311
- = (equals sign) • 102, 438
  - assignment operator • 438
  - QBF blank retrieves • 102

## >

- > (greater than) • 311
  - comparison operator • 311
- >= (greater than or equal to) • 311
  - comparison operator • 311

## 6

- 6 flag • 352, 378
  - report (command) • 352, 378

## A

- ABF • 30, 701
  - abf (command) • 701
  - calling • 701
  - starting • 30
- abs (function) • 313
- absolute dates/times • 297, 587, 592
- accessing databases • 26
  - control methods • 26
- AddDetail operation • 113
  - Update frame • 113
- adding • 70, 83, 181, 183, 185, 186, 489, 491, 492, 496, 506, 510, 515
  - blank lines to forms • 491
  - boxes and lines to forms • 492
  - columns to reports • 186
  - columns to tables • 70
  - components to reports • 181
  - duplicate fields to a form • 515
  - headers and footers to reports • 183

- records in QBF • 83
- report sections • 183
- simple fields to forms • 496
- table fields to forms • 506, 510
- trim to forms • 489
- trim to reports • 185
- aggregate functions • 56, 187, 190, 200, 208, 282, 304, 306, 307, 344
  - average (avg) • 306
  - breaks • 304, 307
  - count (cnt) • 306
  - Create an Aggregate pop-up • 190
  - cumulative • 304
  - Cumulative Aggregation pop-up • 190
  - data selection • 304
  - defined • 187
  - moving • 208
  - non-unique • 304, 307
  - reports • 282
  - Selecting an Aggregate pop-up • 190
  - simple • 304, 307
  - syntax • 306, 344
  - unique • 200, 304
  - views and • 56
- Aggregate operation • 181
  - Create submenu • 181
- aligning • 177
  - RBF report components • 177
- All (clause) • 411
- am/pm (with date data type) • 569
- anchor point • 495
  - boxes • 495
  - rotating • 495
  - vertical lines • 495
- and (logical operator) • 94, 313
- ANSI/ISO Entry SQL-92 compliant databases • 50
  - regular identifiers • 50
- any (aggregate function) • 187
  - reports • 187
- append • 83, 84
  - Append operation • 83
  - data • 84
- Append operation • 83
  - QBF Execution Phase Frame • 83
- appending • 79, 83, 84
  - Append operation • 83
  - in QBF • 83
  - QBF • 79

---

- with table- and simple-field formats • 84
- applications • 30, 626, 642
  - Ingres Menu operation • 30
  - mapping customized • 626, 642
- archiving • 221
  - report definitions • 221
- arithmetic • 310, 342
  - dates • 310
  - expressions • 342
  - operations • 310
  - operators • 310
- arrow keys • 534, 615
  - mapping • 615
  - scrolling with • 534
- asc (keyword) • 378
- ascending sort sequence • 103
  - retrieving information • 103
- atan (function) • 313
- attributes • 39, 503, 525, 534, 544, 682, 683
  - Color • 683
  - derived fields • 544
  - editing • 525, 534
  - field • 39, 503, 534
  - form • 525
  - video • 682
- Attributes operation • 497, 503
  - VIFRED • 497, 503
- autocommit • 349, 374
- automatic joins • 128
- average (aggregate function) • 187, 188, 306
  - unique • 188
  - using in reports • 187

## B

- Blank operation • 44, 89, 110, 555, 558
  - defined • 44
  - Retrieve frame • 89
  - Terminal Monitor frame • 555, 558
  - Update frame • 110
- blanks • 37, 110, 236, 240, 242, 263, 291, 321, 336, 339, 364, 369, 449, 568, 584, 700
  - blank form creation • 449
  - Blank operation • 110
  - cf format • 321
  - cj format • 321
  - comparisons containing • 568
  - flag separator • 700
  - inserting • 339
  - numeric templates • 336, 584
  - owner qualification • 37
  - preceding delimited identifiers • 291
  - separating Report-Writer statements • 263
  - system-level commands • 236, 240, 242, 364, 369
- blinking • 687
- Blinking attribute • 490, 528
  - Attributes for Field frame • 528
  - Box Attributes frame • 490
- block style reports • 254, 277
- blocks • 163, 399, 401, 408
  - .block/.endblock statements • 399
  - .top (Report-Writer statement) • 408
  - .top/.bottom statements • 401
  - block style report • 163
  - formatting • 399
  - printing • 399
- bold • 687
- boolean • 313, 315, 344, 436
  - .if (Report-Writer statement) • 436
  - break • 315
  - expressions • 313, 344
  - is null (operator) • 313
  - not (operator) • 313
  - not null (operator) • 313
  - or (operator) • 313
- Bottom operation • 653
  - predefined menu option • 653
- Box Field attribute • 528
  - Attributes for Field frame • 528
- Box/Line operation • 490, 492, 493, 494, 495
  - anchor point • 495
  - attribute boxes vs • 492
  - attributes • 490
  - compared to keystroke lines • 492
  - creating • 493
  - Edit operation • 494
  - enhancing • 494
  - expanding the box • 493
  - horizontal lines • 493
  - intersections • 494
  - plus sign • 493
  - rotating anchor point • 495
  - underscore character • 494
  - using • 493
  - vertical lines • 493
- boxes • 492, 688
  - boxing characters • 492
  - commands • 688

---

- break (function) • 315
- break Columns • 173
  - pop-up • 173
- BreakOptions operation • 203
- breaks • 159, 162, 164, 166, 173, 183, 203, 250, 265, 304, 307, 378, 384, 386, 397, 402, 746
  - .header (Report-Writer statement) • 397
  - .need (Report-Writer statement) • 384
  - .sort (Report-Writer statement) • 378
  - actions • 250
  - aggregate function • 304, 307
  - Break Columns pop-up • 173
  - break header • 265
  - defined • 203, 250
  - detail • 250, 265
  - formatted values and • 402
  - headers and footers • 166, 183
  - indented reports • 164, 173
  - page • 386, 746
  - reports • 159
  - sort columns • 183
  - start-of-report • 250
  - tabular reports • 162
- bright • 687
- Brightness Change attribute • 490, 528
  - Attributes for Field frame • 528
  - Box Attributes frame • 490
- buffers • 116
  - updates • 116
- byte (data type) • 298, 563, 697
- byte varying (data type) • 298, 563, 697

## C

- C (data type) • 298, 313, 317, 563, 566, 568, 572, 697
  - conversion function • 313
  - data types for report columns • 298
  - default report column format • 317
  - described • 563, 566, 568
  - storage format • 572
  - storage format • 697
- c (function) • 313
- C format • 320, 321
- call (statement) • 701
  - restrictions on parameters • 701
- calling • 73, 74, 79, 446, 700, 706
  - call (statement) • 700
  - operating system • 706
  - QBF • 73, 74, 79
  - VIFRED • 446
- Cancel operation • 44, 136, 218, 511, 653
  - defined • 44
  - Form Layout frame • 511
  - Join Specification frame • 136
  - predefined menu option • 653
  - Save Report frame • 218
- case • 50, 313, 566
  - character strings • 566
  - names • 50
  - uppercase (function) • 313
- catalog frames • 75, 79, 132, 446, 447
  - Forms Catalog Frame • 447
  - JoinDefs • 132
  - QBF • 75, 79
  - VIFRED • 446
- Center operation • 208
  - Move submenu • 208
- centering • 260, 316, 415
  - .center (Report-Writer statement) • 415
  - reports • 316
  - text • 260, 415
- ChangeDisplay operation • 143
- char (data type) • 298, 313, 317, 563, 566, 697
  - conversion function • 313
  - data types for report columns • 298
  - default report column format • 317
  - described • 563, 566
  - storage format • 697
- char (function) • 313
- character data • 572
  - storage format • 572
- character fields • 534
  - display function • 534
- character strings • 301, 313, 321, 341
  - C format • 321
  - concat (function) • 313
  - left (function) • 313
  - length (function) • 313
  - locate (function) • 313
  - lowercase (function) • 313
  - right (function) • 313
  - shift (function) • 313
  - size (function) • 313
  - squeeze (function) • 313
  - T format • 341
  - variables • 301

---

- charextract (function) • 313
- Choose a Report Style • 228
  - pop-up • 228
- clanks • 320
  - B format • 320
  - inserting • 320
- clear (FRS command) • 645
- clearing • 44, 558
  - screen • 44, 558
- clearrest (FRS command) • 645
- coercibility • 122
  - data types • 122
- Color attribute • 490, 683
  - Box Attributes frame • 490
- colors (in fields) • 533
- colors (in forms) • 691
  - termcap descriptions • 691
- colors (in graphs) • 691
  - termcap descriptions • 691
- Column operation • 181, 208
  - Create submenu • 181
  - Move submenu • 208
- column style reports • 254
- ColumnOptions frame • 198
- ColumnOptions operation • 179
  - Report Layout frame • 179
- columns (in reports) • 186, 194, 195, 203, 208, 259, 272, 277, 279, 283, 284, 288, 289, 298, 303, 315, 319, 347, 357, 364, 369, 378, 398, 399, 402, 404, 406, 409, 411, 415, 418, 422, 424, 430
  - breaks • 315, 347, 378
  - Create a Column pop-up • 186
  - creating • 186
  - data types • 298
  - defaults • 283, 319
  - deleting • 194
  - delimited identifiers • 288, 289, 298, 357
  - editing • 195
  - format • 259, 398
  - headings • 195, 399
  - justification • 415, 418, 424
  - line advancing • 422
  - margins • 411
  - moving • 208
  - position\_number variable • 303
  - positioning • 277, 283, 404
  - print formats • 406
  - printing • 430
  - referencing • 298, 364, 369
  - sorting • 203, 272
  - temporary formats • 279, 402, 406
  - w\_column variable • 303, 411
  - w\_name variable • 303, 411
  - width • 284, 404, 409, 411
- columns (in tables) • 51, 56, 61, 62, 63, 65, 66, 70, 121, 136, 137, 139, 143, 510, 676
  - adding • 70
  - copying specifications for • 66
  - defaults • 63
  - defined • 56
  - deleting • 70, 143
  - delimited identifiers • 51
  - hiding • 143
  - join • 121, 136, 137
  - key numbers • 62, 676
  - maximum number • 56
  - moving • 66
  - naming • 56, 61, 510
  - protecting • 139
  - unique • 65
  - unique keys • 65
- comment blocks • 222
  - archived reports • 222
- comments • 257, 268, 351, 620, 630, 660, 756
  - debugging • 756
  - delimiters • 268, 756
  - mapping files • 620, 630, 660
  - nesting • 351, 756
  - report specification • 257, 351
- comparison operators • 92, 94, 311, 536, 537
  - and • 94
  - defined • 311
  - list • 311
  - QBF • 92
  - validation check • 536, 537
- compiled forms • 476
  - VIFRED • 476
- compiling • 476
  - Embedded SQL • 476
  - operations • 476
- Complete operation • 555, 558
  - Terminal Monitor frame • 555, 558
- computation • 296, 304, 307, 310
  - aggregation • 304
  - breaks • 307
  - date expressions • 310

---

---

- exponential notation • 296
- reports • 310
- concat (function) • 313
- conditional statements • 281, 313, 436
  - .if (Report-Writer statement) • 436
  - boolean/logical functions • 313
  - overview • 281, 436
- constants • 291, 293, 296, 297, 353, 429, 438
  - .declare (Report-Writer statement) • 353
  - .let (Report-Writer statement) • 438
  - .nullstring (Report-Writer statement) • 429
  - date • 297
  - numeric • 296
  - reports • 293
  - string • 291, 293
- control characters • 339
  - Q0 format • 339
- control keys • 644
  - mapping • 644
- conventions • 22
  - query languages • 22
  - system-level commands • 22
- conversion • 313, 342, 343, 759
  - dates • 313
  - functions • 313, 342, 343, 759
  - numeric data • 313
  - string/character data • 313
- copyform (command) • 449
  - duplicating forms • 449
- copying • 66, 449, 478
  - column specifications in tables • 66
  - copyform (command) • 478
  - forms • 449, 478
- correlation names • 135, 288, 289
- cos (function) • 313
- count (aggregate function) • 187, 188, 306
- CR abbreviation for credit • 336, 584
- Create a Column pop-up • 186
- create an Aggregate pop-up • 190
- create index (statement) • 56
  - described • 56
- Create operation • 59, 60, 132, 140, 152, 179, 447, 449, 457, 487, 488, 489, 497, 534
  - Form Layout frame • 457, 487
  - forms • 449
  - Forms Catalog frame • 447, 449
  - JoinDefs Catalog frame • 132, 140
  - Report Catalog frame • 152
  - Report Layout frame • 179

- scrollable fields • 534
- Table Utility frame • 59, 60
- VIFRED • 488
- VIFRED forms components • 489, 497
- creating • 60, 169, 173, 183, 186, 248, 474
  - columns in reports • 186
  - creating a Report pop-up • 169
  - forms • 474
  - report sections • 183
  - reports • 173, 248
  - tables • 60
- Creating a Report Layout Section pop-up • 183
- creating a Report pop-up • 169
- cumulative (keyword) • 306
  - aggregate functions • 306
  - cum abbreviation • 306
- cumulative Aggregation pop-up • 190
- currency formats • 571
- current\_date (report variable) • 303
- current\_day (report variable) • 303
- current\_time (report variable) • 303
- cursor • 46, 681, 693
  - activating on terminals • 693
  - moving • 46, 681, 693

## D

- d display format parameter • 330, 331, 333, 335
  - numeric print format E • 330
  - numeric print format F • 331
  - numeric print format G • 333
  - numeric print format N • 335
- d flag • 349, 374
  - report (command) • 349, 374
- D format • 324
- data • 84, 86, 87, 105, 110, 112, 115, 158, 201, 227, 293, 316, 364, 504, 553, 572
  - .query • 364
  - appending • 84
  - constants • 293
  - defining • 553
  - deleting • 112, 115
  - display • 105
  - display format symbols • 504
  - display formats • 572
  - editing • 112
  - entry errors • 86
  - expressions • 293
  - formatting • 316

---

- manipulating • 553
- retrieving • 87
- runtime selection • 201, 227
- sources for reports • 158
- updating • 110
- data display format • 345, 504, 534, 572, 574, 579, 580, 581
  - case • 574
  - data types vs • 572
  - date templates • 581
  - dates • 581
  - f flag • 579
  - floating point numbers • 580
  - incompatible data types • 504
  - integers • 580
  - j flag • 579
  - justification of character data • 579
  - numeric templates • 580
  - scrollable fields • 534
  - symbols • 572
  - syntax summary • 345
- data integrity • 65
  - unique keys • 65
- data type attribute • 531
- data types • 61, 116, 122, 293, 296, 297, 298, 317, 319, 352, 364, 369, 486, 504, 510, 532, 563, 566, 567, 568, 569, 570, 571, 572, 579, 580, 581, 697
  - abstract • 298
  - byte • 298, 563, 697
  - byte varying • 298, 563
  - c • 298, 317
  - c (data type) • 563, 566, 568, 697
  - changing in fields • 532
  - char • 298, 317, 563, 566, 567, 568, 697
  - character • 298, 563, 566, 572
  - coercible • 122
  - date • 297, 298, 317, 563, 569, 572, 581, 697
  - decimal • 296, 298, 317, 563, 571, 697
  - described • 563
  - display formats vs • 486, 510, 572
  - displaying formats • 504, 572
  - entering • 61
  - float • 296, 298, 317, 571
  - float4 • 298, 317, 563, 697
  - float8 • 298, 317, 563, 697
  - floating point • 570, 572, 580, 697
  - integer • 296, 563, 571, 572, 580
  - integer1 • 298, 317, 563, 697
  - integer2 • 298, 317, 563, 697
  - integer4 • 298, 317, 563, 697
  - integrity • 116
  - long byte • 298, 563, 697
  - long varchar • 298, 352, 369, 563, 697
  - long varying • 697
  - money • 298, 317, 563, 571, 572, 580, 697
  - numeric • 296, 298, 563
  - OpenSQL • 697
  - precision • 563
  - QUEL • 319, 697
  - reports • 293, 298, 572
  - smallint • 563
  - SQL • 563, 697
  - string • 293
  - text • 298, 317, 563, 566, 568, 572, 579, 697
  - unsupported • 352, 364, 369
  - user-defined type (UDT) • 563, 697
  - varchar • 298, 317, 563, 566, 568, 697
- data windows • 39, 574
  - defined • 39
  - specifying size • 574
- databases • 26, 32, 33, 34, 50, 119, 287
  - \_SQL-92 compliant • 287
  - accessing • 26, 32
  - accessing non-Ingres • 26, 32, 34
  - ANSI/ISO Entry SQL-92 compliant • 50
  - distributed access • 26, 32, 34
  - Enterprise Access products • 26, 32, 34
  - Ingres Star access • 32
  - local access • 26, 32
  - location • 26, 32
  - relational • 119
  - remote access • 26, 32, 33, 34
  - Star access • 26
  - Star Server access • 34
  - syntax for access • 32
- date (data type) • 298, 317, 563, 569, 572, 697
- date (function) • 313, 570
- date format • 313, 324, 570
  - conversion function • 313, 570
- date\_gmt (function) • 313
- date\_part (function) • 313
- date\_trunc (function) • 313

---

dates • 160, 193, 214, 297, 303, 310, 313, 324, 328, 344, 535, 563, 569, 570, 572, 581, 587, 588, 592  
absolute • 297, 587, 592  
arithmetic operations • 310  
constants • 297  
converting • 570  
current • 535  
current\_date variable • 303  
current\_day variable • 303  
data type • 563, 569, 572, 581, 587  
date (function) • 313  
date\_gmt (function) • 313  
date\_part (function) • 313  
date\_trunc (function) • 313  
display formats • 581  
formats • 324, 587  
functions • 344  
interval (function) • 328, 570  
interval function • 592  
reports • 160, 193, 214  
storing • 297  
templates • 324, 587, 588  
DB abbreviation for debit • 336, 584  
Dbname parameter • 33  
Ddtes • 297  
intervals • 297  
deadlock • 86, 117  
causes • 86, 117  
decimal (data type) • 296, 298, 317, 563, 571, 697  
decimals • 330, 331, 333, 335, 336  
E format • 330  
F format • 331  
G format • 333  
N format • 335  
Numeric templates • 336  
printing standard notation • 331, 333, 335  
declared variables • 353  
runtime • 353  
Declared Variables frame • 227  
default forms • 449  
default report format • 254  
block style • 254  
default values • 531  
Field attribute • 531  
DefaultOrder operation • 517  
defaults • 63, 128, 141, 254, 259, 277, 282, 283, 284, 317, 318, 349, 374, 398, 402, 406, 535, 633, 635, 649  
autocommit • 349, 374  
column • 63, 283  
column format • 284  
column position • 277, 283  
column width • 284  
formats • 282  
joins • 128  
mapping files • 633  
margins • 283, 318  
menu item mapping • 649  
pagewidth • 283  
print format • 317, 402, 406  
reports • 254, 259, 282, 317, 398  
rules • 141  
terminal-type mapping • 635  
VIFRED • 535  
Define term\_ingres • 608  
Delete operation • 179, 194, 457, 487, 516  
Forms Layout frame • 457, 487, 516  
report components • 179, 194  
Report Layout frame • 179  
deletechar (FRS command) • 645  
DeleteLine operation • 555  
Terminal Monitor frame • 555  
deleting • 70, 141, 144, 152, 192, 193, 194  
columns • 70, 194  
Delete operation • 141, 144  
Edit Report Layout pop-up • 193  
report components • 192  
report sections • 193  
reports • 152  
delimited identifiers • 51, 288, 289, 291, 292, 298, 357, 358, 364  
distinguishing from format templates • 291  
distinguishing from strings • 291, 364  
enabling • 357  
object ownership • 289  
QUEL and • 288, 358  
reserved words • 292  
special characters in • 51  
using • 51, 298  
delimiters • 263, 268, 289, 293  
comment • 268  
delimited identifiers • 263, 289  
Report-Writer statement • 263  
string literal • 293

---



---

- derivation formulas • 531, 546
  - described • 531, 546
- Derived attribute • 531
- derived fields • 544
  - behavior • 544
- desc (keyword) • 378
- desc sort sequence • 103
  - retrieving information • 103
- destinations • 225, 228, 229
  - reports • 225
  - Select a Destination pop-up • 228, 229
- Destroy operation • 59, 132, 152, 447
  - Forms Catalog frame • 447
  - JoinDefs Catalog frame • 132
  - Report Catalog frame • 152
  - Table Utility frame • 59
- destroying • 67, 147
  - Destroy operation • 147
  - indexes • 67
  - synonyms • 67
  - tables • 67
  - views • 67
- detail • 250, 251, 265, 283
  - break • 250, 265, 283
  - instructions • 251
  - sections • 251, 265
- direction • 693
- display formats • 196, 197, 316, 317, 320, 321, 324, 330, 331, 333, 335, 336, 339, 341, 402, 406, 500, 509, 572, 574
  - B format • 320
  - C format • 321
  - column • 317, 509
  - data types vs • 316, 572
  - date templates • 324
  - differences • 316
  - DisplayFormat operation • 500
  - E format (numeric) • 330
  - editing • 197
  - F format (numeric) • 331
  - floating point numbers • 330
  - G format (numeric) • 333
  - justification of character data • 321
  - N format (numeric) • 335
  - numeric template • 336
  - parameters • 574
  - Q0 format • 339
  - RBF • 196, 572
  - T format • 341, 402, 406

- Display only attribute • 528
- DisplayFormat operation • 197, 497
  - RBF • 197
  - VIFRED • 497
- displaying • 245
  - reports • 245
- DisplayOnly • 534
  - scrollable fields • 534
- distributed databases • 26, 32, 34
  - accessing • 26, 34
- dow (function) • 313
- downline (FRS command) • 645
- duplicate (FRS command) • 645
- Duplicate operation • 449, 451, 515
  - Creating a Form frame • 449, 451, 515
- duplicates • 65, 203, 352, 515
  - fields • 515
  - removing • 352
  - suppressing in reports • 203
  - suppressing in tables • 65
  - table rows • 352

## E

- E format • 296, 330
  - exponential notation • 296
  - overview • 330
- Edit operation • 132, 152, 179, 194, 195, 447, 457, 487, 489, 490, 495, 511, 517, 555
  - described • 194, 195
  - Form Layout frame • 457, 487, 511
  - Forms Catalog frame • 447
  - JoinDefs Catalog frame • 132
  - Order frame • 517
  - Report Catalog frame • 152
  - Report Layout frame • 179
  - Terminal Monitor frame • 555
  - VIFRED forms components • 489, 490, 495
- EditAttr operation • 513, 525
  - Table Field menu • 513, 525
- editing • 46, 146, 195, 216, 443, 475, 489, 491, 511, 525
  - attributes • 525
  - canceling edits • 216
  - columns in reports • 195
  - Edit operation • 195
  - forms • 443, 475, 489, 491
  - insert and overstrike modes • 46
  - JoinDefs • 146
  - report formats • 195

---

- table fields • 511
- trim • 195, 489
- Editor (FRS command) • 645
- End operation • 44, 653
  - defined • 44
  - predefined menu option • 653
- Enter key • 45
- Enterprise Access products • 21, 26, 34
  - accessing databases • 26, 34
  - product differences • 21
- environment variables/logicals • 26, 107, 296, 429, 571, 587, 605, 755
  - II\_DATE\_FORMAT • 587
  - II\_EMBED\_SET • 755
  - II\_NULL\_STRING • 429
  - II\_NUMERIC\_LITERAL • 296, 571
  - II\_SCROLL\_MSG • 107
  - setting • 26
  - TERM\_INGRES • 605
- Equijoin • 124
- error messages • 49, 116, 349, 561
  - checking • 49
  - Save operation • 116
  - suppressing • 349
  - Terminal Monitor • 561
- errors • 226, 349, 374
  - d flag • 349, 374
  - handling • 349, 374
  - report error log • 226
  - runtime processing • 349, 374
- escape sequences • 339
  - Q0 format • 339
- Examine operation • 59, 68
  - obtaining information on tables • 68
  - Table Utility frame • 59, 68
- examining • 68
  - tables • 68
- examples • 707
  - reports • 707
- exp (function) • 313
- Expand operation • 210, 465, 466
  - described • 465
  - Report Margin submenu operation • 210
  - VIFRED • 466
- expert mode • 446
  - pattern matching • 446
- exponential notation • 296, 310, 333, 335, 570, 572, 580
  - G format • 333

- N format • 335
- expressions • 293, 298, 311, 316, 342, 343, 344, 430
  - abstract • 298
  - arithmetic • 342
  - boolean • 344
  - character • 298
  - columns • 298
  - conditional • 311
  - data type resolution • 342
  - date • 344
  - format specifications • 316
  - numeric • 298, 343
  - printing in reports • 430
  - QUEL conversion functions • 343
  - report variables • 342
  - string • 343
  - string constants • 293
  - types of data • 293

## F

- f flag • 363
  - report (command) • 363
- F format • 331
- Field Attribute menu • 525
- field attributes • 523, 524, 525, 528, 529, 530, 531, 532, 533, 535
  - Attributes frame • 525
  - blinking • 528
  - box • 528
  - Box/Line vs • 529
  - brightness change • 528
  - color • 531, 533
  - columns in table fields • 524
  - data type • 531
  - default value • 531
  - defaults • 524
  - derivation formula • 531
  - derived field • 531
  - display only • 528
  - finishing • 525
  - force lower case • 528
  - force upper case • 528
  - inputmasking • 528, 530
  - internal field name • 531
  - internal name • 532
  - invisible • 528
  - keep previous value • 528
  - list • 528, 531

---

- no auto tab • 528
- no echo • 528
- now value • 535
- nullable • 531
- nullable data types • 533
- query only • 528
- reverse video • 528
- scroll size • 531
- scrollable • 531
- setting • 523
- setting default values • 535
- simple fields • 525
- table fields • 525
- today value • 535
- underline • 528
- validation check • 531
- validation error message • 531
- Field operation • 497, 499
  - VIFRED • 497, 499
- fields • 38, 39, 52, 102, 103, 128, 485, 496, 497, 499, 500, 502, 503, 505, 506, 515, 517, 528, 531, 532, 534, 535, 544, 574
  - attributes • 39, 503
  - blank • 102
  - changing display • 534
  - components • 497
  - creating • 497
  - data display format • 500
  - data window • 39, 500, 503, 505
  - default QBFName • 496
  - default values • 531, 535
  - derived • 544
  - described • 38, 485
  - duplicate • 515
  - editing • 503
  - enclosing in boxes • 528
  - form components • 485
  - internal names • 39, 497, 499, 532
  - JoinDef displays • 128
  - justification • 502
  - mandatory • 528
  - names vs titles • 499
  - order • 517
  - prioritizing in retrievals • 103
  - scrollable • 534
  - simple • 39, 497
  - simple in VIFRED • 503
  - tabbing order • 517
  - table fields • 39, 506
  - titles • 39, 499
  - validation • 52
  - width • 574
- File operation • 555, 556, 557, 558, 561
  - input frame • 555
  - Terminal Monitor input frame • 556, 557
  - Terminal Monitor output frame • 558, 561
- filename extensions • 359
  - .rw • 359
- FilePartial operation • 231
  - filing reports • 231
- files • 359
  - including external • 359
- Find operation • 653
  - predefined menu option • 653
- firstff flag • 757
  - report (command) • 757
- float (data type) • 296, 298, 317, 563, 571
  - default column format • 317
  - described • 571
  - numeric constants • 296, 298
  - precision • 317, 563
- float4 (data type) • 298, 317, 563
- float4 (function) • 313
- float8 (data type) • 298, 317, 563, 697
- float8 (function) • 313
- floating point • 330, 570, 572, 580, 697
  - data type • 570, 572, 580, 697
  - display format • 330, 572, 580
- footers • 166, 193, 251, 265, 397
  - .header (Report-Writer statement) • 397
  - break • 251, 265
  - columns • 251
  - deleting • 193
  - reports • 166, 251
- Force Lower Case attribute • 528
- Force Upper Case attribute • 528
- form feed (command) • 213
- Format operation • 521
- format specifications • 320, 321, 324, 330, 331, 333, 335, 336, 339, 341
  - B format (blank) • 320
  - C format (character) • 321
  - D format (date) • 324
  - E format (numeric) • 330
  - F format (numeric) • 331
  - G format (numeric) • 333
  - N format (numeric) • 335
  - numeric templates • 336

---

- Q0 format (control characters) • 339
- T format (character string) • 341
- format templates • 291
  - distinguishing from delimited identifiers • 291
- formats • 214, 279, 282, 284, 316, 317, 399, 402, 406, 500, 510, 521
  - .format (Report-Writer statement) • 284, 402
  - .tformat (Report-Writer statement) • 406
  - block • 399
  - column • 279, 284, 317, 402, 406
  - defaults • 282, 317
  - display • 500, 510
  - expressions • 316
  - Format operation • 521
  - overriding • 402, 406
  - report data • 316
  - report date and time stamp • 214
  - report page numbers • 214
- FormAttr operation • 457, 458, 487
  - attributes • 458
  - Form Layout frame • 457, 487
  - pop-up location • 458
  - pop-up position • 458
  - pop-up vs fullscreen • 458
- forms • 38, 77, 81, 443, 447, 449, 450, 451, 453, 459, 472, 474, 475, 476, 478, 479, 480, 482, 485, 516, 519, 520, 521
  - assigning QBFNames • 482
  - centering components • 520
  - compiling definitions as C source files • 476
  - components • 485
  - copying • 449, 478
  - creating • 474
  - creating blank • 449, 450
  - creating from existing forms • 449
  - creating from multiple tables • 453
  - default • 81, 449
  - default forms • 451
  - deleting • 475
  - deleting lines/components • 516
  - described • 38
  - display mode • 451
  - editing • 443, 475
  - fields • 38, 485
  - justifying components • 520
  - linking to JoinDefs • 479
  - linking to tables • 479

- moving components • 519, 521
- moving parts of a field • 521
- ownership • 449
- pop-up display • 459
- printing • 478
- QBF • 77
- renaming • 447, 480
- saving • 472
- trim • 38, 485
- forms (Ingres Menu operation) • 30
- Forms Catalog frame • 449
- Forms in Database • 515
  - pop-up • 515
- frames • 38, 175
  - described • 38
  - reports • 175
- from (clause) • 364
- FRS • 615, 644, 651, 692
  - commands • 644, 692
  - defined • 615
  - FRS keys • 651
  - key definition • 615
- FRS commands • 558
  - top • 558
- FullPrompt operation • 227
  - specifying variables • 227
- fullscreen form • 459
  - Pop-up form vs • 459
- function keys • 45, 615, 689
  - activating on terminals • 689
  - defined • 45, 615
- functions • 304, 313, 342, 343, 344, 570, 759
  - aggregate • 304
  - boolean • 344
  - built-in • 313
  - conversion • 313, 342, 343, 759
  - date • 313, 344, 570
  - interval • 570
  - numeric • 313, 343
  - string • 313, 343

## G

- G format • 333
- GetTableDef operation • 66, 136, 138, 509
  - copying column specifications in tables • 66
  - Join Specification frame • 136, 138
  - VIFRED • 509
- Go operation • 44, 89, 132, 133, 145, 152, 154, 555, 558, 653

---

- defined • 44
- JoinDef Catalog frame • 132
- JoinDef Definition frame • 133, 145
- predefined menu option • 653
- Report Catalog frame • 152
- Report MoreInfo frame • 154
- Retrieve frame • 89
- Terminal Monitor frame • 555, 558
- graphics • 688
  - boxes • 688
- graphs • 30
  - Ingres Menu operation • 30
- group by (clause) • 364, 746

## H

- having (clause) • 364
- headers • 166, 193, 213
  - deleting • 193
  - page header • 213
  - reports • 166
- Heading operation • 181
  - Create submenu • 181
- headings • 178, 191, 194, 195, 208, 251, 265
  - break header • 251, 265
  - columns • 251
  - creating additional lines • 191
  - deleting • 194
  - editing • 195
  - moving • 208
  - page header • 265
  - reports • 178, 251, 265
- Help (statement) • 52
  - delimited identifiers • 52
- Help operation • 44, 47, 555, 653
  - defined • 44
  - keys option • 47
  - menu option • 47
  - Terminal Monitor frame • 555
- Help screens • 47
- hex (function) • 313
- hexadecimal • 295, 339
  - constants • 295, 339
- hierarchical control break • 164
  - report style • 164

## I

- i flag • 362
  - report (command) • 362

- II\_DATE\_FORMAT • 587
- II\_DECIMAL • 570
- II\_EMBED\_SET • 755
  - printqry • 755
- II\_MONEY\_PREC • 571
- II\_NULL\_STRING • 429
- II\_NUMERIC\_LITERAL • 296, 571
- II\_PATTERN\_MATCH • 99
- II\_PRINTSCREEN\_FILE • 48
- II\_SCROLL\_MSG • 107, 108
- II\_TERMCAP\_FILE • 669
- II\_TIMEZONE • 569
- indented report style • 164
- indexes • 56, 67, 68
  - defined • 56
  - destroying • 67
  - examining • 68
- ingmenu (command) • 701
- Ingres Menu • 28, 30, 32, 129, 446, 554, 701
  - Applications operation • 30
    - calling • 701
    - capabilities • 28
    - Forms operation • 30, 446
    - forms-based tools access • 28
    - Graphs operation • 30
    - ingmenu (command) • 701
    - JoinDefs operation • 30, 129
    - menu maps • 30
    - Queries operation • 30, 554
    - Reports operation • 30
    - starting from command line • 32
    - Tables operation • 30
    - using • 28
- Ingres Star • 32
  - accessing distributed database • 32
- Ingres tools • 32
  - starting from command line • 32
- IngresMenu • 58
  - Tables operation • 58
- initializing • 270, 301, 353
  - variables • 270, 301, 353
- input masking • 530, 582, 583
  - numeric templates • 583
  - setting on or off • 530
  - turning on/off • 582
- Inputmasking attribute • 528, 530
- Inquire\_forms (statement) • 643
- insert editing mode • 46, 489
- Insert operation • 44, 85, 512

- defined • 44
- Table Field menu • 85, 512
- inserting • 44
  - blank lines • 44
- InsertLine operation • 555
  - Terminal Monitor frame • 555
- int1 (function) • 313
- int2 (function) • 313
- int4 (function) • 313
- integer (data type) • 296, 563, 571, 572, 580, 697
- integer1 (data type) • 298, 317, 563, 697
- integer2 (data type) • 298, 317, 563, 697
- integer4 (data type) • 298, 317, 563, 697
- integers • 580
  - display format • 580
- integrity • 86
  - constraints • 86
- internal field name • 39, 506, 510, 532
  - columns • 510
  - field attribute • 39, 532
  - table fields • 506
- internal field name attribute • 531
- interval (function) • 313, 570
- invisible • 530
  - field attributes • 530
- Invisible attribute • 528
- IQUEL • 554
  - entering statements • 554
  - invoking • 554
- is not null (comparison operator) • 313
- is null (comparison operator) • 313
- ISO Entry SQL 92 compliant databases • 50, 52
  - delimited identifiers • 52
  - regular identifiers • 50
- ISQL • 554, 701
  - calling • 701
  - invoking • 554
  - isql (command) • 701

## J

- J display format parameter • 321
  - character string print format C • 321
- JoinDef • 51, 72, 105, 107, 112, 116, 119, 121, 124, 127, 128, 129, 132, 133, 134, 136, 137, 139, 140, 141, 143, 144, 145, 146, 147, 158, 449
  - basis for form • 449

- catalog • 132
- Change Display frame • 144, 145
- copying to/from text files • 146
- creating • 129
- data source for reports • 158
- defined • 119
- Definition frame • 133
- delimited identifiers • 51
- destroying • 147
- display • 143, 144
- editing • 137
- editing forms • 146
- fields in display • 128
- join columns • 136
- Join Specification frame • 136
- joins (command) • 129
- limitations on sort priority • 105
- Master/Detail • 124, 127
- naming • 134
- overview • 119
- query target • 72
- rules • 116, 121
- saving • 145
- single table • 139
- testing • 145
- Update and Delete Rules frame • 140, 141
- updating • 112, 116
- viewing sort results • 107
- JoinDef Definition frame • 134, 139
- JoinDefs Catalog frame • 132
- JoinDefs operation • 30, 79, 129
  - Ingres Menu • 30, 129
  - QBF Start-up frame • 79
- joining • 119, 139, 140, 741
  - multiple tables • 741
  - table to itself • 119, 139, 140
- JoinLines operation • 555
  - Terminal Monitor frame • 555
- joins • 124, 128, 137, 139
  - defaults • 128
  - equi-joins • 124
  - natural • 128
  - outer joins • 124
  - types • 124
  - validation checks • 139
  - viewing columns • 137
- Joins operation • 133, 139
  - JoinDef Definition frame • 133, 139
- justification • 316, 418, 424, 579

---

- fields • 579
- reports • 316, 418, 424

## K

- Keep Previous Value attribute • 528
- key numbers • 62
- keyboard keys • 42, 45, 46, 534, 607, 691
  - arrow • 534, 691
  - cursor movement with • 46
  - Enter • 45
  - help list • 45
  - mapping • 45
  - Menu • 42
  - Mode • 46
  - Tab • 45
  - terminal considerations • 607
  - VT100 mappings • 607
- keys • 47, 65
  - help list • 47
  - unique • 65
- Keys operation • 47
- keywords • 292, 364
  - all • 364
  - distinct • 364
  - Report-Writer • 292

## L

- l flag • 390
  - report (command) • 390
- labels • 165, 630, 649, 737
  - mapping files • 630, 649
  - report example • 737
  - report style • 165
  - restrictions on customizing • 165
- LastQuery operation • 89, 110
  - Retrieve frame • 89
  - Update frame • 110
- Layout operation • 179, 181, 183, 192, 193
  - adding report sections • 181, 183
  - deleting report sections • 192, 193
  - Report Layout frame • 179
- layouts • 167, 183
  - Creating Report Layout Section pop-up • 183
  - default report • 167
- left (function) • 313
- Left operation • 208
  - Move submenu • 208

- left\_margin (report variable) • 303
- leftchar (FRS command) • 645
- length (function) • 313
- limits • 701
  - call (statement) parameters • 701
- Line operation • 181
  - submenu • 181
- line\_number (report variable) • 303
- LineEdit operation • 555
  - Terminal Monitor frame • 555
- lines • 44, 192, 399, 415, 418, 420, 421, 422, 424, 427, 430, 492, 506, 574
  - .center (Report-Writer statement) • 415
  - .left (Report-Writer statement) • 418
  - .lineend (Report-Writer statement) • 420
  - .linestart (Report-Writer statement) • 421
  - .newline (Report-Writer statement) • 399, 422
  - .right (Report-Writer statement) • 424
  - .tab (Report-Writer statement) • 427
  - adding to reports • 192
  - blank • 44
  - Box/Line operations • 492
  - displaying between rows • 506
  - maximum in report block • 399, 430
  - size • 574
  - text positioning • 415
  - wrapping • 430
- ListChoices operation • 44, 89, 110, 133, 136, 653
  - defined • 44
  - JoinDef Definition frame • 133
  - JoinDef Specification frame • 136
  - predefined menu option • 653
  - Retrieve frame • 89
  - Update frame • 110
- ListForms operation • 515
- listing • 44
  - choices for fields • 44
- local databases • 32
  - accessing/terminating access to • 32
- locate (function) • 313
- Location operation • 456, 457, 487
  - Form Layout frame • 457, 487
  - VIFRED • 456
- log (function) • 313
- logical operators • 94, 313
  - and • 94
  - or • 94

---

- logicals • 26, 107, 108, 296, 429, 571, 605, 755
  - II\_EMBED\_SET • 755
  - II\_NULL\_STRING • 429
  - II\_NUMERIC\_LITERAL • 296, 571
  - II\_SCROLL\_MSG • 107, 108
  - setting • 26
  - TERM\_INGRES • 605
- long byte (data type) • 298, 563, 697
- long varchar (data type) • 352, 369, 563, 697
- lowercase (function) • 313

## M

- Mandatory Field attribute • 528
  - Attributes for Field frame • 528
- mapping • 615, 629, 630, 643, 644, 649, 651, 654, 655, 659, 660
  - described • 615
  - disabling • 630, 659
  - file errors • 660
  - files • 629, 660
  - FRS commands • 644
  - FRS keys • 629, 651
  - getting more information about • 643
  - menu items • 649
  - querying settings • 643
  - statements • 655
  - syntax of statements • 654, 660
- margins • 210, 275, 283, 303, 318, 382, 392, 411, 465
  - .leftmargin (Report-Writer statement) • 382
  - .rightmargin (Report-Writer statement) • 392
  - defaults • 283, 318, 382
  - expanding • 210
  - forms • 465
  - left\_margin variable • 303
  - page footer • 275
  - page header • 275
  - reports • 210, 275
  - right\_margin variable • 303
  - temporary • 411
- Master/Detail • 164
  - report style • 164
- Master/Detail JoinDefs • 105, 108, 124, 127, 740
  - creating • 124
  - in reports • 740
  - retrieving with • 127
  - sorting • 105, 108
- Master/Master JoinDefs • 105, 107, 124
  - creating • 124
  - sorting • 105, 107
- matching • 99
  - patterns in QBF • 99
- maximum (aggregate function) • 187
  - reports • 187
- memory • 760
  - usage • 760
- menu (FRS command) • 645
- menu key • 42
- menu option • 653
- menus • 30, 42, 44, 649
  - exiting • 44
  - keys • 649
  - menu items • 649
  - menu maps • 30, 649
  - using • 42, 44
- minimum (aggregate function) • 187
  - using in reports • 187
- mod (function) • 313
- mode • 46, 399, 415, 418, 422, 424, 427
  - block • 399
  - column formatting • 415, 418, 422, 424, 427
  - insert • 46
  - overstrike • 46
- mode (FRS command) • 645
- mode key • 46
- modify (statement) • 62
  - key columns • 62
- money (data type) • 298, 317, 563, 571, 572, 580, 697
  - data types for columns • 298
  - default column formats • 317
  - described • 563, 571
  - editing defaults • 580
  - storage format • 572, 697
- money (function) • 313
- MoreInfo operation • 132, 152
  - JoinDefs Catalog frame • 132
  - Report Catalog frame • 152
- mouse • 45
  - requirements • 45
- Move operation • 66, 179, 207, 208, 457, 465, 487, 514, 518, 521
  - Form Layout frame • 457, 465, 487, 518, 521



---

- moving columns in tables • 66
- moving report components • 207
- Report Layout frame • 179, 208
- Table Field menu • 514
- moving • 207
  - report components • 207

**N**

- n character in numeric templates • 336, 584
- N format • 321, 341
  - character string print format C • 321
  - character string print format T • 341
- naming • 50, 56, 60, 61, 134, 135, 287, 301, 364
  - columns • 56, 61
  - conventions • 50, 287
  - correlation names • 135
  - JoinDefs • 134
  - tables • 60
  - variables • 301, 364
- natural joins • 128
- nesting • 351, 359
  - .include (Report-Writer statement) • 359
  - comments • 351
- NewLine operation • 492
- NewQueryTarget operation • 79
  - QBF Execution Phase frame • 79
- newrow (FRS command) • 645
- Next operation • 106, 113, 154, 525, 653
  - Attributes frame • 525
  - predefined menu option • 653
  - Report MoreInfo frame • 154
  - Retrieve frame • 106
  - Update frames • 113
- nextfield (FRS command) • 645
- nextitem (FRS command) • 645
- NextMaster operation • 108, 113
  - Retrieve frame • 108
  - Update frame • 113
- NextTable (command) • 143
- nextword (FRS command) • 645
- No Auto Tab attribute • 528
- No Echo attribute • 528
- nodes • 33
  - accessing remote • 33
  - v\_node name • 33
- NoEcho (keyword) • 534
  - scrollable fields • 534
- nofirstff flag • 757

- report (command) • 757
- nonprinting characters • 295, 339
- not (logical operator) • 313
- not null (clause) • 353
- Now date constant • 535, 569
- null strings • 213, 429
  - .nullstring (Report-Writer statement) • 429
  - alternate • 429
  - reports • 213
- null values • 62, 213, 572
  - defined • 62, 572
  - displaying • 213
- nullability • 353, 533, 572
  - .declare (Report-Writer statement) • 353
  - data types • 533
  - described • 572
  - not null (clause) • 353
  - with null (clause) • 353
- Nullable attribute • 531
- numeric (data type) • 296, 330, 331, 333, 336, 563, 572, 583
  - described • 563
  - E format • 330
  - F format • 331
  - G format • 333
  - numbers • 296
  - printing • 330
  - reports • 296, 330
  - storage format • 572
  - templates • 336, 583
- numeric constants • 296
- numeric expressions • 331, 335, 343
  - F format • 331
  - functions • 343
  - N format • 335
- numeric templates • 583
  - input masking and • 583
- NxtDetail operation • 113
  - Update frame • 113

## O

- object\_key (function) • 313
- objects (user interface) • 50, 51, 287
  - naming conventions • 50, 287
  - referencing • 51
- objects in Ingres • 50
  - PC users and • 50
- OnError operation • 555, 561
  - Terminal Monitor frame • 555, 561

---

- OpenSQL • 697
  - data types • 697
- operations • 42, 653
  - aborting • 653
  - menus • 42
  - undoing • 653
- operators • 309, 310, 311, 313
  - arithmetic • 310
  - comparison • 311
  - conditional expressions • 311
  - described • 309
  - functions • 313
  - logical • 313
  - wild card pattern matching • 311
- or (comparison operator) • 94
- or (logical operator) • 313
- order by (clause) • 347, 364
- Order Menu • 517
  - VIFRED • 517
- Order operation • 89, 103, 110, 457, 487, 517
  - Form Layout frame • 457, 487, 517
  - Retrieve frame • 89, 103
  - Update frame • 110
- outer join • 124
- outputting • 211, 225
  - reports • 211, 225
- overriding • 404
  - column position • 404
- overstrike editing mode • 46, 489
- ownership • 37, 50, 52, 135, 264, 289, 352, 364, 449
  - delimited identifiers • 50, 52, 289
  - forms • 449
  - qualifying • 37
  - specify schema • 264, 352, 364
  - synonyms • 37, 264
  - tables • 37, 135, 264
  - views • 37, 264

## P

- pad (function) • 313
- padding • 673
- page (keyword) • 306
- page break • 386
- page\_length (report variable) • 303
- page\_number (report variable) • 303
- page\_width (report variable) • 303

- pages (in reports) • 160, 193, 212, 214, 265, 274, 277, 283, 303, 315, 380, 384, 386, 388, 390, 746, 757
  - .formfeeds/.noformfeeds (Report-Writer statement) • 274, 380
  - .need (Report-Writer statement) • 274, 384
  - .newpage (Report-Writer statement) • 274, 386
  - .pagelength (Report-Writer statement) • 274, 388
  - .pagewidth (Report-Writer statement) • 390
- breaks • 315, 384, 386, 746
- footer • 265
- form feeds • 757
- header • 265
- length • 212, 274, 388
- line\_number variable • 303
- numbering • 160, 193, 214, 386
- page\_length variable • 303
- page\_number variable • 303
- pagination • 274
- width • 274, 277, 283, 390
- parameters • 235, 301, 369, 753
  - passing • 235, 753
  - runtime • 301, 369
- passing • 235
  - parameters • 235
- patterns • 99, 311
  - matching • 99, 311
- performance • 759
  - troubleshooting problems • 759
- permissions • 35
  - accessing database objects • 35
- Place operation • 66, 208, 209, 210, 465, 514
  - described • 465
  - move submenu • 66, 208, 209
  - Report Margin submenu operation • 210
  - VIFRED • 514
- pm/am (with date data type) • 569
- pop-up display style • 458, 459, 460, 461, 464, 468, 470
  - active form • 460
  - adjusting size • 468
  - anchor point • 470
  - borders • 461, 470
  - controlling • 460
  - default location • 464
  - designating • 459
  - FormAttr operation • 458

---

- normal vs • 459
- position • 458
- size • 460, 470
- types • 460
- VisualAdjust operation • 468
- pop-up forms • 460, 461
  - Box/Line operation • 461
  - uses • 460
- position\_number (report variable) • 303
- Preview operation • 152
  - Report Catalog frame • 152
- preview reports • 156, 228
  - defined • 156
  - running • 228
- Previous operation • 154, 525
  - Attributes frame • 525
  - Report MoreInfo frame • 154
- previousfield (FRS command) • 645
- previousword (FRS command) • 645
- Print operation • 447, 561
  - Terminal Monitor output frame • 561
  - Utilities frame • 447
- printing • 48, 234, 274, 277, 279, 295, 316, 317, 324, 330, 339, 364, 399, 402, 415, 418, 424, 429, 430, 478, 757
  - .formfeeds (Report-Writer statement) • 274
  - block mode • 399
  - centered • 415
  - columns • 277, 364, 430
  - control characters • 339
  - dates • 324
  - default formats • 317, 757
  - escape characters • 339
  - expressions • 430
  - formats • 279, 316, 402
  - forms • 478
  - left justification • 418
  - nonprintable character • 295, 339
  - null values • 429
  - numbers • 330
  - printscreen (statement) • 48
  - reports • 234, 430
  - right justification • 424
  - screen contents • 48
  - strings • 430
  - tabs • 430
  - text • 430
  - text positioning • 757
  - variables • 364, 430

- PrintPartial operation • 232
  - printing reports • 232
- printscreen (FRS command) • 478, 622, 645
- privileges • 35
  - database object access • 35

## Q

- Q0 format • 295, 339
- QBF • 30, 71, 72, 73, 74, 77, 78, 79, 83, 88, 92, 94, 97, 110, 472, 482, 702
  - Append Operation • 83
  - calling • 79, 702
  - calling from Ingres Menu • 73, 74, 79
  - calling from operating system • 73
  - comparison operators • 92
  - data types • 78
  - definition phase • 72
  - described • 71
  - Execution Phase frame • 79, 88
  - forms • 77
  - logical operators • 94
  - phases • 72
  - QBF operation • 79
  - QBF Start-Up frame • 83
  - QBFName • 72, 472
  - qf (command) • 702
  - Retrieve frame • 88
  - running from VIFRED • 482
  - starting • 30
  - update (function) • 110
  - wild card characters • 97
- QBF Execution Phase Frame • 110
  - Update operation • 110
- QBF Start-up frame • 79
  - JoinDefs operation • 79
  - QBFNames operation • 79
  - Tables operation • 79
- QBFNames • 51, 72, 146, 479, 480, 482
  - assigning to forms • 479, 482
  - defined • 146
  - delimited identifiers • 51
  - frame in VIFRED • 480
  - Menu in VIFRED • 480
  - query targets • 72
- QBFNames operation • 79, 447, 479
  - QBF Start-up frame • 79
  - Utilities frame • 447
  - VIFRED • 479
- qualifying • 37

- object names with owner • 37
- QUEL • 288, 292, 349, 364, 374, 697
  - data types • 697
  - delimited identifiers • 288, 292
  - specifying queries in reports • 364
  - transaction handling • 349, 374
- queries • 72, 73, 74, 79, 84, 103, 249, 364, 369, 558, 561, 759
  - definition phase • 72
  - executing • 79, 558
  - execution phase • 72, 73
  - performance • 759
  - printing/filing results • 561
  - QBF • 73, 74
  - report specification • 364
  - sorting results • 103
  - targets • 84
  - variables • 249, 364, 369
- Queries operation • 30, 106, 113, 554
  - Retrieve frame • 106
  - starting Terminal Monitor • 30, 554
  - Update frame • 113
- query languages • 30
  - interactive • 30
- Query Only attribute • 528
- Query operation • 59
  - Table Utility frame • 59
- query targets • 79, 81, 84
  - default formats • 84
  - displaying catalog frames • 79
  - forms • 81
  - search order • 79
- Quit key • 483
  - Forms Layout frame • 483
- Quit operation • 44, 555, 653
  - defined • 44
  - end vs • 44
  - predefined menu option • 653
  - Terminal Monitor frame • 555
- quitting • 44
  - applications from submenus • 44

## R

- range variables • 135
- RBF • 30, 149, 150, 155, 217, 248, 703
  - calling • 703
  - overview • 149, 150
  - pop-ups • 155
  - Report-Writer • 248
- rf (command) • 703
- role • 149
  - Save Report frame • 217
  - starting from Ingres Menu • 30
- read mode • 374
- Read operation • 556
  - Terminal Monitor frame • 556
- redraw (FRS command) • 645
- Redraw key • 48
- redrawing the screen • 48, 645
- referencing • 298
  - columns • 298
- remarks (long) • 361
- remarks (short) • 377
- remote databases • 26, 32, 33, 34
  - accessing • 26, 33, 34
- remote nodes • 32, 33
- Rename operation • 132, 152, 447, 475
  - Catalog Frame menu • 475
  - Forms Catalog frame • 447
  - JoinDefs Catalog frame • 132
  - Report Catalog frame • 152
- report (command) • 254
  - described • 254
- report (keyword) • 306
- Report Catalog frame • 152
- Report Layout frame • 175, 179
  - described • 175, 179
  - menu operations • 179
- report Moreinfo Frame • 154
- Report operation • 59
  - Table Utility frame • 59
- report specification • 152, 153, 154, 157, 161, 169, 173, 175, 179, 221, 262, 592
  - archiving • 221
  - creating • 169
  - default • 161, 169, 173
  - defined • 157
  - editing • 592
  - format • 262
  - getting information about • 153, 154
  - list • 152
  - loading • 175
  - ownership • 152
  - saving • 179
- ReportOptions operation • 179
  - Report Layout frame • 179

---

reports • 30, 152, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 169, 173, 178, 181, 183, 192, 193, 194, 195, 196, 198, 199, 207, 211, 212, 213, 214, 218, 221, 225, 226, 227, 228, 230, 233, 245, 248, 249, 250, 252, 253, 254, 255, 257, 258, 265, 268, 270, 274, 276, 282, 293, 316, 317, 342, 347, 351, 352, 361, 362, 363, 364, 369, 377, 378, 380, 382, 384, 399, 401, 402, 406, 408, 415, 418, 424, 436, 438, 572, 704, 740, 757

- adding footers • 183
- archiving • 221
- block mode capabilities • 399, 401, 408
- block style • 254
- block-style • 163
- breaks • 159, 248, 250, 347
- catalog • 253
- centering • 415
- column style • 254
- column-style • 162
- comments in specifications • 257, 351, 361, 377
- conditions • 436
- constants • 293
- creating • 152, 252, 257
- creating report components • 181
- customizing • 161, 248
- data formats • 316
- data selection • 248, 249, 352
- dates • 160, 214
- default • 254, 282, 317
- default destination • 225
- default layout • 167
- default report specification • 161
- deleting • 152
- deleting components • 192
- deleting sections • 193
- designing • 255
- destinations • 225
- detail • 166
- display formats • 196, 402, 406, 572
- editing components • 194, 195
- error log • 226
- footers • 166
- formatting • 276
- headers • 166
- headings • 178
- hierarchical control break • 164
- indented style • 164
- Ingres Menu operation • 30
- joined tables • 740
- justification • 418, 424
- labels style • 165
- layouts • 248, 258, 276, 380
- margins • 382
- Master/Detail style • 164
- moving components • 207
- naming • 362
- nulls • 213
- output file • 363
- outputting options • 211, 245
- page numbers • 160, 214
- pagination • 274, 380, 384, 757
- preview • 156, 228
- printing • 225, 276
- producing • 225
- query language • 218
- remarks • 361, 377
- report (command) • 704
- Report Catalog frame • 152
- report specification • 157
- running in background (batch) • 226, 233
- runtime data selection • 227
- runtime parameters • 369
- runtime variables • 364
- sections • 166
- sending from screen to file • 245
- sending from screen to printer • 245
- sending to file or printer • 230
- sending to screen • 363
- setup procedures • 257, 704
- setup statements • 268
- sorting • 198, 199, 249, 378
- sources of data • 158, 169
- specification • 252, 265, 704
- statistics • 226
- styles • 161, 173, 254
- tabular style • 162, 254
- testing • 254
- time • 160, 214
- underlining • 212
- variables • 227, 270, 342, 438
- wrap style • 254
- wrap-style • 163

Report-Writer • 149, 247, 254, 263, 265, 292, 303, 313, 342, 399, 436, 438, 704, 707

- assignment (statement) • 438

---

---

- block mode • 399
- calling • 704
- conditional statements • 436
- described • 247
- expressions syntax summary • 342
- features • 247
- functions • 313
- overview • 149
- report (command) • 704
- reserved words • 292
- running • 254
- sample report • 707
- special report variables • 303
- specification summary • 265
- statement delimiters • 263
- statement syntax summary • 438
- Report-Writer statements • 256, 347, 349, 352, 353, 357, 359, 361, 362, 363, 364, 369, 374, 377, 378, 380, 382, 384, 386, 387, 388, 390, 392, 393, 395, 397, 399, 401, 402, 404, 406, 408, 409, 411, 415, 418, 420, 421, 422, 424, 427, 429, 430, 433, 435, 436, 438, 708, 718, 721, 729, 735, 737, 740, 746, 757
  - .block • 399, 718
  - .bottom • 401
  - .break • 347
  - .center • 415
  - .cleanup • 349, 740
  - .data • 352
  - .declare • 353, 721
  - .delimid • 357, 721
  - .detail • 393, 708, 721, 729, 735
  - .elseif • 721
  - .endblock • 399, 718, 735, 737
  - .endif • 436, 721
  - .endremark • 361
  - .endwithin • 411, 718
  - .footer • 395, 708, 721, 729
  - .format • 402, 708, 721
  - .formfeeds • 380, 721, 757
  - .header • 397, 708, 721, 729, 735
  - .if • 436, 721
  - .include • 359
  - .left • 418
  - .leftmargin • 382
  - .let • 438
  - .lineend • 420
  - .linestart • 421
  - .longremark • 361, 721
  - .name • 362
  - .need • 384, 708, 746
  - .newline • 399, 422, 718, 735, 737
  - .newpage • 386, 721
  - .nofirstff • 757
  - .nofirstffd • 387
  - .noformfeeds • 380, 757
  - .nounderline • 435
  - .nullstring • 429
  - .output • 363
  - .pagelength • 388, 721
  - .pagewidth • 390
  - .position • 404
  - .print • 430
  - .println • 430, 718
  - .query • 364, 369, 708
  - .right • 424, 708
  - .rightmargin • 392
  - .setup • 374, 740
  - .shortremark • 377
  - .sort • 378, 708, 721
  - .tab • 427
  - .tformat • 406, 721
  - .top • 408, 718
  - .ulcharacter • 433
  - .underline • 435
  - .width • 409
  - .within • 411, 718
  - types • 256
- Report-Writer system catalogs • 253
- reserved words • 50, 292
  - object names • 50
- Resize operation • 470, 495
  - VIFRED • 470, 495
- restrictions • 701
  - call (statement) parameters • 701
- Resume operation • 555, 558
  - Terminal Monitor frame • 555, 558
- retrieve frame • 103, 106
  - Next operation • 106
  - Order operation • 103
  - Query operation • 106
- retrieve operation • 91
  - search conditions • 91
- Retrieve operation • 88
  - QBF Execution Phase frame • 88
- retrieving • 79, 87, 553, 558
  - data with Terminal Monitor • 553, 558

---

---

- QBF • 79, 87
- Retrieve operation • 87
- reverse • 687
- reverse video • 687
- Reverse Video attribute • 490, 528
  - Box Attributes frame • 490
  - described • 528
- right (function) • 313
- right margins • 466
  - forms • 466
- Right operation • 208
  - Move submenu • 208
- right\_margin (report variable) • 303
- rightchar (FRS command) • 645
- rows • 506
  - displaying lines between • 506
  - number displayed • 506
- rows (in reports) • 352, 378
  - distinct vs duplicate • 378
  - sort order • 378
- rows (in table fields) • 676
  - number • 676
- rows (in tables) • 56, 65, 84, 110, 115, 139
  - appending • 84
  - defined • 56
  - deleting • 115
  - duplicates • 65
  - protecting • 139
  - updating • 110
  - width • 56
- rubout (FRS command) • 645
- Rulers operation • 455, 456
  - VIFRED Form Layout frame • 455, 456
- rules • 121, 139, 141, 142
  - deleting • 141, 142
  - JoinDef • 121, 141
  - Rules operation • 139, 141
  - updating • 141, 142
- Rules operation • 133, 136
  - JoinDef Definition frame • 133
  - JoinDef Specification frame • 136
- Run a Report • 228
  - pop-up • 228
- runtime • 201
  - data selection • 201
- runtime system • 353, 364
  - data selection • 353
  - variables • 364

## S

- s flag • 349, 374
  - report (command) • 349, 374
- sample reports • 707, 708, 721, 729, 737
  - account • 721
  - described • 707
  - dictionary • 729
  - label • 737
  - population • 708
- Save Changes to Report • 220
  - pop-up • 220
- Save operation • 113, 116, 133, 154, 179, 217, 218, 457, 472, 487, 653
  - described • 457, 487
  - Form Layout frame • 457, 487
  - Forms Layout frame • 472
  - JoinDef Definition frame • 133
  - predefined menu option • 653
  - Report Layout frame • 179, 217
  - Report MoreInfo frame • 154
  - Save Report frame • 218
  - Update frame • 113, 116
- Save Report frame • 218
  - fields • 218
- saving • 116, 145, 220, 472, 474
  - forms • 472
  - JoinDefs • 145
  - Save Changes to Report pop-up • 220
  - Save operation • 145
  - Saving a Form pop-up • 474
  - table updates • 116
  - VIFRED • 472
- Saving a Form • 474
  - pop-up • 474
- schema • 35, 37, 50, 69, 135, 264, 288, 289, 357
  - defined • 264
  - delimited identifiers • 50, 288, 357
  - object owner qualification • 35, 37, 50, 69, 135, 264, 289
- screen • 44, 48, 608, 676, 687, 688, 691
  - boxing characters • 688
  - clearing • 676
  - color • 691
  - printing contents • 48
  - refreshing • 48
  - returning to previous • 44
  - types • 608

---

- video attributes • 687
- Scroll size attribute • 531
- Scrollable attribute • 531
- scrollable fields • 534
  - defined • 534
  - DisplayOnly • 534
  - NoEcho • 534
- scrolldown (FRS command) • 645
- scrolling • 46, 107, 108, 693
  - keys • 46
  - table fields • 107, 108
- scrollleft (FRS command) • 645
- scrollright (FRS command) • 645
- scrollup (FRS command) • 645
- search conditions • 91
  - retrievals • 91
- search order • 79
  - query targets • 79
- select (clause) • 364, 746
- select (statement) • 364
  - report queries • 364
- Select a Destination • 228, 229
  - pop-up • 228, 229
- selecting an Aggregate • 190
  - pop-up • 190
- Sending a Report to a File • 231, 233
  - pop-up • 231, 233
- Sending a Report to a Printer • 232
  - pop-up • 232
- separation lines • 506
  - displaying • 506
- separators • 263
  - multiple values for a parameter • 263
- sequence numbers • 517
  - in fields • 517
- server • 32, 33, 34
  - type • 32, 33, 34
- set autocommit (statement) • 349, 374
  - overriding default commit behavior • 349, 374
- set functions • 304
- Set lockmode (statement) • 374
  - reports • 374
- shell (FRS command) • 645
- shift (function) • 313
- Shift operation • 208, 209
  - Move submenu • 208, 209
- simple fields • 39, 95, 106, 135
  - defined • 39
- JoinDefs • 135
- logical operators • 95
- viewing retrieved information • 106
- SimpleFields operation • 451
  - Forms Catalog frame • 451
- sin (function) • 313
- size (function) • 313
- smallint (data type) • 317, 563, 697
  - default column format • 317
  - described • 563
  - integer2 (data type) • 697
  - OpenSQL • 697
- Sort Columns pop-up • 183
- sorting • 103, 110, 156, 198, 249, 272, 304, 378, 397
  - .header (Report-Writer statement) • 397
  - aggregate functions • 304
  - breaks • 304
  - columns • 272
  - defining report sort order • 198
  - order • 103
  - preview reports • 156
  - reports • 249, 378
  - rows • 110
- special characters • 336, 339, 595
  - numeric templates • 336
  - Q0 format • 339
  - string templates • 595
- specifications • 316, 510
  - column • 510
  - data format • 316
- SplitLine operation • 555
  - Terminal Monitor frame • 555
- SQL • 235, 298, 349, 364, 374, 563, 697, 705
  - .cleanup • 349
  - .setup • 374
  - autocommit • 349, 374
  - data types • 298, 563, 697
  - parameter passing • 235
  - specifying queries in reports • 364
  - sql (command) • 705
  - transaction handling • 349, 374
- SQL-92 standard • 287
  - database requirements • 287
- sqrt (function) • 313
- squeeze (function) • 313
- sreport (command) • 253, 359, 753
  - .include (Report-Writer statement) • 359
  - role • 253

---



---

- troubleshooting with • 753
- sreport(command) • 706
- star • 26
  - accessing distributed database • 26
- Star server • 34
  - accessing distributed database • 34
- starting • 28, 79, 446, 554
  - Ingres tools • 28
  - Interactive query languages (ISQL and IQUEL) • 554
  - QBF • 79
  - Terminal Monitor • 554
  - VIFRED • 446
- statement separator • 263
  - Report-Writer • 263
- statements • 256, 262
  - syntax • 262
  - types • 256
- straight edge alignment guides • 177
  - RBF • 177
- string templates • 594, 601, 602
  - creating custom template • 594
  - custom character set examples • 602
  - defining • 594
  - described • 594
  - examples • 602
  - mandatory entry • 601
- strings • 291, 293, 295, 313, 321, 341, 343, 353, 430, 566, 653
  - C format • 321
  - constants • 291, 293
  - delimiters • 293
  - functions • 343
  - hexadecimal • 295
  - literal • 566
  - printing • 430
  - reports • 293, 430
  - T format • 341
  - trim (function) • 313
  - with prompt (clause) • 353
  - with value (clause) • 353
  - word wrapping • 321
- styles • 161, 228
  - Choose a Report Style pop-up • 228
  - report • 161
- subtopics • 47
- sum (aggregate function) • 187, 188
  - reports • 187, 188

- synonyms • 36, 37, 56, 67, 68, 264, 288, 289, 352
  - defined • 56
  - delimited identifiers • 288, 289
  - destroying • 67
  - examining • 68
  - ownership • 37, 264
  - using • 36, 352
- syntax • 32, 342, 344, 438
  - aggregate functions • 344
  - database access • 32
  - Report-Writer expressions summary • 342
  - Report-Writer statement summary • 438
- system variables • 26
  - setting • 26
- systems • 647, 653
  - administrator • 647
  - exiting • 653

## T

- t flag • 402
  - report (command) • 402
- T format • 341
- Tab key • 45
- tabbing order • 517
  - changing • 517
- Table Field frame • 506
  - VIFRED • 506
- table fields • 39, 52, 85, 94, 105, 107, 108, 115, 135, 506, 510, 511, 514
  - components • 39
  - creating • 506
  - described • 39
  - display modes • 510
  - editing • 115, 511
  - inserting rows • 85
  - internal names • 506
  - JoinDefs • 135
  - logical operators • 94
  - moving columns • 514
  - retrieving information • 105
  - scrolling • 107, 108
  - validating • 52
- table names, abbreviating • 135
- table\_key (function) • 313
- TableField operation • 451
  - Forms Catalog frame • 451

---

- tables • 36, 37, 51, 56, 60, 67, 68, 72, 84, 115, 119, 135, 138, 139, 140, 158, 249, 264, 272, 288, 289, 352, 357, 449, 740
  - appending data • 84
  - basis for form • 449
  - creating • 60
  - data source for reports • 158
  - delimited identifiers • 51, 288, 289, 357
  - destroying • 67
  - examining • 68
  - joining • 119, 139, 140, 740
  - master deleting records • 115
  - naming • 60
  - ownership • 37, 135, 264
  - query targets • 72
  - reports created from • 249, 272, 352
  - retrieving into/from • 138
  - single-table JoinDefs • 139
  - structure • 56
  - synonyms • 36
- Tables (utility) • 30, 55, 58, 60
  - creating tables • 60
  - described • 55
  - starting from Ingres Menu • 30
  - starting from IngresMenu • 58
- Tables Catalog frame • 58
  - listing user tables • 58
- Tables operation • 30, 58, 79
  - Ingres Menu • 30, 58
  - QBF Start-up frame • 79
- tabs • 430
  - printing • 430
- tabular • 162, 254
  - formats for reports • 162, 254
- templates • 324, 336, 587, 588, 594, 602
  - absolute date and time • 587, 588
  - creating custom string template • 594
  - date format • 324, 587, 588
  - defining string template • 594
  - numeric (data type) • 336
  - string examples • 602
  - string templates • 594
- temporary formats • 402, 406, 411
- TERM\_INGRES • 605
- termcap descriptions • 674, 676, 686, 687
  - advanced features • 686
  - basic features • 676
  - list of commands • 676
  - writing • 674
- termcap file • 605, 626
  - purpose • 605, 626
- Terminal Monitor • 52, 551, 554, 557, 561, 705
  - calling • 705
  - delimited identifiers • 52
  - described • 551
  - error messages • 561
  - printing query results • 561
  - query languages • 551
  - startup • 554
  - writing statements to a file • 557
- Terminal Monitor frame • 557, 558
  - input screen • 557, 558
  - output screen • 558
- terminals • 607, 608, 627, 635, 639, 676, 687, 691, 693, 695, 696
  - activating cursor • 693
  - color • 691
  - Concept 100 • 696
  - Datamedia 3045 • 696
  - Dec VT100 • 695
  - Envision 230 • 696
  - initializing • 676
  - mapping files • 635, 639
  - types functional with Ingres • 608
  - video attributes • 687
  - VT • 607, 608, 627, 635, 639, 695
- text • 278, 433
  - positioning • 278
  - underlining • 433
- text (data type) • 298, 317, 563, 566, 568, 697
  - data types for report columns • 298
  - default report column format • 317
  - described • 563, 566, 568
  - storage format • 697
- text (function) • 313
- text file • 256
  - report definition • 256
- text strings • 566
  - character data type • 566
- time • 160, 214, 297, 303, 313, 328, 569, 587, 588, 592
  - absolute • 297, 587, 592
  - current\_time variable • 303
  - formats • 297
  - functions • 313
  - intervals • 297

---

- reports • 160, 214
- templates • 328, 587, 588, 592
- zones • 569
- Title operation • 497, 499, 521
  - VIFRED • 497, 499, 521
- titles • 39, 499
  - fields • 39, 499
- Today date constant • 535, 569
- Top (FRS command) • 558
  - Terminal Monitor frame • 558
- Top operation • 653
  - predefined menu option • 653
- transactions • 349
  - rolling back • 349
- trim • 38, 185, 208, 485, 489
  - creating • 185, 489
  - defined • 38
  - editing • 489
  - form component • 485
  - moving • 208
- trim (function) • 313
- Trim operation • 181, 489
  - Create submenu • 181
  - VIFRED forms components • 489

## U

- Underline attribute • 490, 528
  - Box/Trim Attributes frame • 490
  - described • 528
- underlining • 212, 433, 435, 687
  - .nounderline (Report-Writer statement) • 435
  - .ulcharacter (Report-Writer statement) • 433
  - .underline (Report-Writer statement) • 435
  - reports • 212, 435
- underscore • 687
- Undo operation • 44, 179, 216, 457, 487, 519, 653
  - defined • 44
  - Form Layout frame • 457, 487, 519
  - predefined menu option • 653
  - Report Layout frame • 179
  - reversing deletions • 216
- union • 364, 741
  - select statements • 364, 741
- unique keys • 65
  - preventing duplicate rows in tables • 65
- UNIX • 236, 240, 359, 369, 668, 693

- .include (Report-Writer statement) • 359
  - delimiting parameters containing
    - parentheses • 236, 369
  - escaping double quotes • 240
  - termcap file for • 668, 693
- Update frame • 113
  - Delete operation • 113
- Update operation • 110, 117
  - described • 110
  - exiting • 117
  - QBF Execution Phase frame • 110
- updating • 79, 110, 112, 116, 141, 142
  - JoinDefs • 112, 116
  - QBF • 79, 110
  - rows in tables • 110
  - rules • 141, 142
  - Update operation • 110
- upline (FRS command) • 645
- uppercase (function) • 313
- user • 642
  - mapping customized • 642
- user identifiers • 50
  - schema names • 50
  - specifying effective user • 50
- user names • 288, 289, 357
  - delimited identifiers • 288, 289, 357
  - object ownership specification • 289
  - specifying effective user • 289
- user-defined abstract data types • 563, 697
- Utilities operation • 152, 447
  - Forms Catalog frame • 447
  - Report Catalog frame • 152

## V

- v flag • 388
  - report (command) • 388
- V\_node • 33, 34
- validation • 52, 139, 531, 536, 537, 543
  - changing criteria • 536
  - check • 531, 536
  - checks on joins • 139
  - comparisons • 537
  - creating criteria • 536
  - delimited identifiers in • 52
  - error message • 543
  - maximum length • 536
  - syntax check • 536
  - table field columns • 537
  - valid comparison operators • 537

---

- Validation Check attribute • 531
- Validation Error Message attribute • 531
- varchar (data type) • 298, 313, 317, 563, 566, 568, 697
  - conversion function • 313
  - data types for report columns • 298
  - default report column format • 317
  - described • 563, 566, 568
  - storage format • 697
- varchar (function) • 313
- variables • 26, 32, 33, 34, 135, 235, 249, 270, 301, 303, 353, 364, 430, 438, 753
  - assigning values to • 353, 438
  - dbname • 33
  - declaring • 353, 753
  - expressions • 301, 438
  - initialization • 301, 353
  - naming • 301, 364
  - passing • 235
  - printing • 364, 430
  - query • 249, 301, 364
  - range • 135
  - remote node • 33
  - reports • 270, 753
  - runtime • 270, 364, 430
  - run-time • 753
  - server\_type • 32, 33, 34
  - special report • 303
  - system • 26
  - v\_node • 33
- VAX/VMS • 236, 240, 359, 369
  - .include (Report-Writer statement) • 359
  - delimiting parameters containing parentheses • 236, 369
  - escaping double quotes • 240
- vchar (data type) • 317
  - default report column format • 317
- vchar (function) • 313
- video attributes • 687
  - termcap descriptions • 687
- viewing • 59, 68
  - database tables • 59, 68
- views • 37, 56, 67, 68, 158, 264, 272, 288, 289, 357
  - data source for reports • 158
  - defined • 56
  - deleting • 67
  - delimited identifiers • 288, 289, 357
  - examining • 68
  - ownership • 37, 264
  - reports • 272
- VIFRED • 30, 52, 443, 444, 446, 447, 449, 451, 453, 454, 458, 468, 472, 474, 475, 480, 483, 485, 488, 491, 492, 497, 500, 503, 506, 509, 523, 581, 706
  - Attributes operation • 503
  - Box/Line operation • 492
  - calling • 446, 706
  - Catalog frame • 447, 449
  - components of form • 485
  - creating blank lines • 491
  - creating form components • 488
  - creating forms • 449
  - data windows • 500
  - default forms • 451
  - destroying forms • 475
  - exiting • 483
  - expert mode • 446
  - features • 443
  - field display formats • 581
  - Field operation • 497
  - fields • 497
  - Form Layout frame • 454
  - FormAttr operation • 458
  - forms creating • 474
  - forms editing • 475
  - forms renaming • 475
  - GetTableDef operation • 509
  - menu operations summarized • 444
  - multiple table forms • 453
  - NewLine operation • 492
  - overview of • 443
  - QBFFNames Catalog frame • 480
  - role • 443
  - saving forms • 472
  - setting attributes on fields • 523
  - starting from Ingres Menu • 30
  - TableField operation • 506
  - utilities operation • 447
  - validation checks • 52
  - Vifred (command) • 446
  - VisualAdjust operation • 468
- vifred (command) • 706
- Vision • 30
  - starting from Ingres Menu • 30
- VisuallyAdjust operation • 462, 468, 470, 471
  - described • 462
  - fixed-position pop-up forms • 468

---

---

- floating pop-up forms • 468
- Move operation • 471
- Resize operation • 470
- rotating anchor point • 470
- VT terminals • 607, 608, 627, 635, 639, 695

## W

- w display format parameter • 320, 321, 330, 331, 333, 334, 335
  - blanking print format B • 320
  - character string print format C • 321
  - numeric print format E • 330
  - numeric print format F • 331
  - numeric print format G • 333
  - numeric print format I • 334
  - numeric print format N • 335
- w flag • 399, 430
  - report (command) • 399, 430
- w\_column (report variable) • 303, 411
- w\_name (report variable) • 303, 411
- where (clause) • 285, 364
- wild card characters • 97, 312, 538, 753
  - asterisk (\*) • 312
  - QBF • 97
  - question mark (?) • 312
  - Report-Writer • 753
  - VIFRED • 538
- Windows NT • 236, 240, 242, 364, 369
  - delimiting parameters containing
    - parentheses • 236, 240, 242, 364, 369
  - escaping double quotes • 240
- WinNT • 359
  - .include (Report-Writer statement) • 359
- with null (clause) • 353
- with prompt (clause) • 353, 364
- with value (clause) • 353, 364
- wrap • 163, 321, 430
  - character string word wrapping • 321
  - line wrapping in reports • 430
  - report style • 163
- wrap style reports • 254
- Write operation • 557
  - Terminal Monitor frame • 557

## Z

- z character in numeric templates • 336, 584