

Mastering DataPerfect®

Ralph Alvy

Mastering DataPerfect®

Copyright © 1997 by Ralph Alvy

All Rights Reserved. No part of the contents of this book may be reproduced in any form or by any means without the written permission of the author. None of the disk files found in this archive may be distributed without permission from the author. Please do not post them in public distribution centers such as BBSs, FTP sites, and CompuServe Libraries without such permission.

Published in the United States by Ralph Alvy

Second Edition

The author/publisher used his best efforts in preparing this book, its accompanying disk files, and the information conveyed. However, the author/publisher makes no warranties of any kind, expressed or implied, with regard to documentation and information contained in this book or accompanying disk files, and specifically disclaims without limitation, any implied warranties of merchantability and fitness for a particular purpose with respect to the disk files, the applications or data contained therein, or the techniques described in the book. In no event shall the author/publisher be responsible or liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential or any other damages in connection with or arising out of furnishing or using the disk files, the applications or data contained therein, or the techniques described in the book.

Trademarks

By using appropriate symbols or initial capitalization, the author has designated many words he has reason to believe trademark, service mark, or other proprietary rights may exist. However, the author may have missed many words or terms in which proprietary rights might exist. The inclusion, exclusion or definition of a word or term is not intended to affect, or express any judgment on, the validity or legal status of any proprietary right which may be claimed in that work or term.

Table Of Contents

Acknowledgments	xi
Foreword	xiii
Introduction	1
What's DataPerfect?	1
What's This Book?	1
Who's Ralph Alvy?	1
The Chapters	2
Products Mentioned	4
Legalese	5
For Beginners	7
Getting Started	7
Creating Panel Text	10
Sizing and Moving the Panel	11
CREATE a Field (F9) and EDIT Field Format (F6)	11
DEFINE an INDEX (Ctrl-F8)	12
Field Formats and Field Format Display Modifiers	14
Browse, Create and Edit Modes	15
Lookups in Browse Mode	16
Remaining Define Panel Menu Options	16
DEFINE PANEL Options (Alt-F8)	17
Define Link for ↓Panel (F5)	20
DEFINE an INDEX (Ctrl-F8)	20
DEFINE FIELD Options (Shift-F8) 1, 2 and 4	21
DEFINE FIELD Options (Shift-F8) 3, 5, 6-9	25
Files and Specifications	27
Program Files	27
Application Files	27
The .STR File	27
The .IND File	28
The .TXX File	28
The Data Files	29
The .TMP Files	29
Specifications	30
Fields: Introduction	31
Field Fundamentals	31
Field Codes and Field Names	32
Field Types	32

Alphanumeric Fields: A, U, and Variable-Length Text	32
Numeric Fields: N, G, H, F, D, and T	33
Display Mode Indicators	38
Fields: Issues	41
Choosing Between ::C and ::N Fields	41
Do You Need the Field to Be Real?	41
Issues Concerning Totaling	42
Keeping Subpanel Data Current	43
Computed Fields and DataPerfect's Work Space	49
Choosing Between G Fields and N Fields	49
The H Field	50
Date Fields	51
When You Might Not Want to Use the D Field for Dates	51
The Date Field as a Special Numerical Field	54
Two-Digit vs. Four-Digit Years	55
A Note about International Dates	56
The Time Field	56
Using Time Fields To Guarantee Uniqueness Of Records	57
Computing Elapsed Time: The Simple Case	58
Calculating Elapsed Time Across the 24-Hour Barrier	58
First Solution	59
An Alternative Solution:	
Using the Concepts of MOMENT and MODULO	61
Formula Changes to Trap Incorrect Data Entry	63
A Special Use for the MOMENT Function in Reports	64
Notes on Deleting a Field	66
Lookups	69
Fundamentals of Lookups	69
Subfield Lookups	70
The Data Link Subgroup Lookup	72
Making Lookups Look Better (Browse Mode)	74
Smart Lookups (Browse Mode)	75
The Smart Lookups Algorithm	77
Strategy in Defining a Browse Mode Lookup	
Using the Smart Lookups Algorithms	81
What if you don't want the lookup field	
to be the first field in its own lookup field list?	82
Reasons for Assigning a Lookup to a Hidden Field	83
Reasons for Assigning a Lookup to a Non-Updatable Field	85
A Note about Saving a Lookup Definition	85
Troubleshooting Lookups	86
Indexes	89
Introduction	89

The .IND File	89
How Indexes Sort	90
Uniqueness and Picking Fields for the Index Field List	92
Sorting Backwards with Reverse Indexes	94
Reverse Sorting by Number	94
Reverse Sorting by Date	97
The .STR File and Index Regeneration	98
Exception Lists	99
What They Are	99
The Lowest Numbered Index: A Caveat	101
Aiding Lookups with Exception List Indexes	101
Speeding Reports with Exception List Indexes	102
Aiding Computed Fields in a Parent Panel with Exception List Indexes	103
Dividing Data File Record Access with Exception List Indexes	105
Warning: Exception List Index Bug in Version 2.2	107

Links 109

Flat-File DBMS vs. Relational DBMS	109
The Four Linking Relationships	110
One-to-Many Linking	110
Many-to-One Linking	110
One-to-One Linking	110
Many-to-Many Linking	111
The Two Types of DataPerfect Links	111
Creating and Defining a Panel Link	113
The Link Index and Link Field List	114
A Note about a Panel Link's Index	116
The Action of a Panel Link	116
The Data Link	117
Defining a Data Link	118
The Link Options Menus	120
Panel Link Options	120
Data Link Options	122
Data Link Options Caveats	125
Data Link Subgroup Lookups	125
Choosing Between the Panel Link and the Data Link	128
A Caveat Regarding Data Links	129
The Recursive Panel Link	133
Conditional Incrementation Using a Recursive Panel Link	133
Absolute Incrementation Using a Recursive Panel Link	134
Reasons for Avoiding Auto-Incrementing Fields	135
Back to Recursive Linking and Absolute Incrementation	136
Absolute Incrementation	
Using a Recursive Panel Link on a Network	137
Virtual Link vs. Subreport Using Virtual Link	140

Making Panel Links Safer	141
Troubleshooting Links	143
Keep A Total	147
Introduction	147
Implementing a Keep A Total	148
When There's No Parent Record	150
Notes on Target Panels for Keep A Total Operations	151
Panel Hierarchies and Keep A Total: a Caveat	151
Using Keep A Total to Update Records in Foreign Panels	151
The Indexes	152
The Keep A Total	153
How it Works: An Example	153
General Principles	154
Keep A Total vs. Cascade Update	155
Reports: Introduction	157
Introduction	157
The Initial Report Definition Screen	158
Report Name	159
Options 1 and 2: Destination	159
Option 6: Disk File Mode	160
Option 7: Print Margins	160
Finding What Panel a Report Is Based On	162
Panel-Dependent Initial Report Definition Screen Options	162
Option 3: Index Number	163
Option 4: Search Conditions	163
Option 5: Sort Direction	163
Option 8: Edit Report Form	164
The Edit Report Form Screen Sections Delineated	166
The Report Algorithm	170
General Theory in Creating a Report	171
Processing the Right Records	171
Getting the Right Information to Print	172
Getting the Report to Look Good	173
Getting the Report to Complete in a Reasonable Amount of Time	173
Knowing Your Place	174
In the Main Report	174
In a Subreport	174
Reports: General Structure	177
The Basic Report	177
Two-Level Reports	181
Subgroup Reports	183
The Subreport	184
Two-Level Reports in Subreports	192

The Primary Sorting Field	193
Subgroup Reports in Subreports	194
Reports: Fields	197
F Fields	197
Print Mode Indicators	198
Print Mode Indicators That Alter Field Output Spacing	198
Print Mode Indicators That Don't Alter Field Output Spacing	199
Summary Table	200
Sneaking Print Mode Indicators into Panel Fields	202
Variable-Length Text Fields in Reports	203
The ;;N Print Mode Indicator (New Occurrence of Field)	205
Examples	205
Report Options	209
The Report Options Menus	209
Global Report Options	211
Select Report Field	211
Eliminate Line if Blank	213
Page Eject and Skip to Bottom of Page	214
Prompt for Report Variable	214
Iteration Control	215
Section-Specific Report Options	215
Section-Specific Report Options for First Page Header	215
Section-Specific Report Options for Other Page Header	216
Section-Specific Report Options for Two-Level Header	216
Section-Specific Report Options for Report Body	216
Section-Specific Report Options for Two-Level Footer	218
Section-Specific Report Options in Page Footer	218
Section-Specific Report Options in Final Footer	218
Report Variables	219
Introduction	219
Printing Data Not Already in Fields	220
Skipping Certain Records	223
Self-Referencing Report Variables	225
Counters	225
Recycled Report Variables	227
Subreports	229
Introduction	229
Option 1 - Include Subreport	229
Subreports: Going From Version 2.2 to 2.3	230
Subreports as Subroutines	232
Printing Totals at the Top of the Invoice	235
Subreport Using Virtual Link	239

The Dummy Report	243
Dummy Report Examples	245
A Report That Branches to Other Reports	245
Gathering Preliminary Information from Various Panels Before Starting the "Real" Report	253
Prompting the User with the Number of Hits	256
Reports That Double-Sort Records	259
Printer Control Issues	267
Open Filename in Report Variable	267
Overwrite Mode from the Report List	269
Overwrite Mode Caveat	270
Printer Control Panel	271
Direct Approach 1	272
Direct Approach 2	273
Indirect Approach 1	273
Indirect Approach 2	276
What These Reports Look Like	277
Direct Report Variable Approach	277
Indirect Report Variable Approach	280
Formulas	281
Introduction	281
Formula Error Messages: A Warning	282
Legal Values and Well-Formed Formulas	282
About The Rest of This Chapter	285
String Identity	285
Perfect Matches and the Identity Operator	286
CASES Statements vs. IF-THEN Statements	289
APPLY.FORMAT and CONVERT	291
APPLY.FORMAT	291
CONVERT	293
SUBSTRING and SUBFIELD	294
SUBFIELD	294
SUBSTRING	298
A String Identity Variation	298
CONTAINS	299
Spaces and Carriage Returns in Formulas	300
Troubleshooting Formulas	301
Iteration Control	303
Introduction	303
Skip Record if RV is False	303
Skip To and Stop If	306
The Variable Entity in Skip To Operations	307
Strategic Placement of the Skip To Code	308

The Internal Logic of the Skip To Code	308
Combining the Skip To Code with the Stop If Code	309
User Chooses Next Record By LookUp	313
Placing the User Chooses code in the First Page Header	315
Placing the User Chooses code in the Report Body	317
Single Record Report From Lookup off a Menu	317
How Report Lookups Display	318
Repeat If	320
A Note about DataPerfect's Notion of Truth	321
Iteration Control Examples	324
Limiting a Report to One Record	324
Limiting a Report to a Particular Number of Records	325
A Report That Prints a Particular Number of Iterations per Record ...	325
A Date-Range Report	326
A Report That Prints Monthly Statements	
for All and Only Accounts with a Positive Balance	326
Getting a Report to Continue after the	
Last Record in the Lookup Is Selected	327
Troubleshooting Iteration Control	328
Export/Import	329
Reasons for Exporting or Importing Data	329
WordPerfect Merge Files	330
The Setup	330
The Nature of the Export File, Including Some Caveats	331
Importing a WordPerfect Merge File	332
Duplicate Records Action	333
What Happens During a WordPerfect Merge File Import	334
Exporting and Importing Transaction Logs	335
Strategies: Merge File vs. Transaction Log	336
Deleting and Creating Fields	336
Moving Fields Without Deleting or Adding Fields	337
DOS Delimited Text	337
Exporting to DOS Delimited Format	338
Importing From DOS Delimited Format	342
The Clipboard	343
Introduction	343
In Define Panel Mode	343
In a Specify Formula Screen	344
In Report Definition Mode	345
Field Formulas and Help Screens: A Caveat	348
Securing the Application	349
Application Passwords	349
Menus	350

The Define Menu Screen	350
Create/Edit Menu Text	351
Move the Menu Prompt	352
Go to Panel	352
Run Report	354
Run Report Option Caveats	355
Go to Panel List	356
Go to Report List	357
Submenu	357
Launch Shell Macro	357
Edit an Existing Entry	357
Delete an Existing Entry	358
User ID Panel	358
User ID Panel Caveats	359
Tracking User Activity: the USER.FIELD[n] Function	360
Controlling User Access	361
The User ID Panel vis a vis Application Passwords	363
User-Stamping Records	364
Controlling Access to Data Accessed from a Menu	365
Controlled Panel Link Access to Subrecords	366
Data Link Subgroup Lookups and the USER.FIELD[n] Function	367
A Note on Deselecting the User ID Panel	368
Troubleshooting Menus	368
Securing Data Entry	371
Preventing Inadvertent Editing	371
The DPMouse© Alternative	372
Keeping a Saved Field from Being Edited	372
Field and Record Protection	373
Controlling Data Entry: The Pick List Field	373
Closing Off a Data Link to Protect a Pick List Panel	374
Closing off Access to the Panel List	374
Controlling Data Entry: Initial Formula or Value	375
Controlling Data Entry: The Basic Default Panel	376
Controlling Data Entry: The Complex Default Panel	377
Controlling Data Entry with ZipKey©	382
Pyramidal Design as a Data Integrity Strategy	383
The Hidden Panel	385
Controlling Record Creation with Indexes	387
Controlling Record Creation with the ::M Field	388
DPMouse© and Field Protection	390
Using DPMouse© To Conditionally Close A Panel Link	390
Using DataPerfect 2.3's Menu Facility	
To Prevent Creation of Records via a Link	391
Controlling Record Deletions with DPMouse©	391
Controlling Data Entry with Reports	392

Reports That Control Record Deletions	393
Providing an Undelete	395
Lookups and the Undelete Scheme	396
Totaling and the Undelete Scheme	396
Physically Deleting the Record	397
Reports That Control Record Creation	399
Reports That Control Editing	401
Hiding Data Entry	403
Application Maintenance Utilities	407
DPExport and DPImport	407
.STE Editing Caveats	407
DPImport Caveats	408
Importing Reports	408
DPDiagnostics	409
Optimization Messages	410
Warning Messages	410
Error Messages	410
DPOrder	411
STE Manager©	413
Application Maintenance Issues	417
Upgrading a Client to a New Version of DataPerfect	417
Compatibility Between Different Versions of DataPerfect	417
When It's Okay to Overwrite an .STR	418
Removing Duplicates in a Panel	422
How Duplicates Are Created	423
Removing Duplicates in a Single Panel	424
Removing Duplicates in the Entire Database	424
Cleaning the .STR Without Re-indexing	425
The Big Clean: Cleaning the Entire Application	426
Fixing a Corrupt .TXX File	426
Crippling Applications	428
Date Crippling	428
Record-Number Crippling	429
Password-Protected Zip File	429
Epilogue	431
Support Avenues	431
CompuServe Support	431
Internet Support	431
The DataPerfect Users Cooperative	432
Index	433

Acknowledgments

It's hard to acknowledge all those who have contributed to this book. I say that in all sincerity. To all participants in the DataPerfect section of WordPerfect Users Forum on CompuServe, as well as the DATAPERF LISTSERV on the Internet, thanks for contributing. You've offered so many DataPerfect application developers, including myself, priceless help. You've shared, and continue to share, without expecting anything in return.

Most of all, I want to thank my kind and gentle friend, Lew Bastian.

For those of you who don't know of me, I'm the original developer of DataPerfect. I am amazed at the very existence of this book. I appreciate Ralph's continued work heading the CompuServe DataPerfect users forum, and his cross country DataPerfect seminars. I applaud his perseverance in producing this work, especially since it is for a product that several companies have tried to pronounce Dead On Arrival. This book is sorely needed, since many powerful features have been added since the last manual was published. Ralph covers just about everything, even the most recent updates. He includes extending the usability of DataPerfect with DPMouse©, an excellent companion product. As with any product, people have found clever ways of doing things with DataPerfect. Using such techniques makes the difference between applications that almost work and those that are complete and polished. Ralph has been in a position to accumulate this wealth of invention and pass it on to other developers. No previous book attempted to show the DataPerfect developer how to produce masterful applications. Ralph does this in his seminars, and capably expands his seminar notes and experiences in the latter half of this book. I know from personal contact that many developers have been anxiously awaiting its publication.

DataPerfect has had an amazing resilience. In spite of being a DOS product in a Windows world, DataPerfect continues to win new friends. To a large extent this is due to the excellent online help available on the CompuServe forum hosted by Ralph Alvy (WPUSERS, Section 11). I am grateful to him and the many DataPerfect experts that frequent that forum, helping novices (and each other) with the intricacies of database applications.

DataPerfect continues to slowly evolve. It owes its popularity to its rapid application development features, its high reliability with large numbers of records, and its ability to be simultaneously used by hundreds of network users. Many users have tried to move to other products and, after experiencing one failure after another with them, been forced to return to DataPerfect. I wish DataPerfect had a more modern user interface. Some work is being done to correct this deficiency. The existence of DPMouse© has helped considerably.

A. Lewis Bastian, Jr.
DataPerfect Lead Developer
January 11, 1997

What's DataPerfect?

DataPerfect is a fully relational character-based DOS database manager that allows the application developer to use a point-and-shoot interface to create networkable database applications, needing a simple programming language only when constructing field formulas. DataPerfect applications can interrelate up to 99 data files, each containing up to 16 million records. These applications can run on a network with up to 9,999 users viewing *and editing* the same record simultaneously.

Authored by Lew Bastian, DataPerfect was initially distributed by WordPerfect Corporation. As of this writing, Novell owns DataPerfect, though they neither sell nor support it. In December 1995, Novell graciously released DataPerfect to the public, encouraging its free distribution as copyrighted free shareware, while allowing Lew Bastian to continue to fix and enhance the product as he sees fit. DataPerfect is currently distributed and supported by a community of DataPerfect application developers. See **Epilogue** for more on this.

What's This Book?

I can't stress this enough: I do *not* intend for this book to be a replacement for the DataPerfect manual I included on diskette for you. I don't cover all topics found in that manual. Rather, with this book I complement that manual, offering information explained a little differently, as well as information not covered at all.

For many years I've been the sysop in charge of DataPerfect discussion on CompuServe (see **Epilogue** for more on the CompuServe forum that supports DataPerfect). In that role, I've been exposed to many tips, tricks and caveats regarding DataPerfect application development. This book is an attempt to share some of that knowledge.

Who's Ralph Alvy?

I attended UCLA from 1967 to 1977 as a philosophy major. The last five of those years I was in the Philosophy Department's doctoral program, during which time, as a teaching assistant, I found myself frequently teaching lower division logic. Tiring of academia, I left to pursue the study of chiropractic. I've been a full time doctor of chiropractic since 1982.

Unhappy with billing applications available to the chiropractic profession, I decided to write my own. Having been a WordPerfect user since WordPerfect 4.1, but not having any experience in programming more than batch files, I turned to

DataPerfect 2.0. Probably because of my background in logic, I took to it very fast. Within a few years I was writing articles for DataPerfect's various newsletters (*DataPerfect Users Newsletter*, *DataLink* and *DataPerfection*). Eventually (I think it was 1991), I was asked to manage the DataPerfect section of WordPerfect Users Forum on CompuServe (see *Support Avenues* in **Epilogue** for more on CompuServe support of DataPerfect).

In 1994 and 1995, in a handful of cities, I taught a national two-day seminar on DataPerfect application development. That seminar was geared towards the experienced DataPerfect application developer. I include all the material taught in that seminar in this book.

Today I still practice chiropractic full time, running my practice with a DataPerfect billing application I wrote, and still sysop the DataPerfect section of WordPerfect Users Forum on CompuServe.

The Chapters

In **For Beginners**, I attempt to get the beginner started. For many, this will be all they need. For others, they'll also have to read the DataPerfect manual I included on diskette.

In **Files and Specifications**, I discuss each file required to run a DataPerfect application, as well as outline file specifications. You won't find some of these specifications in the DataPerfect manual. This chapter is for beginners and experienced DataPerfect application developers alike.

In **Fields: Introduction**, I discuss each field type available to the DataPerfect developer. Though this chapter is mainly directed at beginners, it contains information about DataPerfect fields many experienced DataPerfect application developers don't have.

In **Fields: Issues**, I take my discussion of fields a step further. I choose topics that will interest the beginner and the experienced, though the beginner will get lost fairly earlier in this chapter. Examples of issues covered include ways to optimize Keep A Total operations, how to know when not to use the date field for dates, solving complex elapsed-time problems, and how to decide between using different field formats.

In **Lookups**, I take you from the fundamental to the complex, from explaining how lookups work, to explaining the Smart Lookups algorithm. The latter, introduced with version 2.3, was never mentioned in any of its documentation. I also include a three-step approach to designing lookups so they work in almost all situations, given the logic of the Smart Lookups algorithm. I include a troubleshooting section at the end.

In **Indexes**, I show the beginner what indexes do and how to use them. I also graphically show the experienced DataPerfect application developer how indexes and index regeneration affect the inner working of an application's structure file. This will lay the basis for much of my discussion in a later chapter (**Application Maintenance Issues**), where I outline when it's okay to overwrite a client's structure file with a new one, without regenerating indexes.

In **Links**, I target both beginners and the experienced. I cover such fundamental topics as the four linking relationships of a relational database application, as well as the complex topic of recursive links. I outline reasons for abandoning the auto-incrementing field in favor of one that increments either absolutely or conditionally, using recursive links. I include a troubleshooting section at the end.

In **Keep A Total**, I target both beginners and the experienced. I go from discussing how to set up a Keep A Total operation, to how to use it to update records in foreign data files, and when to use Cascade Update instead.

In **Reports: Introduction**, I mainly target the beginner. But, as in other chapters, there are bits of information interspersed here that frequently surprise experienced DataPerfect developers.

In **Reports: Fields**, I target both the beginner and the experienced. Though I'm mainly concerned here with outlining the various Print Mode Indicators, I spend a lot of time on the complexities of using variable-length text fields in reports. I also show you how to get Print Mode Indicators to affect panel displays, even though they were never intended for that purpose.

In **Report Options**, I offer a straightforward delineation of all possible report options. Though I'm targeting beginners here, I find a few things I mention here surprise some experienced developers.

In **Report Variables**, I target both beginners and the experienced. It's a rather short chapter, with a few straightforward topics, like showing how to use Report Variables as counters and how and why to recycle the same Report Variable within the same report.

In **Subreports**, I'm targeting both beginners and the experienced. This chapter can be quite complex, replete with many report definition examples to demonstrate difficult points. I spend quite some time on what I call *dummy reports*, which allow the definer to create a single report that branches to other reports.

In **Printer Control Issues**, I target the experienced DataPerfect application developer. After spending some time on the new Open Filename in Report Variable option of DataPerfect 2.3c, I go right into the notion of a Printer Control Panel. The latter allows you to build your Report Definitions so that the client can run them on any printer they tell the application they're using. This chapter should lose the beginner.

In **Formulas**, I target both beginners and the experienced. For the experienced DataPerfect application developer, I picked what I consider to be frequently misunderstood and underutilized DataPerfect formula entities. These include CASES, APPLY.FORMAT, and SUBFIELD, to name a few. I include complex, fully commented formulas, as well as a troubleshooting section at the end.

In **Iteration Control**, I target experienced DataPerfect application developers. I'm always surprised when I find developers still relying only on Skip If Report Variable Is False as their only Iteration Control option. Version 2.3 introduced a much more robust library of Iteration Control options. Here I attempt to wean those developers away from that bad habit. I offer many example Report Definitions here, as well as a troubleshooting section at the end.

In **Export/Import**, I target both beginners and the experienced. I discuss all major export and import options available within DataPerfect, including caveats and strategies.

In **The Clipboard**, I target both beginners and the experienced. I discuss DataPerfect's internal clipboard, and contrast it with screen capture available in such products as Shell®, DESQview®, Windows®, and OS/2®. I include serious caveats that, in some cases, demand you use screen capture instead of DataPerfect's internal clipboard. Not heeding these caveats may result in an irretrievably corrupt structure file.

In **Securing the Application**, I target both beginners and the experienced. I discuss the various ways DataPerfect allows the developer to protect an application. This includes menu security, introduced with the initial release of version 2.3, and the User ID facility, introduced with the second release of version 2.3. This is where you find out how to control what each user is allowed to do in the database. I include a troubleshooting section at the end.

In **Securing Data Entry**, I mainly target the experienced DataPerfect application developer. There's a little in here for the beginner, but not much. I talk about such things as keeping a saved field from being edited, conditionally closing off panel links, field protection offered by the third party DataPerfect addon, DPMouse®, and using reports to control data entry.

In **Application Maintenance Utilities**, I target both beginners and the experienced. I discuss four utilities that shipped with DataPerfect: DPExport® and DPImport®, as well as DPOrder® and DPDiagnosics®. I include caveats about all four. I also preview STE Manager®, a third party utility that allows the experienced DataPerfect application developer to make certain types of changes to a structure file more conveniently than using DataPerfect itself.

In **Application Maintenance Issues**, I target the experienced DataPerfect application developer only. Here I discuss such issues as knowing when it's okay to overwrite an old .STR with a new one, and how to fix a corrupt .TXX file.

In **Epilogue**, I talk about DataPerfect's status, as of this writing. This includes how to get support for it, as well as a discussion of recent developments centering around creating a way to move DataPerfect applications into other platforms.

Products Mentioned

DataPerfect® and its allied utilities (DPExport®, DPImport®, DPOrder®, DPDiagnosics®, Shell®, and Editor®) are registered trademarks of Novell, Inc.

DESQview® is a registered trademark of Quarterdeck Office Systems.

DPMouse® is authored and owned by Steven Patamia. See accompanying diskette for more information.

OS/2® is a registered trademark of IBM Corporation.

PKZip® is a registered trademark of PKWARE, Inc.

Windows® is a registered trademark of Microsoft Corporation.

WordPerfect® is a registered trademark of Corel Corporation.

Legalese

This book's publisher and I assume no liability for any consequence, monetary or otherwise, arising from trying any of the techniques outlined here. Please backup all data before trying these techniques.

I wrote this short chapter to help you if you're new to DataPerfect. Here I get you started on your first DataPerfect database application. Please note that I included, on diskette, the reference manual that last shipped with DataPerfect. It's in text file format. Please refer to that often, as this chapter might not be enough for a beginner. Once you're used to browsing data in a DataPerfect application, load Tom Yuhas' DataPerfect Reference Manual application (also on diskette). It lets you browse the DataPerfect manual as though you're reading a book.

Getting Started

First make sure DP.EXE and DP.SYS are in the same directory, and that directory is in the PATH statement in your AUTOEXEC.BAT. If it wasn't in there, put it in with a text editor and reboot. Now create a new directory for your new application. Change to that directory and type *DP*, followed by hitting **Enter**. I'll call that directory E:\NEWAPP.

When running DataPerfect in a directory like E:\NEWAPP, where no previously defined DataPerfect application exists, DataPerfect initially greets you with this menu:

No databases are defined on this disk or directory.
—Choose one of the following—
1 - Create a New Database
2 - Change Directory
0 - Exit
—Selection: 0—

Pathname: E:\NEWAPP\

After hitting **1**, you see this:

Enter a name for the database.
The name is a filename with no extension. The name may specify the drive and/or path.

Pathname: E:\NEWAPP\

In the displayed prompt, enter a legal DOS filename without an extension (i.e., a string of up to eight characters DOS allows in filenames). At this point, many enter such names as PHONEBK, ACCTING, LEGAL, etc. Though such names help you remember what data this or that application holds, I advise you not to use

filenames that long. Instead, pick a two- or three-letter string that stands for that application. As you'll see later, this allows for much easier file maintenance at the DOS level.

Let's say you enter *NA* for your new application, and hit **Enter**. In response to this, DataPerfect creates three files behind the scenes:

```
NA.IND
NA.STR
NA.TXX
```

NA.IND is NewApp's index file, NA.STR its structure file, and NA.TXX the file that will hold variable-length text data (data found in what other database management systems call memo fields). More on these files later. All you need to know at this point is that you only supply DataPerfect with a filename that has no extension. DataPerfect supplies the extensions.

After DataPerfect creates these three files, it then asks you to enter this application's first *Panel Filename*. A few points about this filename:

- It may be any legal DOS filename, with any legal DOS extension (DataPerfect has a few reserved extensions, but that's not important. It'll tell you if you chose one of them).
- It names the file that will hold the data (other than variable-length text data) for a particular panel.
- I suggest you begin each such filename with the same two or three letters you chose for the database itself (in this case, *NA*), and give all such filenames the same extension. I use *.DAT* to signify data. So if this data file is to hold all Names in the application, you might name it *NANAMES.DAT*, or *NA-NAMES.DAT*, or *NA_NAMES.DAT*. This way, to copy, move, or backup *all* files associated with this application, you just use the a simple wildcard expression on your command line:

```
copy e:\newapp\na*.* a:
```

Along the same lines, if you're only copying the *data* files to drive a:

```
copy e:\newapp\na*.dat a:
```

I took this filenaming convention from Bruce Parello's wonderful out-of-print book, *The Theory and Practice of DataPerfect* (publ. WordPerfect Corporation 1989).

Okay, now back to the application. Let's say you chose *NANAMES.DAT* as your first panel filename. You now see this screen:

Define Panel
You can type text in the panel for field labels, or press

CREATE a Field (F9)
EDIT Field Format (F6)
Define Link for !PANEL (F5)

DEFINE an INDEX (Ctrl-F8)
DEFINE FIELD Options (Shift-F8)
DEFINE PANEL Options (Alt-F8)

Use Shift-Arrow keys to Change Panel Size
Move Panel -- F2

NANAMES.DAT-0

That's a blank panel waiting for you to give it some definition. After giving it fields, text, etc., it'll serve as a way for the user to view data in its particular data file (in this case, *NANAMES.DAT*), though it can be designed to let the user view data in related (linked) files as well.

Following the same naming convention, I might later attempt, in different directory, an accounting application. If I give it the two-character name *AC*, DataPerfect would then automatically create *AC.STR*, *AC.TXX* and *AC.IND*, waiting for me to create its various panels. I might use these panel filenames, keeping consistent with my filenaming convention:

Panel Filename	Data File Contents
ACTRAN.DAT	Transactions
ACCLIENT.DAT	Clients
ACVENDOR.DAT	Vendors
ACSTATE.DAT	State Codes

Let's go back to our *NA* application. Note the zero to the right of *NANAMES.DAT*, in this portion of the screen:

NANAMES.DAT-0

That's this panel's record counter. It displays the number of saved records in *NANAMES.DAT*. DataPerfect allows 16 million records per data file.

The next step is to add some fields, descriptive text, indexes, etc. It takes only three basic steps to create a logically complete panel:

- Give it a filename (you just did that).
- Create at least one field on which the cursor can land.
- Create at least one index.

That's all you need for a logically complete panel, but you'll probably want much more.

When you first give DataPerfect the filename for an application's panel, you're thrown into Define Panel mode (as indicated by *Define Panel* in the upper left corner of the screen below). Later, upon entering a panel you defined enough to consider it logically complete (see the three requirements above), you find yourself in Browse mode (as indicated by *BROWSING RECORD*), not Define Panel mode. At that point you can return to Define Panel mode by hitting **Alt-F8**. More on that later.

Let's return to the Define Panel screen. Here's the upper third of the screen, showing your menu options in that mode:

```
Define Panel
You can type text in the panel for field labels, or press

CREATE a Field (F9)                DEFINE an INDEX (Ctrl-F8)
EDIT Field Format (F6)             DEFINE FIELD Options (Shift-F8)
Define Link for !PANEL (F5)        DEFINE PANEL Options (Alt-F8)

Use Shift-Arrow keys to Change Panel Size  Move Panel -- F2
```

Let's detail the above choices.

Creating Panel Text

The first line that follows *Define Panel* refers to how to create what is typically referred to by DataPerfect developers as *panel text*:

```
You can type text in the panel for field labels.
```

Panel text in this case might look like this (which you can go ahead and type in if you want):

```
Define Panel
You can type text in the panel for field labels, or press

CREATE a Field (F9)                DEFINE an INDEX (Ctrl-F8)
EDIT Field Format (F6)             DEFINE FIELD Options (Shift-F8)
Define Link for !PANEL (F5)        DEFINE PANEL Options (Alt-F8)

Use Shift-Arrow keys to Change Panel Size  Move Panel -- F2

NANAMES.DAT-0
NAMES PANEL

    Last Name:
    First Name:
```

Sizing and Moving the Panel

The last line in the menu tells you how to size and move the panel:

Use Shift-Arrow keys to Change Panel Size Move Panel -- F2

The reference to *Shift-Arrow keys* is talking about holding down the **Shift** key while tapping on the *white Arrow* keys (the ones on your numeric pad). On a notebook computer, you don't have *white* arrow keys, so you have to resort to using **Alt** key equivalents:

Shift-Arrow Combination	Alt-Num Equivalent
Shift-Left	Alt-1
Shift-Right	Alt-2
Shift-Up	Alt-3
Shift-Down	Alt-4

The numbers referenced above in the Alt-Num Equivalent column are the ones you find on the top row of your keyboard, not your numeric pad.

That same line on the screen refers to the **F2** key. In Define Panel mode, **F2** is a toggle that alternates between the **Shift-Arrow** (or **Alt-Num**) keys *sizing* the panel on the one hand, and *moving* it on the other. See if you can get the Names Panel to look like this, using the **Shift-Arrow** (or **Alt-Num**) keys to both make the panel smaller and move it up a little:

Define Panel

You can type text in the panel for field labels, or press

CREATE a Field (F9)

EDIT Field Format (F6)

Define Link for !PANEL (F5)

DEFINE an INDEX (Ctrl-F8)

DEFINE FIELD Options (Shift-F8)

DEFINE PANEL Options (Alt-F8)

Use Shift-Arrow keys to Change Panel Size Move Panel -- F2

NANAMES.DAT-0

NAMES PANEL

Last Name:

First Name:

CREATE a Field (F9) and EDIT Field Format (F6)

Now you need a couple fields. To put a field in a panel while in Define Panel mode, move the cursor to the leftmost position of the desired field. To get the cursor there, use **Spacebar**, **Enter** and **Arrow** keys, just like you would in a word processor. Now hit **F9**. DataPerfect now prompts for the new field's format.

Though we'll get more into field formats later (see my **Fields: Introduction** chapter), let's put in a field format for our Last Name field and First Name field. Each

will be A15, which is an alphanumeric field that allows the user to enter letters, numbers and punctuation, up to fifteen characters (including spaces) in all. If we chose the format U15, we would be allowing the user to enter the same characters allowed by the A15 Field Format, but letters will automatically be stored and displayed in uppercase.

Each time you fill in a Field Format when creating a field, DataPerfect asks you for an optional Field Name. You can leave this blank, but I suggest you fill it in with *Last Name* and *First Name*, respectively. You can change this later if you want by cursoring back to the field in Define Panel mode, hitting **F6** and putting a different name.

Note: In Define Panel mode, you can cursor from field to field with **Tab**, but **Shift-Tab** won't let you go backwards. Depending on how you have things arranged on your screen, **Ctrl-RightArrow** and **Ctrl-LeftArrow** will allow cursoring from field to field in both directions.

After putting in two A15 fields, you have this:

Define Panel
You can type text in the panel for field labels, or press

CREATE a Field (F9)	DEFINE an INDEX (Ctrl-F8)
EDIT Field Format (F6)	DEFINE FIELD Options (Shift-F8)
Define Link for !PANEL (F5)	DEFINE PANEL Options (Alt-F8)

Use Shift-Arrow keys to Change Panel Size Move Panel -- F2

NANAMES.DAT-0
NAMES PANEL

Last Name: [REDACTED]
First Name: [REDACTED]

Exit Define Panel mode with **F7** or **F10**. Note that DataPerfect complains you haven't created at least one index for this panel. DataPerfect still lets you exit Define Panel mode, but the next time you enter that panel, DataPerfect forces you back into Define Panel mode, not letting you enter any data—at least not until you create at least one index in that panel. So let's go back into that panel (if you haven't already) and put in an index or two.

DEFINE an INDEX (Ctrl-F8)

An index fulfills two purposes:

- It provides a way to sort records
- It determines the parameters by which a record will be considered unique with respect to other records in the same panel.

Sorting

Let's take the first thing an index does: sorting. Consider an index to be a series of pointers that find records in a panel's data file in a particular order. Each pointer finds exactly one record in that panel. These pointers are determined by an ordered list of fields, called an *index field list*. Suppose our first index's field list consists of the Last Name and First Name fields, in that order, and our second index's field list consists of First Name and Last Name fields, in that order.

Now consider a report that uses the first index to process records in that panel. It will process records based on the Last Name field primarily, and the First Name field secondarily. Say the first Last Name it comes to is *Adams*, but there are two records with this Last Name: John Adams and Sally Adams. When this index sees two records with identical values in their Last Name field (the first field in the index field list), it secondarily sorts them by the First Name field (the second field in the index field list). So John Adams will precede Sally Adams in this index.

Uniqueness

The second thing an index does is determine what counts as a unique record in the panel. Remember I said each pointer finds exactly one record in that panel. DataPerfect won't allow two records to coexist in the same panel if those two records have identical values in all fields in the field list of at least one index in that panel. That's another way of saying DataPerfect won't allow an index pointer to point to more than one record.

In our Names Panel, our first index won't allow two records for John Adams. Neither will the second index. Each index will allow a record for John Adams and one for John Smith, even though they match on one field in each index field list. As long as they don't match on *all* fields in an index's field list, the index won't consider them duplicates. As long as *any* index in that panel considers a record to not be unique, DataPerfect will kick it out of that panel's data file. That is, if adding a record to a panel will cause some pointer in some index in that panel to point to more than one record, DataPerfect will refuse to accept that new record into that panel's data file. What record an index pointers point to, then, is determined by the index field list.

Defining, Editing and Deleting Indexes

To create, edit, or delete an index, whether in Define Panel mode or Browse mode, hit **Ctrl-F8**. To create an index (as we want to do now), choose option 1. To follow the logic outlined above, I want my first index to sort on Last Name. So I **Tab** to the Last Name field and select it with **F4**, and then **Tab** to the First Name field and select it with **F4**. After selecting all the fields I want in that index's field list (in this case, only two fields), I hit **F7** or **F10** to exit and save the index.

I want my next index to sort on First Name. I do the same as I just did above, starting with choosing option 1 to create an index, but this time I select the First Name field before selecting the Last Name field.

You might think that to get the first index to sort on Last Name, you should simply put only Last Name on its field list (and not include the First Name field). And conversely, you might make the second index include only the First Name field in its field list (and not include the Last Name field), causing it to sort on the First

Name field. But doing this would result in the first index allowing only one Jim, John, and Sally, etc. And the second index would allow only one Adams, Doe, and Smith. Not good.

These two different indexes will come in handy later, when accessing and processing data. Note that when you exit out of Define Index mode with **F7**, DataPerfect no longer complains. You now have a logically complete panel (that is, one that will allow you to browse, create, edit, or delete data) because you now satisfied the following three rules regarding panel creation:

- Give it a filename.
- Create at least one field on which the cursor can land.
- Create at least one index.

There's a lot more to say about indexes. I discuss them in painful detail in my **Indexes** chapter.

Field Formats and Field Format Display Modifiers

Let's talk about one of those three rules: *Create at least one field on which the cursor can land*. Can't the cursor land on any field you define here? No. Not if you're talking about modes other than Define Panel mode (Browse, Create, or Edit modes). In Define Panel mode, you can add a display modifier to a field's format, such that it keeps the user from ever landing on that field. Display modifiers always consist of a double-colon followed by a letter, and the modifier as a whole always follows the field's primary format. The three field format display modifiers that keep the user's cursor of the field are ::N, ::H, and ::C. Let's discuss this.

::N makes the field *non-updatable*. The user can see it, but can't land on it. For instance, we might have a Balance field in an Invoice Panel formatted G-**ZZZ,ZZ9.99::N**. As you'll see later, that's a numerical field that allows numbers from -999,999.99 to 999,999.99, but, because of the ::N modifier, doesn't allow the user to ever land on that field. If that field gets its value from totalling carried over from a different panel (the Transaction Panel, for instance), then you want to protect it from the user's fingers. You want the user to *see* it, not to *touch* it.

::H makes the field *hidden*. The user can't even *see* this field. It's there, but hidden from view. One reason you might want a hidden field is to create an index that sorts backwards with respect to a numerical field. Say you have a Customer ID Number field that's five digits long (G**99999**), and you want an index to sort so that it sees the highest Customer ID Number first instead of last. The typical way to do this is to create a G-**99999::H** field that has a formula on it that updates its value to the negative of that found in the Customer ID Number field. So whatever value is found in the Customer ID Number field for a record (say, 03445), its negative (-03445) will be stored in the hidden G-**99999::H** field. An index that sorts on the latter field will sort in reverse with respect to an index that sorts on the Customer ID Number field. There's really no reason to show the user that G-**99999::H** field, so we usually hide it with the ::H modifier. It's only there to allow the developer to create

indexes that need to sort a special way. See *Sorting Backwards with Reverse Indexes* in my **Indexes** chapter for more on reverse indexes.

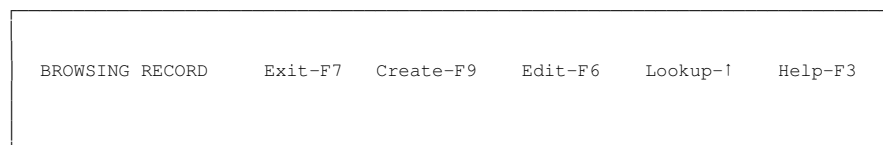
::C makes the field *computed*. Say we have a library application that keeps track of books people check out. In one panel, we want a U7 field to display the string *OVERDUE* when the date found in the Due Date field is earlier than today's date. Such a field would be a good candidate for the ::C display modifier (U7::C). Like the U7 field, the U7::C field will still handle up to seven uppercase characters. And like the U7::N field, it will display to the user without letting the user land on it (you can, however, hide it from the user if you format the field U7::CH).

So what's the difference between a U7::N field and a U7::C field? Well, for now, let's just say you can't put it in an index. We're going to have to talk about field formulas later for you to completely understand what a ::C field is. Essentially, a ::C field exists only on your screen. It has no representation in the panel's data file at all. Not so for the ::N field. It exists in the data file as well as the screen. Some of us say the ::N field is *real* and the ::C *virtual*. In other database programs, ::C fields are called *display* fields. DataPerfect calls them *computed* fields (before version 2.3, they were called *calculated* fields).

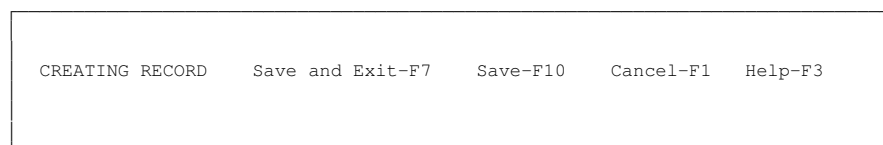
I go much more into field formats in the *Field Types* section of the **Fields: Introduction** chapter. Before I discuss the other options available in Define Panel mode, let's back off that a little and see what you've created so far. I'll cover the other options in Define Panel mode later.

Browse, Create and Edit Modes

If you haven't already entered Browse mode, do so by leaving Define Panel mode with **F7** or **F10**. You'll see the menu options change to this:



Now enter Create mode with **F9**. You see the menu options change to this:



Note: If you had entered Edit mode with **F6** instead of Create mode with **F9**, DataPerfect would have forced you into Create mode nonetheless, since there aren't any records in this panel to edit.

While in Create mode, type *Smith* in the Last Name field and hit **Tab**. Then type *John* in the First Name field. Now hit **F10** to save the record. You have your first record in the Names Panel's data file. Now hit **F6** and change *John* to *Jack* and save your changes with **F10**. That was Edit mode.

Lookups in Browse Mode

Hit **F9** and create another record, this time for Sally Adams, and save it with **F10**. While in Browse mode, cursor to the Last Name field and perform a lookup by hitting **Up Arrow**. Note that records are sorted by Last Name in the first column. As you type letters during this lookup, the highlight bar will go to the first record that starts with the string you've typed so far. If it doesn't find one, it will go to the next one in the index being used to sort the records.

So if this panel has a record for Jack Smiley and one for Jack Smith, and Jack Smith is the first record beginning with *SM* in the Last Name index (the index with Last Name as the first field in its field list), then typing *S* will cause the highlight bar to move to the first record with *S* starting its Last Name field value. Following this by typing *M* will cause the highlight bar to move again, this time to the first record with *SM* starting its Last Name field, which, in this case, is Jack Smiley. I call this lookup facility *type-to-search lookups*. As far as I know, DataPerfect was the first database program to offer this without the definer having to write many lines of code.

While still in Browse mode, cursor to the First Name field and perform a lookup there with **Up Arrow**. Note the difference in the lookup display. It now sorts on First Name, and typing in letters moves the highlight bar closer and closer to the First Name being typed.

Go ahead and create more records in this panel's data file and see how the lookup facility works. It's what I consider to be DataPerfect's most powerful feature. On a network with many users accessing the same database, in a panel with hundreds of thousands of records, a lookup search during Browse mode still only takes a second or two, even if typing in characters that take you to the last record in the index that's sorting that lookup display.

Remaining Define Panel Menu Options

Okay, back to Define Panel mode to discuss the remaining options available in that mode. In Browse mode, return to Define Panel mode by hitting **Alt-F8**:

```
—Define Panel—
  You can type text in the panel for field labels, or press

CREATE a Field (F9)                DEFINE an INDEX (Ctrl-F8)
EDIT Field Format (F6)             DEFINE FIELD Options (Shift-F8)
Define Link for !PANEL (F5)        DEFINE PANEL Options (Alt-F8)

—Use Shift-Arrow keys to Change Panel Size   Move Panel -- F2—
```

DEFINE PANEL Options (Alt-F8)

Note the last menu option above called *DEFINE PANEL Options (Alt-F8)*. Calling the Define Panel Options submenu with **Alt-F8** gives us this menu:

```
Panel Options
1 - Edit Filename          5 - Change Edit Order
2 - Change Color          6 - Edit Panel Name
3 - Auto-Save (Y/N)      7 - Recompute Field Offsets
4 - Auto-Display Record (Y/N) 8 - Auto-Edit/Auto-Create/Menu
Browse Change => Auto-Edit
Display each record during Lookup.
Selection: 0
```

Options 1 and 6: Edit Filename and Edit Panel Name

Sorry, but I need you to exit out of Define Panel mode to see the significance of these two options. **F7** back to Browse mode, and then again to a display like this:

```
Note
A list of panels is displayed. Use ↑ and ↓ to highlight the
desired panel, then press Enter. To define a new panel,
press Insert (Ins).
Press Delete (Del) to delete a panel.

--Beginning--
NANAMES.DAT
--End--
```

That's the *Panel List*. On this particular Panel List you see only one panel: NANAMES.DAT. It bears the name of its data file. That's not a very interesting name to look at, so DataPerfect allows you to change the name that appears on this Panel List without altering the DOS name assigned to the data file itself. To do this, go back into the that panel by hitting **Enter** while the highlight bar is on its Panel List name (NANAMES.DAT). Now return to Define Panel mode with **Alt-F8**, and to Panel Options with **Alt-F8** again.

Options 1 and 6 on the Panel Options menu (Edit Filename and Edit Panel Name) seem similar, but they're radically different. Choosing Edit Filename (option 1) is choosing to change the DOS name you previously assigned to its data file (the file that holds all the saved records in this panel). That's a radical change to a database, and DataPerfect won't let you do that with any data in the panel. Also, if you decide to rename the panel's data file with option 1, DataPerfect actually just creates a new file with the new name and leaves the old in that directory.

Choosing Edit Panel Name (option 6) is choosing to change that panel's name on the Panel List. Hit **6** and tell DataPerfect what name you'd like to appear on the Panel List for this panel. So you and I are consistent here, let's both choose the name *Names Panel*. Now **F7** to exit the Panel Options menu. **F7** again to exit the Define Panel menu. **F7** one more time to exit the panel itself. Now you're in the Panel List and see this:

—Note—

A list of panels is displayed. Use ↑ and ↓ to highlight the desired panel, then press Enter. To define a new panel, press Insert (Ins). Press Delete (Del) to delete a panel.

--Beginning--
Names Panel
--End--

That was option 6 (Edit Panel Name). Though we used that option to change this panel's Panel Name to *Names Panel*, its *data file* still has the DOS name NANAMES.DAT. You can either go to DOS to see this, or exit back to Browse mode again and load the panel by hitting **Enter**. Note that the DOS name still appears in upper left corner of the panel:

└─NANAMES.DAT—0—┐

Of course, the number to the right of NANAMES.DAT might be different in your panel because you might have a different number of saved records in it.

Option 2: Change Color

Explore this on your own. It needs no explanation.

Option 3: Auto-Save

I never use the Auto-Save option, but many other developers love it. It causes DataPerfect to save the record the user is currently creating or editing after hitting **Tab** or **Enter** on the last field in the Edit Order (Edit Order is explained later, under Option 5). The default here is to have that feature off. If you hit **3** you'll see an indicator appear, telling you that feature is on. Hit it again, and that indicator disappears. It's a toggle.

Option 4: Auto-Display Record

This option is on by default. Hit **4** and you'll see the indicator disappear just below it. Like **3**, it's a toggle. If you have a few records in this panel, turn Auto-Display on if it isn't already, and exit the Panel Options menu with **F7**, and exit the Define Panel menu with **F7** again. In Browse mode, perform a lookup with **Up Arrow**. Note that as you move the highlight bar up and down, the record in the panel itself changes to match the one on which the highlight bar sits in the lookup display. Return to Panel Options mode by getting out of the lookup with **Esc**, **F7**, or **Enter**, and then hitting **Alt-F8** twice. Turn off Auto-Display by hitting **4**. Now return to Browse mode and test your lookups again. Note the panel no longer updates to follow the highlight bar. Most developers keep Auto-Display on. You would consider turning it off on slow machines or when the current panel has many computed fields (:C fields), as computed fields tend to slow down the display.

Option 5: Change Edit Order

Edit Order is important. We'll be coming back to this many times, under different themes. Edit Order is the order of fields the cursor follows when the user hits the **Tab** key. You can change the Edit Order whenever you want, even with data in the panel. So you can create fields in this panel in any order, and later change the Edit Order when that's important. Go ahead and access the Change Edit Order option by hitting **5** when the Panel Options menu is active. If you created the Last Name field before you created the First Name field, you'll see this:

```
-Edit Order-
The numbers to the left of each field below show the edit order
for the fields in the panel.  If you want to change this order,
move to the field you want to reorder and enter the new number
for that field.  The edit order is adjusted automatically.

When finished, press Save or Exit.
```

```
-NANAMES.DAT-0-
NAMES PANEL

Last Name: 1
First Name: 2
```

Go ahead and change the Edit Order by cursoring to each field and inserting the correct number in, hitting **Tab** or **Enter** after each insertion, so it looks like this:

```
-NANAMES.DAT-0-
NAMES PANEL

Last Name: 2
First Name: 1
```

The above will be the order the cursor moves through the panel during Browse, Edit, or Create modes, while hitting **Tab**. Exit the Edit Order menu with **F7**, the Panel Options menu with **F7**, and the Define Panel menu with **F7**. Check out the Edit Order now by hitting **Tab** a few times.

Option 6: Edit Panel Name

I covered this with option 1 (Edit Filename).

Option 7: Recompute Field Offsets

This is supposed to increase the efficiency of your database if you've deleted some fields in it. I've never used this, so can't report how much it actually increases efficiency. What it does, essentially, is redefine where the .STR file expects to physically find each field in a panel's data file. You can't perform this operation with data in the database.

Option 8: Auto-Edit/Auto-Create/Menu

This toggles between the following three options:

```
Browse Change => Auto-Edit
Browse Change => Auto-Create
```

Browse Change => Menu

Each time you hit **8**, the indicator to the left (under option 4 on the screen) changes. If you choose *Auto-Edit*, then when the user is in Browse mode and types a character (probably inadvertently), he'll see this menu:

—Note—	
You have changed a field in this record.	
—Do you want to—	
1 -	Create a New Record
2 -	Edit the Displayed Record
0 -	Cancel the Change
—Selection: 0—	

I always use this option (*Menu*) to safeguard against accidental editing. *Auto-Edit* does something different. If that option's active, inadvertent editing during Browse mode will cause the record to go immediately into Edit mode. *Auto-Create* will result in inadvertent Browse-mode editing causing the record to immediately go into Create mode.

Define Link for Panel (F5)

While in Define Panel mode, hitting **F5** will allow you to define a link. If the cursor isn't on a field when you press **F5**, you end up defining a *panel* link. If the cursor *is* on a field, you create a *data* link. In either case, DataPerfect asks you for four things:

- Target panel
- Target panel field to land on
- Target panel index
- Source panel field list

I go over links in painful detail in my **Links** chapter.

DEFINE an INDEX (Ctrl-F8)

Unlike *Create Field (F9)* and *Edit Field Format (F6)*, *Define Index (Ctrl-F8)* can be called from both Define Panel mode *and* Browse mode. Again, here's the Define Index menu:

—Define Index—	
1 -	Create Index
2 -	Edit Index Field List
3 -	Create/Edit Exception List
4 -	Delete Index
0 -	Exit
—Selection: 0—	

Option 1: Create Index

We covered this.

Option 2: Edit Index Field List

This lets you edit an exiting index's field list (to add fields to it or remove fields from it). After calling this option with **2**, you use **Up Arrow** or **Down Arrow** to find the index you want to edit. When you find the one you're looking for, you select it with **F4**. The subsequent Define Index Key Field List menu is self-explanatory.

Option 3: Create/Edit Exception List

Let's skip this for now. It's important, but is really something best discussed when we talk about more advance things. See the *Exception Lists* section in my **Indexes** chapter.

Option 4: Delete Index

This is obvious. To delete an index definition, hit **4**, then use **Up Arrow** or **Down Arrow** to find the index to delete. Select it for Deletion with **F4**.

DEFINE FIELD Options (Shift-F8) 1, 2 and 4

Depending on the format of the field on which your cursor sits, you get a different menu with **Shift-F8** (again, I discuss all possible field formats and their Display Mode Modifiers in my **Fields: Introduction** chapter). In this section, I'll go over their listed options, one at a time, after I list all the possible Define Field Options menus here for you. Note the upper border of each Define Field Options menu. That tells you what sort of field you were on when you called the menu with **Shift-F8**:

```
Options for Alphanumeric and Text Fields
1 - Lookup Field List          5 - Range Check
2 - Initial Formula           6 - Validation Time (Edit/Save)
3 - Initial Value             7 - Define Search Field List
4 - Initialize at Create/Save/Change 0 - Exit
Selection: 0
```

```
Field Options for Numeric Fields
1 - Lookup Field List          6 - Validation Time (Edit/Save)
2 - Initial Formula           7 - Define Search Field List
3 - Initial Value             8 - Keep a Total
4 - Initialize at Create/Save/Change 9 - Remove Last Total
5 - Range Check               0 - Exit
Selection: 0
```

```
Options for Auto-Incrementing Fields
1 - Lookup Field List
2 - Set Value for Next Created Record
0 - Exit
Selection: 0
```

```
Field Options for Computed Fields
1 - Lookup Field List
2 - Define Field Formula
0 - Exit
The formula value will be computed whenever displayed or accessed.
Selection: 0
```

About those menus: An *alphanumeric* field (first menu) is one that accepts both letters and numbers. That would be an A or a U field. A *numeric* field (second menu) is one that accepts only numbers. That would be a G, H, N, D, or T field. An *auto-incrementing* field (third menu) is a numeric field (G, H, or N) that has the ::I or ::J modifier, like G999::I or GZZ9::J. A *computed* field (fourth menu) is one with the ::C modifier, and can be either alphanumeric (A or U) or numeric (G, H, N, D, or T). For example, A15::C or G999::C. Again, I discuss all possible field formats and their Display Mode Modifiers in my **Fields: Introduction** chapter.

Option 1: Lookup Field List

All the above menus have this option. A field's lookup definition always includes a Lookup Field List, and may also include a Lookup Index. Choosing option 1 starts this lookup definition process by asking you for this field's Lookup Field List. This list consists of the fields that display when the user performs a lookup on this field during Browse mode. After selecting each field with **F4** and then Saving the List with **F7** or **F10**, you're then offered a chance to assign an index to this lookup definition. If you want to assign an index, use **Up Arrow** or **Down Arrow** to find the desired index and then select it with **F4**. Otherwise, use **F7** or **F10** to save the lookup definition *without* an index.

As a general rule, think of the Lookup Field List as consisting of the fields the user sees during a Browse mode lookup on this field, and the Lookup Index the way it'll sort. This usually means the first field in the lookup Field List will be the first field in Lookup Index, but not always. Lookups can be a very complicated topic, so I devote an entire chapter to it (**Lookups**).

Option 2: Define Initial Formula or Define Field Formula

This is almost the same for all the above menus. For alphanumeric and numeric fields, option 2 is *Initial Formula*; and for computed fields, it's *Define Field Formula*. These are essentially the same thing. When you choose option 2 for such fields, you're offered the following Specify Formula menu:

Specify Formula

Operands can be numbers, character strings in quotes (" or '), or fields selected with Select (F4). To select a field from another panel, move to a link and press !.

Operators are: + - * / // ^ < > <= >= <> = NOT AND OR.

Parentheses () should be used to group items.

When finished, press Save (F10) or Cancel (F1).

For all intents and purposes, what you're doing here is telling DataPerfect what you want to be the default value of this field, or what will be the value of this field whenever a change is made anywhere in this record. To accomplish this, you type in a formula you want to update either on record creation or whenever a change is made to the record. If the formula is based on another field, you select the other field with **F4** to put it in the formula. When you hit **F4**, DataPerfect gives you a view of the current panel and lets you cursor to the field you need in the formula. When you land on that field, you just hit **F4** again. DataPerfect then throws you back into the Specify Formula screen, with the field you selected showing up there as a string, like *P1F3* (Panel 1 Field 3). Try it.

For instance, suppose the field in question is a U2 field in which the user is to type in a two-letter State Code, like *CA* for California, or *ID* for Idaho. If this application is going to be used only in Los Angeles, and it's used to enter names and addresses of customers in a local video store, you can safely assume that almost all such records will have *CA* in the State Code field. So you give that field a formula like this in the Specify Formula screen:

```

Specify Formula
Operands can be numbers, character strings in quotes (" or '), or fields
selected with Select (F4). To select a field from another panel, move
to a link and press !.
Operators are: + - * / // ^ < > <= >= <> = NOT AND OR.
Parentheses () should be used to group items.
When finished, press Save (F10) or Cancel (F1).

```

"CA"

I know it doesn't look like a formula, but it is. After hitting **F7** to exit the Specify Formula screen, you then decide on the conditions for this formula to update. Note the bottom of the screen that greets you when you exit the Specify Formula screen:

```

Options for Alphanumeric and Text Fields
1 - Lookup Field List          5 - Range Check
2 - Initial Formula           6 - Validation Time (Edit/Save)
3 - Initial Value             7 - Define Search Field List
4 - Initialize at Create/Save/Change 0 - Exit
Selection: 0
Field 1 of panel 1 Field Format: A15

```

Automatically computed when record is created.

Continue to Option 4 to complete this.

Option 4: Initialize at Create/Save/Change

This comes into play in the above screen. The indicator at the bottom of the screen tells you when the formula will update, and it's controlled by Option 4. The default is what you see above: *Automatically computed when record is created*. When you hit **4**, it toggles between these formula update conditions:

- Automatically computed when record is created.
- Automatically computed when created record is saved.
- Automatically computed at any change and when record is saved.
- [blank]

Automatically computed when record created is the one you'd use if you just want to set an initial value (default value) for the field. That's what we would want

in our video store application. We want the user to be able to override the value if they want, but not have to type it in if it's CA.

Automatically computed when created record is saved is used if you need to make sure the field formula is recomputed when the record is saved. This usually comes up in network situations when you want to make sure the field formula checks to see if someone else on the network already saved a record with the value computed by the formula. This comes into play in a crucial way in my discussion of recursive links in the *The Recursive Link* section of my **Links** chapter (*Absolute Incrementation Using a Recursive Panel Link on a Network*).

Automatically computed at any change and when record is saved is a formula update condition you'll use frequently. As a rule, you're going to either have a formula update *on any change* (this condition), or *on creation* (the default condition). When a formula is set to update *on any change*, it'll recompute as soon as the record is put into Create (**F9**) or Edit (**F6**), and every time a value is entered in *any* field or edited in *any* field. In our video store example, if we set the State Code field to update to CA *on any change* instead of *on creation*, any attempt by the user to alter that value (like putting in ID) will be changed to CA.

Why might you want a field formula to update *on any change*? Well, suppose you have a Date of Birth field in your panel, and want another field to display the person's age, based on what was entered in the Date of Birth field. You would want the Age field to update *on any change*, not *on creation*. This allows the user to enter a different Date of Birth value later, when he finds out he made a mistake, and have the Age field update accordingly. If the Age field only updated *on creation*, the new value entered in the Date of Birth field *in Edit mode* would have no effect on the Age field.

The *blank* option looks like it offers you nothing. Go ahead and toggle **4** a few times and see that one of the options is blank. The *blank* option is something developers don't usually use, mainly because they don't know what it does. If the update condition is blank, the field will update when, and only when, the user hits **F6** while the cursor is on that field during Create or Edit modes.

That almost covers Define Field Option 2 (Define Field Formula). I need to add only two points:

Computed Fields (::C)

If you're on a computed field when hitting **Shift-F8**, you don't get to decide on conditions that cause the formula to update. It updates on any change, and whenever the record is displayed. There's an important distinction between a formula that updates *on any change* on a *noncomputed* field and one that updates *on any change* on a *computed* field. I discuss this in detail later (see *Choosing Between ::C and ::N Fields* in my **Fields: Issues** chapter). Suffice it to say for now that a computed field updates *on any change and whenever the record is displayed*. You have no option 4 to set the formula update conditions for computed fields.

Auto-incrementing Fields (::I or ::J)

If you're on an auto-incrementing field when hitting **Shift-F8**, option 2 isn't *Define Field Formula*. Rather, it's *Set Value for Next Created Record*. This makes sense when you think about it.

DEFINE FIELD Options (Shift-F8) 3, 5, 6-9

Here are the two Field Options menus that have these four options:

Options for Alphanumeric and Text Fields	
1 - Lookup Field List	5 - Range Check
2 - Initial Formula	6 - Validation Time (Edit/Save)
3 - Initial Value	7 - Define Search Field List
4 - Initialize at Create/Save/Change	0 - Exit

Selection: 0

Field Options for Numeric Fields	
1 - Lookup Field List	6 - Validation Time (Edit/Save)
2 - Initial Formula	7 - Define Search Field List
3 - Initial Value	8 - Keep a Total
4 - Initialize at Create/Save/Change	9 - Remove Last Total
5 - Range Check	0 - Exit

Selection: 0

Option 3: Initial Value

I never use this. It's no different than simply using *Initial Formula* (option 2) with initialization (option 4) set to *on creation*.

Field Option 5: Range Check

This allows you to define the lowest and highest possible value for the field. This comes in handy when you want to limit the user to, say, 1 through 4 in a particular G9 field; or A, B, and C in a U1 field. Otherwise a user may enter *any* number in the G9 field (0-9), and *any* character in the U1 field (0-9, A-Z, and punctuation characters).

Field Option 6: Validation Time

This determines when the *Range Check* (Option 5) will be examined. You can choose between it being examined on Save or Edit. Hitting **6** toggles between *On Save*, *On Edit*, or *blank*. When blank, the Range Check is useless. Otherwise, you're telling DataPerfect when you want the user to receive the *This field value is not allowed* error message: as soon as he leaves the field with the out-of-range data (On Edit), or as soon as he attempts to save the record with out-of-range data in that field (On Save).

Field Option 7: Define Search Field List

This is an interesting option, but to understand it you have to first examine DataPerfect's search facility, which is pretty unintuitive. Suffice it to say here that when you create a Search Field List for a particular field with option 7, you're telling DataPerfect that when user issues a Search (**F2**) on this field, all the other fields on its Search Field List are to be searched as well.

Field Option 8: Keep A Total

This option is offered only for numerical fields. To use it, the panel in which the field resides must have a link to another panel that contains yet another numerical field. Links are discussed elsewhere (my **Links** chapter), but assuming you know what they are, option 8 allows you to have DataPerfect either Add or Subtract the value in this field to a designated target field in the linked panel. To define a *Keep A Total*, you tell DataPerfect three things, in this order:

- Whether you want this field's value Added to or Subtracted from the target numerical field.
- What link you want this Keep A Total to use to find the target numerical field.
- What field in the link's target panel you want to Add this field's value to, or Subtract it from.

I go over Keep A Total in more detail in my **Keep A Total** chapter.

Field Option 9: Remove Last Total

If, for some reason, you want to delete the last Keep a Total you defined for this field (Option 8), use option 9. Be careful. It'll delete that Keep A Total definition without prompting you.

Though this chapter should interest experienced DataPerfect application developers, it should also interest users of other database management systems. It explains the various files used to run a DataPerfect application, including DataPerfect's two flavors of runtime executables. Some of the file specifications mentioned at the end of this chapter aren't found in the DataPerfect manual.

Program Files

To create and run a DataPerfect application, you only need two files:

```
DP.EXE  
DP.SYS
```

These files are part of the Developer version of DataPerfect 2.3. You may, however, want to distribute your application with a runtime version of DataPerfect. There are two such versions available. One is the Single User Runtime DataPerfect (DPDRT.EXE and DRDRT.SYS), which allows only one user to access the same database at any time, and keeps all users from altering the application itself.

There's also a Network Runtime DataPerfect (DPRUN.EXE and DPRUN.SYS), which allows up to 9,999 users to access the same database simultaneously on the same network. Like the full Developer version, it allows multiple users to even edit the same record simultaneously. Unlike the Developer version, this version won't allow users to alter the application itself.

Application Files

The .STR File

This is the application's *structure* file. You name it when you first create your application, though you may not supply its filename extension (DataPerfect insists on *.STR*). The structure file determines how the application works. It's what you're creating or modifying when you do things in Panel Define mode. It's the only essential file of the application. It's all you need to run a DataPerfect application, besides the DataPerfect program files. If DataPerfect doesn't find the other application files (*.IND*, *.TXX* and data files) upon loading the application, it will prompt you to create them.

The .IND File

You have little control over the name assigned to this file. DataPerfect will look at the name you gave your .STR file and use that, coupled with the .IND extension. This is the application's *index* file. It tells the .STR the location of each record in the various data files. DataPerfect creates but one index file per application, covering all the data files. It also sorts the records of the various data files in ways defined by the developer of the application. For instance, an application might contain a Names Panel. This panel would have a one-to-one relationship with a data file—probably one the definer designated to contain Names. The index file for this application would most likely contain, among its various other indexes, an index that sorts records in the Name Panel by the Last Name field, and one that sorts those very same records by the First Name field.

So the index file consists of all the indexes for the application. An index is a kind of map of a panel's records. It allows reports to send records in a certain order to the printer, and lookups to display records in a certain order to the user's eyes. Like the index of a book, it's sorted alphanumerically, but with a twist. Since a record consists of fields, an index sorts records by a certain field primarily, another field secondarily, etc.

DataPerfect limits your application's index file to eight gigabytes.

The .TXX File

Like the index file, you have little control over the name assigned to this file. DataPerfect will look at the name you gave your .STR file and use that, coupled with the .TXX extension. The .TXX file holds all data found in an application's variable-length text fields. There's only one .TXX file per application, whether that application has hundreds of variable-length text fields, or none. A variable-length text field has the following properties not shared by other DataPerfect fields:

- It may occupy more than one line on the screen
- It allows the user to enter more data than can be seen in its actual screen display. That is, it scrolls during data entry to accommodate more and more data. So a variable-length text field that occupies but three lines on the screen will allow many more lines than that during data entry.
- Each variable-length text field allows 64,000 characters.
- None of the data entered in a variable-length text field is stored in its panel's data file. Rather, it's stored in a common storage area: the .TXX file.

The Data Files

Except for a very simple application, an application will have many data files. Perhaps one for Invoices, one for Transactions, one for Customers, etc. The data file is the only one of the files I mention here that doesn't have a hardwired extension. That is, you, the developer, may choose its complete filename, including its extension. I discuss strategies for naming data files (and .STR files, for that matter) in the **For Beginners** chapter (see *Getting Started*).

There's a one-to-one correspondence between a panel and a data file. A DataPerfect application is limited to 99 panels, and therefore 99 data files.

The .TMP Files

Every time you load an application, DataPerfect creates three temporary files with the extension .TMP. They take on the following patterns, where #### stands for a four-number string:

```
DP{I####.TMP
DP{S####.TMP
DP{T####.TMP
```

Each network user loading that application gets a different trio of temporary files, all stored in the directory where DataPerfect was loaded, or where directed by the

```
/d=[directory]
```

startup option. All the temporary files in a given trio share the same four-number string. Each user gets a different randomly generated four-number string. Needless to say, you, the developer, have no control over these filenames.

DP{I####.TMP is a temporary index file. It has the same format as the .IND file. DataPerfect uses this only to hold the index generated for a Two Level Report when that report needs a generated index.

DP{S####.TMP is a temporary scratch file. It's just an output stream, so has no specified format. DataPerfect uses it to hold report output information during its label routine and to facilitate the Conditional Page Eject feature.

DP{T####.TMP is a temporary text file. It has the same format as the .TXX file and .STR file. DataPerfect uses it for many purposes, holding all kinds of information. Here's a partial list:

- Fast lookup information
- Formula cascade information
- Report variable values
- Copied data for copy/paste operations

Upon exiting, DataPerfect always deletes the temporary files unless the system crashes. However, if DataPerfect aborts due to some error, it still deletes the temporary files.

Temporary files can accumulate on networks when the temporary files are assigned to a shared drive. This is a poor practice. Instead of allowing that to happen, if you're running an application on a network, use the /d start-up option to direct the temporary files to a local drive. This also improves performance.

Specifications

Here's a table outlining DataPerfect's various specifications:

DataPerfect Specifications		
	<i>Number</i>	<i>Size</i>
.STR file	1 per application	No limit determined yet
.IND file	1 per application	8 gigabytes
.TXX file	1 per application	534,773,728 bytes
Panels	99 per application	2 billion bytes each
Data Files	99 per application	2 billion bytes each
Fields	125 per panel (or data file)	Numeric: 14 digits Fixed-length: 78 characters Variable-length: 64k characters
Records	16 million per panel	
Indexes	200 per panel (or data file)	2 billion bytes
Open Files	Browse: 19 Create/Edit: 14	
Reports	169 per application	64,000 bytes
Report Variables	255 per report	
Users on Network	9,999 editing same record	

Most of this chapter discusses pretty fundamental stuff. However, in what appear to be very basic sections, I included a few points that the experienced DataPerfect application developer should find interesting. I suggest perusing the entire chapter, experienced or not.

Field Fundamentals

A field is the most elemental unit of data organization in a database. Data is essentially organized in a database via three entities: the field, the record, and the data file. In DataPerfect, of course, there's a one-to-one relationship between a data file and a panel. A panel (or data file) is a group of records (Students), and a record is a group of fields (Last Name, First Name, Date of Birth, etc.). In some database management systems, you'll see data files referred to as tables, records as rows, and fields as columns.

DataPerfect allows 125 fields per panel (or data file), providing a structure for data entry. Each field has a format (numerical, alphanumeric, date, etc.) which determines what sort of data it will accept.

When you, the definer, create a field, DataPerfect automatically gives it a *field code* which references that field's panel number and field number for referencing it in reports and formulas. For instance, if you see

P2F12

in a formula, that's the field code for field 12 in panel 2. You have no control over what a field's field number will be, and therefore its field code. DataPerfect always assigns a new field the lowest possible field number available for that panel, so deleting a field allows its number (and, of course, its field code) to be reused when creating another field later.

With the exception of the computed field (explained later), all DataPerfect fields are *real*. A real field is one that stores its data somewhere in the application's data files. A computed field's data, however, exists nowhere else but on the screen. Further, with the exception of the variable-length text field (also explained later), real fields store their data in their panel's data file. Variable-length text fields, however, store their data in a different file: the .TXX file. That file holds all data from all variable-length text fields in all panels in a single application.

Field Codes and Field Names

Don't confuse a field's field *code* with its field *name*. A field may or may not have a field name. This is up to the definer. When creating a field, DataPerfect always asks you for its format and name. Though you must give it a format, you need not give it a name. Further, if you refrain from giving a field a name during its creation, you can return later to give it a name (or change its name) in Define Panel mode. To do that, all you have to do is cursor to the field and hit **F6**. There you'll be offered another chance at editing its format and name.

Differences between field codes and field names arise in different areas. During a Quick Merge report (**Ctrl-F9** while in Browse mode), DataPerfect will recognize field *names*. For each field you chose for the Quick Merge report creation process, DataPerfect will show you that field's field name and ask you if you want that name in the resulting merge report. Also, when selecting a field with **F4** in a Specify Formula screen (while creating or editing a formula, either for a panel field, a Report Variable, or a Search Condition), the resulting code that DataPerfect places on the Specify Formula screen is that field's field *code*, not field name. But when the cursor rests on a selected field in a Specify Formula screen, you can see its field *name* (if it has one) on the bottom line of the Help window in the upper third of the screen.

To facilitate debugging and future development, name fields strategically. Consider including field numbers in names, as well as detailed descriptions. For example, in Define Panel mode, with the cursor on field 21 in Panel 3, I would usually give that field a name like the following:

```
F21 - Last Name in Customer Info Panel
```

Field Types

Alphanumeric Fields: A, U, and Variable-Length Text

A and U Fields

These accept and hold any ASCII characters. A and U fields are essentially the same, except that a U field converts all data entry to uppercase. A and U fields are fixed-length fields with a 78-character limit. An A23 or U23 field, for instance, allows data entry of up to 23 characters, including spaces. You can have an A78 field, but an A79 field is illegal. The DataPerfect panel allows up to 78 columns, thus the fixed-length alphanumeric field format limit of 78 characters.

Variable-Length Text Fields

These are important variations of A and U fields. A Variable-Length Text field allows more than one line to display in that field, and, as with a traditional word processor, data entry is wrapped automatically with soft returns. The Variable-Length Text field format consists of either a double-A entry, like A23A10, or a combination of A and U, like U23A10. Both of these examples produce a field that displays 23 columns and 10 rows of data, but the second one converts all characters to uppercase

as they're typed. The second component is always an A, whereas the first may be A or U.

A Variable-Length Text field allows up to 64,000 characters of data entry, and such data can occupy more space than is displayed. That is, the user can enter more than 10 lines of data in an A23A10 or U23A10 field because DataPerfect will scroll the field allowing for continuous data entry until the 64k limit is reached. In Browse mode, if not all the text in the field can be displayed in the given Variable-Length Text field format, DataPerfect displays an downward pointing arrowhead in the lower right corner of the text field, indicating additional, yet undisplayed, text is held in this field. You can view the additional text by scrolling with the **Arrow** keys while in the Variable-Length Text field in Browse, Edit, or Create mode. The display limit of a Variable-Length Text field is 78 columns and 15 rows, so the largest format allowed would be either A78A15 or U78A15.

Unlike fixed-length alphanumeric fields (A and U), the Variable-Length Text field not only accepts letters and numbers during data entry, but also bolding (**F6**), underlining (**F8**), and carriage returns. Also unlike fixed-length alphanumeric fields, Variable-Length Text field data isn't stored in its panel's data file. Rather, it's stored in that panel's .TXX file. Since there's only one .TXX file, all Variable-Length Text field data from all panels in a single application is found in that single .TXX file. Though the .TXX file does have a size limit, it's pretty large: 534,773,728 bytes.

In an Edit Report Form screen, if the second A component of a Variable-Length Text field format is zero, like A23A0 or U23A0, the entire field is printed. Changing the zero to a number specifies how many lines should be printed. For instance, a report field with format A23A5 prints five lines of the field. Changing the report field format's first component determines how wide the output will be. For instance, changing A23A0 to A20A0 prints 20 characters per line. However, this doesn't result in truncating the data. It essentially changes the margins of that output. See *Variable-Length Text Fields in Reports* in my **Reports: Fields** chapter for more details on this.

Numeric Fields: N, G, H, F, D, and T

N Fields

These are left-aligned numeric fields, and shouldn't be used in formula arithmetic calculations. Actually, you can use them in formula arithmetic calculations as long as none of the N fields in the formula have decimal points, and the formula itself isn't assigned to an N field that has a decimal point. If this confuses you, just don't put N fields in formula arithmetic calculations.

N fields take 9s to stand for digits in the format. For instance,

N9999

will take numbers containing from one to four digits. However, please note that, though you can create an N field with a minus sign, the value it holds isn't really a negative number. For instance,

N-9999

will always display the negative sign. When the user enters, say, 56 in that field, the field will display and store the following:

-5600

If you create a second N-9999 with a field formula that adds 1 to the first N-9999 field, the second N-9999 field will display and store

-5601

which is *not* the equal to

-5600 + 1

Also, if a third field is formatted G-9999 (see the next section for a discussion of G fields) instead of N-9999, and its field formula also adds 1 to the value found in the first N-9999 field, it will also display and store

-5601

Again, this is *not* equal to

-5600 + 1

All this results from the fact that N fields aren't really supposed to be used in arithmetic formulas. Think of them as A or U fields that take numerical characters.

G and H Fields

These are decimal (right-aligned) numeric fields, and can be freely used in formula arithmetic calculations. When its value is zero, the G field displays and stores zeros, whereas the H field displays and stores nothing (a blank); otherwise, these two field types are identical.

Like the N field, the G and H fields use 9s as placeholders for digits. But they can also use Zs for leading digits you don't want to display when zero. So, if the user enters, say, 56 in a G9999 field, DataPerfect displays and stores

0056

But if the user entered 56 in a GZZZ9 field, DataPerfect displays (as right-aligned) and stores

56

Further, you can insert a minus sign to the left of the leftmost digit to allow the user to enter either positive or negative numbers. So a G-9999 field allows for numbers from -9999 to 9999. Unlike the N-9999 field, however, the leftmost minus

sign in a G field only displays and stores if the user actually enters a negative number.

Now, note also the following possible field formats:

G9-9999
G99-999
G999-99

I don't recommend them as formats. They have odd consequences. No matter what the user enters in those fields, DataPerfect will understand the number to be negative. That will be true whether the user entered a negative sign or not. If the user enters, say, 567 in each of the above three fields, DataPerfect will display and store the following numbers respectively:

0-0567
00-567
005-67

But DataPerfect will consider the numerical value of each of the above, for purposes of calculation when accessed by, say, field formulas,

-00567

This is very unintuitive, so don't put the negative sign anywhere but to the left of the leftmost digit.

All the above discussion of G fields also applies to H fields.

F Fields

These are floating decimal fields, available only in reports (that is, you can't this format when formatting a field in a panel). I discuss them in detail in the *F Fields* section of my **Reports: Fields** chapter.

D Fields

These are date fields, which are special numerical fields. Their format always begins with a *D*, followed by one of many possible combinations. Let's say the value in a D field is January 9, 1996. Here are various ways a D field can display that value, depending on its format:

January 9, 1996	
<i>Format</i>	<i>Display</i>
1 D99/99/99	01/09/96
2 D99/99/9999	01/09/1996
3 DZ9/Z9/99	1/ 9/96
4 DZ9/Z9/9999	1/ 9/1996
5 D99/99	01/09
6 D99	01
7 DDMY99/99/99	09/01/96
8 DYMD99/99/99	96/01/09
9 DYMD99/99	96/01
10 DYMD99	96

For data entry purposes, you're probably going to choose one of the first four formats above. The other six formats are usually used to display or print the contents found in another field that's probably formatted like one of the first four. I suggest, however, you shy away from using the Z place holder in DataPerfect date fields. Regarding fields into which the user will be entering data, the Z place holder is very unintuitive for months prior to October (the single-digit months). Even though the date field is formatted DZ9/99/99, the user still must do something with the first character in the field (skip it with the cursor key, or enter 0). This easily confuses the user.

For computed fields, though, you might prefer the DZ9/99/99::C format. Some like the DZ9/Z9/99::C format, which I think looks terrible for days 1 through 9. For the sake of consistency, I format even my computed date fields without Z place holders. I want the user to get used to entering leading zeros for single-digit months and days, so that's all I want them to see on the screen or reports.

Also note the order of displayed date field parts can be changed, as demonstrated in the last four formats in the above table. And lastly, formats 5, 9, and 10 also demonstrate how DataPerfect lets you display only selected parts of the date.

A few comments about leaving parts of a date out of a date format, as in 5, 9, and 10. First, the way you do this is to make sure the part you want left out is at the end of the format's date part ordering. So to leave out the year, no special ordering need be added (example 5), since the year is the last part of this example, by default. But to get rid of the day in example 9, I reorder things to make the day last in that order. To get rid of both the month and the day in example 10, I reorder that as well, putting these parts at the end of that order.

Second, you must realize that DataPerfect always stores a complete date for any value saved in a date field, even if that date field's format has parts missing. DataPerfect stores 1904 for a missing year, January for a missing month, and the first of the month for a missing day. So if you enter 05 into a D99 field, DataPerfect will store 05/01/1904, even though it displays 05. This holds true for date fields with parts

missing unless the user enters a date in a date field by hitting **F6** while on that field in Edit or Create mode. DataPerfect assigns a special function to the **F6** key when hitting it while sitting on a date field during Create or Edit modes: it inserts today's date in that field as long as no field formula is attached to the field (otherwise, it updates the date field using that field formula). So hitting **F6** on a date field with parts missing will store today's date and not follow the rules I just outlined.

Date fields aren't special numerical fields just because of their formatting capabilities. They're also special numerical fields because they allow date fields to participate in arithmetic calculations. I get more into this later, but suffice it to say here that if P1F1 and P1F2 are date fields where P1F1 holds 01/18/96 and P1F2 holds 01/21/96, then the formula

$$P1F2 - P1F1$$

yields

3

Date fields actually store the number of days since 03/01/1900, while displaying the actual date intended. Again, I explain in detail how all this works later.

This entire discussion about date fields has assumed the date default you have set for your DataPerfect application is U.S. format (month/day/year). Whatever you have the default set for, that's what will display when no order indicators are present in the format. The default is controlled in the Format Defaults menu (**Shift-F9, 2**):

Format Defaults

1 - Date Order:

1-DMY 2-MYD 3-YDM 4-DYM 5-YMD 6-MDY

2 - Time Order:

1-SMH 2-MHS 3-HSM 4-SHM 5-HMS 6-MSH

3 - Decimal Point Character:

.

4 - Thousands Separator Character:

,

5 - Month Abbreviations

JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC

Selection: 0

Note item 1 above. You may choose any of the six Date Orders you see there. The sixth, of course, is U.S. format.

T Fields

These are time fields. Like date fields, they're special numerical fields that can be used in arithmetic formulas. The time field can be thought to hold a quantity of time (in hours, minutes, and seconds), or a specific time on the military clock. They're formatted like this:

$$T99:99:99$$

By default, that's understood to be in U.S. format:

$$\text{hours:minutes:seconds}$$

As with other numerical fields, you can suppress the display of leading zeros with a Z:

TZ9:Z9:Z9

Also like the date field, the time field can be formatted to change its default ordering of components, including amputated permutations:

02:06:15		
<i>Format</i>	<i>Display</i>	<i>Quantity Meaning</i>
T99:99:99	02:06:15	2 hours, 6 minutes, 15 seconds
TZ9:Z9:Z9	2: 6:15	2 hours, 6 minutes, 15 seconds
TSMH99:99:99	15:06:02	15 seconds, 6 minutes, 2 hours
TMSH	06:15:02	6 minutes, 15 seconds, 2 hours
T99	02	2 hours
T99:99	02:06	2 hours, 6 minutes
TSMH99	15	15 seconds
TMSHZ9	6	6 minutes

Again like the date field, subtract one time field's value from another, and you get the difference. Time fields actually store the number of seconds since midnight. I get much more into time fields later.

Display Mode Indicators

Computed fields (::C)

The computed field has a few distinctive characteristics. It's basically a screen entity, not existing in the data file at all. It derives its value only from its field formula (**Shift-F8, 2**), ignoring any attempt by the user to **Tab** to it or enter data in it. Its field formula will recalculate as soon as the panel is displayed, every time a change is made in any field in its panel, and whenever a field's formula or report accesses it. It's very similar to a Non-updatable field (::N), but different in important respects you'll see next. In other database management systems the *computed* field is frequently called a *display* field.

Non-updatable fields (::N)

Like the computed field, the Non-updatable field derives its value only from its field formula (**Shift-F8, 1** or **Shift-F8, 2**), ignoring any attempt by the user to **Tab** to it or enter data in it. Unlike the computed field, the Non-updatable field's data is written to the data file (stored in its record). Also unlike the computed field, it can be used in indexes and link field lists (this is because its value is written to its panel's data file). Also unlike the computed field, merely displaying its panel doesn't update it. A Non-updatable field updates within definer-determined parameters, which can only

come into play when in Edit or Create mode; whereas computed fields update even when just in Browse mode.

Auto-Enter fields (::E)

This is straightforward. When a field is formatted Auto-Enter, the user's cursor will automatically exit that field (as though they hit **Tab**) after filling that field with its maximum number of characters.

Hidden fields (::H)

These have all the same characteristics as Non-Updatable fields, with one added characteristic: they're hidden from the user.

Auto-Incrementing fields (::I, ::J)

These modifiers can only be placed on G, H, or N fields. It results in that field, in Create mode, incrementing to the next number up. You set its *Value for Next Created Record* with **Shift-F8, 2**. So if you place, say, a G9999::I or G9999::J field in a panel, each time the user creates a record, that field will be assigned the next number that field can take. If that field's *Value for Next Created Record* was set at 0001, then the first record created gets a 0001 in that field, the second gets a 0002, etc.

Each of these two modifiers updates as soon as the user hits **F9** (Create). They differ in two significant ways that are exposed in an application that's running on a network. If two different users, each at a different workstation on a network, have the same application loaded and hit **F9**, the following happens:

Users' Actions	::I field		::J field	
	User 1	User 2	User 1	User 2
Both users hit F9 in Create.	001	001	001	002
User 1 hits F10 . Then User 2 does.	001	002	001	002

Here's what's happening above. The field in question is initially set by the definer to update to 001 as its *Value for Next Created Record*. Two cases are shown: one where the field in question is a ::I field, and one where it's a ::J field. Note that both cases yield the same final result (bottom row): User 1 gets a 001 in that field and User 2 a 002. But the path the ::I field takes to that result is different than the path the ::J field takes. Note User 2's shaded cells. In the ::I case, User 2 temporarily had 001 in that field, but in the ::J case he always had 002.

This becomes important later, as we'll see. Suffice it to say that the difference between these two modifiers is that the ::I modifier gives all users in the same Create session the same number, but reevaluates that choice on Save, making sure it's unique. If it isn't unique at Save, it's incremented again. That's what happened to User 2 in the ::I field case. DataPerfect saw that another user saved a record in that panel with 001 in that field (User 1), so it incremented it further. A ::J field, on the other hand, assigns a unique number to its field on Create and doesn't reevaluate that value

on Save. So if ten users all hit **F9** at the same time in that panel (ten different workstations), they'd all get a different number in that field immediately if that field is a ::J field. Thus there's no need to reevaluate the field's value on Save.

The difference between the ::I and ::J fields has consequences when the user decides to Cancel (**F1**) a Create operation before Saving it. If the auto-incrementing field in question is an ::I field, the number the user saw in that field just before hitting **F1** will be available for the next created record. If the auto-incrementing field is a ::J field, that number is gone.

The ::J modifier was introduced late in DataPerfect's evolution when it was discovered developers were making improper use of the data link. I discuss this later, in *A Caveat Regarding Data Links* in my **Links** chapter. Also, in *The Recursive Panel Link* of that same chapter, I discuss reasons for avoiding Auto-Incrementing fields altogether, in favor of a different scheme.

A final note. Many developers think they need to protect an auto-incrementing field from the user accidentally editing its value. This isn't the case. Even if you allow the user's cursor to land on that field (that is, you do *not* make it a G999::IN field or a G999::JN field), DataPerfect won't allow the user to alter its value.

Must Be Updated fields (::M)

If a field has the ::M modifier, and that field is empty (blank if a alphanumerical field, or zero if numerical), DataPerfect won't let you save the record. For instance, if a Client Panel record has a Social Security Number field formatted N999-99-9999::M, DataPerfect won't let you save a record in that panel with a 000-00-000 in that field. This modifier *cannot* be applied to a hidden field.

In this chapter, I discuss what I consider to be interesting issues that revolve around the Field. Though I primarily target the experienced DataPerfect application developer here, the beginner should at least peruse this chapter.

Choosing Between ::C and ::N Fields

Do You Need the Field to Be Real?

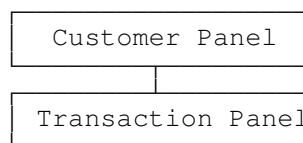
A field is *real* just in case DataPerfect stores its data in its panel's data file. All DataPerfect fields are real except computed fields (::C) and panel links. (Don't forget a panel link, unlike a data link, is a field. DataPerfect assigns it a field number.)

Whether or not a field is real has significant consequences. If it is *not* real, the following limitations arise:

- The field can't be included in an index.
- The field can't receive or send values in a Keep A Total operation.

So, if you must use the field in an Index Field List or have it send or receive totals with DataPerfect's Keep A Total facility, you're committed to making it real, thus shunning the computed field. This issue easily arises in a two-panel hierarchy like the one that follows.

Suppose your application has a Customer Panel and a Transaction Panel, the latter being a subpanel of the former:



Though the Customer Panel might have a Customer Number field and an index that sorts on the Customer Number field, you'll probably want that panel to also have an index sort on the Last Name field. Fine. Now what about the Transaction Panel?

Assuming you want a Last Name field in the Transaction Panel, do you want the Last Name field to be part of an index in that panel? If you want to perform Transaction Panel lookups sorted on the Last Name field, you'll have to place the Last Name field in a Transaction Panel index. The Last Name field, then, would have to be real, and thus not a computed field. Note, however, that though this situation requires the Last Name field be real, it's *not* because the Last Name field is on a

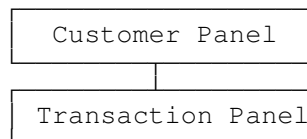
lookup field list. Rather, it must be real because the lookup must be tied to an index that includes the Last Name field on its field list. DataPerfect allows computed fields in *lookup* field lists—it just banishes them from *index* field lists.

Note: DataPerfect began allowing computed fields on *lookup* field lists with version 2.2, and began allowing computed fields on *link* field lists with its second official release of version 2.3 (September 1993).

Issues Concerning Totaling

As a general rule, you can speed totalling operations by favoring computed fields over non-updatable fields in panels that *receive* the totals. In a panel that receives totals, replacing twenty computed fields with twenty non-updatable fields that update on any change, dramatically slows the totalling. This is because each totalling event will trigger the twenty non-updatable fields to update. However, if these twenty fields were *computed* fields, the same totalling events wouldn't trigger them to update because computed fields update *only when they're displayed or directly accessed by a formula*. The speed difference here is considerable, even with a fast computer.

Consider the following two-panel hierarchy again:



Let's say you'd like some fields in the Customer Panel to show the Date of the First Transaction and the Date of the Last Transaction. These fields will typically be either non-updatable fields that update on any change, or computed fields:

CUSTOMER PANEL			
ID No.	Last Name	First Name	Balance
.....
Date of First Transaction:			
Date of Last Transaction:			

Given the existence of the Balance field, the above panel obviously receives totalling data from a Keep A Total taking place in at least one of its subpanels. Every time it receives totalling data from a subpanel, it goes into Edit mode temporarily. This causes all its real (noncomputed) fields that update on any change to update, but leaves all its computed fields alone. So, if the two date fields you see above are non-updatable fields that update on any change, every Transaction Panel record that triggers the Keep A Total facility to carry totalling data to the Customer Panel will also cause those two fields to update. But if those two fields are computed fields, the

Keep A Total facility won't cause them to update. The slowdown I'm referring to here is the time it takes to save a record in the Transaction Panel.

With only two such fields in the Customer Panel, you probably won't see much of a speed difference between formatting them as computed versus non-updatable. But if you have many such non-updatable fields that update on any change in the Customer Panel, Transaction Panel record Saves will take much longer than if those non-updatable fields were computed instead. The difference is very noticeable.

Keeping Subpanel Data Current

This issue became almost moot starting with the second release (September 1993) of DataPerfect 2.3. Let's see the fields in both the Customer and Transaction Panels (I'll take out the Customer Panel date fields because they're not important for this point):

CUSTOMER PANEL

ID No.	Last Name	First Name	Balance
To Transaction Panel			

TRANSACTION PANEL

ID No.	Last Name	First Name
Date	Description	
Charge	Payment	Adjustment

Note the three fields common to both panels: ID Number, Last Name, and First Name. If this application is defined so the user always enters the Transaction Panel via the panel link in the Customer Panel, there's no reason to allow the cursor to ever land on these three common fields in the Transaction Panel. So these three Transaction Panel fields will be either computed fields or non-updatable fields.

The ID Number field value will never change over time for the same Customer, and it must be a real field (not computed) so that it can be used in the index tied to the Customer Panel panel link. But the values in one or both of the Name fields could change over time for the same Customer. The Customer may marry and change her Last Name, for instance. Any change in the Last Name or First Name fields you make in that Customer's record in the Customer Panel should somehow reflect itself in that Customer's subrecords in the Transaction Panel.

Beginning application developers inevitably stumble here. There's no essential reason for the Last Name and First Name fields to be in the Transaction Panel, other than that it's nice to see them there. In this scheme, the user always picks the Customer in the Customer Panel before entering the Transaction Panel, so they already know whose Transaction Panel record is on the screen without having to see Name fields in that Transaction Panel. And the definer can position each panel so that the user always sees the Customer Panel record while working on the Transaction Panel record. But there are other reasons to have the Name fields in the subpanel, other than screen aesthetics and functionality. For one, it'll make report defining much easier. But let's go back to how beginning developers typically stumble here.

The mistake frequently made at this stage is to format the Name fields in the Transaction Panel as non-updatable and then put the Customer Panel's Name fields on the Customer Panel's panel link field list. This works fine as long as the Customer never changes either of his names. On this scheme, DataPerfect will insert the appropriate values in the Names fields in the Transaction Panel when the user penetrates the panel link, and in Browse mode, it will keep the proper Transaction Panel records tied to the appropriate Customer in the Customer Panel.

But as soon as the user changes the value found in one of the Name fields in the Customer Panel, and then attempts to penetrate the panel link to Browse that Customer's Transaction Panel records, they'll see this message:

No records are found in this subset. If you want to add records, Press Create Record in Linked Panel (F5). Otherwise, you will Remain in this panel.
NOTE: You will get this message again if you press F5 and you are Not authorized to create a record in the linked panel.

Why is DataPerfect spitting out this message when it didn't before the user changed the value in one of the Name fields? Well, because the definer put those Name fields on the field list of the Customer Panel's panel link, when the user attempt to penetrate that link, DataPerfect is attempts to find records in the Transaction Panel with the current values in the ID Number, Last Name, and First Name fields. But they don't exist.

There are three typical ways to handle this, with pros and cons for each:

Backward-Referring Computed Fields

Here you take the Name fields off the Customer Panel panel link field list, and then format the Name fields in the Transaction Panel as computed fields with field formulas that take the value found in the matching fields in the Customer Panel parent record. This requires there be a link in the Transaction Panel to accomplish the writing of the field formulas for these two fields.

On this scheme, you put in a hidden panel link in the Transaction Panel with the following characteristics:

Target:	Customer Panel
Index:	ID Number, Last Name, First Name

Field List: ID Number
Status: Hidden

That panel link takes the user from any Transaction Panel record to that record's parent in the Customer Panel.

Here's what we have so far in our two panels:

Customer Panel fields		
<i>Field Code</i>	<i>Field Name</i>	<i>Field Format</i>
P1F1	ID Number	G99999::J
P1F2	Last Name	A15
P1F3	First Name	A15
P1F4	Balance	GZZ,ZZ9.99::N
P1F5	Panel link to Transaction Panel	Displayed

Transaction Panel fields		
<i>Field Code</i>	<i>Field Name</i>	<i>Field Format</i>
P2F1	ID Number	G99999::N
P2F2	Last Name	A15::C
P2F3	First Name	A15::C
P2F4	Date	D99/99/9999
P2F5	Description	A20
P2F6	Charge	GZ,ZZ9.99
P1F7	Amount	GZ,ZZ9.99
P2F8	Adjustment	GZ,ZZ9.99
P2F9	Panel link to Customer Panel	Hidden

When defining the field formula for the Last Name field in the Transaction Panel, you use that panel's hidden panel link. The Last Name field in the Transaction Panel would end up with this formula:

P2F9P1F2

The Transaction Panel's First Name field would end up with this formula:

P2F9P1F3

The Customer Panel panel link that takes the user to the Transaction Panel will have only the ID Number field on its field list, not the Last Name or First Name fields. Since a Customer's ID Number will never change for that Customer, the linkage now won't break when they change their name. And any change to their name will always be reflected in their Transaction Panel records because the computed Names fields will update every time that panel displays.

The downside to this approach is that the Names fields aren't real, and therefore can't be used in an index. If you need an index in that panel with one or both of these fields in its index field list, you'll need to use the *Cascade Update* approach, discussed next.

Cascade Update

Use this approach if you need the fields like the Name fields in the Transaction Panel to update whenever their parent record in the Customer Panel experiences changes in one of the two Name fields. To make sure changes to the value found in a field in a parent panel are reflected in all its linked records in one or more of its subpanels (when the matching fields to update in its subrecords are *real* fields), make sure that, for each subpanel to receive the automatic updates of these fields, the source panel has a panel link with the following three characteristics:

- It has the fields in question on its link field list.
- It has Cascade Update on.

To turn a link's Cascade Update facility on, cursor to that link and **Shift-F8, 7**. Note that **Shift-F8, 7** toggles between *Cascade off* (the default), *Cascade Update on*, and *Cascade Update/Delete on*. Be careful not to turn Cascade Update/Delete on when you only want Cascade Update on.

When a link has Cascade Update turned on, that link will make sure any changes to values in fields *on its field list* will cascade to all subrecords linked to that parent record via that link. So change the Last Name from *Adams* to *Smith*, and a link with Cascade Update turned on will make sure all subrecords in the Transaction Panel linked that Customer will have their Last Name field update from *Adams* to *Smith*.

Don't get too excited about using Cascade Update. It has a downside that must be taken seriously on large databases. If a parent record has hundreds, or worse, thousands of subrecords to which to cascade a field list change via a Cascade Update panel link, the user will find that changing any of the field list fields in question will cause a serious delay in processing, with *Please Wait* appearing on the screen. Until Cascade Update finishes updating dependent records, it will keep the current user from using the keyboard, and lock out *all other network users* from saving any records in that database!

This can be a real pain. With large databases, use backward-referring computed fields unless you need these fields in an index. If you need them in an index, and you expect their panel's data file may eventually hold many many records, I suggest considering the next solution.

Backward-Referring Non-Updatable Fields

This method can be used only if supplemented with a report. It involves formatting the fields in question as non-updatable instead of computed, using the same formulas used in the computed field approach, setting them to update on any change. But that's not enough, since such fields will only update their values when in Create or Edit mode. Putting the *parent* record in Create or Edit mode won't have any impact on these fields. To get around this, you can create a report that puts all records in this panel in Edit mode. A simple report like this would be one that just places a value (any value at all) in one of the non-updatable fields that update on any change. That will trigger all such fields in that panel to update. Don't worry what value you have the report put in that field—as long as it updates on any change, its field formula will override what the report puts in there.

But suppose the user only changed one or two names in a database of a few thousand Customers. You don't want to have this report go through all the records in the Transaction Panel, as opposed to just those linked to Customers with name changes. To show how to optimize this, let's use our Customer-Transaction Panel example again. For each of our Name fields, we'll create a hidden field with the same format in the same panel, like this:

<u>CUSTOMER PANEL</u>			
ID No.	Last Name	First Name	Balance
To Transaction Panel			

Those two fields at the top are hidden versions of the Last and First Name fields below them. They update to the values found in their matching fields on record creation (*not* on any change).

Now put in a hidden G9 field in the Customer Panel, such that it updates on any change to 1 if the value of either of the Name fields fails to be identical to that found in its matching field. If the Last Name and First Name A15 fields are still P1F2 and P1F3, respectively, and the two new hidden A15 fields are P1F6 and P1F7 (matching the Last and First Name fields, respectively), then the formula for the new hidden G9 field would be something like this:

```
if P1F2"%"=P1F6"%" or P1F3"%"=P1F7"%" then 0
else 1 endif
```

The above formula tests P1F2 and P1F6 for a perfect match, and tests P1F3 and P1F7 for a perfect match, and updates to 0 if either isn't; otherwise it updates to 1. Read *Perfect Matches and the Identity Operator* in my **Formulas** chapter to see why this formula will *not* work:

```
if P1F2=P1F6 or P1F3=P1F7 then 0
else 1 endif
```

Now that we have a G9::H field that updates on any change to 1 if we have a mismatch between the Last Name field and its matching A15::H field, or between the First Name field and its matching A15::H field (otherwise this G9::H field update to 0), we create an Exception List Index that holds all and only those Customer Panel records that have a 1 in that G9::H field. That index, then, will hold all and only Customer Panel records that have mismatches between the fields in question. See *Exception Lists* in my **Indexes** chapter for more Exception List Indexes.

Next, we create a report that runs on the new Exception List Index. That is, it sees all and only Customer Panel records that have mismatches between the fields in question. For each such record, it takes the values found in the Last Name and First Name fields and puts them in their matching A15:H fields, and then puts them in the Last Name and First Name fields in all its subrecords in the Transaction Panel. The first step (stuffing the A15::H fields) causes the G9::H field to update back from 1 to 0, effectively taking that record out of the Exception List Index on which this report is running. The second step, of course, updates that Customer's Name fields in his subrecords in the Transaction Panel.

Here's what such a report would look like:

```
Based on Exception List Index
-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Store Value in Report Variable 1 -----
-----Store Value in Report Variable 2 -----
-----Store Report Variable 1 in Field 6 -----
-----Store Report Variable 2 in Field 7 -----
=====SUBREPORT LINK/PANEL: 5 2=====
-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Store Report Variable 1 in Field 2 -----
-----Store Report Variable 2 in Field 3 -----
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
=====END OF SUBREPORT=====
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
```

Main Report:
Customer Panel

Store Name fields
RV1 and RV2.
Put those values in
A15::H fields.

Subreport:
Transaction Panel

Update the Trans
Panel records with
new values.

Note that in the above report, we really don't need to stuff the Transaction Panel subrecords with Report Variables 1 and 2. All we need to do in the subreport is stuff *anything* in a field in that panel that updates on any change. That will cause all fields in that panel that update on any change to update, independently of what was stuffed in the field with the Report Variable. Such fields that qualify here are the Last Name and First Name fields.

Computed Fields and DataPerfect's Work Space

You need to be aware of another issue concerning computed fields: DataPerfect's work space. When DataPerfect first loads, it allocates a finite amount of memory for its work space. This is where DataPerfect computes your field formulas, remembers the panel you just left when you landed in the current panel, etc. All computed fields in the current panel are held in this work space until you leave that panel. So computed fields, by their nature of living only in the current display, put demands on the work space.

When DataPerfect's work space is taxed beyond its limits, it issues the Error 104 message. To work around this, DataPerfect has a startup switch (/L) that increases its work space. It accomplishes this by decreasing the number of panels the work space will remember from seven to three. This, of course, could cause a minor performance hit during data entry, by causing DataPerfect to access the disk more often, looking for panels accessed by field formulas in your current panel.

Though computed fields take up a lot of DataPerfect's work space, the most common cause of an Error 104 isn't the computed field *per se*. Rather, it's a computed field whose field formula accesses yet another computed field, even if both fields are in the same panel. This is probably the greatest burden on DataPerfect's work space. Make sure to rewrite any such formula from scratch so that it doesn't access another computed field.

One other note about DataPerfect work space. You might have noticed your application ran without Error 104 under DataPerfect 2.3's initial release (Feb 1993), but ever since upgrading to DataPerfect 2.3 Sep 1993 or later, you got Error 104 frequently, even when using the /L switch. This is because some of the features introduced by the September 1993 release caused a decrease in the available work space. These features include the User ID Panel facility and its related USER.FIELD.[n] function. Don't expect this to change with newer versions. Start optimizing your applications right now by looking for computed fields that access other computed fields, rewriting their field formulas.

Choosing Between G Fields and N Fields

Unless you have an overriding reason to use an N field, use a G field for numerical data. As noted earlier, you should format a numerical field G if the field is to be used in a formula. Plan ahead. You might need to access this numerical field with a formula later.

The main consideration that should lead you to favor the N field over the G field is when this field is a data entry field into which entering data would be unintuitive when right-aligned. But if this is a non-updatable or hidden field, don't use the N format unless you have a good reason to do so.

Another consideration. Hidden flags should be numerical fields (preferably G fields, of course) unless other considerations prevent this. Such flags would be, for instance, fields that are used in Exception List Indexes. You may need to use such fields for totalling later, and you'll be out of luck if you formatted them alphanumeric.

For instance, my application's Transaction Panel might have an Exception List Index that tracks payments, doing this by using a hidden G9 field as the sole occupant of the Exception List. I may later decide that I want the total number of such payments per account to be displayed in the Account Panel. This is a relatively painless change if the hidden Exception List field is numerical. Not so if it's alphanumeric.

The H Field

The H field format isn't used enough by DataPerfect developers. It's essentially G format that doesn't display zeros or masks when the field is blank. Though formats GZZZZ, HZZZZ, and HZZZZ9 are, for all intents and purposes, the same (they all display the same values in all cases), differences arise when masks appear. Consider the typical phone number format: (999)999-9999.

G and H field formats are lousy for phone number fields in panels. Entering numbers into a right-aligned field is unintuitive. Such fields are properly formatted as N fields:

N (999) 999-9999

Such a field could be formatted U13, but that format requires the user insert the masks himself.

So we're committed to using the N format for phone number fields in a panel. But this poses a slight problem in reports. Consider records with blank phone number fields. If you don't want a report to print a (000)000-0000 for each blank phone number, use the H format *in the report* (even though the field is still an N field in the panel):

H (999) 999-9999

Though the H field is technically incompatible with the N field, it works fine when simply grabbing a phone number from a panel and sending it to the printer. This is because both fields store their data as a 10-digit number, putting masks on only for display purposes. But the H field won't print anything if the value is zero.

Here's what each format does in a report with a blank phone number entered into the panel's N(999)999-9999 phone number field:

<i>Report Field Format</i>	<i>Report Output</i>
G (ZZZ) ZZZ-ZZZZ) - (
G (999) 999-9999	(000) 000-0000
N (ZZZ) ZZZ-ZZZZ	(000) 000-0000
H (999) 999-9999	

The last format is what you want for a phone number field in reports, but not panels.

There's one more possibility for blanking out phone numbers in reports when no data exists in the field. You could format it in the report like this:

N(999)999-9999;;E

The ;;E Print Mode Indicator (Delete Zero Subfields from the End) is designed to drop the final subgroup of digits if all those digits are zero (see *Print Mode Indicators* in my **Reports: Fields** chapter). If DataPerfect detects all digits in all subgroups are zero, it fails to print that field at all. This is fine if no phone number ever has 0000 as its final subgroup of digits. But if a phone number *does* have 0000 as its final subgroup, the above format will print as with that subgroup dropped. That is,

(310)555-0000

will print as

(310)555

I was assured by a student at one of my DataPerfect seminars that phone companies don't allow 0000 as the final subgroup, but I'm not counting on that. I prefer to stick with my H field report field conversion technique.

Date Fields

When You Might Not Want to Use the D Field for Dates

You have a few choices in deciding how to format date fields. As previously outlined, DataPerfect has a special date field format (the D field), but you could also use an A8 or U8 field, allowing dates to be entered as short as

3/6/96

or as long as

03/06/96

or an A10 or U10 field, allowing four-digit years, like

03/06/1996

Likewise, you could use numerical fields, like N99/99/99, N99/99/9999, G99/99/99, or G99/99/9999 fields, allowing all the same possibilities as the A and U field formats.

Alternatively, you can—and almost always *should*—use DataPerfect's date field format. There are, however, good reason for occasionally entering or expressing dates in fields other than the D field. Let's discuss these.

You want a date expressed in English.

Instead of displaying

01/12/93

to the user, you might want to display

January 12, 1993

Of course, such a field's data can't be used in a computation by some other field. But such a field must convert data received from a real date field somewhere else in the application, and that other field (the real date field) can be used in calculations. Typically, fields that express dates in English, which must be A or U fields, will be computed fields. For a formula that converts a the value found in a date field to its English expansion, see *APPLY.FORMAT* in my **Formulas** chapter.

Your application will be taking dates prior to March 2, 1900 or beyond December 31, 2078.

The DataPerfect date field won't accept such fields. You'll need to choose an A, U, or N field for that. G fields would be silly, since they force the user to enter data in them from right to left.

You want to express a date in a computed field only under certain conditions, and display a text message in other conditions.

In the application I use to run my practice, I have an A8::C field that displays the date the last diagnosis was made on the patient, assuming one was made. If I failed to enter a diagnosis in the Diagnosis Panel for this patient, the same field displays

Diag?

reminding me I have yet to enter a diagnosis for this patient. This application has many such fields that display dates under certain conditions, and text messages in other conditions.

A formula for such an A8 field might look something like this:

```
if P1F10P2F1="" then "Diag?"  
else apply.format["D99/99/99";P1F10P2F2]  
endif
```

Here DataPerfect looks through a panel link (P1F10) to see if anything is in the Diagnosis field (P2F1) in Panel 2. If that field is blank, this computed field displays

Diag?

Otherwise, it displays a text conversion of the date it sees in a date field in Panel 2 (P2F2). This text conversion will look just like a date field display:

The above display is text in an A8 field, even though it looks like a numerical display in a D99/99/99 field. Note that the formula for this requires use of the `APPLY.FORMAT` function to convert a numerical value (that found in the date field, P2F2) to text. For more on `APPLY.FORMAT`, see *APPLY.FORMAT* in my **Formulas** chapter.

You want to provide the user with an intuitive date lookup field.

[Refer to UD.STR here.
Find the single-panel *Finding Dates* series.]

Perhaps you've noticed this already. Incremental searching on a lookup is impossible on date fields formatted with the U.S. format (DMDY99/99/99 or DMDY99/99/9999). The highlight bar won't move until you've typed an entire date, slashes included. Then it will move to the first it finds equal to or greater than that date in the active index. The same unsatisfactory lookup searches arise when using the alternative DYMD99/99/99 format or DYMD99/99/9999 format.

If, in a particular panel, finding records by date in a lookup is important, consider creating a special field used solely for date-sorted lookups. That is, you provide the user with the usual date field using the D field format. That field will be for data entry, not for finding records incrementally in a lookup. Another field will be reserved for that function, and it won't be a D field. If you're using four-digit years in your data entry field, this other field—the date lookup field—will be a U10 field if you want slashes in the date, or a U8 field if you don't want the slashes. The U10 date lookup field will have a field formula like this, to update on any change:

```
apply.format["N9999";year[P1F1]]
"/"
apply.format["N99";month[P1F1]]
"/"
apply.format["N99";day[P1F1]]
```

P1F1 is the date entered by the user in the data entry field. If the user entered

03/14/1994

in P1F1, which is formatted D99/99/9999, then the above U10 field will display

1994/03/14

A lookup on a field like that will accommodate incremental keystroke searches on each number typed (with slashes). If you leave out the second and fourth lines in the above formula (the *slash* lines), a U8 date field would display

19940314

allowing for incremental keystroke searches without typing in slashes. Here's the U8 formula, which updates on any change:

```
apply.format["N9999";year[P1F1]]  
apply.format["N99";month[P1F1]]  
apply.format["N99";day[P1F1]]
```

***You want to provide the user with an intuitive
Birthday lookup field.***

Suppose you have a Birthdate field in a particular panel, but would like another field to display that person's *birthday*. Here the issue is slightly different. *Birthdays* differ from *birthdates*. The former doesn't include the year. You have two choices for birthdays. Either case will grab its data from the Birthdate field, which we'll call P1F1.

Assuming U.S. format, the first possibility is to create a U5 field that updates on any change to

```
apply.format["D99/99";P1F1]
```

The second possibility is to create a D99/99 field for that updates on any change to

```
P1F1
```

Incremental keystroke searches will only work, however, on the U5 field. Note that the D99/99 format shows only the month and day, leaving out the year if the date is the U.S. format.

The Date Field as a Special Numerical Field

Okay, those are the typical reasons for shunning the D field when creating a field that will accept or express a date. Now let's go into more detail than before, regarding DataPerfect's actual date field (the D field) as a special numerical field. DataPerfect's date field accepts dates from March 2, 1900 to December 31, 2078. If you enter March 1, 1900 in a date field and then save the record, DataPerfect saves it with 00/00/00 (or 00/00/0000) on the screen, and will consider that date, for purpose of date field calculations (talked about later), March 1, 1900. Any earlier date will cause DataPerfect to beep and prevent you from leaving that field, let alone saving the record.

Further, a DataPerfect date field *displays* one thing but *stores* another. On the screen, you'll see a date in the format demanded by the field format, like 03/06/96 or 03/06/1996. But that's not what DataPerfect stores for that date in that panel's data file. Instead, DataPerfect stores the number of days since March 1, 1900 that date represents. So 03/01/1900 is stored as 0, 03/02/1900 as 1, and so on, up to 12/31/2078, which is stored as 65319.

Again, when I say the date field *stores* a number like 65319, it stores it in that panel's data file. You never see it. All you see is the actual date. DataPerfect stores date field values this way so that calculations can be performed on them. If date field

P1F1 (D99/99/9999) displays 03/06/1996 and date field P1F2 (D99/99/9999) displays 03/21/1996, then numerical field P1F3 (GZZZZ9), if formulated to update on any change to

P1F2 - P1F1

displays

15

which is the number of days between the two dates. Behind the scenes, DataPerfect is actually performing this calculation:

35084 - 35069

35084 is the number of days between 03/01/1900 and P1F2, and 35069 is the number of days between 03/01/1900 and P1F1.

Two-Digit vs. Four-Digit Years

As of this writing, we're painfully close to the year 2000. I don't know what others will do with their data at that time, but I don't store critical data with two-digit years. I'm sure there'll be many utilities offered near the turn of the century, written to convert data and date fields in corporate databases, but don't plan on that. Anyway, DataPerfect isn't popular enough for such a utility to be written for it. Format all date fields with four-digit years: D99/99/9999 or DZ9/Z9/9999.

Let's explain why this is important. To enter a date with the year 2000 in a D99/99/99 field the user must enter the year portion of that date as *00*. But DataPerfect understands this to be the year 1900. Entering the year 2001 as *01* in the year portion of a D99/99/99 field is really entering the year 1901, and so forth. So you can't enter a date beyond 12/31/1999 in a D99/99/99 field. No matter how you do it, it will be stored as a date one hundred years before the date you intended it to be.

This caveat against using two-digit years only applies to *real* (i.e., not computed) fields in the *panel*. A computed date field may be formatted with a two-digit year because it doesn't store its data in the data file. As far as reports are concerned, you can use any format you want for dates sent to the printer (I usually use D99/99/99) because doing so doesn't affect the data file. You just need to inform the user that a date in a report that looks like 01/01/10 is probably in the year 2010, not 1910. If you want, just make a simple rule: No two-digit years anywhere, not even reports. Four-digit years may not be what your end user is used to, but they're never ambiguous.

This problem of how DataPerfect stores D99/99/99 field data has implications in a few areas. First, a calculation problem arises when you have, say, a G-9999 field that displays the Days Remaining for loans. Its field formula determines how many days remain until the Maturity Date of each loan. If the Maturity Date field is P1F1, then the Days Remaining field formula would be

If the number is positive, it represents how many days *until* maturity; if negative, how many days *since* maturity. But if the Maturity Date field is a D99/99/99 field, the Days Remaining field will display incorrect values with Maturity Dates after 12/31/99 because they'll all be considered to be a hundred years earlier.

A second problem with D99/99/99 fields arises during data entry. Don't forget that DataPerfect doesn't allow dates earlier than March 2, 1900 (and March 1, 1900 displays as 00/00/00). This means that a date field won't allow dates from 01/01/00 to 03/01/00, even if intended as being in the year 2000. DataPerfect will interpret such data entry as attempts to enter dates in the year 1900, before March 2, 1900.

A third problem with D99/99/99 fields arises when performing lookups on these fields. Dates after 12/31/99 won't sort properly, placing them *before* 12/31/99, not after. Again, this is because DataPerfect stores them as a hundred years earlier. Likewise, reports using indexes based on such fields will sort dates after 12/31/99 improperly.

A Note about International Dates

If you want to make your application available outside the United States, you need to know that DataPerfect allows you to change the default Date field format (MDY) to, say, European (DMY). You access that screen with **Shift-F9, 2** and then choose the default. This keeps you from constantly having to format each Date field in your European application as DDMY99/99/9999, overriding the default DMDY99/99/9999 that would be assumed if you left field D99/99/9999 and never changed the system default with the **Shift-F9, 2** screen.

Also, running DP.EXE /INT at a DOS prompt lets you change this default as well. Doing that allows you to change not only the Date and Time field defaults, but now certain error messages will read on the screen. The end result is the a new executable named DPN.EXE. You can then rename that DP.EXE and use with the original DP.SYS, or just use DPN.EXE with the original DP.SYS. When you do this, you no longer have to use the **Shift-F9, 2** screen to change the Date field default, provided you created the new application with the new DP.EXE. If you created the new application with DP.EXE *before* altering it with the /INT switch, you'll have to use the **Shift-F9, 2** screen to change that application's defaults. Altering DP.EXE with the /INT switch affects *future* applications.

The Time Field

To take full advantage of the way DataPerfect handles time, you must pay close attention to the distinction between what a field *displays* on the screen and what it *stores* in the data file. Again, the date field, typically formatted D99/99/99, *displays* a month, day and year, but *stores*, in the data file, the number of days since March 1, 1900. On the other hand, the time field, typically formatted T99:99:99, *displays*

hours, minutes, and seconds, but *stores*, in the data file, the number of seconds since midnight.

Here I discuss DataPerfect's special time field, and how to use it to solve a few special problems. DataPerfect offers a great number of time field formatting possibilities that allow us to manipulate the way a slice of time is *displayed* to the user. And because of the way DataPerfect *stores* the time field's data in the data file, we can enter slices of time into ordinary mathematical computations, like elapsed time computations. With the exception of the first section that follows (on using time fields to guarantee uniqueness of records), much of this discussion will turn on the distinction between what the time field *displays* versus what it *stores*.

Using Time Fields To Guarantee Uniqueness Of Records

First, a very simple way to use DataPerfect's time field. If you've been using DataPerfect for even a short time, you must by now realize that one of the most common uses for the auto-incrementing field (::I or ::J) is to guarantee the uniqueness of records in an index. Placing an auto-incrementing field in a panel can be all you need assure that two records with otherwise matching data in a particular index are unique. Think of records in a Transactions Panel, where a customer may come in twice in the same day and purchase the same item both times. Customer Number, Date, and Item Number might all be the same in each record. An auto-incrementing Transaction Number Field would work here, giving each record a unique number within the database.

Note: DataPerfect application developers are used to making sure each record in a data file is unique across all indexes that govern that file, but not all database programs require this. RBase, for instance, doesn't require all records in a data file be unique across *any* index.

You might, though, have reasons to shy away from auto-incrementing fields (I discuss these reasons in *The Recursive Link* in my **Links** chapter). In place of using auto-incrementing fields to guarantee the uniqueness of records in a given panel, we can use the two-field combination of a date field and a time field. This, though, should be done with an important caution in mind. Let's explain.

The method, as you might surmise, is to place in each index in question, one date field and one time field. Format each as either non-updatable (::N) or hidden (::H), and then formulate the date field to update to TODAY on creation, and the time field to update to NOW on creation. Because the smallest unit of the time field is the second, uniqueness is guaranteed as long as no more than one record in the panel is created in any given second.

Now this final condition about limiting record creation to one per second isn't going to be a problem during manual data entry. But suppose you have a WordPerfect Merge file of records to import into this panel, none with the date and time fields mentioned above. In such a case, importing the data will take care of the blank date

and time fields (the import process will update each field automatically as each record is imported). But if your computer is fast enough, and the importing doesn't involve any totalling to other panels, it's certainly conceivable that you may import more than one record per second. If so, you'll end up with DataPerfect skipping what it considers to be duplicate records during the import process (each record that got the date/time values given to the one that preceded it).

Keep this caveat in mind when using date and time fields to guarantee uniqueness of records in an index.

Computing Elapsed Time: The Simple Case

You can use a time field to express either a *specific* time or a *quantity* of time. That is, 08:23 can be used to express either the specific time 8:23am, or the quantity 8 hours and 23 minutes. Both senses of the time field can easily come into play in a single panel. Consider, for instance, a Consultations Panel that keeps track of consultations you do during each day in your business, allowing for easy billing of clients based at an hourly rate. You could decide to design such a panel to have three time fields:

```
Start Time Field
End Time Field
Elapsed Time Field
```

The Start Time Field and the End Time Field would be data entry fields, and the Elapsed Time Field would be formatted as either non-updatable (::N) or computed (::C), formulated to yield the elapsed time between the two other time fields.

DataPerfect makes writing the field formula in the Elapsed Time Field very easy because it offers a straightforward way to use a time field value in computations. In the case outlined here, the Elapsed Time Field would simply be formulated as the difference between the End Time Field and the Start Time Field. If the Start Time Field is P1F1 and the End Time Field is P1F2, then the Elapsed Time Field formula would be

$$P1F2 - P1F1$$

If the Elapsed Time Field is a non-updatable field instead of a calculated field, then its formula would be set to update on any change.

Nice and simple. This is because DataPerfect stores a time field's data as simply the number of seconds after midnight, allowing us to use these fields in computations just like other numerical fields, adding and subtracting them with ease.

Calculating Elapsed Time Across the 24-Hour Barrier

[Refer to UD.STR.

Find the single-panel *Elapsed Time* series.]

Suppose that, instead of a Consultations Panel that tracks events that span hours and minutes within a single day, you want to create a Projects Panel that tracks events that span days. In such a panel we'll need at least the following fields:

```
Start Date Field
Start Time Field
End Date Field
End Time Field
Field(s) showing elapsed days, hours, and minutes
```

Unfortunately, if the value in the End Time Field is less (earlier) than the value in the Start Time Field, simple subtraction won't help us. To see this, consider a project that started on 08/01/92 at 10:30, and ended on 08/03/92 at 09:00 (don't forget that we're talking about military time here). Subtracting the Start Date from the End Date, and the Start Time from the End Time, yields

```
2 days and a negative 01:30 hours
```

The true elapsed time, though, is

```
1 day and 22:30 hours
```

Let's outline two solutions to elapsed time that crosses midnight or that exceeds twenty-four hours.

First Solution

Suppose we lay out our Projects Panel time fields something like this (I'll use field codes in place of actual fields):

Start Day: P1F1	End Day: P1F3
Start Time: P1F2	End Time: P1F4
P1F5 days, P1F6 hours, P1F7 minutes	

One way to formulate the elapsed time fields P1F5, P1F6, and P1F7 is as follows (explanations follow the two outlines):

Elapsed Days field (P1F5)

```
Formatted GZZ9::C, with the following field formula:

if P1F1=P1F3 then 0 else
if P1F4>=P1F2 then P1F3-P1F1 else
P1F3-P1F1-1
endif endif
```

Elapsed Hours field (P1F6) and Elapsed Minutes field (P1F7)

```
P1F6 formatted THMSZ9::C, and P1F7 formatted TMHSZ9::C,  
each with the following field formula:
```

```
if P1F4>=P1F2 then P1F4-P1F2  
else P1F4-P1F2+86400  
endif
```

Let's explain the above, starting with the Elapsed Days Field, P1F5. The format of the Elapsed Days Field is straightforward, and so is the first line of its field formula. The second line of the formula covers the case when the End Time is greater than or equal to the Start Time, which is a case where simple subtraction actually works. The third line of the formula covers the remaining type of case, when the End Time is *smaller* than the Start Time, which is like the case talked about earlier that resulted in a negative value for the Elapsed Time Field when simple subtraction was performed.

Understanding how I formatted the Elapsed Hours Field, P1F6, and the Elapsed Minutes Field, P1F7, requires taking a look at the special power and flexibility of DataPerfect's time field. DataPerfect allows us to change the display order of the components of any particular time field (again, remember that we're talking here of manipulating how time is *displayed*, not *stored*). The default display order of a time field's components is

```
hours:minutes:seconds
```

If needed, we can alter the display order of any given time field's components by simply using one of the following before the first 9 or Z place holder in the format scheme:

Format	Meaning
TMHS	minutes:hours:seconds
TSMH	seconds:minutes:hours
THSM	hours:seconds:minutes

So, for instance, if a time field was formatted

```
TMHS99:99:99
```

and had a value corresponding to the time

```
10:30am and 23 seconds
```

it would display

```
30:10:23
```

Like it offers us with the date field, DataPerfect offers us a way to isolate just one or two components of a time field by simply leaving out the Z's and 9's in the appropriate places. For instance, to display only the hours and minutes of a given time, we can leave the order of the components in their default order, and then leave out the place holders for seconds:

```
T99:99
```

But to display only the minutes of a given time, we first need to alter the display order of its components so that minutes is the first component, and then leave out the place holders for hours and seconds. Any of the following formats will cause a time field to display only the minutes component of its value:

```
TMHS99  
TMHSZ9  
TMSH99  
TMSHZ9
```

So if the value held by a time field corresponds to

```
10:30am and 23 seconds
```

and it's formatted in one of the above ways, it will simply display

```
30
```

With formatting out of the way, we now take a look at the formula used for both the Elapsed Hours Field and the Elapsed Minutes Field. Again, the field formula for each field was the same:

```
if P1F4>=P1F2 then P1F4-P1F2  
else P1F4-P1F2+86400  
endif
```

The first line covers cases where the End Time is greater than or equal to the Start Time—cases where simple subtraction works. The second line covers remaining cases, where simple subtraction yields a negative number. In such cases we add the number of seconds in a twenty-four-hour period (86,400) to that negative number. Whether we want to display only hours or only minutes, is, again, determined by the field format, not the field formula.

An Alternative Solution: Using the Concepts of MOMENT and MODULO

Another way to successfully formulate the Elapsed Days Field, the Elapsed Hours Field, and the Elapsed Minutes Field, is to use a concept I call the *moment*. Just as what a DataPerfect date field actually records in the data file is the number of days since March 1, 1900, I define a moment as the number of *seconds* since March 1, 1900.

Given there are 86,400 seconds per day, the present moment is expressed as

$(86400 * \text{today}) + \text{now}$

In that formula,

$86400 * \text{today}$

expresses the number of seconds in a day, multiplied by the number of days since March 1, 1900, which gives us the number of seconds since March 1, 1900, *as of* midnight last night. To add the number of seconds *since* midnight last night, we add *now*. Given the highest date DataPerfect allows is December 31, 2078, the highest moment DataPerfect can allow is December 31, 2078, 23:59:59, which can be expressed with following DataPerfect formula:

$(\text{date}[31;12;2078] * 24 * 60 * 60) + 86399$

The above formula yields the following number of seconds:

5,643,647,999

Going through this multiplication is necessary only to determine how many characters we should allocate to a moment field. We now know a moment field should allow for a ten-digit number (e.g., G9999999999).

With the moment concept in mind, let's go back to our example of an elapsed time that crosses midnight. Consider our prior scheme:

Field Code	Field Format	Field Name
P1F1	D99/99/9999	Start date
P1F2	T99:99	Start time
P1F3	D99/99/9999	End date
P1F4	T99:99	End time
P1F5	GZZ9::C	Elapsed Days
P1F6	TZ9::C	Elapsed Hours
P1F7	TMHSZ9::C	Elapsed Minutes

Relevant moment calculations would be the following:

Field Formula	Field Name
$(86400 * \text{P1F1}) + \text{P1F2}$	Start Moment
$(86400 * \text{P1F3}) + \text{P1F4}$	End Moment

So the actual elapsed time in seconds is the End Moment less the Start Moment. Now we need to convert the number seconds in the actual elapsed time, which is probably in the thousands, if not tens or hundreds of thousands, into days, hours and minutes.

We can use the field formats already outlined for our Elapsed Days, Hours, and Minutes Fields. We'll formulate the Elapsed Days Field as before, but formulate the Elapsed Hours Field and the Elapsed Minutes Field using the moment concept:

Elapsed Days field (P1F5)

As before, formatted GZZ9::C, and also as before, with the following field formula:

```
if P1F1=P1F3 then 0 else
if P1F4>=P1F2 then P1F3-P1F1 else
P1F3-P1F1-1
endif endif
```

Elapsed Hours field (P1F6) and Elapsed Minutes field (P1F7)

P1F6 formatted THMSZ9::C, and P1F7 formatted TMHSZ9::C, each with the following field formula:

```
((86400*P1F3)+P1F4-(86400*P1F1)-P1F2) // 86400
```

The format and formula of the Elapsed Days Field have already been explained in the original solution to this problem. So have the formats for the Elapsed Hours Field and the Elapsed Minutes Field. Let's discuss the formula used for each of the latter two fields.

Note the use of the modulo operator, '//'. It yields the *remainder* of dividing the expression on its left by the expression on its right. The expression on its left is the End Moment minus the Start Moment, which is the actual elapsed time in seconds. On its right is, of course, the number of seconds in a day. This operation, for all intents and purposes, yields the remaining number of seconds after subtracting all the full twenty-four-hour days in the actual elapsed time figure. Again, the TZ9::C format displays only the hours in this figure, and the TMHSZ9::C format, only the minutes.

Formula Changes to Trap Incorrect Data Entry

To accommodate the user entering incorrect Date and Time values (values that make the Start Date/Time combination occur after the End Date/Time combination), our formulas should be changed as follows:

The Elapsed Days field (P1F5)

```
if P1F1>=P1F3 then 0 else
if P1F4>=P1F2 then P1F3-P1F1 else
P1F3-P1F1-1
endif endif
```

The Elapsed Hours (P1F6) and Minutes (P1F7) formula for the first set of Elapsed Time fields

```
if (86400*P19F1)+P19F2 >= (86400*P19F3)+P19F4
then 0 else
if P19F4>=P19F2 then P19F4-P19F2
else P19F4-P19F2+86400
endif endif
```

The Elapsed Hours (P1F6) and Minutes (P1F7) formula for the second set of Elapsed Time fields

```
if (86400*P19F1)+P19F2 >= (86400*P19F3)+P19F4
then 0 else
((86400*P19F3)+P19F4-(86400*P19F1)-P19F2) // 86400
endif
```

Now both sets of Elapsed Hours and Minutes formulas use the moment function, in order to take into account the possibility of the user incorrectly entering an older End moment.

A Special Use for the MOMENT Function in Reports

Suppose you design a panel that includes a field that tells you the last time any given record was edited in that panel's data file. Actually this takes two fields, one a date field that updates on any change to *today*, and the other a time field that updates on any change to *now*. This pair of fields always show either when the record was created, or the last time it was edited, whichever is later. Simple, so far.

Further suppose you need a report that, among other things, puts data into one or more of this panel's fields. For instance, my doctor's office billing application has an Insurance Billing report that date-stamps each record it sends from the Transactions Panel to the printer. It does this by entering the current date into the otherwise blank non-updatable Insurance Billed Field of that record. The Insurance Billed field is a date field. This gives me a way to see if and when I billed a particular transaction. It also gives me a convenient way of excluding billed transactions from the report process—I merely have to run the report on an Exception List Index that excludes Transaction Panel records with a blank Insurance Billed field.

But, as described previously, I also placed date and time fields in the Transactions Panel that let me know when a record has been edited. Let's call them the Last Edited fields. There's a reason I need to know when a record was last edited, but that's not important right now. But what *is* important is that I don't want the Insurance Billing report to trigger these two Last Edited fields to update to *today* and *now* respectively. I only want them to update when the user manually enters data into the panel.

Do you see the problem here? Those Last Edited fields are date and time fields that update *on any change* to *today* and *now*. But when a report places a value in any field of a record (in this case the report is placing *today* in the Insurance Billed field), that puts that record into Edit mode, consequently triggering updates in all

fields in that record that are supposed to update on any change. So after running this report that, say, sends a hundred Transaction Panel records to the printer in the form of a series of insurance bills, the Transaction Panel will show each of those hundred records as having just been edited, showing in each of these records' Last Edit fields, the date and time this report printed them on their respective insurance bill. But I want the Last Edited fields only to show the last time a *user* put that record in Edit mode, not my Insurance Billing report.

To keep a report from triggering the two Last Edited Fields when it enters data into some other field in the panel, we'll need to use our moment function again. Let's explain.

[Refer to UD.STR for this technique.

Load the *Moment field; Proofing; Incrementing on a network panel*.

See how you can't edit a record there and save it with a date in the Proofed field.

But the *Mark Records as Proofed* report successfully puts today's date in that field.]

With the moment function we can create a window of time where a report fails to trigger selected fields while editing a record. First we need a G999999999::H field that houses moment values. It's hidden for obvious reasons, and is ten characters long for reasons explained previously. It has no formula or initial value assigned to it.

Next, we need to reformulate each field that we want to be able to offer update protection when selected reports place the record in Edit mode. To do this, let's suppose such a field that needs update protection is P1F1. Further suppose that the hidden field that will receive report-entered moment values is P1F2. With that, we can formulate protected P1F1 as follows:

```
if (86400*today)+now<P1F2
then P1F1
else .....
endif
```

The first two lines of the formula basically say that

```
if the current moment is less than the value held in the
hidden Moment field, don't update P1F1
```

The third line (the ELSE statement) is to be filled in by the definer. It would contain the formula that is supposed to update on any change in the *usual* conditions.

To have our report take advantage of this, we make sure that when it's about to enter data into the panel's fields, its first act of data entry is to enter the current moment, plus a few seconds, into the hidden Moment field, P1F2:

```
(86400*today)+now+3
```

The other acts of entering data in that record by this report must immediately follow. If you feel that all the data entry, coupled with saving of the record, can't take place within 3 seconds of the report entering the above value into the hidden Moment field, then increase the number of added seconds, thus increasing the window of protection

for the vulnerable field or fields. This method allows altering the size of the window of update protection on a report-by-report basis.

Let's summarize this technique. Its purpose is to have P1F1 update on any change other than that caused by a particular report. We create a hidden Moment field that has no formula attached to it. It's just blank for now. We adjust our P1F1 formula so that instead of reading like

```
blah blah blah
```

it now reads like this:

```
if (86400*today)+now<P1F2
then P1F1
else blah blah blah
endif
```

This way, P1F1 only updates after first examining the Moment field P1F2. If the current moment is smaller than the value found in the Moment field, P1F1 won't update; otherwise, it will. At first, the value in the Moment field is 0, so P1F1 will certainly update on the next Edit. When our report runs, however, it temporarily interferes with P1F1 updating by putting a value in the Moment field that is three seconds into the future. This means that for the next three seconds, P1F1 won't update in Edit mode. After those three seconds are up, P1F1 will update in Edit mode. But by that time the report has left that record, so the record is no longer in Edit mode. The next time the user hits **F6** in the record, the current moment will again be greater than (later than) the value in the Moment field, so P1F1 updates.

Notes on Deleting a Field

Deleting a field (while in Define Panel mode) can have serious consequences. With the exception of one problem indicated below, all the following problems will be found by simply running DPDIAG on the .STR after the field is deleted. This is an essential practice after deleting a field. Run DPDIAG often during heavy .STR development periods, and always after deleting a field when you can't be absolutely sure such a deletion won't cause one of the following problems:

- Because it results in the deleting of all indexes that contain that field, it might delete an index that's used by a lookup definition.
- Because it results in the deleting of all indexes that contain that field, it might delete an index that, though it's not used in a lookup definition, is the best index DataPerfect can use during a lookup on a particular field that doesn't have an index in its lookup definition. *DPDiag will not detect this problem.*
- Because it results in the deleting of all indexes that contain that field, it might delete an index that's used in a link definition, corrupting that link.
- It will corrupt a link if the deleted field was on that link's field list.
- It will corrupt a link that has a computed field (::C) on its field list, if the deleted field is accessed by that computed field.

Though I include a lot here for the beginner, most of this chapter targets the experienced DataPerfect application developer. There's a lot here not found in the DataPerfect manual.

Fundamentals of Lookups

One of the most powerful facilities DataPerfect offers is its lookup facility. Here the user can type-to-search his way to the desired record rapidly, with little degradation in search speed with large increases in database size. If the user's cursor is on the Last Name field in a database, he can perform a Browse Mode lookup (**Up Arrow** or **F8**) and see all the records in that data file displayed alphabetically by Last Name in the upper or lower third of the screen. A highlight bar will initially sit on the first record waiting for the user to move it to the desired record and hit **Enter**. Other than just using the typical cursor keys to move the highlight bar (**Up Arrow** and **Down Arrow** move it one record at a time, and **PgDn** and **PgUp** move it five records at a time), you can just start typing the Last Name desired. As you type, the highlight bar moves closer to the desired record. It does this by reading your keystrokes, one keystroke at a time, zeroing in on the desired record. When you type *S*, the highlight bar moves to the first record that begins with *S* in the Last Name field. Then when you type *M*, it moves to the first record it finds that begins with *SM* in the Last Name field, and so on.

DataPerfect lets the developer define a different lookup for each field in a record, as well as choose a different sorting index for each lookup. For instance, in my office I frequently want to call a Patient to see how they're doing, but can't, for the life of me, remember their Last Name (and I sure don't remember their Account Number). With the DataPerfect application I wrote and use to run my practice, I can cursor to the Last Name field perform a lookup that displays all records sorted by Last Name, or cursor to the First Name field and perform a lookup that displays all records sorted by First Name, or cursor to the Account Number field and perform a lookup that displays all records sorted by Account Number. If needed, I could, in a manner of seconds, define a lookup on the Account Balance field that would let me perform a lookup sorted by that field.

You don't need to enter Define mode to define a lookup. Just cursor to the field on which you want a lookup and hit **Shift-F8, 1**. Next, you're asked to choose the fields that will comprise this field's lookup field list. For each field you want on that list, you cursor to it and select it with **F4**. If you make a mistake, hit **2** to delete the last entry on the list, or **1** to delete the whole list. Then **F7, F10**, or **0**, to exit and save. You're then asked for the index to sort the lookup, allowing you to browse all the indexes of that panel using **Up Arrow** or **Down Arrow**. You select an index by

hitting **F4**. Now the user can perform a lookup on that field in Browse mode by simply hitting **Up Arrow** or **F8**.

It's important to be clear on the difference between the two central entities involved in designing a lookup. The lookup *field list* determines what fields the lookup will display, and in what order. The lookup *index* determines how DataPerfect will sort that display. If defined properly, the user should be able to type-to-search to a desired record as described before.

The DataPerfect manual says the following about this way of using a lookup:

```
For this type of search to work properly,  
the first field in the lookup field list  
should be the same as the first field in  
the index.
```

[Reference Manual, p. 178]

For now, it's a good idea to make sure the first field in the lookup field list is the first field in the lookup Index. Later, we'll see when this is unnecessary, and even undesirable.

Subfield Lookups

Starting with version 2.3, DataPerfect let's you search for records using both the first and the second fields in the lookup display, using **Tab**. Let's say you perform a lookup on the Last Name field in your Customer Panel, where that field's lookup sorts on an index having as its first two fields, Last Name and First Name, respectively. Further, the lookup field list also has as its first two fields, Last Name and First Name, respectively.

Now, looking for *John Smith*, you perform a lookup on the Last Name field and hit **SM**. You land on the first occurrence of a *Smith*. If you have many Smith's in the database, you can now hit **Tab**. This results in further keystrokes narrowing down the search by sorting on the *second* field of the lookup field list. Hitting **JO** gets you to John Smith's record.

A Subfield Lookup is designed to work only if the first two fields of the lookup field list match the first two fields of the lookup index. At least that's the way it works if you entered the panel directly from the Panel List. If, instead, you entered the panel with a panel link or with a menu and a keyword, Subfield lookups work slightly differently. In such a situation, the first two fields of the lookup field list must either be the first two fields of the index DataPerfect is using to sort the lookup, *or* the first two fields that follow the *common fields*. The common fields are the fields in the subpanel that correspond to the fields of the field list of the panel link that got you there (the parent panel panel link's field list). Let's explain all this with an example Attorney's Office application.

The Panel Link

Say you go from the Attorney Panel to the Client Panel with a panel link. Further, let's say you decided to link these panels on the Attorney ID Number field (that is, the panel link's field list consists of just one field, the Attorney ID Number field). Now let's say the Client Panel has at least the following four fields:

```
Attorney ID Number
Client ID Number
Last Name (Client)
First Name (Client)
```

The panel link places the user on the Last Name field in the Client Panel.

The Lookup Index

The Client Panel index the panel link uses to get you into the Client Panel is

```
Attorney ID Number, Last Name, First Name,
Client ID Number
```

Let's use the same index for the lookup on the Last Name field.

The Lookup Field List

You probably don't care to have the Attorney ID Number field in the lookup field list on the Last Name field because you already know who the Attorney is when you're in the Client Panel (you picked him or her before penetrating the panel link). So having their Attorney ID Number field in the lookup is a waste. Instead, we'll have the lookup field list start with Last Name, followed by First Name.

Note that the above scheme refrains from making the first field in the lookup field list be the first field in the lookup index, thusly violating the previously cited rule in DataPerfect manual. Nonetheless, as long as you entered that panel with the noted panel link, lookups designed this way will sort by Last Name; keystroking by Last Name will narrow down the correct record or records; and **Tab** will cause a subsort on the First Name field. This won't work properly, however, if you enter the panel directly from the Panel List.

What's happening here is that, after panel link penetration, DataPerfect is allowing the Last Name lookup to sort on the first field that follows the common fields in the panel link's field list, thusly having the sort ignore the Attorney ID Number field all together. The field that immediately follows those common fields is the Last Name field.

So, I can define one Attorney Panel panel link that uses the

```
Attorney ID Number, Last Name, First Name,
Client ID Number
```

index in the Client Panel, and define that panel link to take me to Client Panel's Last Name field, allowing lookups there by Last Name. I can then define a second Attorney Panel panel link that uses the

```
Attorney ID Number, First Name, Last Name,  
Client ID Number
```

index in the Client Panel, and define that panel link to take me to the Client Panel's First Name field, allowing lookups by First Name. And finally, I can define a third Attorney Panel panel link that uses the

```
Attorney ID Number, Client ID Number, Last Name,  
First Name
```

index in the Client Panel, and define that panel link to take me to the Client ID Number field, allowing lookups by Client ID Number.

You'll see later that, by virtue of DataPerfect 2.3's *Smart Lookups* algorithm, I can now reduce those three parent panel panel links to one, without losing the ability to perform all three distinctly sorted subpanel lookups. For now, though, just know that the first field in the lookup field list need not be the first field in the lookup index if you typically enter the panel via a panel link and the lookup field is the field that follows the common fields of the panel link.

The Data Link Subgroup Lookup

[Refer to UD.STR
Find the *Data Link Subgroup Lookup* series.
Load the Transaction Panel.]

The Data Link Subgroup Lookup offers the definer a way to strategically filter what records the user sees during a Create or Edit mode lookup on a data link. Though the DataPerfect 2.3 manual fails to mention the Data Link Subgroup Lookup, the README file that was found on the shipped diskettes does, without referring to it as the Data Link Subgroup Lookup. Here's the relevant README passage:

DATAPERFECT: Lookup

```
Performing a lookup on a data link while  
you are creating or editing a record will  
display records from the linked panel based  
upon the data link Key Field List. If the  
first field in the Key Field List is the  
data link field, you will be able to see  
all records in the linked panel; otherwise,  
you will see only the records that match  
the fields of the Key Field List.
```

[README 02/01/93]

Starting with version 2.3, the data link's field list also acts as a record filter for the lookup display during Create or Edit. This allows you to define a data link in such a way that, during Create or Edit, a lookup performed on it will display a defined subgroup of the linked panel's records—thus the name, Data Link Subgroup Lookup.

If you perform a lookup on a data-linked field during Create or Edit, the lookup will display the linked panel's records filtered by all fields preceding the data-linked field in the data link's field list (i.e., all the fields preceding the data-linked field in the field list must match the corresponding fields in the displayed record).

So, if the data-linked field is the *first* field in the data link's field list, then no fields precede it in the data link's field list. Performing a lookup on this field during Create or Edit will effectively display an *unfiltered* display of the linked panel's records (*all* the linked panel's records will be displayed). This effectively keeps things the way they were with DataPerfect 2.2, because it fails to create a Data Link Subgroup Lookup.

To create a Data Link Subgroup Lookup, the data-linked field must come somewhere after the first field in the data link's field list. If the data-linked field is the *second* field in the data link's field list, the lookup will display the linked panel's records filtered by the *first* field in the field list. If the data-linked field is the *third* field in the field list, the display will be filtered by the *first* and *second* fields in the field list. And so on. Note, though, that if the data-linked field is *absent* from the field list, the lookup display will be *empty*, displaying the *No Data* message.

Let's clarify this with some examples. Suppose you run a local software retail store, like Egghead. Like Egghead, you sell more than just software—your customers can also purchase books and peripherals, like printer cables. Let's say the categories of Software, Books, and Peripherals exhausts everything you sell.

We'll place all those items for sale in the Item Panel, which will have at least two fields: Item and Category, where the latter field will be a U4 field that takes three possible values: SOFT, BOOK, or PERI. We'll create two indexes in the Item Panel, one sorting by Item, the other by Category. Pretty straightforward.

Now the Transaction Panel. There, we'll have at least the Category and Item fields again, along with the usual Date and Amount fields. On the Item field we'll place a data link that ties the field to the Item Panel, and have it land on the Category field. The data link will use the {Category, Item} index in the Item Panel, and have a field list consisting of Category and Item, in that order.

Here are the two panels, showing minimal configurations:

Category: [text box]
Item: [text box]

Item Panel

Date	Cat.	Item	Amount
[text box]	[text box]	[text box] ♦	[text box]

Transaction Panel

When the user enters the Transaction Panel and goes into Create mode, he first fills in the Date and the Category (SOFT, BOOK or PERI). Then he **Tabs** to the Item field and hits **F8** or **Up Arrow**. What DataPerfect does at this point is where version 2.3 differs from 2.2. The lookup performed on the Item field during Create or Edit will show only records matching the Category field to its left. So if the user filled in *SOFT* in the Category field, hitting **Up Arrow** on the Item field during Create or Edit will display only a list Software records. Under DataPerfect 2.2, you would have seen *all* records in the Items Panel during this Create or Edit lookup.

Again, if a lookup is performed on a data-linked field during Create or Edit, DataPerfect first examines what field you're performing the lookup on (in this case, the Item field of the Transactions Panel). Then DataPerfect looks to see if that field is on the data link's field list. If it isn't, you get a recordless lookup, displaying the *No Data* message. If it *is* in the field list, the lookup display will be filtered by all fields that precede it on the data link's field list (in this case, it was filtered by the Category field).

So, a good rule to follow here is this. If you attach a data link to a field that is to be used as a sort of Pick List field during data entry, then make sure the field list assigned to that data link has the data-linked field in a desirable spot. This will be determined by whether or not you want the lookup display to be filtered. If so, make sure the lookup field immediately follows the filter fields in the link field list.

Making Lookups Look Better (Browse Mode)

[Refer to UD.STR.

Find the *Making Lookups Look Better* single-panel series.]

One way to spruce up your lookups is to use hidden fields in lookup field lists. Say you have a lookup that displays Last Name, First Name, and Address. Right now, the columns look like this:

Abramson	Helen	Los Angeles	CA 90024
Jones	Jim	Santa Monica	CA 90403
Watson	Alan	Beverly Hills	CA 90291

Hidden fields allow for this sort of lookup:

Abramson, Helen	Los Angeles, CA 90024
Jones, Jim	Santa Monica, CA 90403
Watson, Alan	Beverly Hills, CA 90291

The lookup is still performed on the Last Name field, and the index used has Last Name and First Name as its first two fields, but the lookup field list starts with the two hidden fields shown above.

A simple version of the first hidden field in the lookup field list would have a formula like this:

```
cat.t[P1F1; ", "P1F2]
```

where P1F1 is the Last Name field and P1F2 is the First Name field.

The second hidden field in the lookup field list would have a formula something like this:

```
cat.t[P1F3;"", "P1F4;  
      " "apply.format["N99999-9999;;E";P1F5]  
      ]
```

where P1F3 is the City field, P1F4 is the State Code field, and P1F5 is the Zip Code field. Note that the `;;E` modifier is used here, even though this isn't a report field. It can be used in formulas like this, even when part of a panel field instead of a report field. It just won't have any effect if you use it as a panel field modifier directly. Its effectiveness comes via the `APPLY.FORMAT` function. Try it.

Smart Lookups (Browse Mode)

[Refer to UD.STR.
Find the *Smart Lookups* series.
Load the Attorney Panel.]

I want to thank Ray Babbitt of Novell/WPCorp for his help in getting this information to you. The information in the flow charts was derived mainly from personal telephone and eMail conversations with Ray.

Though DataPerfect lookups are, on the surface, pretty straightforward, their inner workings can be very complex. DataPerfect application developers found this out when they upgraded from DataPerfect 2.2 to 2.3. Many of their lookups, all of which worked beautifully under 2.2, worked terribly under 2.3. They worked, but not the way they did under 2.2.

Version 2.3 introduced a new logic behind Browse mode lookup display. What fields will display, and what index will sort that display, is more under the control of this new behind-the-scenes algorithm than under the control of the designer of the database. Sort of, anyway. Let's explain.

Under version 2.2, when a developer defined a lookup, he assigned two things to that lookup: a *lookup index* and a *lookup field list*. Let's say the user entered that panel from the Panel List. Here, when the user sits on that field under DataPerfect 2.2 and performs a Browse mode lookup, those two elements control the event. If no lookup was defined for that field, a Browse mode lookup would be governed by the lowest numbered index (probably index 1, unless it was deleted) and would display that index's field list.

Alternatively, if they entered that panel from a panel link and attempted the same Browse mode lookup, the index that will sort that field's lookup will be the one that was attached to the panel link, no matter what index was assigned by the developer to that field's lookup definition. Let me explain this with an example.

In the billing application I use to run my practice, I have an initial panel called the Doctor Panel. There the user picks the Doctor of the patient in question. Next the user penetrates a panel link to the Accounts Panel, where they pick the Account

(Patient) in question. Then they penetrate a panel link to the Case Panel. Then the Transaction Panel where they create, edit or delete records. Very logical and predictable.

Let's back up a little. I said that after picking the Doctor in the Doctor Panel, the user penetrates a panel link to the Account Panel. Then the user performs a Browse mode lookup to find the Account in question. In this case, the Doctor Panel panel link that takes the user to the Account Panel is tied to this index:

```
Doctor Code, Last Name, First Name, Middle Name,  
Account Number
```

where the Name fields there are the Patient's Names, not the Doctor's. Of course that panel link has a single field on its field list:

```
Doctor Code
```

If the user penetrates that panel link and lands on the Last Name field in the Account Panel, and then does a lookup, all is well. They'll see a lookup displaying Patients sorted by Last Name. But if the user only remembers the Patient's First Name, and then moves to the First Name field and does a lookup, all will *not* be well under DataPerfect 2.2. The index that will sort that lookup under DataPerfect 2.2 is the index that's tied to the panel link that took the user to that panel. So, though the lookup field list for that field might begin with the First Name field, it won't sort that way. It'll sort by the Last Name field. Before version 2.3, a panel link's index always governed how Browse mode lookups would sort in the target panel, no matter what the developer put in the lookup definition for the field on which the lookup is being performed.

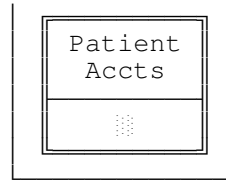
The way around this was to create more than one panel link in the parent panel (Doctor Panel), where each one used a different target panel (Account Panel) index. That's what I had to do before version 2.3. I had three panel links, with panel text describing what they did:

Patient Accounts		
Sorted by		
Last Name	First Name	Acct Number
☐	☐	☐

*Lower Left Corner
of Doctor Panel*

So, if the user wanted to enter the Account Panel and find a Patient by Last Name, they use the first link. Otherwise they used one of the other two. This worked fine, but was a little cumbersome. They had to keep exiting the Account Panel and re-entering it whenever they wanted to change the sort for their Account Panel lookup.

This changed with version 2.3. Here's what that same lower left corner of the Doctor Panel looks like now:



That's the first of the three links in the old Doctor Panel—the one previously labeled *Last Name*. But now, when the user penetrates that link, they find a Browse mode lookup in the Account Panel sorts records by Last Name if the cursor is on the Last Name field, sorts records by First Name if the cursor was on the First Name field, and sorts records by Account Number if the cursor is on the Account Number field. All this, even though the user arrives in the Account Panel via a panel link that's tied to an index that sorts by Last Name.

With this new power in lookups comes added complexity in understanding the logic that governs Browse mode lookup displays. Put simply, if you were used to designing lookups in version prior to 2.3, you might find they just don't seem to display the way you want with version 2.3. Let's go over the new logic.

The Smart Lookups Algorithm

Here I'll attempt to explain how DataPerfect chooses to sort and display Browse mode lookups. This algorithm, introduced with version 2.3, governs what has come to be called *Smart Lookups*. The flow charts I present convey a logic that may seem terribly complicated, but I have faith that after a few readings you'll find the beauty in it all. Neither the DataPerfect 2.3 manual nor any accompanying README file even mentions Smart Lookups.

In the charts that follow, I speak of an index *activating* a field in this or that situation. This only applies to situations where you entered a panel from a panel link that has a field list, or from a menu using a keyword. In such situations, DataPerfect assumes you don't want records in the target panel sorted by the *common fields* of the link index (the fields that correspond to the link field list or menu keyword), because all the records you're about to access in the target panel have the same values in those fields. Consequently, DataPerfect strips the *common fields* from the sorting process, sorting instead on the next fields in the index. That's what I mean by the index *activating* that field: *it upgrades that field's status to primary sort field for that lookup event*.

For instance, if you just went from the Customer Panel to the Transaction Panel, and did so with a panel link that has as its field list, the Customer ID field, you know all the records you're about to see in the Transaction Panel are of that particular Customer. So why bother including the Customer ID in the sorting process of a lookup in the Transaction Panel? Consequently, DataPerfect takes the liberty of

stripping the common fields from the sorting process, sorting instead on the next field in the index.

Using a different example, if we penetrate a panel link that has as its field list, the Attorney ID field, then any index in the subpanel that has Attorney ID as its first field will activate its second field. In this situation, the

Attorney ID, Client ID

index will activate the Client ID field. Likewise, the

Attorney ID, Trial Date

index will activate the Trial Date field. But the

Client ID, Attorney ID

index won't activate any field at all in this situation because it doesn't start with the link's field list (the Attorney ID field).

Using the same logic, if we penetrate a panel link with

Attorney ID, Client ID

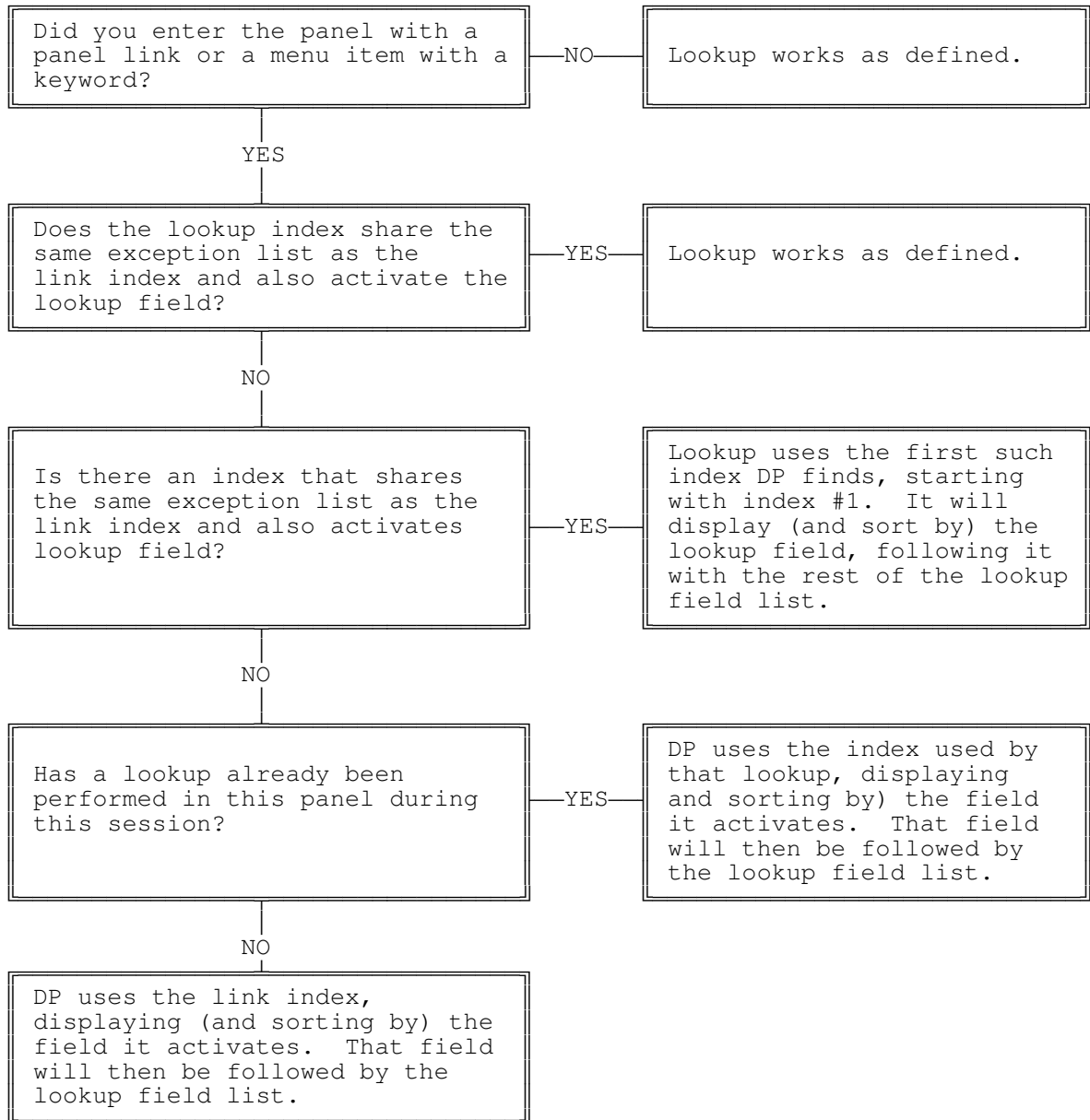
as its field list, then any subpanel index with Attorney ID *and* Client ID as its first two fields (in that order) will activate its *third* field.

Again, when I say this or that field in the target panel's index is *activated*, I mean a lookup at that point in time will cause the display to sort by that field, even though that field isn't the first field in the index in question. So, though the link index that got us into, say, the Client Panel starts with the Attorney ID field, a Browse mode lookup performed on the Client ID field may sort on the Client ID field instead of the Attorney ID field.

Now the logic (algorithm). When you tell DataPerfect you want a Browse mode lookup by hitting **Up Arrow** or **F8**, DataPerfect will try very hard to find an index that activates the lookup field (the current field). That is, DataPerfect will look for an index that makes the lookup field the primary sorting field. DataPerfect will do this while keeping it consistent with the panel link's two filters: the field list of the link that got you into that panel in the first place, and any exception list tied to that link's index.

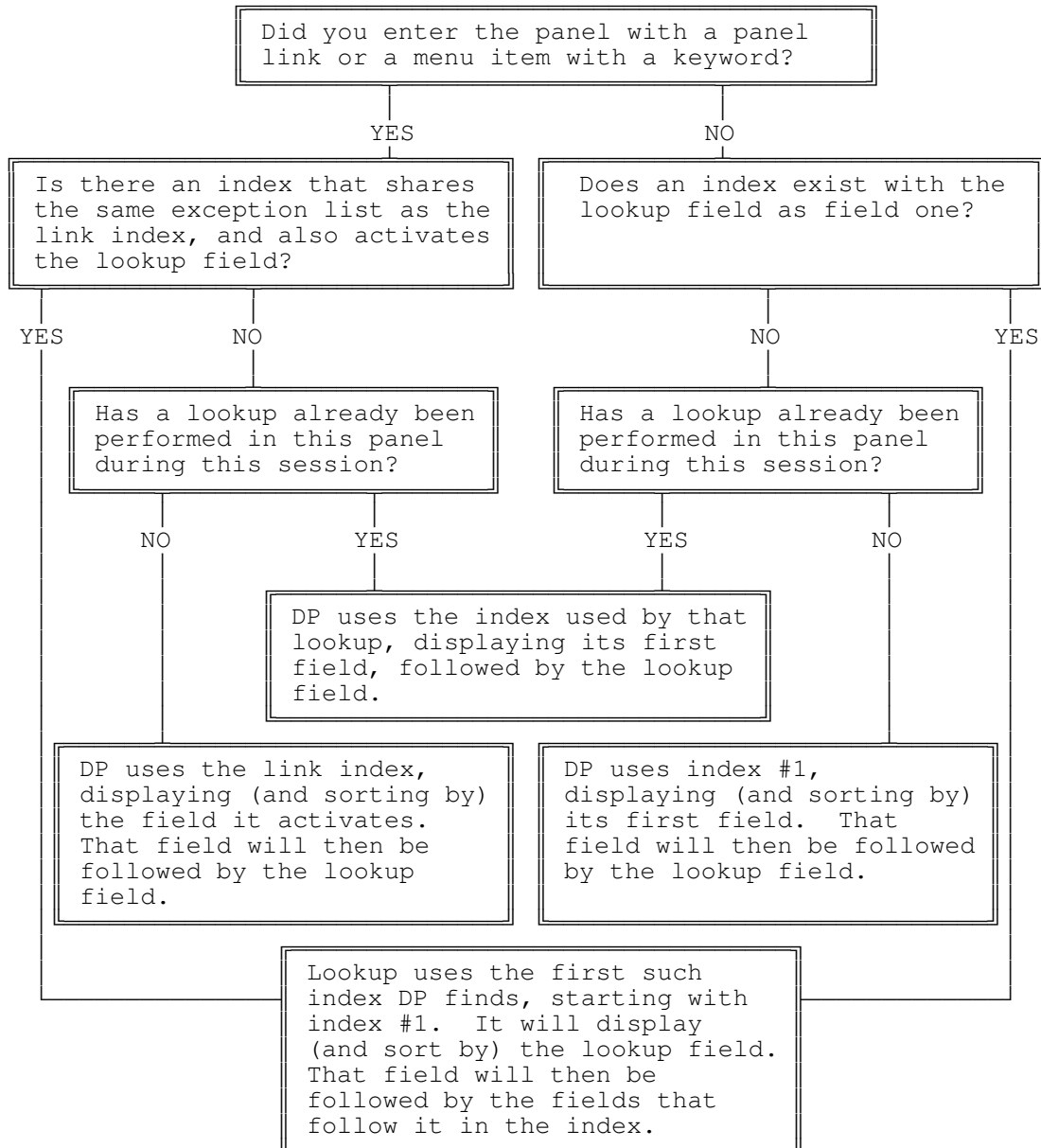
Smart Lookup Algorithm 1

When the Lookup Field Has a Lookup Definition



Smart Lookup Algorithm 2

When the Lookup Field Doesn't Have a Lookup Definition



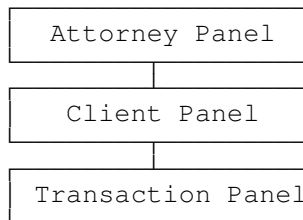
Strategy in Defining a Browse Mode Lookup Using the Smart Lookups Algorithms

You don't really need to understand those flow charts if you adopt three rules in defining a lookup. These should work for you in almost all situations:

1. Construct the lookup field list the way you would like it to be in all situations, whether entry into that panel is keyed or unkeyed.
2. Define the lookup index to be one you would like to sort lookups after an unkeyed entry into that panel.
3. For each possible keyed entry into that panel, make sure there exists an index that activates the lookup field.

In the above rules, I refer to entry into the panel in question as being keyed or unkeyed. A *keyed* penetration into a panel is either via a panel link with a field list or via a menu item with a keyword. An entry into a panel is *unkeyed* when it's via the Panel List, a panel link without a field list, or a menu item without a keyword.

Now some examples to clarify this. Let's go back to our Attorney's Office application. Its overly simple panel hierarchy is the following:



Each Attorney Panel record has a unique Attorney ID Number. Each Client Panel record has a unique Client ID Number. Each Transaction Panel record has a unique Transaction ID Number. The two panel links are defined this way:

Panel Link 1

Purpose: Takes user from Attorney Panel to Client Panel
Index: Attorney ID Number, Client ID Number
F/List: Attorney ID Number

Panel Link 2

Purpose: Takes user from Client Panel to Transaction Panel
Index: Attorney ID Number, Client ID Number, Transaction ID Number
F/List: Attorney ID Number, Client ID Number

Now I go to the Client Panel and define a lookup on the Last Name field. Following the three-step rule I outlined, I'll decide how I'd like to see my records displayed during Browse mode when I hit **Up Arrow** or **F8** on that field *in all situations*, regardless of whether I enter that panel via key fields or not. I decide that I'd like the its lookup field list to be

```
Last Name, First Name
```

I want the lookup on the Last Name field to display those two fields whether or not I enter that panel keyed. That satisfies rule 1.

Next, I choose an index I'd like this lookup to use *after an unkeyed entry into that panel*. I choose the following:

```
Last Name, First Name, Client ID Number,  
Attorney ID Number
```

That satisfies rule 2.

Finally, I make sure that, for each keyed penetration into this panel, there's an index that the Smart Lookup algorithm will pick, and will sort consistent with the field list chosen in step 1. The only panel link right now that takes me to the Client Panel is one that has

```
Attorney ID Number
```

as its field list, so I want at least one index that starts with Attorney ID Number (which is the single *common field* in this case), immediately followed by the Last Name field. The following index works:

```
Attorney ID Number, Last Name, First Name,  
Client ID Number
```

When penetrating Panel Link 1, the above index will activate the Last Name field (the field immediately following the *common* field in the index).

Now, for any other keyed entry into the Client Panel, I'll need to make sure there exists an index that activates the Last Name field. In such an application, it's unlikely there'll be a panel link that has a field list other than {Attorney ID Number}, but you get the idea.

What if you don't want the lookup field to be the first field in its own lookup field list?

This takes a little work. In case you wonder why you may want to do this, let's return to the Attorney's Office application.

Instead of having the user perform a Client ID Number lookup on the Client ID Number field (it's protected with the ::N modifier, so the user can't **Tab** to it), we'll use the Middle Name field for that purpose. Further, we'll let the user know about that in our help screens. To do this, we use a different method than previously

outlined, because we don't want the lookup routine to activate the lookup field (the Middle Name field). Rather, we need it to activate the Client ID Number field.

Given that we're entering the Client Panel via a panel link with Attorney ID Number as its field list, we assign

Attorney ID Number, Client ID Number

as the index for this lookup, thus activating the Client ID Number field in the Client Panel. We then assign the lookup a field list to the Middle Name field that starts with the Client ID Number field. That's all, as long as we enter the Client Panel via the panel link.

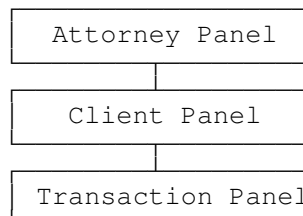
How is this different than our more typical example, where we want the lookup field to be the activated field? It's the index here that makes the difference. When we want the lookup field to be the activated field, we use an index that would work when entering this panel unkeyed (rule 2). But in this case we want to activate a field other than the lookup field. Here we choose an index that activates this other field in a keyed entry into the panel. This is only going to work, however, if you arrive into this panel in the same keyed way every time. That is, you never enter this panel unkeyed (e.g., from the Panel List), and you never enter this panel keyed on a different field list (e.g., from a panel link with a different field list than the one originally planned for). In any of these other situations, the lookup won't sort properly. For instance, if you enter the panel from the Panel List, the lookup will sort by Attorney ID Number, but with a lookup field list starting with Client ID Number. But, since you shouldn't give users access to the Panel List anyway, this may be a moot point in most applications.

Reasons for Assigning a Lookup to a Hidden Field

I talked about why you might want to put a hidden field (::H) on a lookup field list. Why might you want to assign an entire lookup definition to a hidden field? If the user never lands on it, why bother assign a lookup to it? Here are two reasons:

Facilitating a Subrecord Lookup

The most common reason I know of to define a lookup on a hidden field is to affect how a panel link displays dependent records during an **F8** Browse mode lookup on that panel link. For instance, consider our Attorney's Office application again:



Suppose you would like to position the cursor on the panel link that takes you from the Attorney Panel to the Client Panel and view, via an **F8** Browse mode lookup on that link, Clients sorted by Last Name. Don't forget that an **F8** lookup on a panel link in Browse mode displays *dependent* records in the *subpanel*, not records in the current panel. Further, you would like this lookup to display the following fields:

Last Name, First Name, Client ID Number, Balance

Further suppose that's *not* the type of lookup you want to assign to the first field you land on when penetrating that panel link. That is, you would like that panel link to take the user directly to the Last Name field, but you want the Last Name field's lookup to display the following fields:

Last Name, First Name, Client ID Number, Home Phone,
Work Phone

Or, alternatively, suppose you want that panel link to allow an **F8** Browse mode lookup that sorts by Last Name, but hitting **Down Arrow** on that link in Browse mode takes the user directly to the Client ID field, not the Last Name field.

Well, at first, that may seem impossible, because hitting **F8** on a panel link during Browse mode is supposed to produce the lookup display based on the lookup definition *of its target field*. But don't forget that the target field of a panel link can be hidden, which will make it a field the user never lands on after penetrating a panel link. So you're free to give the panel link an **F8** subpanel lookup that's independent of the field the user actually lands on in the subpanel, by making the target field of panel link a hidden subpanel field. Define the lookup on that hidden field to your heart's delight, but make sure the field you want the user to land on after penetrating the panel link is the first editable field in the Edit Order that follows this special purpose hidden field.

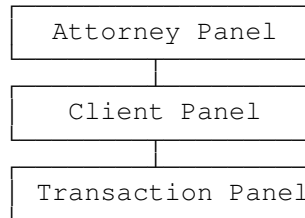
Facilitating a Lookup in a Report

The User Chooses Next Record By Lookup code (see *User Chooses Next Record By Lookup* in my **Iteration Control** chapter) allows the definer to present the user with a lookup during report generation. The field list for that report lookup is the field list found in the lookup definition assigned to the active sorting field of the index that's active at that point in the report. If no lookup definition has been assigned that field, the report lookup will display all and only fields in the active index, starting with the active sorting field.

Suppose you put a report lookup in a section of a report that uses a reverse index (see *Sorting Backwards with Reverse Indexes* in my **Indexes** chapter). This index will probably sort by a hidden field at that point in the report. If you don't assign a lookup field list to that hidden field, the report lookup will probably display what appears to be meaningless numbers (the reverse field itself).

Reasons for Assigning a Lookup to a Non-Updatable Field

Other than the two reasons outlined above for assigning lookup definitions to a hidden field, both of which can be good reasons for doing so to a non-updatable field, there's a third reason you might want to assign a lookup to a non-updatable (::N) field. Consider our Attorney's Office application again:



Suppose you want the Client ID Number field in the Client Panel to be defined as non-updatable, keeping its value out of the hands of the user. It might be a field that updates on creation based on a recursive link scheme (see *The Recursive Panel Link* in my **Links** chapter). Suppose also you want the user to be able to perform lookups on that field, and have them sorted by Client ID Number. In such a case you could go ahead and define the lookup on that field and let the user know via a help screen that he can access the Client ID Number field for lookup purposes by using the shifted white cursor keys (or Alt-*num* keys on a notebook computer—see *Sizing and Moving the Panel* in my **For Beginners** chapter for a discussion of the Alt-*num* method of cursoring).

Also note I discuss an alternative to using a lookup on a non-updatable field above, in *What if you don't want the lookup field to be the first field in its own lookup field list?*

A Note about Saving a Lookup Definition

It's easy to inadvertently save a lookup definition without an assigned index. This happens after examining or editing an existing lookup definition. While sitting on a field, you decide to examine that field's lookup definition. You **Shift-F8, 1** and then look at the fields chosen for its field list. You either leave them alone or change them, and then hit **F10**. Now you find yourself in the lookup index screen. The index you selected before appears there. It looks fine, so you hit **F10**.

You just removed the index from the lookup definition! When you access the **Shift-F8, 1** screen to inspect a lookup definition, you're first presented with the Define Lookup Field List screen. Fine. If it looks okay, just hit **F10**. Next you see the Index Selection screen. If you just hit **F10** or **F7** here (or **F1**, for that matter), you just told DataPerfect you don't want any index assigned to this lookup definition. That's telling DataPerfect to feel free to just use the Smart Lookups algorithm to decide how to sort this lookup. When you see that Index Selection screen, and you want an index

assigned to this lookup definition, select it with **F4**, even if it was already selected previously.

This issue arose with version 2.3. Before version 2.3, you had no option to save a lookup definition without an index. With version 2.3, you do this by simply hitting **F7**, **F10**, or **F1** while in the index selection screen.

To see what fields and index are in fact assigned, without going through these two screens, just hit **Shift-F8** and take a look at the area near the bottom of the screen, where you'll see your lookup definition (if one exists at all). It should look something like this:

```
Record Lookup Index and Field List: 5 - 35 2 1
```

That line (which would be missing if this field doesn't have a lookup definition) tells you this field has a lookup definition that uses index 5, and a field list of fields 35, 2, and 1.

If you go into that lookup definition and hit **F10** on the Define Lookup Field List, and again on the Index Selection screen, you'll get this line on your **Shift-F8** screen:

```
Record Lookup Index and Field List: 0 - 35 2 1
```

That zero on the left says the lookup doesn't have an index assigned to it. By assigning it no index, you're telling DataPerfect you're fine with the Smart Lookups algorithm deciding what index to use with that field list on any given occasion.

This section is extremely important. Reread it if you don't see it's significance. You can easily blow away a working lookup definition if you don't understand this.

Troubleshooting Lookups

Problem

Your lookup seems to display properly, but keystroking doesn't seem to move the highlight bar properly.

Solution

The lookup field list is out of sync with the lookup index. This would be like having the Last Name field as the first field in the lookup field list, but the index is sorting primarily on the First Name field. This sort of thing can arise, of course, if you simply chose the wrong index for that field's lookup definition. But if the index seems appropriate for the current field's lookup field list, consider how entry into that panel is taking place here.

If you enter that panel via a panel link with a field list or via a menu option with a keyword, then the field that's acting as the primary sorting field during the lookup may very well not be the field you think it is. For instance, if the panel link that gets you into this panel has Account Number on its field list, then, during a lookup on the Last Name field DataPerfect will look for an index in that panel that

begins with Account Number and Last Name. It won't use an index that begins with Last Name, even if it's assigned to the lookup definition of that field. If there's no index in that panel beginning with Account Number and Last Name, this lookup will essentially have no index on which to sort the displayed records. For more on this, see *Smart Lookups (Browse Mode)* in this chapter.

Problem

A report lookup (resulting from a User Chooses Next Record By Lookup code) shows an undesired field in the first column of the lookup display. That field is displaying a five-digit number instead of a date, or a negative number instead of a positive one, even though lookups in that panel during Browse mode always look fine.

Solution

Examine the index this report lookup is using. Locate the field in that index that's the primary sorting field in that part of the report. If this is in the main report, that field would be the index's first field. If this is a subreport, it's the field that immediately follows the link field list as defined by the subreport definition.

Now see if that field has a lookup field list defined. That's what the report lookup uses, if it exists. If it doesn't have a lookup field list, the report lookup will display all the fields in the index active in that part of the report. So, to control a report lookup display, give that active sorting field the proper lookup field list. Even if the active sorting field at that point in the report is a hidden G9999 field holding a five-digit Julian Date value, or a hidden G-9999 field, you can still give it a lookup field list that begins with a displayed date field or displayed G9999 field.

Problem

Sometimes a particular field's lookup displays properly, and sometimes it doesn't. It seems to display *improperly* after performing lookups on other fields.

Solution

Unless you actually assign a lookup definition to that field, DataPerfect will choose the lowest number index that makes that field the primary sorting field. If no index makes that field the primary sorting field, and a lookup has already been performed in that panel this session, DataPerfect will use the index used by that other lookup. Take a look at the flow chart on this in *Smart Lookups (Browse Mode)* in this chapter. So, to solve this problem, assign a lookup definition to that field.

Problem

When you enter a particular panel from the Panel List, no matter what field you're on, performing a lookup fails to show all records in the panel. You've checked the lookup definitions on these fields and find none of them have indexes with Exception Lists assigned to them.

Solution

Check the lowest numbered index in that panel. It probably has an Exception List on it. When you enter a panel from the Panel List, you're *initially* under the control of the lowest numbered index. Though DataPerfect gives way to a field's lookup definition, it does so only if *consistent with an Exception List that may be assigned to it*. This is precisely why the lowest numbered index should *never* have an Exception List on it.

Problem

When you enter a particular panel from a particular menu item or panel link, no matter what field you're on, performing a lookup fails to show all records in the panel. You've checked the lookup definitions on these fields and find none of them have indexes with Exception Lists assigned to them.

Solution

Check the index assigned to that menu item or panel link. It probably has an Exception List on it. When you enter a panel via a menu item or panel link, you're *initially* under the control of the index assigned to that entity. Though DataPerfect gives way to a field's lookup definition, it does so only if *consistent with an Exception List that may be assigned to it*.

Problem

Though lookups in other panels seem okay, lookups in a particular panel seem uncharacteristically slow. Moving the highlight bar up and down the lookup display is as slow as molasses.

Solution

Either turn off Auto Display or go with less computed fields in that panel. Either of these slows down lookups. Auto Display is accessed with **Alt-F8, Alt-F8**, which gives you the Panel Options menu:

```
Panel Options
 1 - Edit Filename           5 - Change Edit Order
 2 - Change Color           6 - Edit Panel Name
 3 - Auto-Save (Y/N)        7 - Recompute Field Offsets
 4 - Auto-Display Record (Y/N) 8 - Auto-Edit/Auto-Create/Menu
Browse Change => Menu
Display each record during Lookup.
Selection: 0
```

Note *Display each record during Lookup* in the lower left corner. That disappears when you hit **4** above, and reappears when you hit it again. Turning that off will speed lookups, but will keep the user from seeing records in the panel change while moving the highlight bar in the lookup display. This holds for report lookups in that panel too.

This chapter is directed at both inexperienced and experienced DataPerfect application developers.

Introduction

As you find out early in this game, you don't have a working panel without at least one index defined for that panel. What exactly are you defining when you define an index?

An index is an ordered set of pointers that direct the .STR file to specific records in a particular panel's data file. An index's set of pointers are determined by the ordered field list the definer assigns that index. So, when you create an index, you create a way of accessing its panel's data file. You don't affect its data file directly—that is, the data file on disk isn't changed at all when you create an index for its panel. Rather, you offer the .STR an ordered access to those records.

An index determines how a data file's records sort during particular events. For instance, a definer-determined index will determine the sort order of records processed by a report, or the sort order of records displayed during a lookup. A report that prints Monthly Statements sorted by Last Name uses a different index than one that prints them by Account Number. The same for lookups: one may display Clients sorted by Last Name, and another by First Name, or even Birthdate. Each such lookup will be defined in a way that uses a different index.

So, again, an index determines how records in its panel's data file sort when a particular event uses that index. And, though defining an index amounts to creating an ordered list of fields found in a panel, defining that index has no affect on that panel's data file on disk. However, defining an index directly affects *both* the .STR and .IND files. That is, unlike a panel's data file, both the .STR and .IND files are rewritten to disk when you create, edit or delete an index. Let's examine what role each of these files plays in indexing.

The .IND File

The .IND file stores all indexes for all data files in a single application. You're limited to 200 indexes per panel. The .IND's file size limitation is 8 gigabytes. When you create or edit an index, DataPerfect puts information about that index in the .IND file, and puts pointers in the .IND file that direct that particular index to each and every record. Unless there's an Exception List placed on that index (see *Exception Lists* later in this chapter), there will be a pointer in that index's portion of the .IND file for *every* record in the data file associated with that index's panel.

Within a single data file's group of indexes, what makes one index's set of pointers different from another's is their sort order. One index might have pointers that point to all records in the Client Panel (one pointer per Client), sorted by Last Name. Another index might have pointers that point to all the same records, but sorted by First Name. The Last Name index considers, say, John Adams' record as the first record in the Client Panel, and Alice Zane's as the last; whereas the First Name index will see Alice's record before John's. Put another way, the Last Name index will store pointers for records in the Client Panel data file in the .IND file in such a way that John's record gets a lower number (earlier in the index) than Alice's. The opposite is true of the First Name index.

These pointers in the .IND file direct the .STR file to records in a panel's data file in an ordered fashion, depending on what database event is active. By a database event, I mean an event like a lookup that's displaying, or a report that's running. An essential part of the definition of each of these sorts of database events is the index the definer assigns to it.

How Indexes Sort

An index definition consists simply of an ordered set of real (i.e., not computed) fields in a particular panel. We call this set of fields an *index field list*. Understanding how an index will sort records if it has only one field on its field list is easy. It sorts by that field alone.

Consider a panel's data file to have only two records:

```
John Adams  
Alice Zane
```

Say Index 1 in that panel has a field list consisting simply of the Last Name field. Then a lookup using that index will display records just as you see above.

Now consider Index 2. Its field list consists simply of the First Name field. A lookup using *that* index will display records like this:

```
Alice Zane  
John Adams
```

What confuses some users in the first stages of application development is how an index sorts records when its field list has more than one field. In the above data file example, say Index 3 has this field list:

```
Last Name, First Name
```

Assume Index 4 has this field list:

```
First Name, Last Name
```

What do fields after the first field in an index field list do? Well, think of it this way. Index 3 will first sort records alphabetically by the first field in its field list (the Last Name field). If no two records have the same value in that field, the sorting is done. If two or more records have the same value in the first field in the index field list, then DataPerfect will attempt to break the tie by looking at the second field in the index field list. If there are two or more records with the same value in the first *two* fields, DataPerfect will attempt to break the tie in the third field in the index field list. Index 4 will use the same sorting logic, but start with the First Name field and move to the Last Name field only when needed to break a tie.

Now, in our example above, DataPerfect doesn't need to look any further than the first field of either index's field list. But suppose that data file has these records:

```
John Adams
Alice Zane
John Smith
Sally Zane
John Jones
Jack Zane
```

In the above example of a data file, Indexes 1 and 2 won't work. Again, these two indexes each have only a single field in its index field list. Why won't they work? Because each one can't break ties on the first field in its index field list. There are two or more records in the above data file that have the same value in the Last Name field, so Index 1 can't break the tie on its single-field field list. And there are two or more records there that have the same value in the First Name field, so Index 2 can't break the tie on its single-field field list. To break these ties, we need at least one more field in each of these index field lists. Here's how Index 3 sorts the above list of records:

John Adams	}	<i>Sorted by Last Name field, which is first in index field list.</i>
John Jones		
John Smith		
Alice Zane		<i>Broken ties using First Name field, which is second in index field list.</i>
Jack Zane		
Sally Zane		

And here's how Index 2 sorts that same list of records:

Alice Zane	}	<i>Sorted by First Name field, which is first in index field list.</i>
Jack Zane		
John Adams		<i>Broken ties using Last Name field which is second in index field list.</i>
John Jones		
John Smith		
Sally Zane		

Uniqueness and Picking Fields for the Index Field List

As you can surmise, there might be cases where there aren't enough fields in an index to break all ties that might occur in a panel's data file. What if the above data file has another record, this one for another John Jones? This gives us two records with identical values in all fields of an index—in this case, *both* indexes. After finished creating or editing an index definition, if DataPerfect finds two or more records in that data file that match on all fields in that index field list, you get the following error message:

```
Warning: Redefine this index. The index keys are not unique.
```

This means two or more records in that data file match on all fields in that index field list. So you'll need to either add more fields to the index field list, or use different fields completely.

A situation that would cause above error message would be this. Your panel has a data file with the following records:

```
John Adams
John Smith
Sally Zane
```

So far you only defined one index in this panel, which we'll call Index 1. Its field list consists of but one field, the Last Name field. Fine. So far, no two records in that list share the same value for the Last Name field, so as far as DataPerfect is concerned, the data file has no duplicates. Next you create Index 2, whose field list consists solely of the First Name field. Well, note that two records in that list share the same value for that field. When you exit the Define Index screen, DataPerfect will beep with the *Redefine this index* message.

Alternatively, you may have created such an index before ever adding records to that panel's data file. In that case, the moment you attempt to save a record that matches an already existing record on all fields of an index field list, you'll get this message:

Note
This record is not unique. At least one index key was Found to be duplicated in the index(es).
In order to save the record, data must be entered into the Fields which will make the record unique in panel: 1

Note that the above error message refers to Panel 1. You might think it should be obvious which panel has the indexing error, because it must always be the current panel. Right? Wrong. DataPerfect allows you to define elements of a panel so that saving a record in it affects records in other data files. Such devices include the Cascade Update and the Keep A Total facilities. So when you save a new index definition, you might notice that message above appear with a foreign panel referenced.

To get around problems like two Clients matching on all their name fields, even their Middle Name fields, most application developers add an ID Number field such panels. This might be the Client's Social Security Number or one assigned by the field itself with the ::I or ::J. If we put that field on every index field list in that panel, we can have more than one Client with the same name. So, instead of having

- (A) Last Name, First Name
- (B) First Name, Last Name

as two indexes in the Client Panel, we might have the following indexes:

- (1) ID Number
- (2) Last Name, First Name, ID Number
- (3) First Name, Last Name, ID Number

A few comments about the above Indexes 1-3. Unlike Indexes A and B, Indexes 1-3 all include the ID Number field. Also note that Index 1 has only one field: the ID Number field. This last point has double-edged importance. It guarantees that no ID Number will ever be used twice. Think about this a second. Suppose Index 1 wasn't there, and all we have in that panel is Index 2 and 3. That would allow us to create two records with the same ID Number, like these:

	<i>ID Number</i>	<i>Last Name</i>	<i>First Name</i>
<i>Record 1</i>	1001	Adams	John
<i>Record 2</i>	1001	Zane	Sally

Each of those records is unique if our only two indexes are these:

- (2) Last Name, First Name, ID Number
- (3) First Name, Last Name, ID Number

That's because these two records don't match on all of their index list fields. In fact, they match on only one. Now add back Index 1:

- (1) ID Number
- (2) Last Name, First Name, ID Number
- (3) First Name, Last Name, ID Number

Index 1 won't allow the creation of both those records because they'll match on all its index field list fields (there's only one field on its field list). So Index 1 guarantees we never create two records with the same ID Number, and Indexes 2 and 3 allow for more than one Client with the same name because they'll have different ID Numbers.

Also, it's good practice to make Index 1 the single-field index that contains a record's uniquely identifying field (like the ID Number field). When you enter a panel from the Panel List, DataPerfect accesses records in that panel via the lowest numbered index. I say *lowest numbered* instead of *Index 1* because you might have deleted Index 1 sometime during development. This is a good place to put that

single-field index. And because DataPerfect uses the lowest numbered index to access the panel from the Panel List, don't ever put an Exception List on that index (see *Exception Lists* later in this chapter).

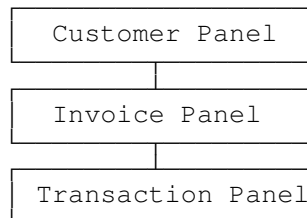
Sorting Backwards with Reverse Indexes

A developer frequently needs to use *reverse indexes* in his application. These are indexes that sort in reverse order with respect to a field in the same panel. Usually that means sorting in reverse order with respect to a date field or a numerical ID field, like an Account Number field or Transaction Number field. Let's explain.

Reverse Sorting by Number

Via a Panel Link

Let's say you have the following familiar panel hierarchy:



Further assume the Customer Panel has an auto-incrementing G99999::J Customer Number field P1F1 that's the sole occupant of that panel link's field list. Likewise, the Invoice Panel has a matching G99999::N Customer Number field P2F1, as well as having an auto-incrementing G9999::J Invoice Number field P2F2 that's used to uniquely identify each Invoice. Simple enough.

Now you'd like the panel link that takes the user from the Customer Panel to the Invoice Panel to land on that Customer's most recent Invoice. Here's how to do this:

- Create a hidden G-99999::H field in the Invoice Panel that updates on any change to

-P2F2

Let's assume its field code is P2F3. We'll name it the Reverse Invoice Number field.

- Create an Invoice Panel index consisting only of the Customer Number and Reverse Invoice Number fields, in that order.
- Assign the above index to the Customer Panel panel link. That panel link has the Customer Number as the only field on its field list already.

That panel link will take the user to the Customer's most recent Invoice in the Invoice Panel. For lookups to work correctly in the Invoice Panel, however, it must have a

```
<Customer Number, Invoice Number>
```

index. If that panel only has the

```
<Customer Number, Reverse Invoice Number>
```

index, Invoice Panel lookups performed on the Invoice Number field won't work quite right. In the latter case, performing such a lookup will indeed produce a lookup display that sorts records in reverse order (with the highest Invoice Number at the top). But when you try to type numbers in that lookup, the lookup highlight bar won't move. The lookup is actually waiting for you to type *negative* numbers because the index that's active after penetrating the panel link is the Reverse Invoice Number index. But if there *is* a

```
<Customer Number, Invoice Number>
```

index in the Invoice Panel, the Smart Lookups algorithm will use that index, provided you don't assign the

```
<Customer Number, Reverse Invoice Number>
```

index to the Invoice Number field's lookup definition.

If the lookup uses the

```
<Customer Number, Invoice Number>
```

index, it will produce a lookup with records sorted so the *lowest* numbered Invoice for that Customer is at the top, but the record you initially land on after penetrating the panel link will be the *highest* numbered Invoice for that Customer. Again, the reason you land on the *highest* numbered Invoice when penetrating the panel link is that the link uses this index:

```
<Customer Number, Reverse Invoice Number>
```

But the reason the actual lookup display on the Invoice Number field sorts with the *lowest* Invoice Number on top is because the Smart Lookups algorithm will choose this index for a lookup on that field after penetrating that panel link:

```
<Customer Number, Invoice Number>
```

So it works fine now. What this is doing is letting the panel link use a reverse sort, which places the user on the most recent Invoice (highest Invoice Number) in the Invoice Panel, while assigning a forward sort to the Invoice Number field in the Invoice Panel, allowing the user to type-to-search in the Invoice Panel to find the

desired Invoice if the most recent one isn't the one they want. See *Smart Lookups* in my **Lookups** chapter. This can be a very complicated topic.

In the Level 1 Panel

But what if you want a reverse sort on the Customer Number field in the top panel (the Level 1 panel) in this panel hierarchy: the Customer Panel. I don't know why you'd want that, but let's say you do. Well, here's the problem you face.

Let's say you assign the Customer Number field a lookup definition that has a field list beginning with the Customer Number field. Then you assign that lookup definition this single-field reverse index:

```
<Reverse Customer Number>
```

The Reverse Customer Number field will be a G-99999 field that's formulated to update on any change to

```
-P1F1
```

We'll call that field P1F6.

What's wrong with this picture? Well, when you perform a Browse mode lookup on the Customer Number field with that lookup definition, it will sort with this index:

```
<Reverse Customer Number>
```

But that index is sorting by P1F6 (negative numbers), not P1F1 (positive numbers). What will happen is that you'll find the first Invoice Number at the bottom of the lookup display (in virtue of the lookup's reverse index), and you'll see only positive numbers (in virtue of the lookup field list), but you won't be able to type-to-search on that lookup. Because the lookup is using the reverse index to sort its display, it's expecting you to type in negative numbers corresponding to values that would appear in P1F6, not P1F1.

There are a couple of ways around this. *First*, if your application gives the user access to the Panel List (I suggest you never do this), make index 1 in that panel the reverse index. Then just make sure that panel also has an index that sorts forward by Customer Number. When entering a panel directly from the Panel List, DataPerfect puts you on the first record of that panel's lowest numbered index. In this situation, then, DataPerfect will put you on the record with the highest Customer Number. But when you perform a lookup on that field, the Smart Lookups algorithm will sort by the first index it finds that sorts primarily by the Customer Number, if you haven't already assigned one to that field's lookup definition.

Second, if your application only gives users access to panels via menus (I highly recommend this), then define entry to the Customer Number Panel with a menu option that uses the reverse index. Again, as long as you create a forward-sorting Customer Number field index in that panel, and don't put the reverse index in that field's lookup definition, the Smart Lookups algorithm will sort by the first

index it finds that sorts primarily by the Customer Number, if you haven't already assigned one to that field's lookup definition. In this situation, the lowest numbered index need not be the reverse index.

Again, if you don't understand what I mean by the Smart Lookups algorithm, read up on it in *Smart Lookups* in my **Lookups** chapter.

Reverse Sorting by Date

Now let's consider the panel link that takes the user from the Invoice Panel to the Transaction Panel. Let's say you'd like that link to land on the most recent Transaction for that Invoice. This panel link has only the Invoice Number field on its field list. Let's assume the Transaction Panel has, among other fields, an auto-incrementing G99999::J Transaction Number field P3F1 and a D99/99/9999 Date field P3F2. Here's what we need to do:

- Create a hidden G-99999::H or D-99/99/9999::H field in the Transaction Panel that updates on any change to

-P3F2

Let's assume its field code is P3F3. We'll name it the Reverse Date field. Note that you can use either a G or a D field for this. Most use the G field, since it takes less screen space. Don't forget that a D field holds a five-digit number in the data file, not a date as we know it. Both fields are just holding the negative of the five-digit Julian number held by P3F2 (the Date field). If this confuses you, see *The Date Field as a Special Numerical Field* in my **Fields: Issues** chapter for a discussion of the difference between what D fields store and what they display.

- Create a Transaction Panel index consisting only of the Invoice Number and Reverse Date fields, in that order.
- Assign the above index to the Invoice Panel panel link. That panel link has the Invoice Number as the only field on its field list already.

This panel link will now take the user to the Invoice's most recent Transaction in the Transaction Panel. Lookups on that Date field will now sort with the earliest on the bottom and the latest on the top, but, like the Invoice Number field discussion above, type-to-search won't work unless that panel has a

<Invoice Number, Date>

index for the Smart Lookups algorithm to grab. If such an index exists, and that field doesn't have the

<Invoice Number, Reverse Date>

index assigned to its lookup definition, the lookup will allow type-to-search on that field. But there are certain issues to consider when designing a database that allows for type-to-search lookups on date fields. Read my discussion on Date field lookups in *Date Fields* in my **Fields: Issues** chapter. Also make sure you read *Reverse Sorting by Number* above, so you understand the issues that arise in the subsections called *Via a Panel Link* and *In the Level 1 Panel*. Those issues apply to reverse sorting by date as well.

The .STR File and Index Regeneration

When you create, edit, or delete an index definition, you don't just affect the .IND file. You also dramatically affect the .STR file. So just how are these two files related in terms of indexes?

Like the .IND file, the .STR file also has index pointers. Unlike the .IND file, these pointers don't point to records in a data file. Rather, they point to the pointers in the .IND file. These .STR index pointer live in *index blocks* in the .STR file. Further, these .STR index blocks live in one or two possible physical locations of the .STR file, which I'll call the *Old Index Block Pool* and the *New Index Block Pool*. All newly created index blocks are placed in the New Index Block Pool. After an all-panel Index Regeneration (**Shift-F9, 1, 2**), index blocks found in the New Index Block Pool are swept into the Old Index Block Pool. During that time, any holes created by deleting an index block in the Old Index Block Pool are filled. Graphically, this would look like this:

Old Index Block Pool	1	2	3	4	5	6	7
New Index Block Pool							

Indexes Just Regenerated

Old Index Block Pool	1	2	3	4		6	7
New Index Block Pool	8	9					

2 Indexes Created, 1 Deleted

Old Index Block Pool	1	2	3	4	6	7	8	9
New Index Block Pool								

Indexes Regenerated Again

Think of the above index blocks as physical places in the .STR where the .STR hooks into the .IND file. If the .STR file is a hand and the .IND file a glove, these index blocks are hand fingers that fit and move glove fingers. The gloved hand then manipulates data files.

In the first graphical display, the .STR hooks into the newly generated .IND in a particular way. Index blocks 1 through 7 hook the .STR into indexes 1 through 7 in the .IND file. That .STR hand has fingers in particular locations that fit an .IND glove that has its fingers in compatible locations.

In the second graphical display, the user just created Indexes 8 and 9, and deleted Index 5. DataPerfect creates two new index blocks in the New Index Block Pool, and leaves a hole where Index Block 5 was in the Old Index Block Pool. The .IND file associated with the *second* .STR expects to find Index Block 8 in the New Index Blocks Pool, in that particular physical location. Invoking our hand-and-finger metaphor, the second .STR hand has some of the same fingers as the first one, but is missing one (Index Block 5), and has a couple of new ones where fingers didn't exist in the first one (Index Blocks 8 and 9). It won't work with the .IND the first .STR worked with. That hand won't fit the old glove.

The above considerations become very important when deciding whether or not you need to regenerate indexes upon installing a new version of an .STR at your client's site. I talk about this in *When It's Okay to Overwrite an .STR* in my **Application Maintenance Issues** chapter.

Exception Lists

What They Are

When you call your Define Index menu with **Ctrl-F8**, you see the following options:

```
Define Index
1 - Create Index
2 - Edit Index Field List
3 - Create/Edit Exception List
4 - Delete Index
0 - Exit

Selection: 0
```

Options 1, 2, and 4 are straightforward, but 3 isn't. What exactly is an Exception List?

You can attach an Exception List to any existing index, though you should never attach it to the lowest numbered index for a given panel. I'll explain this caveat later, but for now, let's talk about Exception Lists. When you attach an Exception List to an index, you're restricting what records will be seen by that index (what records will have pointers in that index). Simply put, for any given record, it gets into an index that has an Exception List if *any* of its fields on the Exception List are filled in. Put another way, a record is kicked out of an index if *all* its fields in the Exception List are empty. A numerical field is considered empty if its value is zero, and a text field is considered empty if it's blank.

To attach an Exception List to an index you first enter the Define Index menu with **Ctrl-F8**, then choose **3**. With the cursor keys, find the desired index and select it with **F4**. **Tab** to each field you want to include in the Exception List and select it with **F4**. When done selecting fields for the Exception List, **F7** your way back to the panel display. Though the fields you choose need not be part of that (or any other) index in that panel, each field in the Exception List must be a *real* field (i.e., it can't be a computed field).

After an Exception List is attached to an index, you can always go back at any time to see what fields are on that index's Exception List by browsing the index definition the usual way, with **Ctrl-F8, 2**. On that screen (the Index Selection screen), after you cursor to the index in question, you'll see an uppercase *E* to the immediate left of each field that's in its Exception List.

An example of an index with a single-field Exception List might be one in an Account Panel that has the Balance field as its only Exception List field. That index will include only records with a positive or negative balance, and exclude all records with zero balance. This index will see (have pointers for) only records with a balance, so any entity that uses that index, like a lookup or a report, will likewise see only records with a balance. More on strategically using Exception List indexes later.

An example of an index with an Exception List that has more than one field is an index in the Transaction Panel of my doctor's office application. This Transaction Panel index *includes* (has pointers for) all Transactions that are *payments* (as opposed to charges or adjustments) in virtue of having an Exception List with two hidden G9 fields. One of these hidden G9 fields is formulated to update on any change from 0 to 1 if the Transaction is a payment from an insurance carrier, and the other hidden G9 field is formulated to update on any change from 0 to 1 if the Transaction is a payment that's *not* from an insurance carrier. A Transaction Panel record with one or both of these G9 fields filled in (i.e., with a value of 1 instead of 0) will be considered a payment from *any* source, thanks to this Exception List Index. Likewise, a different Exception List Index—one with only the G9 field that updates on any change from 0 to 1 if the Transaction is a payment from an insurance carrier—will point to all and only Transactions that are payments from an insurance carrier.

A couple of notes about DataPerfect's Help screen on Exception Lists. In the first few releases of DataPerfect 2.3, there's an error in this Help screen. Until recently, **Ctrl-F8, F3** displayed the following:

```
An index exception list determines which
fields are to be checked for an empty
value. If a field in the exception list is
empty, the record will not be placed in the
index.
```

This is false. A record is excluded (has no pointers in the index) here only if *all* the Exception List fields are empty. It's included (has pointers in the index) if *any* are not empty.

That Help screen changed a bit in later releases of DataPerfect 2.3:

An index exception list determines which fields are to be checked for an empty value. If all fields in the exception list are empty, the record will not be placed in the index.

Though that second sentence is true, it doesn't tell you what happens if one *but not all* fields in the Exception List are empty. Here's how I'd rewrite it:

An index exception list determines which fields are to be checked for an empty value. If *and only if* all fields in the exception list are empty, the record will not be placed in the index.

That covers situations where one but not all Exception List fields are empty.

It's sometimes confusing to think about which records are *excluded* by an Exception List, and which are *included*. I think it's a lot easier to think about which records will be *included* in an Exception List Index, instead of thinking about which records will be *excluded*. A record is *included* in an Exception List index if *any* of the Exception List fields are *not* empty. If one or more of those fields are filled in, the record gets in the index.

The Lowest Numbered Index: A Caveat

Earlier, I mentioned you should never attach an Exception List to the lowest numbered index in a panel. The only thing the DataPerfect reference manual says about this is this:

One exception list per index is allowed. At least one index per panel (usually the first index) should not include an exception list.

That caveat is inadequate. The index that should *never* have an Exception List is the *lowest numbered* index in the panel. Why? Because when you enter that panel from the Panel List, you're under the control of that panel's lowest numbered index. All subsequent lookups after that entry will be restricted by that Exception List. This will tend to be very confusing. Leave that lowest numbered index without an Exception List. Also note that I don't say you should protect Index 1, but instead talk about the lowest numbered index. That's because you may end up deleting index 1 at some point, resulting in some other index being the lower numbered index in that panel.

Aiding Lookups with Exception List Indexes

[Refer to UD.STR for this.
Find the *ELI in Lookups* single-panel series.]

One basic use for an Exception List Index is to exclude certain records in a lookup display. In my Phonebook Database, I might want to have a series of G9 fields help

me find certain groups of people. One such G9 field might be titled Consultant field. I could define it so that I would enter 1 if the person is a consultant for my business, and leave it 0 otherwise. If I were to then create an Exception List Index for that panel—one that excludes records with a blank Consultant field—I could then assign this Exception List Index to the lookup for the Consultant field. This lookup now displays all and only consultants.

The above is an example of using an Exception List Index that excludes records from a lookup based on whether or not the value of a displayed flag field is blank. Another use might be to base record exclusion on the given value of a field, not just whether or not it's blank. This is a little more complicated and employs an Exception List that includes a hidden G9 field. Let's explain.

In my Customers Database I might want the Accounts Panel to contain a lookup that displays all and only those accounts with positive balances. This time it won't suffice to have an Exception List Index exclude records with a blank Balance field because that won't exclude *negative* balances.

For this I can use a hidden G9 field that updates on any change to 1 when the Balance field is greater than 0, otherwise it remains 0. In UD.STR's *ELI in Lookups* panel (on your diskette), such a field is the hidden field 5, which has this formula:

```
if P6F4>0 then 1 else 0 endif
```

In that formula, P6F4 is the Balance field. I can now define an Exception List Index that excludes records with field 5 blank. Using this Exception List Index for a lookup will result in the lookup displaying all and only records with positive balances.

To create a lookup that uses this Exception List Index, I created a U1 field that updates on any change to blank, so the user can't save any characters in it. It's just there to land on and do a lookup. I assigned our special Exception List Index to its lookup definition. Now its lookup displays all and only records with a positive balance. To test it, do a lookup on the Last Name for First Name fields to see records with both positive and negative balances. Then do a lookup on the Positive Balance lookup field.

Speeding Reports with Exception List Indexes

Perhaps the way to speed a report is to assign it an Exception List Index when such an index is appropriate. For instance, suppose you want to print Monthly Statements sorted by Account Number, but only want to print those with a positive balance. You can use the Initial Report Definition Screen's Search Conditions (option 4) to do this, with a formula like this on the Specify Formula screen:

```
P1F4>0
```

But what exactly is that going to do? It will cause the report to *examine* every record in its assigned index, but *print* only those with a positive balance. That can take a long time if most of your clients keep their balances clear.

Alternatively, you can create an Exception List Index that sees only those records with a positive balance. An Exception List Index like that one we created in *Aiding Lookups with Exception List Indexes* above will do fine here. If it's just a copy of the Last Name index, but with an Exception List consisting of the Positive Balance Flag field attached to it, it will still sort by Last Name, but point only to records with a positive balance. If, instead of putting Search Conditions on this report, we assign the report this Exception List Index, the records it *examines* will be precisely those it *prints*. That is, it won't even see records with other than positive balances. The speed difference here can be the difference between a report that takes hours versus one that takes seconds or minutes.

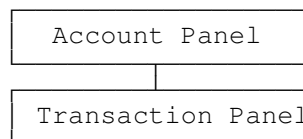
Slightly more complicated is to create a Subreport that uses a panel link governed by an Exception List Index. This will cause the subreport to see only those records that satisfy a particular condition (the condition implied by the Exception List Index's fields not all being blank).

So don't even think about using Search Conditions if you can create an Exception List Index to satisfy the same conditions. Using Search Conditions will still result in the report examining every record in the panel. Using an Exception List Index won't.

Aiding Computed Fields in a Parent Panel with Exception List Indexes

[Refer to UD.STR for this.
Find the *ELIs Aiding Parent ::C Fields* series.
Load the Account Panel.]

Let's consider this simple panel hierarchy:



Most Recent Transaction

Suppose you'd like a computed field in the Account Panel to display the most recent Transaction for the currently displayed Account. The typical way to do this in the Account Panel is to could create a hidden panel link that leads to the Transaction Panel, making sure the panel link is governed by a Transaction Panel index that sorts records in reverse chronological order by Account Number. Then formulate the computed fields to grab values found in the Date and Amount fields of the Transaction Panel record using the new panel link. Let's explain.

First, we create a Transaction Panel index that sorts Transaction Panel records in reverse chronological order by Account Number. In the Transaction Panel in UD.STR's *ELIs Aiding Parent ::C Fields* panel series, that's Index 3. It turns on the P8F10 in that panel (the Reverse Date field). Records sorted in the Transaction Panel by that index are sorted primarily by Account Number (the first field in its

index field list), and secondarily by Reverse Date (the second field in its index field list). The Reverse Date is just the negative of the value found in the Date field in that panel. A Reverse Date field is typically formatted

G-99999::H

and formulated to update on any change to

-P1F1

where P1F1 is the Date field from which it's grabbing its date. Though you can also format the Reverse Date field

D-99/99/9999::H

and use the same field formula,

G-99999::H

takes less screen space. See *Reverse Sorting by Date* above for more on Reverse Dates.

Second, we create a hidden Account Panel panel link that targets the Transaction Panel, uses this our new index, and has only the Account Number field on its link field list.

Third, we formulate a couple of Account Panel computed fields that use this panel link to grab values found in the Date and Amount fields of the most recent Transaction Panel record for the currently displayed Account Panel record. In UD.STR's *ELIs Aiding Parent ::C Fields* panel series, these two Account Panel computed fields are fields 8 and 9 (H\$ZZZ9.99::C and D99/99/99::C, respectively), and the panel link they're using is field 10.

Most Recent Payment

But suppose you would like to have a pair of computed fields in the Accounts Panel display the most recent *payment* on that account, and, like UD.STR's *ELIs Aiding Parent ::C Fields* panel series, you don't have a separate panel for payments (your Transactions Panel holds both debits and credits). To do this, we create a Transaction Panel index that sorts in reverse chronological order by Account Number and assign it an Exception List that contains a G9 field that updates on any change from 0 to 1 if the Transaction is a payment. Then we formulate the computed fields to grab values found in the Date and Amount fields of the Transaction Panel record using the new panel link. Again, let's explain.

First, we create a Transaction Panel G9::H field that updates from 0 to 1 if its Transaction Panel record is a payment. In my office's billing application, I can easily create G9::H fields for Exception List Indexes like this because its Transaction Panel has a U3 field called the Quick-Entry Code field. All Transactions Panel records that are payments have a Q/E Code beginning with *P* (e.g., *PKO* stands for Payment by Check in Office, *PKM* for Payment by Check in Mail, *PCO* for Payment

by Credit Card in Office, etc.). This facilitates formulating a G9::H field that looks for payments. In UD.STR's *ELIs Aiding Parent ::C Fields* panel series Transaction Panel, field 1 is the Q/E Code field, and field 8 is the G9::H field that updates on any change to 1 if the Q/E Code field value begins with *P*. The formula for the G9::H field is

```
if P8F1="P" then 1 else 0 endif
```

The above updates to 1 when the first character of the Q/E Code is *P*, otherwise it updates to 0. See *String Identity* in the my **Formulas** chapter if you don't understand why that formula updates to 1 if the first character of the value in P1F1 is *P*.

Second, we need to create a Transaction Panel index like we did in the previous example (consisting of Account Number and Reverse Date fields, in that order), but assign it an Exception List. That Exception List will consist solely of our new G9::H field that updates from 0 to 1 when the Transaction is a payment (i.e., the Q/E Code begins with *P*).

Third, we create a hidden Account Panel panel link that targets the Transaction Panel, uses our new Exception List Index, and has only the Account Number field on its link field list.

Fourth, we formulate Account Panel computed fields that use this panel link to grab values found in the Date and Amount fields of the most recent Transaction Panel record for the currently displayed Account Panel record. In UD.STR's *ELIs Aiding Parent ::C Fields* panel series, these two Account Panel computed fields are fields 3 and 4 (again, H\$ZZZ9.99::C and D99/99/99::C, respectively), and the panel link they're using is field 7.

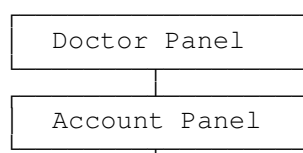
In my billing application I have a series of such computed fields in the Account Panel, each using a different hidden panel link, showing such things as the following:

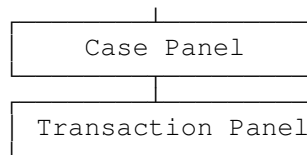
- Most recent insurance payment (Q/E Code is *PIN*)
- Most recent private payment (Q/E Code starts with *P* but isn't *PIN*)
- Most recent office visit (Q/E Code starts with *V*)
- Most recent transaction (any Q/E Code)

For each of the above I have a special G9::H field in the Transaction Panel, coupled with a corresponding Exception List Index to be used by a hidden Account Panel panel link.

Dividing Data File Record Access with Exception List Indexes

Consider this panel hierarchy:





Male vs. Female Account

Suppose you'd like to set up a menu so one menu item takes the user to all females in the Account Panel, and another menu item takes the user to all males in the same panel. Or perhaps you want a panel link in the Account Panel to take the user to all Active Cases in the Case Panel for the currently displayed Account in the Account Panel, and another panel link to take him to all and only Inactive Cases for this Account.

To do this, we use Exception List Indexes that divide a data file's records in two, allowing one menu option (or panel link) to give access to one portion, and another menu option (or panel link) to give access to the other portion. Each member of this pair of menu items (or panel links) targets the same panel.

Think about this a little. In the first case (female versus male Accounts in the Accounts Panel), all we need in the Accounts Panel is an Exception List Index that sees only females, and one that sees only males. To do this, we could create a G9::H field that updates on any change from 0 to 1 if the Sex field (formatted U1) in the Account Panel has *F* in it. We then create another G9::H field that updates on any change from 0 to 1 if the Sex field has *M* in it. An Exception List Index that has one or the other of these new G9::H fields on its field list, sees only Accounts of a single sex. Assigning such an Exception List Index to a menu option limits that menu option's access to a single sex.

Active vs. Inactive Case

In the second case (active versus inactive Cases in the Case Panel), you could have a single displayed and editable Case Status field in the Case Panel, formatted G9. That field will have a formula that updates on creation to 1. The user is instructed to leave it as 1 until the Case is to be considered inactive, at which time they change its value to 0. If those are the only two options you want for that field, you'd put a Range of *0 to 1* on it. Next create a G9::H field that updates on any change from 0 to 1 if the Case Status field is 0.

Now we can create two Exception List Indexes. One that sees only active Case records, and one that sees only inactive ones. The *active* Exception List Index has the displayed G9 field on its Exception List. The *inactive* Exception List Index has the hidden G9 field on its Exception List. Assigning the *active* Exception List Index to a panel link in the Account Panel gives access to only active Cases in the Case Panel. Likewise, the *inactive* Exception List Index gives an Account Panel panel link access to only inactive Cases in the Case Panel.

Warning: Exception List Index Bug in Version 2.2

DataPerfect 2.2 has a fairly dangerous bug that involves Exception List Indexes. If the user deletes a record that is in an Exception List Index, while operating within an environment governed by that Exception List Index, DataPerfect 2.2 may corrupt the record counter for that panel, causing indexing problems. Version 2.3 corrects this problem.

By operating within the environment governed by an Exception List Index, I mean placing yourself within the following situations:

- You're in a lookup display governed by an Exception List Index.
- You entered the currently displayed panel via a panel link or a menu option governed by an Exception List Index.
- You're running a report governed by an Exception List Index.

In any of these three environments, deleting records may result in serious problems (in the third environment I'm speaking of having the report itself doing the deleting). After such deletions, you'll frequently notice the record counter in the upper left corner of the panel display is artificially high.

I've also found that assigning an Exception List to the last index of a panel can result in another serious problem. In such a case, deleting records with **Shift-F5** may cause the same corruption mentioned above without even operating within an environment governed by an Exception List Index.

To work around the problems outlined above, I suggest you take the following precautions:

- Don't use Exception List Indexes for lookups unless you have DPMouse© installed with the Delete Protect flag active.
- Hide all panel links governed by Exception List Indexes. This will keep users from penetrating these links and then deleting subrecords within an environment governed by an Exception List Index.
- Don't attach an Exception List to the last index of a panel.

The above precautions apply to DataPerfect 2.2, and are unnecessary with version 2.3.

This chapter targets both beginners and the experienced.

Flat-File DBMS vs. Relational DBMS

Database management systems (DBMSs) are divided into two very different groups: flat-file DBMSs and relational DBMSs. DataPerfect is a relational DBMS. Let's outline the differences between a flat-file DBMS and a relational DBMS.

A flat-file DBMS allows you to create applications that manage data in only one file; whereas, a relational DBMS allows you to create applications that manage data interrelated across many files. Flat-file DBMSs are great for things like phonebooks, where you're only interested in data conveniently housed by a single file of records. Here, each record contains all the information you need for a phonebook entry.

But how would you run your auto parts store with a flat-file DBMS? Assuming that, as well as parts descriptions, you would like to record customer transactions with this application, how could you do this with only one data file? This would be cumbersome in a single-file system because each time you enter a transaction for a customer you would have to type in the customer's name and part description by hand. It would be a lot easier if you could do the following:

- For each customer, enter customer information *once* in a Customer file.
- For each part, enter part information *once* in a Part file.
- For each transaction, pick the customer from a pick list of records in the Customer file, and pick the part from a pick list of records in the Part file, to create a unique record in the Transaction file.

In the above scheme, the user types customer and part information only once for each customer and part. With a relational DBMS program like DataPerfect, you can create an application that interrelates these three files of records each time the user enters a transaction. This then allows users to create many records in the Transaction file for the same customer, or many records in the Transaction file with the same part, without having to type customer or part information repeatedly.

With DataPerfect, we assign each data file a single panel, which serves as both a data entry form and a view of the data. We might design an application with three panels:

- Part Panel
- Customer Panel
- Transaction Panel

Now we need links that allow the application to interrelate these three data files.

The Four Linking Relationships

In any DBMS program, a link expresses one of the following relationships between the records of the two files it links:

- One-to-many
- Many-to-one
- One-to-one
- Many-to-many

Let's use examples related to our Auto Parts application to clarify these relationships.

One-to-Many Linking

The most common linking relationship is the *one-to-many* relationship. Consider our Customer Panel and Transaction Panel. The Customer-Transaction relationship is typically a one-to-many relationship, in that each Customer can have many Transactions. Or, to put it another way, each record in the Customer Panel can have many related records in the Transaction Panel. This is sometimes expressed as a *parent* record in the Customer Panel having many *children* records in the Transaction Panel.

Many-to-One Linking

The second most common linking relationship is the converse of the one-to-many relationship: the *many-to-one* relationship. Such a relationship is the Transaction-Customer relationship. This is pretty obvious if you understood my explanation of the one-to-many relationship, so let's move on to the next relationship.

One-to-One Linking

Probably the next most common linking relationship is the *one-to-one* relationship. In DataPerfect applications, this type of linking usually shows up when the developer wants to add an extension to the panel seen on the screen. We might consider this for our Auto Parts application if we decide we want only basic information in the Part Panel, and very technical information about each part in an extension of the Part Panel. This extension might be a Part Specifications Panel. There would be one record in the Part Specifications Panel for each record in the Part Panel, and vice versa. For now, though, we'll limit our discussion to the three panels outlined above.

Many-to-Many Linking

Probably the most complicated linking relationship is the *many-to-many* relationship. Instead of expressing this relationship with a link, DataPerfect applications express it with a panel that joins data from two other panels. Consider the relationship between the Part Panel and the Customer Panel. From the vantage point of the Part Panel, one Part can show up in many Customers' shopping bags (one-to-many), and many Parts can show up in a single Customer's shopping bag (many-to-one). From the vantage point of the Customer Panel, one Customer can buy many Parts (one-to-many) and many Customers can buy the same Part (many-to-one). The Part-Customer relationship, then, is many-to-many. In its own way, the Transaction Panel expresses this many-to-many relationship by joining a Customer and a Part in each record.

The Two Types of DataPerfect Links

With DataPerfect, we use panel links and data links to express these linking relationships. Defining *either* type of link simply amounts to giving it four characteristics:

- Target panel
- Target panel field to land on
- Target panel index
- Source panel field list

Think of links, be they panel links or data links, as simply bridges between a *source panel field list* and a *target panel index*. Each takes the user from the displayed record in the source panel to related records in the target panel, filtering the target panel's records with the link field list (which consists of source panel fields), and sorting them with the link index (which is a target panel index).

The differences between a panel link and a data link arise from the fact that a data link is attached to a field and a panel link stands alone. Though each takes the user from the displayed record in the source panel to related records in the target panel, the data link, in virtue of being tied to a field, provides for data checking during data entry. This difference between the two links is sometimes expressed as *the panel link links panels, whereas the data link links a field with a panel*. Don't think of the difference only in this way, however. The difference between the two types of links is much more than that, and you need to understand their differences if you want to prevent problems with your database applications.

In deciding between creating a panel link or a data link, always choose a panel link unless you need data checking on a particular field. That is, never create a data link for navigation between panels. Never. All these should be panel links:

- A link used only to take the user to a subpanel.
- A link used only to take the user to a parent panel.
- A link used only for the purpose of grabbing a field in another panel during a formula computation.
- A link used only to facilitate a Keep A Total operation to a different panel.
- A link used only to make a subreport possible in particular reports.

If none of the above links need provide the user with a pick list during data entry, they should all be *panel* links, not data links.

But, again, on the definitional level, these two links are essentially the same: both simply link a source panel's field list with a target panel's index. I can't stress this point enough. When I first started defining panel links, I thought I had to guarantee the *target* panel index was an ordered subset of some *source* panel index. That is, consider the typical Customer application that has a Customer Panel, Invoice Panel, and a Transaction Panel. Suppose I wanted to create a panel link in the Customer Panel that takes the user to the Invoice Panel. DataPerfect is going to ask me for an index in the Invoice Panel and a field list in the Customer Panel. I won't be asked for the existence of any particular index in the Customer Panel. I can tie this panel link to the following two entities:

- An Invoice Panel index consisting of Customer ID and Invoice Number, in that order.
- A Customer Panel field list consisting only of the Customer ID field.

Though I probably would have defined a Customer Panel index with Customer ID as its first field, the panel link defined above doesn't require such an index exist. The set-subset relationship that must exist for a panel link isn't between an index of the source panel and an index of the target panel; rather, it's a relationship between an ordered set of fields in the source panel (the link's field list) and an index in the target panel. There need not be any index in the *source* panel that works with the field list for the link in question.

That the target panel index need not be a subset of a source panel index can be useful. Consider a powerful enhancement made in the first interim release of DataPerfect 2.3 (September 1993):

```
DataPerfect now allows Computed fields
(::C) in Link Field Lists and in Index
Exception Lists.
```

[README (09/01/93)]

This means we can now place fields that aren't even *real* fields in link field lists, and therefore aren't even *candidates* for indexes (computed fields don't exist in the data file—that's why, we can't put them in indexes). That said, consider the following, which revolves around the User ID facility introduced with that same release of DataPerfect:

- Hide a computed field that updates to the User ID of the current user.
- Create a panel link that has this hidden computed field in its link field list, and uses a target panel index that includes a corresponding *real* User ID field in the target panel.

The above panel link only allows access to records created with the current User ID. This might be attractive in certain high security areas of a database. Or you could use this panel link as access to a subpanel that serves as a notepad for users, letting them create and access only their own notes.

Creating and Defining a Panel Link

While in Define Panel mode, if you hit **F5** *with your cursor not on a field*, DataPerfect places a single-character block in the panel if there's room. This will be a panel link after you define it further. To define that panel link, move the cursor over it and hit **F5** again. DataPerfect now throws you into the Panel List, asking you for this link's target panel. After you select the target panel by hitting **Enter**, DataPerfect asks you for the link index (again, this is a *target* panel index). After cursoring to the index you want, select it with **F4**. Next you build the link field list (consisting, again, of *source* panel fields). The help screen on this is very clear, but it doesn't really tell you the logic behind picking a link field list.

The link field list serves as a filter that determines what records the user *accesses* when penetrating that link. At least that's one way of thinking about it. But the link field list also can be seen as a list of fields that, when combined with the link index, determine what records will be *linked* to the current record in the link's panel. In order for the link field to work properly with the link index, however, it must be a set of fields in the source panel that mirror, in the same order, fields in the target panel index chosen as the link index. And the index fields in the link field list must mirror the target panel's fields starting with the index's first field.

Let's clarify this with an example. You want a panel link in the Invoice Panel that allows the user to access Transaction Panel records related to the current Invoice Panel record. That is, you want the user to be able to easily move from an Invoice to the Transactions that are part of that and only that Invoice.

A typical way to define this link is to choose the Transaction Panel as its target panel, and a Transaction Panel index that looks like this:

```
Invoice Number, Transaction Number
```

Given the index chosen for this link, we would assign only one field to the link's field list:

```
Invoice Number
```

What would a link like this do for the user who's sitting in the Invoice Panel? Just this. When the user penetrates that link he accesses Transaction Panel records

that have the Invoice Number found in the record he just left in the Invoice Panel. Put another way, that link *links* any given Invoice to all and only *its* Transactions. The link field list, in concert with the link index, links each record in the Invoice Panel with records in the Transaction Panel *that have the same value in the Invoice Number field*. Put more generically, the link links each Invoice Panel record with Transaction Panel records that have the same value *in the fields comprising the link field list*. This link works fine if its field list mirrors the fields in the link's index, starting with the first field of the link index

Let's clarify what I mean when I say the link field list must *mirror* the fields in the link index. First, don't forget that the link *index* is composed of fields in the *target* panel, but the link *field list* is composed of fields in the *source* panel. So they're never the *same* fields. Second, when you examine each field in the link field list, and do so in order, you must find that each such field is of a format compatible with the corresponding field in the link index field list (here you compare the first field in the link field list with the first field in the link index, the second field in the link field list with the second field in the link index, etc.).

By *compatible format*, I mean that the two fields being compared must both be the same length and of the same type. They're the same type if both are G fields, or both are N fields, or both are A fields, etc. One can be, say, a GZZ9 field and the other a G999 field (same length and same type, even though one has Zs and the other doesn't). Or one can be a G999 field and the other an H999 field, because an H999 field is really just a G999 field that displays nothing when 0 (both are simply right-aligned numerical field formats). But you can't have one field an N999 field and the other a G999 field. Those two aren't the same type—the first is left-aligned and the second right-aligned. And you can't have one field a G999 field and the other a G99 field because they're not the same length.

That's what it takes to define a panel link. A panel link is essentially a screen entity that links one panel with another, and does so via a target panel index and a source panel field list. A typical panel link takes you from an existing record to dependent records in another panel. Some developers talk about such a link as taking you to the current record's *subrecords*; others talk about it taking you to the current record's *attached* records; and the DataPerfect manual talks about it taking you to *dependent* records. This link is typically used, as you have probably surmised, to express a *one-to-many* relationship.

The Link Index and Link Field List

Let's return to our application that has an Invoice Panel and a Transaction Panel. To link these two panels in a meaningful way with a panel link in the Invoice Panel, we need an appropriate Transaction Panel index (the target panel index). If we want the panel link to provide access only to Transactions of the Invoice the user just left in the Invoice Panel, we don't want a Transaction Panel index that starts with the Transaction Number field. Rather, we need a Transaction Panel index that starts with the Invoice Number field. This way, if we assign a link field list consisting of just the Invoice Number field in the Invoice Panel, the link field list filters the user's access

to Transaction Panel records to just those with the Invoice Number of the Invoice Panel record he or she just left.

In general, database programming would call the Invoice Number field in this case the key field on which this link links these two panels. The value found in the Invoice Number field will be shared by all records found in the Transaction Panel when penetrating this panel link.

The Link Field List's Role in Record Attachment

In deciding on a field list for this link, we need to think about what records we want attached to the current record during Browse mode. This applies equally to both the panel link and the data link. During Browse mode you want the user to use the Invoice Panel panel link to access records in the Transaction Panel that have the same Invoice Number. To determine what field list to use here, think about what fields in the attached records (Transactions) must match the current record (Invoice). In this case, that's just the Invoice Number field.

If this panel link has a field list of just Invoice Number, and an index that starts with Invoice Number, then penetrating that link during Browse mode puts the user in the Transaction Panel with access to all and only Transactions with the Invoice Number of the currently displayed record in the Invoice Panel.

Suppose we may want a *data* link on the Item field in the Transaction Panel. This would allow the user to pick an Item from an Item Panel pick list during Create or Edit mode in the Transaction Panel. Again, in forming this link's field list, consider what records you want it to attach to the current record during *Browse* mode. In this case, you want only *one* record attached to the current record. That is, the record in the Item Panel that's related to the current record in the Transaction Panel is simply the record the user chose from the pick list during Create or Edit mode. So you need a field list that limits Browse mode access to that single record in the Item Panel. In this case, such a field list consists of Item Name or Item ID Number, or both.

The Link Field List's Role in Field Filling

But the field list does more than just determine what records are attached to the current record during Browse mode. The link field list also determines what fields will be filled in during Create or Edit mode. In the case of the panel link, the link field list causes its corresponding fields in the target panel to be filled in with the values in the current source panel record. Whereas, in the case of a *data* link, the link field list causes its corresponding fields in the *source* panel to be filled in with the values found in the *target* panel record the user chooses from the pick list.

In the case of the Transaction Panel data link (on the Item field), if the Transaction Panel has both an Item Name field and an Item Number field, then you can have both fields filled in when the user chooses a record from the Create or Edit mode pick list. Just put both Item Name and Item Number on the link field list, and choose the

<Item Name, Item Number>

index; or put both Item Number and Item Name on the link field list, and choose the

<Item Number, Item Name>

index.

In Summary

In summary, the link field list does two things. *First*, it acts as a filter in determining what records will end up attached to the current record during Browse mode. *Second*, it determines what fields will be automatically filled in during Create or Edit mode.

It's precisely the second point above where panel links and data links diverge. The field list of a *panel* link takes values from the *source* panel's record and puts them in the corresponding fields in the *target* panel's record; whereas, the field list of a *data* link takes values from the record the user selects from the *target* panel (via a pick list lookup) and puts them in the corresponding fields in the *source* panel's record (the current record).

That's all you need to know to construct a link's field list. Just make sure the index you choose begins with fields that mirror the field list, in the same order. Follow these rules, and all your links will be well-formed.

A Note about a Panel Link's Index

Though the entity that generally determines what records will be available to the user after penetrating a panel link is its field list, be aware that not all indexes give full access to records. If a panel link's index has an Exception List attached to it, it may very well not give access to all the records it would have if it had no Exception List attached to it. Attaching an Exception List index to a panel link is a particularly powerful thing to do, so make sure you read *Exception Lists* in my **Indexes** chapter if you're not sure of them.

The Action of a Panel Link

Back to the Invoice-Transaction application. If you assign this Invoice Panel panel link a Transaction Panel index that starts with the Invoice Number field, and assign it a field list consisting of simply the Invoice Panel's Invoice Number field, the following will happen:

- If the cursor is on the panel link in the Invoice Panel in Browse mode, and there are Transaction Panel records that correspond with the current Invoice Panel record (children records of the current Invoice Panel record), hitting **Down Arrow** will put the user in the Transaction Panel with access only to records with the same Invoice Number he just left in the Invoice Panel.

- If the cursor is on the panel link in the Invoice Panel in Browse mode, and there are *no* records in the Transaction Panel that correspond to the current Invoice Panel record, hitting **Down Arrow** will inform the user that no related subrecords exist in the Target Panel with this message:

No records are found in this subset. If you want to add records, press Create Record in Linked Panel (F5). Otherwise, you will remain in this panel.

Hitting **F5** will put the user in Create mode in the Transaction Panel, with the Invoice Number field automatically filled in with the Invoice Number of the parent Invoice Panel record. That is, DataPerfect starts the creation of a new subrecord, filling in fields in the Transaction Panel with values found in the field list fields of the Invoice Panel panel link. In this case, the field list only has one field.

- If the cursor is on the panel link in the Invoice Panel in Browse mode, whether or not there are records in the Transaction Panel that correspond to the current Invoice Panel number, hitting **F5** will put the user in Create mode in the Transaction Panel, with the Invoice Number field automatically filled in with the Invoice Number of the parent Invoice Panel record.

- If the cursor is on the panel link in the Invoice Panel in *Create* or *Edit* mode, all the above apply, but DataPerfect will save the Invoice Panel record before moving to the Transaction Panel.

- If the cursor is on the panel link in the Invoice Panel in Browse mode, hitting **Up Arrow** does just what this does when on any other field in the panel: it produces a Lookup of records in the Invoice Panel.

- If the cursor is on the panel link in the Invoice Panel in Browse mode, hitting **F8** does something different than it does when on a data field. When on a data field during Browse mode, hitting **F8** produces a Lookup of Invoice Panel records. When on a panel link, hitting **F8** produces a Lookup of *Transaction Panel* records with the current Invoice Number. So hitting **F8** while on a data field in Browse mode produces a Lookup that displays records of the current panel; whereas, hitting **F8** while on a panel link in Browse mode produces a Lookup that displays dependent (related by the link's field list) records in the linked panel.

The Data Link

Of course, DataPerfect also offers another type of link: the *data link*. Whereas a *panel link* links a panel to another panel, a *data link* links a *field* to another panel. Whereas a panel link is used to take the user from a record to its subrecords (e.g., from the current record in the Invoice Panel to its dependent records in the Transactions Panel), a data link is used to provide the user a *pick list* during data entry in the current panel, where he's allowed to pick a value that exists in another panel's record.

Let's explain. In our Invoice-Transaction application, we would probably also have a third panel: the Items Panel. The Items Panel contains all the sellable Items, representing each as a single record. In the Transaction Panel, we probably would want to put a data link on Item field. If it's properly linked to the Items Panel then, during Create or Edit mode, when the user performs a Lookup by hitting **Up Arrow** or **F8** on the Item field in the Transaction Panel, he'll see a lookup displaying records in the *Items* Panel (the *linked* panel), not the *Transaction* Panel (the *current* panel). DataPerfect is providing the user with a list of possible values from which to choose for the current field (the Item field in the Transaction Panel).

So, whereas a panel link is attached to a *panel*, providing the user a means of traveling from a record to its related records in another panel, a data link is attached to a *field*, providing the user a pick list for data entry into that field during Create or Edit mode.

Defining a Data Link

To define a data link, you need to be in Define Panel mode. Cursor to the field on which you want to attach the data link and hit **F5** (the cursor must be *on* the field before hitting **F5**, otherwise DataPerfect thinks you're creating a *panel* link). Now DataPerfect asks you for the same information about this link that it asks when defining a panel link:

- Target panel
- Target panel field to land on
- Target panel index
- Source panel field list

That's it. You now have a data link on that field.

Controlling the Data Link's

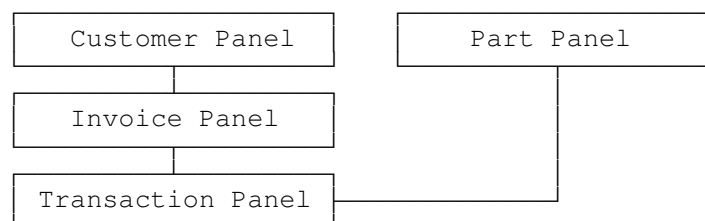
Create/Edit Mode Lookup Display

When the user is in Create or Edit mode, sitting on a data-linked field, hitting **Up Arrow** or **F8** produces a lookup display of record in the linked panel. To make sure that lookup display works properly, please focus on this rule:

A data link's index and field list determine its Browse mode linkage; whereas, a data link's target field's lookup definition determines its Create or Edit mode lookup display.

I have fixed many developer's applications by simply applying that rule here and there, in panels they said weren't working properly.

Let's explain. When defining a data link, picking an index and field list will determine what will be linked to the current record in the source panel during Browse mode. Say you have an Auto Parts application, something like this:



Here the user picks a Customer in the Customer Panel, penetrates the panel link to the Invoice Panel, picks an Invoice, and then penetrates the panel link to the Transaction Panel. There, they hit **F9** to go into Create mode, if they aren't in that mode already, and **Tab** to the Part Number field. The Part Number field is data linked, so when they hit **Up Arrow** or **F8**, they get a display of Parts in the Part Panel. They hit **Enter** on their choice and then, if necessary, continue filling in other fields in the Transaction Panel record.

In defining the data link on the Part Number field in the Transaction Panel, the index and field list you assign it determine what records in the Part Panel will be attached or linked to the current record in the Transaction Panel during Browse mode. Another way of saying this is that the data link's index and field list will determine what records it allows you to access in the Part Panel when in Browse mode in the Transaction Panel. The data link's index and field list play almost no part in how the link Create or Edit mode lookup display behaves.

So when deciding on a data link's index and field list, think only about what you want it to attach to the current record during Browse mode. In this case, we want the data link to attach exactly one record in the Part Panel to the current Transaction Panel record. That is, we want this data link to allow Browse mode access from the current Transaction Panel record the single Part Panel record this Transaction Panel record is about. So the Part Panel (target panel) index we would choose for this data link would begin with the Part Number field, and the Transaction Panel (source panel) field list would consist of only the Part Number field. Assuming no two records in the Part Panel have the same Part Number, this will effectively link a single Part Panel record to each Transaction Panel record.

But what do we do to control the Create or Edit mode lookup display behavior of this data link? Suppose we want the Create or Edit mode lookup on that data link to display Parts in the Part Panel alphabetically by Part Name, not numerically by Part Number. Does this mean we must change the index we chose for our data link? Don't forget that we chose an index that sorts by Part Number, not Part Name. The answer is no. What controls the way target panel records display during a lookup on a source panel data link during Create or Edit mode is the *lookup definition on the target field*. It has essentially nothing to do with the index and field list chosen for that data link.

So, even though we assigned this data link a Part Panel index that sorts by Part Number, and a field list consisting solely of Part Number, we can still get its Create or Edit mode lookups to sort Part Panel records alphabetically by Part Name. To do this, we first make sure the target field of this data link is the Part Name field in the Part Panel, not the Part Number field in the Part Panel. Then we exit the Transaction Panel and load the Part Panel and take a look at the lookup definition on the Part Name field. That lookup definition will determine how Create or Edit mode lookups will sort when hitting **Up Arrow** or **F8** on the data linked Part Number field in the Transaction Panel. So we would now choose for that lookup definition, an index that sorts by Part Name and a field list beginning with Part Name. Now Create or Edit mode lookups on the Part Number field in the Transaction Panel will sort by Part Name, not Part Number, even though the data link on that field is defined with an index that sorts by Part Number.

So, please, paste this to your computer terminal!:

A data link's index and field list determine its Browse mode linkage; whereas, a data link's target field's lookup definition determines its Create or Edit mode lookup display.

The Link Options Menus

There's actually a little more to discuss regarding defining a link, whether it be a panel link or a data link. When you're done creating the link (giving it its target panel, field to land on, index, and field list), you're then presented with one of these two menus just before returning to Browse mode with **F7** or **F10**:

```
—Define Panel Link—
Link to Panel:3 Field1 Index1 Field List to Build Key:1
 1 - Edit Target Field/Target Index/Field List
 2 - Define Related Records Window      5 - Display/Hide Link
 3 - Create/Edit Window Field List      6 - Define Lookup List
 4 - Delete Window                      7 - Cascade Off
Window Off
—Selection: 0—

—Define Data Link for Field—
Link to Panel:2 Field1 Index1 Field List to Build Key:1
 1 - Edit Target Field/Target Index/Field List
 2 - Remove Data Link                  5 - Prompt-Create
 3 - Auto-Create                      6 - Check During Data Entry Off
 4 - No-Create                        7 - Cascade Off
Prompt for creating related record if not found.
—Selection: 0—
```

The first menu, of course, is for the panel link, and the second, the data link. At this point, you don't have to do anything at all with such a menu. Just hit **F7** or **F10** to return to Browse mode, accepting the defaults above.

You can always return to the *Panel Link* Options menu later, in either Browse mode or in Define Panel mode. In Browse mode, you can access the Panel Link Options menu with **Shift-F8**; whereas in Define Panel mode you can access it with either **F5** or **Shift-F8**. On the other hand, you can return to the *Data Link* Options menu only in Define Panel, and only with **F5**.

Let's discuss the options on each menu.

Panel Link Options

```
—Define Panel Link—
Link to Panel:3 Field1 Index1 Field List to Build Key:1
 1 - Edit Target Field/Target Index/Field List
 2 - Define Related Records Window      5 - Display/Hide Link
 3 - Create/Edit Window Field List      6 - Define Lookup List
 4 - Delete Window                      7 - Cascade Off
Window Off
—Selection: 0—
```

Option 1 (Edit Target Field/Target Index/Field List)

This lets you change any or all of the four essentials of the link:

- Target panel
- Target panel field to land on
- Target panel index
- Source panel field list

Option 2 (Define Related Records Window)

A panel link, unlike a data link, may have a Window. This is a view, in the current panel, of records in the link's target panel. After hitting **2**, DataPerfect allows you to place and size the Window, and then asks you what fields you want to display in it. You're also given the option of framing the Window with a single-line border, as well as the option of having the Window display its records from the bottom (end) of the index instead of the top (beginning). For an example of a panel link Window, load UD.STR. Find the *Default Schemes*, *Hidden Panel*, & *Undelete* series and load the Client Panel.

Option 3 (Create/Edit Window Field List)

Option 4 (Delete Window)

These options let you later return to modify the properties of a panel link Window, or delete it altogether.

Option 5 (Display/Hide Link)

Hitting **5** toggles between displaying the panel link (the default), or hiding it. You should consider hiding it if it's on there for things like field formula or report access to other panels. If you don't intend for the user to ever penetrate that link in Browse mode, hide it.

When you hide a link with option 5, the menu reflects that change to the right of option 5:

5 - Display/Hide Link	Hidden
6 - Define Lookup List	
7 - Cascade Off	

Option 6 (Define Lookup List)

This lets you assign a lookup definition to a panel link, just as you do with any other field.

Option 7 (Cascade Off, Cascade Update, Cascade Update/Delete)

This toggles between these three states:

- Cascade Off
- Cascade Update On
- Cascade Update/Delete On

What changes is the way option 7 reads on the screen with each press of 7:

```
5 - Prompt-Create
6 - Check During Data Entry Off
7 - Cascade Off
```

```
5 - Prompt-Create
6 - Check During Data Entry Off
7 - Cascade Update On
```

```
5 - Prompt-Create
6 - Check During Data Entry Off
7 - Cascade Update/Delete On
```

Cascade Update On will cascade changes made to values found in the current record to subrecords through that link. For fields to cascade, they must be on the link's field list. So, suppose the Last Name field is on this link's field list, and, in Edit mode, you change the Last Name field value from *Smith* to *Adams*. If that link is setup with Cascade Update On, DataPerfect will change the Last Name field from *Smith* to *Adams* in all records related to the current record via that link. This preserves the linkage that would be lost if Cascade was off. Cascade Update has a downside that must be taken seriously on large databases, however. See *Keeping Subpanel Data Current* in my **Fields: Introduction** chapter for a discussion of this.

Cascade Update/Delete On does everything Cascade Update On does, but also deletes all records related to the current record via that link when the current record is deleted. This prevents orphaned subrecords. This is a fairly dangerous option and should be used with caution. You can allow the user to accidentally delete thousands of records with a single keystroke this way.

Data Link Options

```
—Define Data Link for Field—
Link to Panel:2 Field1 Index1 Field List to Build Key:1
1 - Edit Target Field/Target Index/Field List
2 - Remove Data Link                    5 - Prompt-Create
3 - Auto-Create                        6 - Check During Data Entry Off
4 - No-Create                          7 - Cascade Off
Prompt for creating related record if not found.
—Selection: 0—
```

Option 1 (Edit Target Field/Target Index/Field List)

This is just like it is under Panel Link Options above.

Option 2 (Remove Data Link)

Unlike the panel link, you can remove a data link after you create it. Option 2 lets you do this.

Option 3 (Auto-Create)

Option 4 (No Create, No Access)

Option 5 (Prompt-Create)

Option 6 (Check During Data Entry Off)

You can leave that menu just as it is, in its default state you see above, with

```
Prompt for creating related record if not found.
```

sitting there, just under option 4. If so, then, when the user is in Create or Edit mode and enters data in that field that isn't found in the linked panel, DataPerfect will prompt him with this screen:

Value Not Found	
The field value entered is not found in the other panel.	
Do you want to	
1 - Create a New Record in the Other Panel	
2 - Lookup a Record in the Other Panel	
0 - Reenter the Value for this Field	

Alternatively, you can change this behavior with other options that follow.

Option 3 (Auto-Create). Hitting **3** changes the menu to this:

Define Data Link for Field	
Link to Panel:4 Field1 Index1 Field List to Build Key:1	
1 - Edit Target Field/Target Index/Field List	
2 - Remove Data Link	5 - Prompt-Create
3 - Auto-Create	6 - Check During Data Entry Off
4 - No-Create	7 - Cascade Off
Automatically create related record if record not found.	
Selection: 0	

This option will probably confuse the user. If the user places a value in the current data-linked field that doesn't exist in the linked record, and this option is active, the user will suddenly find himself in the Create mode in the *linked* panel. There they must save the record with **F10** or **F7**, which throws them back into Create mode in the current panel.

Option 4 (No Create, No Access). Hitting **4** changes the menu to this:

Define Data Link for Field	
Link to Panel:2 Field1 Index1 Field List to Build Key:1	
1 - Edit Target Field/Target Index/Field List	
2 - Remove Data Link	5 - Prompt-Create
3 - Auto-Create	6 - Check During Data Entry Off
4 - No Access	7 - Cascade Off
Do not allow user to create related record if not found.	
Selection: 0	

Note that

```
Prompt for creating related record if not found.
```

has changed to

```
Do not allow user to create related record if not found.
```

If you don't want the user to be able to create a new record in the linked panel, this is the option you use for that data link. Note that option 4 on that menu now

reads No Access instead of No Create. If you then hit **4** again, the menu changes to this:

```
Define Data Link for Field
Link to Panel:2 Field1 Index1 Field List to Build Key:1
1 - Edit Target Field/Target Index/Field List
2 - Remove Data Link          5 - Prompt-Create
3 - Auto-Create              6 - Check During Data Entry Off
4 - No-Create                7 - Cascade Off
No Access
Selection: 0
```

Note that

Do not allow user to create related record if not found.

changed to

No Access

The No Access option not only keeps the user from creating new records in the linked panel (via the data link), but keeps the user from penetrating that link at all, even in Browse mode.

At this point, hitting **4** will toggle between these two menus and options:

```
Define Data Link for Field
Link to Panel:2 Field1 Index1 Field List to Build Key:1
1 - Edit Target Field/Target Index/Field List
2 - Remove Data Link          5 - Prompt-Create
3 - Auto-Create              6 - Check During Data Entry Off
4 - No Access                7 - Cascade Off
Do not allow user to create related record if not found.
Selection: 0
```

```
Define Data Link for Field
Link to Panel:2 Field1 Index1 Field List to Build Key:1
1 - Edit Target Field/Target Index/Field List
2 - Remove Data Link          5 - Prompt-Create
3 - Auto-Create              6 - Check During Data Entry Off
4 - No-Create                7 - Cascade Off
No Access
Selection: 0
```

The active option is always displayed just under option 4.

Option 5 (Prompt-Create). Hitting **5** gets you back to this menu's original state, with

Prompt for creating related record if not found.

displayed as the active status for that data link:

```
Define Data Link for Field
Link to Panel:2 Field1 Index1 Field List to Build Key:1
1 - Edit Target Field/Target Index/Field List
2 - Remove Data Link          5 - Prompt-Create
3 - Auto-Create              6 - Check During Data Entry Off
4 - No-Create                7 - Cascade Off
Prompt for creating related record if not found.
Selection: 0
```

Option 6 (Check During Data Entry Off). Hitting **6**, changes the menu to this:

```

--Define Data Link for Field--
Link to Panel:2 Field1 Index1 Field List to Build Key:1
  1 - Edit Target Field/Target Index/Field List
  2 - Remove Data Link                    5 - Prompt-Create
  3 - Auto-Create                        6 - Check During Data Entry Off
  4 - No-Create                          7 - Cascade Off
Don't check data link path during data entry.
--Selection: 0--

```

This results in the data link no longer checking what the user enters in that field. They can enter anything compatible with the field's format, whether or not it exists in the linked panel.

Option 7 (Cascade Update, Cascade Update/Delete)

This is just like it is under Panel Link Options above.

Data Link Options Caveats

Option 5 (Prompt Create)

If the user enters data in a data-linked field with the Prompt Create option active, and that data has no match in the linked panel, the user can still save that record. The closest thing you can do to prevent this is to formulate that field to blank out when the user enters data not found in the linked panel, and format the data-linked field ::M. The formula would look something like this:

```
if P1F1<>P1F1P2F1 then "" else P1F1 endif
```

That formula tests whether or not the data the user entered in P1F1 (Panel 1) is identical to that found in P2F1 (Panel 2), using the data link found on P1F1. If it isn't (which it won't be if the user enters a value not found in the linked panel), P1F1 blanks out. If it is (which it *will* be if the user enters a value *found* in the linked panel), P1F1 accepts the value the user entered. If P1F1 is formatted ::M, then the user won't be able to save the record if P1F1 is blank.

Option 4 (No Create, No Access)

Note also that if you change the data link to a No Create or No Access data link, though DataPerfect won't let you save the record if the data in that field doesn't match a record found in the linked panel, it *will* let you save that record with a blank in that field. This can be overcome by making that field a ::M field.

Data Link Subgroup Lookups

The Data Link Subgroup Lookup was introduced with version 2.3 of DataPerfect. It offers the definer a way to strategically filter what records the user sees during a Create or Edit mode lookup on a data link. Though the DataPerfect 2.3 manual fails to mention the Data Link Subgroup Lookup, the README file found on the shipped diskettes does, without referring to it as the Data Link Subgroup Lookup. Here's the relevant README passage:

DATAPERFECT: Lookup

Performing a lookup on a data link while you are creating or editing a record will display records from the linked panel based upon the data link Key Field List. If the first field in the Key Field List is the data link field, you will be able to see all records in the linked panel; otherwise, you will see only the records that match the fields of the Key Field List.

[README 02/01/93]

Starting with version 2.3, the data link's field list also acts as a record filter for the Create or Edit mode lookup display, not just the Browse mode lookup display. This allows you to define a data link in such a way that, during Create or Edit mode, a lookup performed on it will display a defined *subgroup* of the target panel's records—thus the name, Data Link Subgroup Lookup.

If you perform a lookup on a data-linked field during Create or Edit mode, the lookup will display the target panel's records filtered by all fields preceding the data-linked field in the link field list. That is, all the fields preceding the data-linked field in the link field list must have values that match the corresponding values found in the fields in the displayed record.

So, if the data-linked field is the *first* field in the data link's field list, then no fields precede it in the data link's field list. Performing a lookup on this field during Create or Edit will effectively display an *unfiltered* display of the linked panel's records (*all* the linked panel's records will be displayed). This effectively keeps things the way they were with DataPerfect 2.2, because it fails to create a Data Link *Subgroup* Lookup.

To create a Data Link Subgroup Lookup, the data-linked field must come somewhere after the first field in the link field list. If the data-linked field is the *second* field in the link field list, the lookup will display the target panel's records filtered by the *first* field in the link field list. If the data-linked field is the *third* field in the link field list, the display will be filtered by the *first and second* fields in the field list. And so on. Note, though, that if the data-linked field is *absent* from the field list, the lookup display will be *empty*, displaying the *No Data* message.

Let's clarify this with some examples. Suppose you run a local software retail store, like Egghead Discount Software. Like Egghead, you sell more than just software—your customers can also purchase books and peripherals. Let's say the Categories of Software, Books, and Peripherals exhausts everything you sell.

We'll place all those items for sale in the Item Panel, which will have at least two fields, Item and Category, where the latter field will be a U4 field that takes three possible values: SOFT, BOOK, or PERI. We'll create two indexes in the Item Panel, one sorting by Item, the other by Category. Straightforward, so far.

Now the Transaction Panel. There, we'll have at least the Category and Item fields again, along with the usual Date and Amount fields. On the Item field we'll place a data link that ties the field to the Item Panel, and have it target the Category field in the Item Panel. That data link will use the

<Category, Item>

index in the Item Panel, and have a field list consisting of

Category, Item

Here are the two panels, showing minimal configurations:

Cat.	Item	Amount
.....

Item Panel

Date	Cat.	Item	Amount
.....	◆.....

Transaction Panel

When the user enters the Transaction Panel and goes into Create mode, he first fills in the Date and the Category (SOFT, BOOK or PERI). Then he cursors to the Item field and hits **F8** or **Up Arrow**. What DataPerfect does at this point is where version 2.3 differs from 2.2. The lookup performed on the Item field during Create or Edit mode will show only records matching the Category field to its left. So if the user filled in SOFT in the Category field, hitting a lookup on the Item field during Create or Edit mode will display only a list of Software records.

Again, if a lookup is performed on a data-linked field during Create or Edit mode, DataPerfect first examines what field you're performing the lookup on (in this case, the Item field of the Transactions Panel). Then DataPerfect looks to see if that field is on the data link's field list. If it isn't, you get a recordless lookup, displaying the *No Data* message. If it *is* in the field list, the lookup display will be filtered by all fields that precede it on the data link's field list (in this case, it was filtered by the Category field), making sure the only records that display are those that have the value the user entered in the Category field.

So a good rule to follow here is this. If you attach a data link to a field that's to be used as a Pick List field during data entry (which is the only reason you should ever create a data link), then make sure the link field list has the data-linked field in a desirable spot. This will be determined by whether or not you want the lookup display to be filtered. If so, make sure the data-linked field immediately follows the filter fields in the link field list. Also be sure to adjust the Edit Order so the user will always fill in the filter fields before landing on the data-linked field.

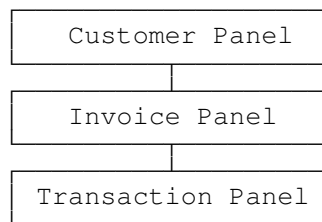
See *Data Link Subgroup Lookups and the USER.FIELD[n] Function* in the *User ID Panel* section of my **Securing the Application** chapter for more on this idea.

Choosing Between the Panel Link and the Data Link

It's easy to develop a bias in working with panel links and data links. That bias is that the panel link is for going *down* the typical panel hierarchy, and the data link is for going *up*. Or, put another way, the panel link provides a path to a *child* record, and the data link to the *parent*. Or, in keeping with our previous discussion, the panel link expresses either a one-to-many or a one-to-one relationship, and the data link expresses the many-to-one relationship. This bias, though it may describe as many as ninety percent of the typical links in a complex application, will limit your awareness of possibilities for application development.

Again, whether it be a panel link or a data link, a link is simply a bridge between a field list in the source panel and an index of the target panel, regardless of the relative position of each panel in the panel hierarchy. A panel link, then, depending on how it's defined, can just as easily take the user *up* the panel hierarchy as *down*.

Consider, again, the typical Customer-Invoice-Transaction application. Its panel hierarchy would look like this:



Here, going down the hierarchy is going from the one to the many, or from the parent to the child, typically expressed with panel links. But suppose you want to define a Keep A Total operation in the Transaction Panel, such that credits and debits entered there carry to fields that hold their totals in the Invoice Panel. Many assume the only way to do this is to use a *data* link—perhaps hidden—in the Transaction Panel, using it as the conduit to the parent Invoice Panel in the Keep A Total operation. But a *panel* link not only works just as well here, it's easier to define and takes less screen space. Let's explore this.

What pushes many to use a *data* link for the totalling routine is that we typically total to a parent record. And in the panel hierarchy, we typically use *panel* links to take us to child records, not parent records, leaving *data* links for Create or Edit mode lookups that target parent records. So, many may opt for the *data* link in their totalling routines because it's typically used to connect to a parent panel.

Let's go back to our Customer-Invoice-Transaction application. The totalling routine requires a link that takes us to the current Transaction Panel record's parent in the Invoice Panel. Whichever link we choose—panel link or data link—the link's field list must contain just the right key fields such that, when combined with an appropriate Invoice Panel index, it lands on the parent of the current Transaction Panel record.

If we assume every Invoice has a unique Invoice Number generated by the ::I or ::J format, then a Transaction Panel link that lands on the parent Invoice Panel record of the current Transaction Panel record will have a field list composed simply of Invoice Number. But *both* a data link *and* a panel link can be defined this way, because each has the same simple elements:

- Target panel
- Target panel field to land on
- Target panel index
- Source panel field list

In this case, the link—be it a data link *or* a panel link—will have its parent panel as its target panel. That is, *either* link will go *up* the panel hierarchy.

Choosing the panel link for a totalling routine has certain advantages over the data link. For one, we can create it in a panel filled with data; whereas, if you choose a data link, and want that link hidden, you may have to create a hidden field to accommodate the scheme. This would require purging the panel of data. And the panel link takes up less screen space, being only one character in size.

Before version 2.3, though, there was a good reason to use data links in totalling schemes if the link had to be hidden. You might want to hide a link to a record in a higher panel so as not to tempt the user to penetrate it and get confused—if it only exists for totalling to a higher panel, you *should* hide it. Every hidden panel link looks the same, but hidden data links can be attached to fields of different sizes, signalling the definer as to which higher panel this or that data link leads to. For instance, the Customer Number field might be G9999 and the Invoice Number field might be G999 (one character smaller in length). A data link on a hidden Customer Number field is then easy to discern from a data link on a hidden Invoice Number field.

But DataPerfect 2.3 brought us field names. When in Define mode, cursoring to a field will display its field name on the last line in the help screen. This field name is optional, and can be as simple as *Date*, or as informative as *Panel link to the parent Invoice Panel record*. You're allowed up to 47 characters. With this enhancement, I see no reason to use hidden data links at all.

So, in summary, the issue of whether or not the link goes up or down the panel hierarchy is irrelevant. If data checking isn't required, use a panel link for either direction. Data links should only be used for data checking (Create or Edit mode Pick Lists).

A Caveat Regarding Data Links

I've said, many times, you shouldn't create a data link for any reason other than for data checking (i.e., Create or Edit mode pick lists). Why do I insist on this?

Data links are intended for data checking and not for navigating between panels. The latter is the job for a panel link. I've seen many developers use data links where panel links should be used. Most of the time this mistake is done where a one-

to-many link is required. One-to-many linkage is the forte of the panel link, but DataPerfect allows you to create a one-to-many *data* link as well. There's really no difference in what goes into the creation of either link. Both the data link and the panel link definition processes involve assigning the same components:

- Target panel
- Target panel field to land on
- Target panel index
- Source panel field list

So you *could* create a data link that provides one-to-many linkage. But why *shouldn't* you? Though there are no *definitional* differences between the panel link and the data link, there are three *practical* differences between them:

- A data link is attached to an existing field. A panel link *is* a field.
- During Create or Edit mode, a lookup on a data linked field provides a lookup that displays records in a foreign panel for data entry into the current field in the current panel.
- Penetrating a data link while in Create or Edit mode leaves the source panel record in its current state (Create or Edit mode) while placing the user in the target panel; whereas, penetrating a panel link while in Create or Edit mode *saves* the current record before placing the user in the target panel. Following this logic through, you'll find yourself in Create or Edit mode when you return to the source panel in the data link situation, but find yourself In Browse mode when returning to the source panel in the panel link situation.

The third difference above has serious consequences when opting for a one-to-many *data* link over a one-to-many *panel* link. Remember that a one-to-many link is the kind that's typified by the Invoice-Transaction relationship. Such a link takes you from a parent record to its many subrecords.

Say you just created Invoice 012 in the Invoice Panel and want to penetrate a link to the Transaction Panel to enter some Transactions for that Invoice. Whether you use a data link or a panel link for this, in either case, when you arrive in the Transaction Panel you'll see the Invoice Number field in each Transaction Panel record you create has 012 already filled in for you (if you designed this one-to-many linkage properly).

But let's say the link you use for this is a *panel* link. After you save these Transaction Panel records for Invoice 012 and exit back to the Invoice Panel, you'll find Invoice 012 in Browse mode. Now you decide to delete Invoice 012 (the Customer changed his mind). If you have Cascade Update/Delete on that panel link (**Shift-F8, 7**), its subrecords in the Transaction Panel will also be deleted. Likewise, if you leave Invoice 012 in place, but change it's Invoice Number to 013 for some reason, Cascade Update or Cascade Update/Delete on that panel link will preserve the linkage, modifying the Invoice Number for each of those Transaction Panel subrecords so they're still attached to the Invoice with its new Invoice Number.

But if this link is a *data* link, things are very different. Again, you just returned from the Transaction Panel to the Invoice Panel, but in this situation you find yourself in either Create or Edit mode, not Browse mode. So Invoice 012 was never saved. If you now decide to Cancel the creation of Invoice 012, you now have Transaction Panel records attached to a nonexistent Invoice 012. Don't forget those Transaction Panel records were saved with 012 in their Invoice Number field. Alternatively, say you decide you really want 013 as this new Invoice Number, not 012. So you **Tab** to the Invoice Number field and change 012 to 013. Now you have Invoice 013 with no Transactions, and Transactions in the Transaction Panel that are attached to a nonexistent Invoice 012. Again, don't forget those Transaction Panel records were saved with 012 in their Invoice Number field, not 013. And Cascade Update and Cascade Update/Delete won't help here. That only comes into effect if a *saved* record is altered or deleted. Invoice 012 was not saved upon first re-entering the Invoice Panel after creating those Transaction Panel records.

This is especially troublesome on a network, which will give rise to a another flavor of this problem. Suppose Sue and Tom, each on a different work station on the same network, both access the Customer Panel in our example application. Each is creating a new Customer record, though for different Customers. Further suppose the Customer ID field is a G999::I field, which is being used by the developer to give each new Customer a unique number, incrementing each time the user goes into Create mode. Sue hits **F9** to create a new Customer record and sees the Customer ID field fill in the next number in sequence, which is, say, 012. This application, unfortunately, requires she use a data link to access the Invoice Panel from the Customer Panel. So, while still in Create mode in the Customer Panel, she **Tabs** to that data link and hits **F5** to create the first Invoice record for this Customer. She finds herself in the Invoice Panel in Create mode. Assuming the Invoice Number field is also a G9999::I field, let's say she sees 0134 in that field. After filling in the appropriate information in that Invoice record, while still in Create mode, she cursors to a data link in that panel that takes her to the Transaction Panel. She hits **F5** to create some Transactions in the Transaction Panel for Invoice 0134. Now she finds herself in the Transaction Panel in Create mode.

In the mean time, Tom has done everything Sue has done, only for *his* Customer. What's wrong with this picture? Well, the ::I field modifier will increment the G999::I field to the next number in the series in Create mode. So both Sue and Tom are working on a Customer record with 012 in the Customer ID field—a record that has yet to be saved. Likewise, both Sue and Tom are working on an Invoice record with 0134 in its Invoice Number field—a record that has yet to be saved. Both Sue and Tom are in the Transaction Panel, creating Transactions for their respective Customer.

When Sue is done saving her last Transaction for her new Customer, she has a bunch of Transactions linked to Customer 012, Invoice 0134. When Tom is done saving Transactions for *his* new Customer, he also has a bunch of Transactions linked to Customer 012, Invoice 0134. Now Sue and Tom have created Transactions for the same Customer instead of two different Customers. Further, they have created Transactions for the same Invoice instead of two different Invoices.

When Sue exits from the Transaction Panel to the Invoice Panel, she finds she's still in Create mode in the Invoice Panel. When she exits the Invoice Panel, DataPerfect saves Invoice 0134 and returns her to the Customer Panel. There, she finds she's in Create mode. She saves that record for Customer 012.

On the other hand, when *Tom* exits from the Transaction Panel to the Invoice Panel, he *also* finds himself in Create mode in the Invoice Panel, but when he exits to the Customer Panel, DataPerfect saves that Invoice Panel records with 0135 in the Invoice Number field, not 0134. This is because the ::I modifier increments the field value by one when you hit **F9**, and checks again on Save to see if that number has been used sometime since you hit **F9**, probably by someone else on the network. In this case, DataPerfect sees 0134 has been saved by someone else, so it increments to 0135. But now all those Transaction Panel records Tom created are linked to the wrong Invoice, since they were saved with 0134 in their Invoice Number field.

To add insult to injury, when Tom exits back to the Customer Panel and saves that Customer Panel record, he finds its Customer ID field goes from 012 to 013. Again, the same logic rules: DataPerfect sees that someone else saved a record in that panel with 012 in that field before Tom did, so it increments a second time on Save, to 013.

So, Tom has created a Customer 013 record with no Invoice, an Invoice 0135 record with no Transactions, and Transactions that belong to *Sue's* newly created Customer 012 and Invoice 0134.

The above possibilities arise when using a data link instead of a panel link for a one-to-many linkage because of the unfortunate interplay between these two facts:

- Penetrating data link while in Create or Edit mode leaves the source panel record in its current state (Create or Edit mode) while placing the user in the target panel.
- A ::I field increments when the user hits **F9**, and then rechecks on Save to make sure that new number hasn't already been saved by someone else since the user hit **F9**. If so, it increments again.

The possibility for the above improper use of data links is the reason DataPerfect's author eventually created the ::J field modifier (introduced with the initial release of version 2.3). A ::J field increments when the user hits **F9** and yields a unique value, even if someone else hits **F9** on the network in that panel at the same time. If two network users hit **F9** at the same time in the same panel, a ::J field will give each user a different number, one being one higher than the other. That value will never change, even on Save. Further, say you hit **F9** and a ::J field increments to 012. Then you Cancel with **F1**, deciding not to create a new record after all. Well, the next time you hit **F9**, that field will increment to 013, not 012. Cancelling a record creation loses the number the ::J increments to, but not the number the ::I field increments to.

If you replay the Tom and Sue case outlined above, but this time with ::J fields instead of ::I fields, you'll see the problems that arose from using the data link as a one-to-many link resolve themselves. Still, just don't use the data link as a one-to-many link. It was only created for data entry pick lists.

The Recursive Panel Link

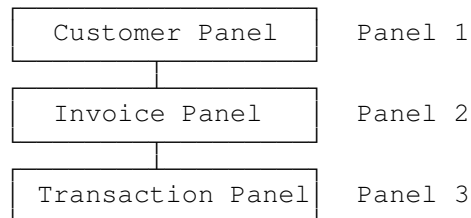
[Refer to UD.STR for this.
Find the *Recursive Links* series and load the Customer Panel.]

If you frequent the DataPerfect section of the WordPerfect Users Forum on CompuServe, you inevitably find recursive panel links discussed. A panel link is *recursive* when it takes the user back to the same panel. Don't look to the Define Panel Link menu for a special selection that makes a link recursive—it's not there. You make a panel link recursive by simply choosing the same panel as its source and destination panel.

Why do this? On the surface it sounds rather silly, but there are good reasons for creating recursive panel links. Consider the topic of *incrementation*.

Conditional Incrementation Using a Recursive Panel Link

Though DataPerfect offers the auto-incrementing field (:I or ::J), this only increments a field *absolutely* with respect to its panel. You might, though, have reasons for a field to increment *conditionally*. Consider the three-panel Customer application to clarify the difference here:



Suppose we identify each record in each of the panels with a number: Customer Number, Invoice Number, and Transaction Number. Further suppose each Customer has a unique Customer Number and each Invoice a unique Invoice Number, *but Transactions will have Transaction Numbers that are unique only within the world of a single Invoice*. That is, the Transaction Number field will increment *conditionally* with respect to its Invoice Number, starting over with each new Invoice.

Suppose our Transaction Number field is P3F1 and formatted G99::N. How do we formulate it to increment with each new Transaction, but start from 01 with each new Invoice? The auto-incrementing field won't help because it doesn't give a hoot what this particular Transaction's Invoice Number is. It's going to increment to the next number by adding one to the Transaction field's highest number used *in the panel as a whole*, as opposed to adding one to the Transaction field's highest number *for this particular Invoice*.

To properly formulate P3F1, you can take the following steps, all while in the Transaction Panel (Panel 3):

1. Create a hidden Reverse Transaction Number field (P3F2) and format it G-99::H. Formulate it to update on any change to -P3F1.
2. Create an index whose first field is the Invoice Number field (P3F3), and whose second field is the hidden Reverse Transaction Number field (P3F2). This index sorts forward by Invoice Number, but backwards by Transaction Number.
3. Create a panel link whose destination panel is the Transaction Panel. Assign it the above index, and a field list consisting only of the Invoice Number field (P3F3). Hide the link.

Note: Ponder the panel link you just created:

- Because its field list consists only of the single Invoice Number field (P2F3), it takes the user to all and only Transactions of a particular Invoice.
- It's recursive, in that its source and destination panels are the same.
- It takes the user to the highest numbered Transaction for each Invoice.

Such a panel link seems more likely to reside in the Invoice Panel than the Transaction Panel, taking the user from a particular Invoice to its Transactions, but we'll keep it the Transaction Panel anyway.

4. Formulate the Transaction Number field (P3F1) in Specify Formula screen: **F4** and **Tab** to the new panel link. **Down Arrow** through the link. **Tab** to the Transaction Number field. **F4**. Add one to this field. Have it update on record creation.

Following the above formula procedure, if the recursive panel link is P3F4, the formula will be

$$P3F4P3F1 + 1$$

The Transaction Number field will now increment conditionally, starting from 01 with each new invoice. Put simply, we created a Transaction Panel panel link that sees only Transaction Panel records for that particular Transaction Panel record's Invoice, and lands on the record with the highest Transaction Number. We then formulated P3F1 to increment by adding one to the Transaction Number it sees on the other side of this link.

Absolute Incrementation Using a Recursive Panel Link

That's how to use the recursive panel link to *conditionally* increment a field. Now, though DataPerfect offers the auto-incrementing field to *absolutely* increment a field, you might have reasons to have that field *absolutely* increment its records without formatting it as auto-incrementing (::I or ::J).

I'll take this moment to outline *my* reasons for shunning auto-incrementing fields, even when I need a field to increment *absolutely*.

Reasons for Avoiding Auto-Incrementing Fields

Here are my reasons for avoiding auto-incrementing fields whenever possible:

1. Auto-incrementing fields make the export and subsequent re-import of merge file data less than smooth for the uninitiated. Upon import, DataPerfect asks the user what to do with the import file's auto-incrementing fields. If no such fields exist in the panel in question, DataPerfect won't prompt for this, making import smoother for the uninitiated.

2. When developing an application, you no doubt use dummy data to test and retest your panels and reports. If your application includes auto-incrementing fields, then when ready to release your application to a client, you must remember to reset all the auto-incrementing fields after deleting the dummy data.

3. More importantly for the developer, though, is the ease of upgrading a client's application. Suppose you have a copy of your client's .STR file (without their data) and decide to simply add, alter, or delete a few reports. Now you want to upgrade their application. The standard method (that is, that method that was supported by WordPerfect Corporation, and later, Novell) for upgrading your client's application in such a situation would be the following five steps:

- a. Run DPEXP on the new .STR file, creating an .STE file.
- b. Load the new .STE file in your text editor and delete all that precedes *REPORTS:*. Then delete all but the new and altered reports. Make sure the file has a blank final line. Save the file as, say, REPORT.STE.
- c. On your client's computer, run DPIMP on REPORT.STE, importing it into their old .STR file.
- d. Delete reports that are now obsolete.
- e. Re-order the Report List with DPOrder.

There's a simpler way to do this, though it was never supported by DataPerfect's manufacturers. If the only change you made to the previous .STR file was to add, alter, or delete reports, *and no field in the application is formatted as auto-incrementing*, you might be able to simply overwrite the old .STR with the new. This means you can just send client the new .STR with a batch file that copies the new .STR to the appropriate directory.

What are some of the things the developer might do to the new .STR that would preclude simply overwriting the old .STR with the new? Here are a few:

- Adding, altering or deleting an index.
- Altering the format of a field (there are some exceptions to this)
- Adding or deleting a field (also some exceptions)

If you think about it, this is pretty obvious. If you do one of the above to the .STR in your office, its relationship with the data it will govern in your client's office

may be critically altered. But if you simply add, alter, or delete a few reports, you haven't altered how the .STR relates to the data files it governs.

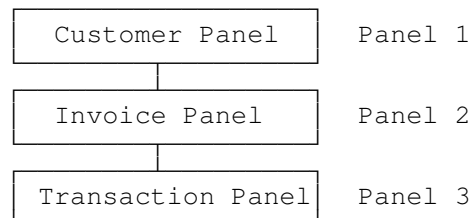
Now where do auto-incrementing fields fit in here? Well, auto-incrementing field pointers are stored in the .STR. That is, the .STR holds information telling it what the next highest number is for each auto-incrementing field in the database. If you don't reset the auto-incrementing fields to correspond with their current states in your client's database, overwriting the old .STR with the new will obviously cause problems.

If you insist on having auto-incrementing fields in your client's database, you might consider giving your client a report that prints out the state of each auto-incrementing field in the database—hidden or otherwise—and have them fax it to you just before you ship the new .STR. This way you can reset the auto-incrementing fields at the last minute, coordinating this with the client.

Please don't start overwriting client .STR files with new ones until you've read *When It's Okay to Overwrite an .STR* in my **Application Maintenance Issues** chapter! I discuss this in more detail there.

Back to Recursive Linking and Absolute Incrementation

Many of the above concerns disappear with proper use of the recursive panel link. Consider our Customer application again:



Unlike the Transaction Number field, we want the Customer and Invoice Number fields to increment *absolutely*. To do this, we again call on our recursive panel link, though this time we'll also make the link *virtual* (I'll explain what a *virtual* panel link is in a second).

Here's what I suggest. Let's do this for the Customer Panel, leaving it up to you to do the same for the Invoice Panel. Assuming the Customer Number field is P1F1, do the following:

1. Create a hidden Reverse Customer Number field (P1F2) and format it G-9999::H. Formulate it to update on any change to -P1F1.
2. Create an index consisting simply of the hidden Reverse Customer Number field (P1F2). This index sorts backwards on Customer Number.
3. Create a panel link whose destination panel is the Customer Panel. Assign it the above index. When asked for a field list, don't select any field—just hit **0**, **F7**, or **F10** (don't hit **F1** here, or else you'll have to define the link again). Hide the link.

Note: Ponder the panel link you just created:

Like the panel link for conditional incrementation, it's recursive, taking the user to its own panel, based on a reverse sort. What makes this link different than the other recursive panel link is that it works without a field list to filter the records the user can access in the destination panel: it's a *virtual* link. Penetrating a virtual link has the same effect as entering the destination panel from the Panel List, except the virtual link has the definer choose which index will govern entry into the destination panel (remember that entry into a panel from the Panel List is always controlled by the lowest numbered index in that panel).

4. Formulate the Customer Number field (P1F1) by first having the formula penetrate the recursive virtual panel link and grab the Customer Number field. Then add one to it. Have it update on record creation.

Following the above formula procedure, if the recursive virtual panel link is P1F4, the formula will be

P1F4P1F1 + 1

The Customer Number field will now increment *absolutely*.

Let's go back now to the concern I outlined involving upgrading a client's application. If all of the application's *absolutely* incrementing fields increment with the *recursive virtual* panel link scheme I just outlined, then information as to the next highest number for each such field is stored in the that field's *data* file, not its .STR file. This frees you, the developer, from being concerned about the next highest number for each auto-incrementing field in the client's database the next time you want to upgrade their application.

Absolute Incrementation Using a Recursive Panel Link on a Network

[Refer to UD.STR for this.

Find the panel called *Moment field; Proofing; Incrementing on a network*.

The Record Number field increments this way in that panel.]

This concerns you if you decide to implement my suggestions regarding using absolute incrementation with recursive links instead of using ::I or ::J modifiers, *and are doing so on a network*. Note that I said such an absolutely incrementing field should update *on record creation*:

Formulate the Customer Number field (P1F1) by first having the formula penetrate the recursive virtual panel link and grab the Customer Number field. Then add one to it. Have it update **on record creation**.

In the past, I taught that, to use this scheme *on a network*, just set the absolutely incrementing field to update *when created record is saved* instead of

simply *when record is created*. Well, a fellow DP developer who attended my 1995 Atlanta seminar (Ireland's Frank Hannah) recently pointed out that doesn't work as nicely on his network. Even Lew Bastian (DataPerfect's author) thought the recursive linking approach worked fine on all networks, and he, like me, uses it instead of ::I or ::J. I'll outline the problem, and a simple solution.

To review, the typical way to implement the suggested *absolutely incrementing field using a recursive link* scheme would resemble this:

Absolute Incrementation Using One Recursive Link	
P1F1	Absolutely incrementing field without ::I or ::J. Formulated to update <i>when created record is saved</i> (not <i>when record is created</i>) to $P1F3P1F1+1$. $P1F3P1F1$ is obtained by penetrating link P1F3.
P1F2	Reverse field. Formulated as $-P1F1$ to update <i>on any change</i> .
P1F3	Recursive panel link. Takes the user to the same panel, uses Index 1, and lacks a field list.
Index 1	Reverse index. Its field list only contains one field: P1F2.

The problem arises this way. On a network, one user hits **F9** in that panel. Then, while the first user is still in Create, another user hits **F9** in the same panel. Now they both have the same number in P1F1. One user then saves his record successfully. Then the other attempts to save. Unfortunately, the second user is told he's attempting to save a *nonunique record*. He must now cancel (**F1**) and create (**F9**) all over again. This time, it works because he gets a higher number in P1F1. Though no corruption occurs this way, it's annoying.

Because setting a field formula to update *when created record is saved* causes a field to recalculate its formula on Save, this should have safeguarded us here, producing a different number when the second user saves his record. But apparently when a formula is set to update *when created record is saved*, DataPerfect 2.3 caches (stores in memory) a copy of the panel link and the record on the other side of the link it sees on Create. Then, on Save, it does indeed take another look through the link, but it takes its second look through the *cached* link-record combination (the copy in memory). So it sees the value it saw on Create again, even if the field formula that uses that link updates *when created record is saved*. So, in such cases, setting P1F1 to update *when created record is saved* gives the same result as having it update *when record is created*. Thus when the second user saves his record he gets the *nonunique record* error message and has to cancel and create again.

You can solve this by adding a new panel link to your already existing recursive panel link scheme (and you can do this to a panel with data in it). Let's say your new panel link is P1F4. All you have to do is this:

- Make sure the Edit Order places P1F1 before P1F2. This is important.
- As before, P1F2 is formulated as $-P1F1$ to update *on any change*.

- Create a second panel link. The two panel links are identical. Each takes the user to the same panel, uses Index 1, and lacks a field list. This isn't different than before, except you now have two links instead of one.
- As before, formulate P1F1 to update *when created record is saved* (not *when record is created*). What's new with P1F1 is its field formula. Use

```
(if P1F2=0 then P1F3P1F1 else P1F4P1F1 endif)+1
```

or its equivalent

```
if P1F2=0 then P1F3P1F1+1 else P1F4P1F1+1 endif
```

In the above, *P1F3P1F1* is obtained by selecting P1F1 via the *first* recursive link, and *P1F4P1F1* by selecting P1F1 via the *second* recursive link. This formula takes advantage of the fact that, given the Edit Order, P1F2 is momentarily 0 on Create. When it sees P1F2 is 0, P1F1 grabs the value on the other side of the *first* recursive link and adds 1 to it. Later, on Save, when it sees P1F2 has changed to something other than 0, it grabs the value on the other side of the *second* recursive link and adds 1 to it. Because the *second* recursive link was not accessed on Create, it was never cached. The value the field formula sees on the other side of this link on Save, then, is new. Edit Order is important here. The incrementing field must come before the reverse field. The Edit Order of the *links* is irrelevant.

Again, this entire discussion only concerns you if you use the *absolute incrementation using a recursive link scheme* on a network. If you're using it on a single-user setup, you can implement the *single-link* method. And don't forget you can always update your scheme to the *two-links* method without exporting any data, because all you have to do is add a panel link and change a field formula. Both of these can be done with data in the database. Here's a summary of the *two-links* method:

Absolute Incrementation Using Two Recursive Links	
P1F1	Absolutely incrementing field without ::I or ::J. Formulated to update when created record is saved (not when record is created) to if P1F2=0 then P1F3P1F1+1 else P1F4P1F1+1 endif P1F3P1F1 is obtained by penetrating link P1F3. P1F4P1F1 is obtained by penetrating link P1F4.
P1F2	Reverse field. Formulated as -P1F1 to update on any change.
P1F3	Recursive panel link. Takes the user to the same panel, uses Index 1, and lacks a field list.
P1F4	Second recursive panel link. Identical to link P1F3.
Index 1	Reverse index. Its field list only contains one field: P1F2.
Note	P1F1 must precede P1F2 in Edit Order.

Late Breaking Note

Just before shipping this manuscript to the publisher, Lew Bastian (DataPerfect's author) contacted me. After reading a copy of the manuscript I sent him, he said he's now considering changing DataPerfect to make the *two-links* method unnecessary on a network. That is, he's considering having DataPerfect clear the link cache before Save. This will slightly slow Saves, but make things more logical in terms of the Update When Created Record is Saved option. By the time you read this, Lew may have already made this change to DataPerfect. See *Support Avenues* in **Epilogue** for ways to keep current on this. If Lew makes this change, the *single-link* method will suffice. The *two-links* method, though unnecessary, will still work. So this change won't break an application that employs the *two-links* method.

Virtual Link vs. Subreport Using Virtual Link

The recursive link discussed above is what DataPerfect calls a *virtual link*. A virtual link, though still assigned an index, *lacks a field list*. A virtual link need not be recursive, however. It's simply a panel link that lacks a field list. Don't confuse this with something else that sounds related: *Subreport Using Virtual Link*. Though this isn't the place to get into the Subreport Using Virtual Link (I talk in detail about this in *Subreport Using Virtual Link* in my **Subreports** chapter), suffice it to say this is option 4 in the following menu, called by **Ctrl-F7, 6** in the Report Body of the Edit Report Form screen:

Subreports & Record Creation

1 - Include Subreport

2 - Create Record Through Link

3 - Create Record From Panel List

4 - Subreport Using Virtual Link

0 - Return to Edit

Selection: 0

Until version 2.3, DataPerfect offered only option 1 above (Include Subreport). To create a subreport that way, you need the appropriate panel link waiting for you in the Report Body's associated panel. If this panel link is one you prefer the user not see, you create and hide it for this report (see *The Link Options Menus* earlier in this chapter for more on hiding a link). Many of us have tons of these hidden panel links in the applications we created with DataPerfect 2.0 through 2.2. We created them with the sole purpose of facilitating the definition of subreports.

With version 2.3, we don't need these hidden panel links—at least those created for subreports. Option 4 allows a report to essentially create a temporary panel link while it runs. When defining such a subreport, you tell DataPerfect what index and field list to use. This is just like defining a panel link (with one exception, which I'll outline later). And, just like the panel link definition process, you may elect to forego the field list altogether.

DataPerfect's use of the term *virtual* in these two concepts is confusing. A panel link is virtual when it lacks a field list. But DataPerfect's *Subreport Using Virtual Link* option involves creating a subreport without an existing panel link. The phrase *Subreport Using Virtual Link* makes sense to me. It involves a subreport that really uses a virtual object: a temporary panel link the report creates on the fly. But DataPerfect's virtual panel link really isn't virtual: it's as real as any other panel link. What makes it different is that it's missing something other panel links have: a field list. Perhaps it should be called *fieldless*, *unkeyed*, or *unfiltered*.

Again, I discuss the Subreport Using Virtual Link extensively in *Subreport Using Virtual Link* in my **Subreports** chapter.

Making Panel Links Safer

Starting with version 2.3, DataPerfect's panel link is characterized by a behavior many developers and users don't like. When sitting on a panel link, hitting **F5** puts you in the subpanel in *Create mode*. Because of this, the user can easily find himself unintentionally in Create mode in the subpanel. Then, to add to this problem, the user may become confused when he sees he's in Create mode, and hit **F7** (Exit) instead of **F1** (Cancel). Whereas **F1** will gracefully resolve this situation, **F7** won't. **F7** exits the subpanel and saves the record the user never wanted to create in the first place.

Computed Fields Next to Links

By far, the most common way of dealing with this is to place an A2::C field next to each panel link, and then formulate each of these A2::C fields to update to ASCII 25 (Down Arrow) if it sees a record on the other side of its adjacent link. To define this formula, you need to penetrate the panel link on the Specify Formula screen and

select (**F4**) a numerical field in the subpanel that will never be 0, or a text field in the subpanel that will never be blank. Such a formula would look something like

```
if P1F10P2F1 <> "" then "↓" else "F5" endif
```

where P1F10 is the adjacent link and the selected subpanel field P2F1 is text, or

```
if P1F10P2F1 <> 0 then "↓" else "F5" endif
```

if P2F1 is numerical. However, the first formula works for both situations because DataPerfect evaluates "" as a blank if the field in question is text field, and 0 if the field in question is a numerical field.

In any event, the above A2::C field displays Down Arrow if it sees a record on the other side of the link, otherwise it displays *F5*. If it abuts the panel link in question, it reminds the user which key to hit. If he follows the directions this A2::C field offers, he hits **F5** only if there aren't any subrecords attached to the currently displayed record in the parent panel. If there are such records, he hits **Down Arrow**, even if he wants to create a new subrecord. If he indeed wants to create a new subrecord when some already exist, he hits **Down Arrow** in the parent panel, relying on **F9** in the subpanel to get into Create mode there.

A popular alternative to the above scheme is to simply have an A1::C field display an asterisk if it sees a record on the other side of the link. This formula would look like this:

```
if P1F10P2F1 <> "" then "*" else "" endif
```

The DPMouse© Alternative

But there's a powerful alternative to using computed fields here. If you install a copy of DPMouse© alongside DataPerfect, you can make it so the user can only penetrate a panel link or data link by hitting **F5**. At that point DPMouse© presents the user with a popup that offers him the choice to access the subpanel in Browse mode or Create mode. This way the user's hand is held in a way that greatly limits the possibility of inadvertent record creations in a subpanel. Take a look at the DPMouse© manual I provided on diskette for more details on this DataPerfect addon.

A related issue is finding a way to keep users out of a subpanel under certain conditions; that is, closing a panel link under certain conditions. This might involve keeping certain users out all the time, or all users out under certain conditions. DPMouse© will allow the developer to do this. For more on this, see *Using DPMouse© To Conditionally Close A Panel Link* in my **Securing Data Entry** chapter, as well as the DPMouse© manual I provided on diskette. Also see *User ID Panel* in my **Securing the Application** chapter for ideas on how to combine DPMouse© and the USER.FIELD function to control not only *when* a link can be penetrated, but *who* penetrates it.

Troubleshooting Links

Problem

The subrecord isn't linked after exiting back to the parent panel. That is, you create a few records on the other side of a panel link, and then exit back to the parent panel. When you attempt to penetrate the panel link with **Down Arrow** to browse the subrecords, you receive the *No records are found in this subset* message.

Solution

One of these is probably true:

- The link field list isn't a perfect match on an ordered subset of fields that start the link index. That is, some field on the link field list isn't matched on the link index; or it's matched by a field of different type or length; or the fields in the link index that match the link field list aren't in the same order as they are in the link field list; or the first field in the link index isn't the matching field for the first field in the link field list.
- Some matching field in the link index is an auto-incrementing field.
- Some matching field in the link index has a field formula that, on Save, changes the value it inherits from the field it matches.
- The indexes in the target panel need to be regenerated.

Problem

A Create or Edit mode data link lookup fails to sort properly no matter how you define the link.

Solution

You're working on the wrong entity. A Create or Edit mode data link lookup display is controlled by the target field, not the source field. Exit the source panel and load the target panel. Define the target field's Browse mode lookup the way you want it to display in the Create or Edit mode data link lookup you just left. Now return to the source panel and try again.

Problem

You edit certain fields in the parent record and later find its subrecords in a linked panel are no longer linked to this parent record.

Solution

You're editing one or more fields in the field list of the panel link in question. You have a few choices here to stop this from happening:

- Put the ::N modifier on each link field list field.
- Install DPMouse© and place an ASCII 250 character (·) next to any link field list field that isn't a ::N field. This will keep the field from being edited after the record is saved. Read up on DPMouse© in the manual I provided on diskette.
- Put Cascade Update or Cascade Update/Delete on that panel link. This will make sure that whenever the user edits a link field list field, its change will

cascade to all dependent subrecords accessed via that panel link, thus preserving linkage. Cascade Update/Delete goes a little further than just Cascade Update. For more on Cascade Update and Cascade Update/Delete, see *Panel Link Options* in *The Link Options Menus* in this chapter.

Problem

In the parent record you see what looks like data from subrecords overriding the display. This seems to hide some fields you don't want hidden.

Solution

You inadvertently defined a window for some panel link in that panel. Go find it and, with the cursor on that link, turn off its window with **Shift-F8, 4**.

Problem

Some fields seem to be hidden ever since defining a panel link in the same panel. But those fields aren't ::H fields.

Solution

Same as above. Get rid of the inadvertently created panel link window.

Problem

You got fancy with a panel link and put a ::C field on its field list. It worked fine in your office, but when your client installed the new .STR, that link no longer properly controlled access to subrecords.

Solution

Your client is using a version of DataPerfect prior to the September 1993 version 2.3. Computed fields weren't allowed on link field lists until this second release of version of 2.3.

Problem

Referential Integrity options (Cascade Update and Cascade Update/Delete) worked fine in your office, but when you sent the .STR your client, it didn't work properly.

Solution

Same as above. Referential Integrity options weren't introduced until the second release of DataPerfect 2.3 (September 1993).

Problem

You're using a Keep A Total to update subrecords through a panel link. It's supposed to act as sort of date stamp, where a change in the parent record causes a Keep A Total to subpanel records, which in turn causes a date field in each one of the subrecords to update to today's date. But when you examine each parent record's subrecords, you find only one of its subrecords was updated with the Keep A Total operation.

Solution

A Keep A Total operation only targets the first record it sees on the other side of a link. If you want all subrecords of a parent record to update via some formula or another, you'll need to come up with a Cascade Update scheme, or else use a report.

Problem

When you penetrate a panel link, lookups in the subpanel work fine, but not when you enter that same subpanel from the Panel List. Or vice versa: when you enter the panel via the panel link, lookups don't work properly, but they work fine when enter that same subpanel from the Panel List. Or lookups in that subpanel work fine when entering that panel from the Panel List, and also when entering it from *some* panel links, but not others.

Solution

This isn't a link problem. It's a Smart Lookups problem. Read up on this in *Smart Lookups* in my **Lookups** chapter.

Problem

In Create or Edit mode you keep getting a *No Data* message when performing a lookup on a data link field.

Solution

Other than the obvious, which is that the target panel has no records, check the link definition to make sure that either the data link field is the first field of both the link field list and index, or else all fields that precede the data link field in the link field list and index are filled with data at the time the user is to perform the lookup on the data link field. Read *Data Link Subgroup Lookups* in this chapter for more on this.

Problem

Any attempt to create or edit a record after penetration of a particular data link is met with the *You don't have access rights to do that* message.

Solution

This is a common problem when upgrading a DataPerfect database created with DataPerfect 2.2 or earlier, to 2.3 status. Make sure that the data link is set for *Prompt for creating related record if not found* in the Define Data Link menu (option 5). Also, if a menu item governs entry into the panel with the data link, make sure you have *Restrict Modification to First Level* set to *No* for that menu item.

Problem

Your recursive link doesn't seem to result in your incrementing field incrementing properly.

Solution

The most common reason for this is selecting the *reverse* field when passing through the recursive link in the Specify Formula screen, instead of the incrementing field

itself. The *reverse* field is the one that updates on any change to the negative of the incrementing field.

Problem

When you delete a record in one panel, records seem to be deleted in another panel.

Solution

At least one link (perhaps hidden) has Cascade Update/Delete on. Cascade Delete is the only way to delete a record in a foreign panel while deleting a record in the current panel.

Problem

When you create a record in one panel, sometimes records in another panel get created.

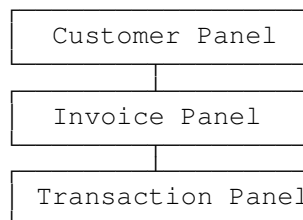
Solution

There's at least one Keep A Total in the current panel that targets that other panel. The Keep A Total facility is the only way to create a record in a foreign panel when saving a record in the current panel.

This chapter targets both the beginner and the experienced.

Introduction

Keep A Total is one of DataPerfect's most powerful features. Before DataPerfect, you were hard put to find this feature in a database management system. What Keep A Total does in DataPerfect is what other database management systems typically call *master file updates*. Let's say you create an application with following panel hierarchy:



The Invoice Panel will hold totals carried from the Transaction Panel (Total Charges, Total Payments, Balance, etc.). The Customer Panel will have similar fields. The fields that hold totals in the Customer Panel and the Invoice Panel will somehow update by receiving data from the Transaction Panel. When the user enters a record in the Transaction Panel, you want the value in its Amount field to somehow influence the appropriate fields in the Invoice Panel and Customer Panel so the user will always have current values in the latter two panels.

DataPerfect has always made it easy to get the database to update the appropriate fields in the top two panels with each record created, edited or deleted in the lowest panel. This has been true beginning with DataPerfect's initial release as 2.0. Not so with other database management systems. If you wanted each Transaction Panel record Create, Edit or Delete to dynamically update the appropriate fields in the top two panels, even if your database management system had an application generator, you had to write many lines of code for each such Transaction Panel field, telling it what to do with each Create, Edit or Delete, with respect to totalling to the top two panels. Until recently, one of DataPerfect's major competitors (Alpha Four), didn't allow any dynamic master file updates. Rather, you had to provide the user with a master file update routine to run daily, weekly, or monthly. So totals in the top two panels wouldn't be up to date until after this master file update routine was run. For some developers, it was DataPerfect's Keep A Total facility that most influenced their choice of DataPerfect over other database management systems available.

Implementing a Keep A Total

Say those three panels look something like this:

CUSTOMER PANEL			
ID	Last Name	First Name	Balance
To Invoices			

INVOICE PANEL			
ID	Last Name	First Name	Balance
Inv #	Charges	Payments	
To Transactions			

TRANSACTION PANEL			
ID	Last Name	First Name	
Inv #	Description	Charge	Payment

You'd like the following to happen when the user *creates* a Transaction Panel record:

- If he places a value in the Charge field, on Save DataPerfect will do the following with that value:
 - Add it to the Charges field in the Invoice Panel
 - Add it to the Balance field in the Invoice Panel
 - Add it to the Balance field in the Customer Panel
- If he places a value in the Payment field, on Save DataPerfect will do the following with that value:
 - Add it to the Payments field in the Invoice Panel
 - Subtract it from the Balance field in the Invoice Panel
 - Subtract it from the Balance field in the Customer Panel

Further, if the user *edits* or *deletes* a Transaction Panel record, you'd like the totals in the top two panels adjusted up or down appropriately.

All the above is easily accomplished with only a few keystrokes when defining a panel. If you want a field to carry its value to a foreign panel, for the purpose of updating a field in the foreign panel that holds a total, you simply need to make sure there exists a link in the current panel, where that link leads to the current record's parent record in that foreign panel.

In the above case, let's consider the Charge field in the Transaction Panel. To get DataPerfect to update the Charges and Balance fields in the Invoice Panel, and the Balance field in the Customer Panel, we need a link in the Transaction Panel that targets the current record's parent record in Invoice Panel, and another link that targets its parent record in the Customer Panel. Each link may be either a panel link or a data link. As long as passing through that link in Browse mode lands the user on the parent record of the current Transaction Panel's record, it'll work. We might as well make them hidden panel links. Then all we do now is create three instances of Keep A Total on the Charge field in the Transaction Panel.

To create the first instance of Keep A Total on the Transaction Panel Charge field, we cursor to that field in either Browse mode or Define Panel mode, and **Shift-F8, 8** to get a Totaling menu:

Totaling	
You have chosen to keep a total in another field.	
Do you want to:	
1 - Add the value to another field	
2 - Subtract the value from another field	
Selection: 0	
Field 5 of panel 3 Field Format: GZZZ9.99	

When we choose **1** (Add the value to another field), DataPerfect presents us with the current panel and waits for us to **Tab** to the link we want to use to target, say, the Invoice Panel. After hitting **Down Arrow** on the desired link, DataPerfect presents us with the Invoice Panel, where we must **Tab** to the target numerical field and hit **F4** to select it. That's all there is to attaching a Keep A Total option to a numerical field. Now when a Transaction Panel record is saved, DataPerfect will see if the value in the Charge field changed. If it did, DataPerfect will adjust the total in the Charges field in the Invoice Panel appropriately.

We repeat the above procedure on the Charge field in the Transaction Panel two more times, once to create a Keep A Total that targets the Balance field in the Invoice Panel (using the same link we did above), and once to create a Keep A Total that targets the Balance field in the Customer Panel (using a different link—one that targets the parent record in the Customer Panel).

After you create these three Keep A Total codes for the Charge field in the Transaction Panel, your Field Options screen (**Shift-F8**) for that field should have something like this at the bottom:

```
The value in this field is added to field 6 of panel 2 through link 7
The value in this field is added to field 4 of panel 2 through link 7
The value in this field is added to field 4 of panel 1 through link 8
```

The wording of these three lines is straightforward, except, perhaps, for the references to numbered links. If the link is a data link, the number refers to the field to which it's attached. If the link is a panel link, the number refers to that link's field number. Don't forget that a panel link is a field, so it gets a field number.

This accomplishes all I need for the Charge field to carry its value from the Transaction Panel to the Charges and Balance fields in the Invoice Panel, and the Balance field in the Customer Panel. DataPerfect will also adjust those target values

in the Invoice Panel and the Customer Panel when the Charge field in the Transaction Panel is edited, or a record in the Transaction Panel is deleted.

To setup the Keep A Total codes on the Payment field in the Transaction Panel, you take similar steps. You'll have its value added to the Payments field in the Invoice Panel instead of the Charges field, and you'll have its value subtracted from the Balance field in the Invoice Panel and the Customer Panel instead of added. Otherwise, the methodology here is the same as with the Charges field.

When There's No Parent Record

Suppose you enter the Transaction Panel directly from the Panel List, not knowing if there's a corresponding parent record in the Invoice Panel and Customer Panel to receive totals generated by the Charge and Payment fields in the Transaction Panel. That is, in Create mode in the Transaction Panel you enter values in all these fields (panel links to the current record's parent records in the Invoice Panel and Customer Panel are hidden):

TRANSACTION PANEL			
ID	Last Name	First Name	
Inv #	Description	Charge	Payment

Now you save the record. If there's no Invoice Panel and Customer Panel parent records for this Transaction Panel record, where will the Keep A Total codes deposit their values?

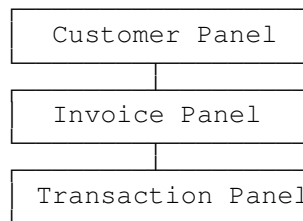
DataPerfect always makes sure there's a target for a Keep A Total operation. So in the above example, DataPerfect makes sure there's a Invoice Panel record with the proper Invoice Number and a Customer Panel record with the proper Customer ID Number. If there isn't, DataPerfect creates what's needed in those two higher panels before completing the Keep A Total operations. That is, when you finish saving the record in the Transaction Panel, exit to the Panel List, and then enter each of the top two panels from the Panel List, you'll find the appropriate Customer Panel and Invoice Panel records there, even though you didn't create them before that last Transaction Panel record creation. The records created in the two upper panels in this situation will have values only in fields matching those in the field list of the link used by the Keep A Total. No other fields will be filled in.

Notes on Target Panels for Keep A Total Operations

Please read *Issues Concerning Totaling* in the *Choosing Between ::C and ::N Fields* section of my **Fields: Issues** chapter for special considerations in designing the target panel of Keep A Total operations.

Panel Hierarchies and Keep A Total: a Caveat

Let's return to the following panel hierarchy:



Note that in the previous discussion of this panel hierarchy, I placed multiple Keep A Total codes on each relevant field in the Transaction Panel (Charge and Payment fields). Some of these Keep A Total codes targeted the Invoice Panel, and some the Customer Panel.

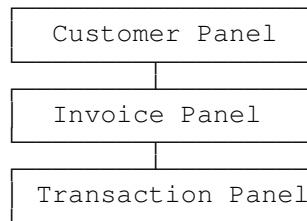
Now, I could just as easily have created Keep A Total codes in the Transaction Panel that target only the Invoice Panel, and then place Keep A Total codes on relevant fields in the Invoice Panel, making them target the Customer Panel. I call this *piggybacking* Keep A Total operations. The first set of Keep A Total codes (in the Transaction Panel) update the Invoice Panel. After that Keep A Total operation completes, the second set of Keep A Total codes (in the Invoice Panel) update the Customer Panel.

In earlier versions of DataPerfect, I found piggybacking Keep A Total operations unreliable. This may not be the case with recent versions of DataPerfect. I haven't tested it for a long time. I have, however, heard users complaining that, in some of their applications, it didn't update the higher panels, even with DataPerfect 2.3. I suggest you refrain from piggybacking Keep A Total operations, opting to always place a Keep A Total code in the lowest possible panel, even if that means placing multiple Keep A Total codes on a single field. In general, I consider this rule to ease the strain on a relational database.

Using Keep A Total to Update Records in Foreign Panels

Once you realize that the Keep A Total code updates records in foreign panels (panels other than the current panel), it's tempting, and sometimes wise, to use this facility solely for the purpose of triggering foreign panel records to update when the user creates a record in the current panel, or to actually *create* foreign panel records

when a user creates a record in the current panel. Consider the following scenario, with the following familiar panel hierarchy:



When a new Customer makes their first purchase in your store, you'd like this application to automatically create a record in a fourth panel that serves as a reminder to send a Thank You letter to that new Customer thirty days after their initial purchase. You plan on creating a report that prints these Thank You letters, recording the print date in a special date field.

Such a Thank You Panel might look like this:

THANK YOU PANEL		
ID	Last Name	First Name
Initial Purchase:		
Thirty Days Later:		
Letter Generated:		

In the above panel, if the Initial Purchase field is P4F4, then the Thirty Days Later field would have a field formula that updates on any change to

P4F4 + 30

The Indexes

Now we create the following three indexes for this new panel:

- (1) ID
- (2) Thirty Days Later, Last Name, First Name, ID
- (3) ID, Last Name, First Name, Initial Purchase

The first index guarantees we never have more than one record per Customer. The second is for our future report (I won't bother constructing the report here). The third is what we'll use to allow our Keep A Total to work.

The Keep A Total

Now we need to create the Keep A Total in the Transaction Panel that forces creation of the proper record in the new Thank You Panel for each Customer on their first day's purchase. Before we do that, we need an additional field in the Transaction Panel. It's a G9::H field that has a field formula that updates to 1 if there's no record for this Customer in the Thank You Panel; otherwise it updates to 0. That field will be the one that has the special Keep A Total that totals to the Thank You Panel. We can make the Keep A Total target the Thirty Days Later field because that field will update to thirty days after the Initial Purchase field no matter what, even if a Keep A Total value drops into it. Don't forget that a Keep A Total needs to target a numerical field of some kind. This one will do nicely, even though it's not really storing any total at all. We're simply using it as a dummy receptacle to allow the Keep A Total to cause the Thank You Panel record to be created.

The Formula on the Transaction Panel G9::H Field

The field formula for this G9::H field uses a hidden link that targets the Thank You Panel. This hidden link uses the first index noted above and, like the index itself, it has only the ID field on its field list. The field formula will update on any change to 1 if it sees a record on the other side of this link, otherwise it remains 0.

The Keep A Total on the Transaction Panel G9::H Field

The Keep A Total on this G9::H field will use a different hidden link than its field formula uses. That link would also be hidden. It targets the Thank You Panel. It uses the third index we listed for the Thank You Panel. This link will have a field list matching all the above fields (the Initial Purchase field above, of course, will be matched by the Transaction Panel's Date field).

How it Works: An Example

Let's outline what the above scheme does with a real example. The user creates a record for Sally Adams in the Transaction Panel. The G9::H field formula looks on the other side of the special link we created that uses Thank You Panel's index 1. Two possibilities arise here. DataPerfect takes only one of them:

- If the G9::H field formula sees *no* record in the Thank You Panel (i.e., it sees no record there with Sally Adams' ID Number), it updates to 1. Since the field value goes from 0 to 1, its Keep A Total code detects a change in the field's value. The Keep A Total facility then attempts to add 1 to Sally Adams' record in the Thank You Panel. Since none exists, DataPerfect creates one automatically. In virtue of the link used by this Keep A Total to create a Thank You Panel record for Sally Adams, that record's ID, Last Name, First Name, and Initial Purchase fields are filled in. While creating this record, DataPerfect adds 1 to its Thirty Days Later field. This event is only temporary, however, as DataPerfect overrides that with that field's field formula. Since that field formula updates on any change, it

overrides the adding of 1 to its value. That formula updates the Thirty Days Later date field to thirty days after the date it sees in the Initial Purchase field in that record.

- If the G9::H field formula *does* see a record in the Thank You Panel (i.e., it sees a record there with Sally Adams' ID Number), it updates to 0. Since the field goes from 0 to 0, its Keep A Total code detects *no* change in the field's value. So the Keep A Total facility refrains from attempting addition to the Thank You Panel. This results in DataPerfect leaving the Thank You Panel record for Sally Adams alone.

General Principles

Here are the general principles for using a Keep A Total to update a foreign panel a single time on record creation in the current panel:

- Create a G9::H field in the current panel that updates to 1 when the foreign panel needs updating, otherwise it updates to 0.
- Put a Keep A Total on that G9::H field, and have that Keep A Total target a numerical field in the foreign panel that needs updating.
- If available, make the target of the Keep A Total a numerical field that has a field formula that updates on any change. This way you don't need a field in the foreign panel to hold an ever increasing total. The target field's field formula will override any addition made to that field by the Keep A Total. Otherwise, you can just create a G9::H field in the foreign panel that updates to 0 on any change. That, too, will override the Keep A Total addition. Or, if you really can use a running total in the foreign panel, create a field there to hold the ever increasing total, and let that be the receptacle for the Keep A Total operation.

If, instead of wanting a Keep A Total to update a foreign panel a single time on record creation in the current panel, you want it to update that record in the foreign panel *every* time the current panel's record changes, you'd alter the principles slightly:

- In the current panel, create a G9999999999::H field. This must be a ten-character field.
- Create a formula on that G9999999999::H that updates on any change to

$$(86400 * \text{today}) + \text{now}$$

which is the number of seconds since March 1, 1900, otherwise called the *Moment* in this book (for a discussion of the concept of *Moment*, see *An Alternative Solution: Using the Concepts of MOMENT and MODULO* in the **Fields: Issues** chapter).

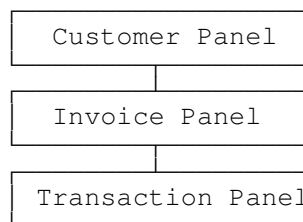
- Put a Keep A Total on that G9999999999::H field, and have that Keep A Total target a numerical field in the foreign panel that needs updating.
- If available, make the target of the Keep A Total a numerical field that has a field formula that updates on any change. Otherwise, create a G9::H field in the foreign panel that updates to 0 on any change.

The above scenario will trigger a Keep A Total to the foreign panel every time the user saves a created *or edited* record in the current panel, because hitting **F9** or **F6** will each cause the G9999999999::H field to increase in value.

Keep A Total vs. Cascade Update

Perhaps it's obvious, but I need to make this clear right now. A Keep A Total only updates a single record in its target panel. What record it updates depends on the link it's using. Whatever record you land on when penetrating that link in Browse mode is the record the Keep A Total will update. If no record exists on the other side of the link during Browse mode, then the Keep A Total operation will create whatever record you'd create by penetrating that link with **F5**.

So, given the above, you won't go far attempting to use Keep A Total to update more than one record on the other side of a link. For instance, consider the following familiar panel hierarchy:



Suppose you want Last Name and First Name fields in all three panels, and you want all of them to be real fields (i.e., not computed fields). Further, you want any change made to one of the two name fields in the Customer Panel to be reflected down the panel hierarchy, updating those fields in subrecords for that Customer Panel record. That is, when Sally marries, and the data entry person changes *Sally Adams* to *Sally Jones* in the Customer Panel, you want *Sally Adams* to change to *Sally Jones* in all and only Sally's records in the Invoice Panel and Transaction Panel.

Keep A Total won't help here. Suppose I create a G9999999999::H field in the Customer Panel, formulating it to update on any change to

`(86400*today)+now`

I then put two Keep A Total codes on this field. One targets the Invoice Panel, the other the Transaction Panel. Each targets a numerical field that itself updates on any change to some value, overriding the addition Keep A Total event. I'm following the general principles I outlined above.

Further, I make sure the Last Name and First Name fields in the Invoice Panel and Transaction Panel each are formulated to update on any change to the values they see in the Last Name and First Name fields in the Customer Panel. These field formulas would use links (perhaps hidden) that go to the parent record in the Customer Panel. This way, any time an Invoice Panel or Transaction Panel record is placed in Edit mode, which happens every time a Keep A Total sends a value to it, the Last Name and First Name fields will update to values found in the Customer Panel.

Well, such a scheme will only cause an update to occur in the single record the Keep A Total sees on the other side of the link. That's all. So only one Invoice Panel record and one Transaction Panel record will update each time, not *all* of Sally's subrecords. This will most likely update Sally's first Invoice Panel record, and her first Transaction Panel record (or perhaps the last of each, depending on how you index the links that take you from the Customer Panel to the Invoice Panel and Transaction Panel).

There are basically two ways to get *real* fields (i.e., not computed fields) in the subpanels to update to changes made in fields higher in the panel hierarchy. One is to construct a report that does the job for you, perhaps to be run at the end of each week. This is time consuming, so I don't recommend it. The other is to use Cascade Update on a panel link that contains the fields to update on its field list. In any event, it wasn't the point of this section to discuss Cascade Update in as much as it was to point out a common mistake developers commit with Keep A Total codes. I discuss Cascade Update in the *Keeping Subpanel Data Current* section of my **Fields: Issues** chapter.

Though I mainly target the beginner here, experienced DataPerfect application developers should still peruse this chapter.

Introduction

DataPerfect's report facility is one of its most powerful features. Unfortunately, with this power comes complexity that easily discourages the beginning DataPerfect application developer.

We usually think of a database report as a hardcopy form of data gathered from a database and sent to a printer, like a Day Sheet outlining all activity in an auto parts store, or a receipt handed to a patient for insurance reimbursement. But besides reports that send data to a printer, DataPerfect also allows the developer to create reports that send the same data to a disk file (perhaps to be printed later, or for electronic archiving purposes), as well as reports that send data to the screen. Actually, with DataPerfect, you can have a report do any one, two, or three of these three possibilities as well. But there's one more thing a developer can do with the DataPerfect report facility: it can be used to define routines that manipulate data in the database, like purge records in particular data files, or update particular fields in particular records. This latter option is usually reserved for a separate facility in a database program—a facility that allows the user to run routines from a menu. In DataPerfect, the report facility is used for this.

So, DataPerfect's report facility offers the developer a way to do one, two, three, or all four of the following in a single report:

1. Gather data from a database and send it to a printer.
2. Gather data from a database and send it to a disk file in either DOS Text or WordPerfect format
3. Gather data from a database and send it to the screen.
4. Manipulate data in a database in unattended fashion.

As I mentioned, sometimes these options are combined in the same report, or in a pair of reports that are typically run one right after the other. For instance, at the end of each month of practice, I load my chiropractic billing application and run an Aging of Accounts report to update the aging fields in every record in the Case Panel. I've defined it to show me on the screen what's happening at each stage of its work, so it combines features 3 and 4 above. When that's done, I then run my Monthly Statements report, which sends data to the printer and creates a Transaction Panel record for each patient, recording the date and amount of the billing for that month for that patient. So the Monthly Statements report uses features 1 and 4 above. If I wanted, I could have the Monthly Statements report simultaneously create a disk file

of all the output that was sent to the printer (feature 2), to be printed later, or for archiving purposes.

So that's typical of what the DataPerfect report facility can do. How do we actually create reports with it?

The Initial Report Definition Screen

First, from Browse mode in a panel display, hit **Shift-F7** to call the Report List. If no reports have been created in this application, you'll see only one item on that list: *Built-In Short Reports*. That report serves as a template for creating a new report, and can't be deleted. Any previously created report on the list can be used as a template as well. In such a case, you can just highlight the old report and hit **Insert**. This gives you a copy of the old report to work on. But let's go back to Built-In Short Reports, which is what you use when you want to create a report when none exist on the Report List, or when you want to create one that isn't based on an existing report.

To create your first report, back out of the Report List with **F7** and load the panel on which to base the report. If that's the panel you just landed on when backing out of the Report List, you can go back to the Report List with **Shift-F7**. If that's not the panel on which you want to base this new report, hit **F7** and pick one from the Panel List by highlighting it and hitting **Enter**. Now go back to the Report List with **Shift-F7**. The panel you previously loaded will be the panel on which you're about to base this new report. By saying a report is based on, say, the Invoice Panel, we're saying that that report is working in the world of Invoice Panel records. That is, when that report starts running, it sees nothing but Invoice Panel records, and if you don't add certain other features to its Report Definition—like a subreport or two (more on this later)—that report will never see anything but Invoice Panel records.

So, to start creating a report based on the currently loaded panel, highlight *Built-In Short Reports* in the Report List and hit **Insert**. This will bring you into what I call the Initial Report Definition Screen (DataPerfect doesn't call this screen anything in particular, so I'll label it this way in this book). You should see the following:

REPORT: New Report				
Destination:		Screen Only		
1 - Printer On/Off				
2 - Disk File On/Off				
3 - Index Number		1		
4 - Search Conditions		No Search		
5 - Sort Direction		Forward		
6 - Disk File Mode WP/DOS		No Disk File		
7 - Print Margins		Top	Bottom	Left
		6	0	0
8 - Edit Report Form		Text Lines		
		54		
9 - Edit Report Name				
Selection: (Press Shift-F7 to begin the report) 0				

Let's go over the various options you see above.

Report Name

Notice the first line above. The name of the report is *New Report*. That's the first thing I change. Hit **9** (Edit Report Name) to give your new report a better name. You can change your mind later, so just give it any old name now.

Options 1 and 2: Destination

For now, leave option 1 set for *Screen Only*. This way you don't waste paper while trying to figure out why reports aren't working. Even the most seasoned DataPerfect expert can waste tons of paper experimenting with a new Report Definition that isn't working just right. It's better to see the report at least looks somewhat right on the screen before committing to printing it. Also, before I use Option 1 to tell DataPerfect to send the report the printer, I frequently use Option 2 and view the subsequent output with a file viewer. I hate wasting paper.

By the way, using Options 1 or 2 doesn't result in the report suddenly being sent to the printer or to disk. These are just settings that determine where the report output will go when you later decide to run the report.

If you fiddled with Options 1 and 2 already, you may have noticed you can actually get the Destination to be both Printer *and* Disk File, like so:

```
Destination:    Printer LPT1    Create Disk File
```

That's not a glitch. This will really send the data to *both* the printer and a disk file. To eliminate one or the other, hit the appropriate Option number again to toggle it off. If you didn't get that double output configuration, see if you can get it by tapping on **1** and **2** a few times, and then return the Destination value back to *Screen Only*.

Option 6: Disk File Mode

With Destination set for *Screen Only*, Option 6 should look like this:

```
6 - Disk File Mode WP/DOS      No Disk File
```

That makes sense because you're not directing output to the disk. Now do the following:

- Hit **2** to direct output to a disk file.
- Hit **1** to create a new file.
- Type *TEST.RPT* for the file's name and hit **Enter**.

Three changes should now appear in the Initial Report Definition screen:

- *Create Disk File* appears in the Destination area.
- A new line displays just under Option 2, showing the output filename.
- Option 6 (Disk File Mode) now says either *DOS Text* or *WordPerfect* instead of *No Disk File*.

Hit **6** to see it toggle between *DOS Text* and *WordPerfect*. Now go back and hit **2** to toggle Disk File off again. This reverses the three changes outlined above.

Option 7: Print Margins

This is the only remaining Option on the Initial Report Definition Screen that doesn't have much to do with the panel you were in when you first started this process. Its default values read like this:

```
7 - Print Margins      Top      Bottom      Left      Text Lines
                       6        0          0        54
```

Here's how the above parameters work. DataPerfect thinks in terms of the number of lines on a page. Unlike its big sister, WordPerfect 5.x and above, it doesn't understand inches. DataPerfect works more like WordPerfect 4.x in this regard. So you first have to consider the paper you want to use for this or that report. Assuming it's letter size, you're talking about eleven inches in length. At 6 lines per inch, that's 66 lines per page.

Second, you have to consider the type of printer you're going to use. If given the chance, a dot matrix printer will print on all 66 lines. Not true for a laser, which will print on no more than 60 lines per page by default (still assuming 6 lines per inch). Lasers have a half-inch *No Print* zone on all four sides.

Let's take a look at the four fields in the Print Margins line in our initial menu. The first three are fairly straightforward. They represent margins at the top, bottom and left of each page. The top and bottom margins are in terms of lines, and the left margin is in terms of columns or characters (we're talking about monospaced fonts here, like Courier, not proportional fonts, like CG Times). The fourth field (Text

Lines) represents the total number of lines DataPerfect will count before automatically issuing a page eject.

The default values for these four fields result in the following:

- DataPerfect will start each page one inch from the top (Top = 6).
- DataPerfect will consider the first column to be on the far left of the paper (Left = 0).
- DataPerfect will issue a page eject after counting 54 lines (Text Lines = 54).

The above yields a flush-left report that prints with one-inch margins (6 lines), top and bottom. Note that the Text Lines value makes the Bottom value of 0 irrelevant here. If the paper is ejected after printing 54 lines, which followed the 6-line margin at the top, it's essentially ejecting after 60 lines, leaving a margin of 6 lines at the bottom. For a typical report, I always leave the Bottom value at 0, and determine my bottom margin by the combination of the Top value and the Text Lines value.

The rule here is that

$$\text{Top} + \text{Text Lines} + \text{Bottom}$$

must not exceed the printable zone of your paper. For a dot matrix, this total must not exceed 66, and for a laser, 60. If you always accept the default

$$6 + 54 + 0$$

for paper length parameters, then your report will work with letter-size paper on both a dot matrix and a laser. Alternatively, if you like to print reports that use more than 60 lines per letter-size page, then you'll have to modify the report when using a laser.

Left Margin

I didn't talk much about the Left margin. Let's do that now. Note that there isn't any Right margin field. DataPerfect assumes you'll be determining right margins within the guts of the report itself (8 - Edit Report Form, which we have yet to cover), by using carriage returns at the end of each line. So it just needs to know where to start each line. Or better, it just needs to know where on the physical page to consider column one to be. A value of zero here tells DataPerfect to consider column one to be the far left of the paper (as far left as the printer allows).

I usually don't accept the default for this value. Many of my reports, like Daily Activity reports and Monthly Performance reports, are going to end up in three-ring binders. For such reports, I usually increase this value to at least three, leaving room for the binder holes. A value of three for Left margin means that DataPerfect will consider column one to be three characters to the right of the leftmost printable column on the physical page.

The Left margin field is deceptively convenient. It frees you from having to determine how many columns to the right to start each line in the Edit Report Form screen itself (item 8, which, again, we discuss later). Just go ahead and consider the

first column to be the far left of the Edit Report Form screen and print out the report. If it's too far to the left, adjust only one figure: the Left margin in the Initial Report Definition Screen. This saves you having to adjust each line of the Edit Report Form screen itself.

Labels

With respect to a report that generates labels, item 7 is handled a little differently. Here the only field you care about is the Left margin field. Set all the others to zero. After testing the report by printing out a few labels, see if you have to adjust the Left margin.

Finding What Panel a Report Is Based On

We've covered all items on the Initial Report Definition Screen that don't depend on the panel that was displayed when you started this process. As mentioned earlier, that's the panel the report is *based on*. To see what panel a report is based on, take a look at the Report List (where you were after you first hit **Shift-F7**, but just before the Initial Report Definition Screen). To the far right of any given report name, you should see the panel number (P1 for Panel 1), like so:

Monthly Statements

P1

There's another way to see what panel the report is based on, especially if you really don't remember exactly what, say, Panel 1 is. Hit **8** (Edit Report Form) and, with your cursor still in the section called First Page Header, hit **F4**. This will take you to the panel that report is based on. In any given section of a Edit Report Form screen, this is how to find out what panel that section is based on. This becomes more important when defining subreports within the report, and you're wondering where you are in this or that section. When done looking, **F7** to return to the Edit Report Form screen. **F7** again to return to the Initial Report Definition screen.

Panel-Dependent Initial Report Definition Screen Options

There are four more options on our Initial Report Definition Screen, all of which relate to the panel on which the report is based:

- 3 - Index Number
- 4 - Search Conditions
- 5 - Sort Direction
- 8 - Edit Report Form

The first three can be covered pretty quickly, reserving the bulk of our work here to explain Edit Report Form.

Option 3: Index Number

This defaults to the lowest numbered index for that panel (that's not always Index 1 because you may have deleted Index 1). This index is from the panel the report is based on. To take a look at just what that index is, simply hit **3**. From there, you may select a different index by using **Up Arrow** or **Down Arrow**, hitting **F4** to select it.

If, after browsing the indexes, you decide you should have based this report on a different *panel*, you'll have to start the Report Definition process over again. Though DataPerfect allows you to change the index for an existing Report Definition, it doesn't allow you to change the panel on which you based the Report Definition. To change the panel on which to base the report, exit the Report Definition with **F7** enough times to return to the Panel List. Load the panel on which you want to base the report, hit **Shift-F7**, and then, with the highlight bar on Built-In Short Reports, hit the **Insert** key.

Your choice of index here is crucial. It determines the order the report will process that panel's records. It may even exclude an entire class of records without you realizing it, if you inadvertently select an index with an Exception List attached to it. If you see an *E* attached to a field in the Index view after hitting **3**, then that index has an Exception List attached to it. You probably don't even know what an Exception List is at this stage, so don't worry about this. The upshot here is that the report index is an important setting, not to be taken lightly. See *Exception Lists* in my **Indexes** chapter for more on this.

Option 4: Search Conditions

This is used to narrow the scope of a report. It works just like **F2** works when a panel displays in Browse Mode, allowing you to define the report to search based on formula, template, etc. I can safely say I *never* use this option. Never. *No Search* always displays to the right of Search Conditions on my Initial Report Definition Screen. I suggest using a combination of Prompts and Iteration Control to narrow down the scope of a report. The Search Conditions option is cumbersome, unintuitive, and easy to forget it's set when you run a report a few months later. See *Prompt for Report Variable* in the *Global Report Options* section of my **Report Options** chapter for a discussion of prompts, and my **Iteration Control** chapter for a discussion of Iteration Control.

Option 5: Sort Direction

This is straightforward. It determines whether the report processes records forwards or backwards, based on the index selected in Item 3.

Option 8: Edit Report Form

This is the guts of your Report Definition. I said DataPerfect's report facility is one of its most powerful facilities. This is where you access that power. When you choose Edit Report Form, you see this:

```

-----Column 1-----
Type the text to be included in the report. To include a data field,
press Select (F4), move to the field (possibly through links,) and again
press Select. While the cursor is on a report field mark, you can press
F6 to edit the report format. To include variable fields, prompts or
special control instructions, press Report Options (Ctrl-F7).

-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
--Empty--
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
```

Your cursor initially sits in the First Page Header, as seen above.

The DataPerfect report facility is, like the report facilities in other database management systems, a *band* report facility. That is, the different sections or groupings of the report (First Page Header, Other Page Header, etc.) are displayed as banded sections of the screen. Let's go over the top help box you see there, one sentence at a time:

```

-----Column 1-----
Type the text to be included in the report. To include a data field,
press Select (F4), move to the field (possibly through links,) and again
press Select. While the cursor is on a report field mark, you can press
F6 to edit the report format. To include variable fields, prompts or
special control instructions, press Report Options (Ctrl-F7).
```

Type the text to be included in the report.

It means just that. In any section of the Edit Report Form screen, you can just start typing. What you type will, when the report runs, either display on the screen or be sent to a printer or disk file, depending on other things you may have put in that Edit Report Form screen.

To include a data field, press Select (F4), move to the field (possibly through links) and again press Select.

Go ahead and hit **F4** and notice that DataPerfect throws you back into the panel on which you based this report. At this point, you can just hit **F7** to get back to the Edit Report Form screen (as I mentioned before, when I'm not sure what panel I based the

report on, I use **F4** to see, and then **F7** to return to the Edit Report Form screen). Alternatively, before returning to the Edit Report Form screen with **F7**, you can select a field with **F4**. Go ahead and do that. Note that DataPerfect throws you back into the Edit Report Form screen, but with an important difference. Now there's a copy of the field you selected from the panel, showing up in the section of the Edit Report Form screen you were in when you hit **F4** initially.

If your cursor was in the First Page Header when you hit initially **F4**, your screen should now look something like this after selecting a field:

Column 16

Type the text to be included in the report. To include a data field, press Select (F4), move to the field (possibly through links,) and again press Select. While the cursor is on a report field mark, you can press F6 to edit the report format. To include variable fields, prompts or special control instructions, press Report Options (Ctrl-F7).

FIRST PAGE HEADER

OTHER PAGE HEADER

--Empty--

TWO-LEVEL REPORT HEADER

--Empty--

REPORT BODY

--Empty--

TWO-LEVEL FOOTER

--Empty--

PAGE FOOTER

--Empty--

FINAL FOOTER

--Empty--

If the field you selected was, say, an A15 field, then the cursor would now be sitting in column 16, as noted at the top of the screen.

Move the cursor to the left, so it sits on the selected field in the Edit Report Form screen. Note the screen changed to show you information about that selected field:

Column 1

Type the text to be included in the report. To include a data field, press Select (F4), move to the field (possibly through links,) and again press Select. While the cursor is on a report field mark, you can press F6 to edit the report format. To include variable fields, prompts or special control instructions, press Report Options (Ctrl-F7).

Path to field: P1F1
Last Name

Field Format: A15

FIRST PAGE HEADER

OTHER PAGE HEADER

--Empty--

DataPerfect shows you where the field comes from (*P1F1* means *Panel 1 Field 1*), what name (if any) you assigned it in that panel, and what format it has in the Edit Report Form screen.

Note that the sentence I cited here has a parenthetical *possibly through links*. If you don't know what links are yet, this won't make much sense to you. If you do, suffice it to say here that DataPerfect allows you to select a field in a linked panel if

you hit **F4** in the Edit Report Form screen, and then **Tab** to the link in question and hit **Down Arrow**. At that point, you then **Tab** to the linked panel's field you desire and hit **F4**.

*While the cursor is on a report field mark,
you can press F6 to edit the report format.*

This is pretty clear, though it probably should have said *to edit the report field format*. Hit **F6** while the cursor is on that field. You now can edit the format of that field. If you do, this will only affect how that field's data is sent to the screen, printer, or disk file when running this report. It doesn't affect any other reports, nor does it affect the format of that field in the panel from which it was selected. For instance, you might want to alter the field's format in a report to get it to fit on the paper, or to convert it to all upper case characters for address labels.

*To include variable fields, prompts
or special control instructions,
press Report Options (Ctrl-F7).*

This can be complicated. What you get when you hit **Ctrl-F7** is different in each section of the Edit Report Form screen. That is, **Ctrl-F7** gives you a different menu of options in the First Page Header, versus, say, what it gives you in the Report Body or Final Footer. When I spoke of the power of DataPerfect's report facility, I was really referring to the Report Options you access via **Ctrl-F7** in the various sections of a Edit Report Form screen. I could write a book just on this. But before I get into the **Ctrl-F7** Report Options in each section, let's discuss something more basic: the *Edit Report Form Screen Sections*.

The Edit Report Form Screen Sections Delineated

By *Edit Report Form Screen Sections*, I mean the First Page Header, Other Page Header, Two-Level Report Header, Report Body, Two Level-Report Footer, Page Footer, and Final Footer.

First Page Header

As long as there are records to process by the Report Body, DataPerfect processes the First Page Header first, and processes it only once per report. So when a report is run, DataPerfect first takes a look to see if there are any records for the Report Body to process. Why might DataPerfect see no records to process by the Report Body? Well, here are the most common reasons for DataPerfect to stop the report cold because it fails to see records for the Report Body to process (in the following list, *the panel* refers to the panel on which the report is based):

- There are no records in the panel.
- The report's index has an Exception List that excludes all records in the panel.
- The report's Search Conditions exclude all records in the panel.
- Iteration Control options used in the report exclude all records in the panel.

If any of the above are true, *no* section of the Report Definition will run because all sections of the Report Definition are merely appendages to the Report Body. And the Report Body runs on records.

The First Page Header would be whatever you want at the top of the first page of your report. It shows up nowhere else, no matter how many pages are printed. So, if you're defining a report that prints out all teachers and their schedules for this academic year, you might put

FALLBROOK HIGH
Teachers and Schedules

in that report's First Page Header.

Other Page Header

As long as there are records to process by the Report Body, DataPerfect processes the Other Page Header at the beginning of each page, beginning with the second page. So, unlike the First Page Header, which prints only once per report, the Other Page Header prints at the top of every page of the report except for the first page. There is, however, a way to make it print on the first page as well. We'll discuss that later.

So, in the same report mentioned above, you might consider putting the following in its Other Page Header:

Teachers and Schedules (cont'd.)

The Other Page Header is also a typical place to put a Page Number field, which I talk about later.

Two-Level Report Header

As long as there are records to process by the Report Body, DataPerfect processes the Two-Level Report Header once per Two-Level Report group. This only happens if you define your report to be a Two-Level Report. We'll talk more later on how to do this, but suffice it to say that the place you tell DataPerfect that this is a Two-Level Report is the First Page Header.

So what's a Two-Level Report? Well, suppose our report is running on the Class Panel, which has records that show the name and time of each class, as well as the teacher assigned to it. It has one record per class. Now, suppose we'd like this report to show all the classes grouped together by teacher.

If we run this report on an index that sorts by Teacher and Class, in that order, we're likely to produce a report that prints like this:

Alexander, Joseph	Art 1	TuTh 10
Alexander, Joseph	Art 2	MWF 9
Alexander, Joseph	Art 3	MWF 11
Donaldson, James	Math 2	MWF 10
Donaldson, James	Math 5	TuTh 10
Green, Sally	History 1	MWF 9
Green, Sally	History 2	MWF 10
Green, Sally	History 3	TuTh 9

But this is more readable:

Alexander, Joseph	Art 1	TuTh 10
	Art 2	MWF 9
	Art 3	MWF 11
Donaldson, James	Math 2	MWF 10
	Math 5	TuTh 10
Green, Sally	History 1	MWF 9
	History 2	MWF 10
	History 3	TuTh 9

The latter is a Two-Level Report. Its Two-Level Report Header is the space occupied by the teacher's name in each grouping. That's the header of each group, as opposed to the header of the report (First Page Header) or the header of each page (Other Page Header). More on the details, tricks and traps of Two-Level Reports later.

Report Body

DataPerfect processes the Report Body once per record. So, each line of the following represents one pass of the Report Body:

Alexander, Joseph	Art 1	TuTh 10
Alexander, Joseph	Art 2	MWF 9
Alexander, Joseph	Art 3	MWF 11
Donaldson, James	Math 2	MWF 10
Donaldson, James	Math 5	TuTh 10
Green, Sally	History 1	MWF 9
Green, Sally	History 2	MWF 10
Green, Sally	History 3	TuTh 9

An Edit Report Form screen that would produce a report like you see above would look something like this:

```

-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

In the above Edit Report Form screen, the fields you see in the Report Body were placed there by hitting **F4** (which places me in the panel on which the report is based), **Tabbing** to the Teacher Name field, and hitting **F4** again. This places the Teacher Name field in the Report Body, represented by the horizontal block you see above. The same procedure was used for the Class Name and Class Time fields.

I might want to put in some text to make the report more readable:

FIRST PAGE HEADER		
FALLBROOK HIGH		
Teachers and Schedules		
Teacher	Class	Time
=====		
OTHER PAGE HEADER		
Schedules (cont'd.)		
Teacher	Class	Time
=====		
TWO-LEVEL REPORT HEADER		
--Empty--		
REPORT BODY		
=====		
TWO-LEVEL FOOTER		
--Empty--		
PAGE FOOTER		
--Empty--		
FINAL FOOTER		
--Empty--		

This would produce a report like this (I'll put a ruler above the printout for later reference, when I start playing with the Left Margin setting—it's not part of the report printout):

1		2		3		4		5		6		7	
01234567890123456789012345678901234567890123456789012345678901234567890123456789													
FALLBROOK HIGH													
Teachers and Schedules													
Teacher				Class				Time					
=====													
Alexander, Joseph				Art 1				TuTh 10					
Alexander, Joseph				Art 2				MWF 9					
Alexander, Joseph				Art 3				MWF 11					
Donaldson, James				Math 2				MWF 10					
Donaldson, James				Math 5				TuTh 10					
Green, Sally				History 1				MWF 9					
Green, Sally				History 2				MWF 10					
Green, Sally				History 3				TuTh 9					

And if there are enough records to produce more than one page, each page after the first page would have this as the top:

Schedules (cont'd.)		
Teacher	Class	Time
=====		

If we want to make sure the text is centered on the page instead of flush left, we would go the Initial Report Definition screen and set the Left Margin to accommodate that. This is where the ruler I put in above comes in handy for discussion. A setting of 13 for the Left Margin makes it print like this (again, the ruler isn't part of the report printout):

1 2 3 4 5 6 7
0123456789012345678901234567890123456789012345678901234567890123456789

FALLBROOK HIGH Teachers and Schedules		
Teacher	Class	Time
=====	=====	=====
Alexander, Joseph	Art 1	TuTh 10
Alexander, Joseph	Art 2	MWF 9
Alexander, Joseph	Art 3	MWF 11
Donaldson, James	Math 2	MWF 10
Donaldson, James	Math 5	TuTh 10
Green, Sally	History 1	MWF 9
Green, Sally	History 2	MWF 10
Green, Sally	History 3	TuTh 9

Note there isn't any setting in the Initial Report Definition Screen that centers a report on the printed page. For this, you must rely on calculating what the Left Margin should be to accomplish this. In the above example I counted the total characters across the Report Body. That's 52. Subtracting from 80 (the total number of characters across the page, using Courier font), we get 28. Half that would be the Left Margin (14).

Of course things are often not that simple. Inevitably you print the report out and look at it and then adjust the Left Margin accordingly. Also, you might want to accommodate another half inch or so for the holes needed for a three-ring binder. At 10 characters per inch (for Courier), that adds another 5 for the holes, bringing the Left Margin to about 19. Again, print out the report, punch the holes, examine, and change the Left Margin if needed.

Two-Level Report Footer

As long as there are records to process by the Report Body, DataPerfect processes the Two-Level Report Footer once per Two-Level Report group. Like the Two-Level Report Header, this only happens if you define your report to be a Two-Level Report (again, we talk about this in detail later).

Page Footer

DataPerfect prints a Page Footer at the end of each page.

Final Footer

DataPerfect prints the Final Footer only once per report: at the very end. Unless you put in a specific code that forces it to print at the bottom of the final page, it will simply begin printing after everything else in the report has printed, always starting on its own line.

The Report Algorithm

Okay, here's the process DataPerfect goes through, step by step, when running a report. First, DataPerfect sees if any records should be evaluated by the Report Body. There might not be any to process because there aren't any in the panel on which the report is based, or the report index has an Exception List that excludes all records in the panel on which the report is based. If there are *no* records to evaluate, the report aborts immediately, issuing the *No Data* message. If there *are* records for the Report

Body to evaluate, DataPerfect processes the various sections of the Edit Report Form screen in this order:

1. First Page Header is processed.
2. If there's an *Include Header Before Data* code (**Ctrl-F7, 3**) in Other Page Header, Other Page Header is processed; otherwise, Other Page Header is skipped.
3. What happens next depends on whether a Two-Level Report is defined in First Page Header.
 - a. If a Two-Level Report is *not* defined in First Page Header, Report Body is processed once per record.
 - b. If a Two-Level Report *is* defined in First Page Header, Two-Level Report Header is processed. Then Report Body is processed once for each record that belongs in the first group of records as determined by the Two-Level Report definition found in First Page Header. Then Two-Level Report Footer is processed. If another group of records exists (where such a group is determined by the Two-Level Report Definition in First Page Header), Two-Level Report Header is processed again, followed by Report Body once per record in that group of records, followed by Two-Level Report Footer again. This cycling ends when there are no records left to process by Report Body.
4. During all the above, whenever the report is near the end of a page, with just enough room to print Page Footer, Page Footer is processed, a page eject is issued, and Other Page Header is processed at the top of the next page. Then it continues where it left off in step 3.
5. At the end of the entire report, Final Footer is processed.

General Theory in Creating a Report

The primary reason you're going to create a report is to send data to the printer. Your main concerns are these:

- Processing all and only the right records, in the right order.
- Getting just the right information from each record to print.
- Getting the report to look good.
- Getting the report to complete in a reasonable amount of time.

Processing the Right Records

Here, you're best off going from the general to the specific. Choose an appropriate panel on which to base the report. Then choose an appropriate index. This will determine not only the right order, but may impact which records will be processed. If there's no Exception List attached to an index, it includes *all* records in the panel.

If there *is* an Exception List attached to an index, it may not include all records in the panel. You need to be aware of this when assigning an index to your report.

Narrow down the records to be processed with Search Conditions or Iteration Control Options. I devote an entire chapter to formulas (which is what you need to understand to construct Search Conditions), and an entire chapter to Iteration Control. You're almost always better off using Iteration Control options for this instead of Search Conditions.

Getting the Right Information to Print

The primary way you're going to do this is to use **F4** to select fields from the panel, placing them one at a time into the Edit Report Form screen. If needed, change the field format of a field *after* you place it in the Edit Report Form screen. You can do that by cursoring to the field in the Edit Report Form screen and hitting **F6**. This allows you to, say, shorten an A35 field to an A20 field, and, perhaps, change it so it prints all upper case (changing it to U20), all without changing the field's format in the panel itself.

Also note that if the panel on which this report is based has, say, an A15 Last Name field and an A15 First Name field, selecting them in the report with **F4** and typing a comma after the Last Name field, will result in a Edit Report Form screen like this:

```

-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
XXXXXXXXXX, XXXXXXXX
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

But that Edit Report Form screen prints like this:

```

Alexander      , Joseph
Donaldson     , James
Green         , Sally

```

Think about this a few minutes to see why this is so. To get rid of those spaces before each comma, you need to alter the field format a little. DataPerfect offers Print Mode Indicators to handle situations like this. The ones relevant here are those that strip off blanks at the ends of fields. Here are the three that do that:

```

;;B   Truncate Both Leading and Trailing Blanks
;;S   Suppress Leading Blanks
;;T   Truncate Trailing Blanks

```

The above are the mnemonics in the DataPerfect manual. I use different mnemonics to remember Print Mode Indicators:

```
;;B   Remove both  
;;S   Remove starting  
;;T   Remove trailing
```

Reformatting the first field in the Report Body from A15 to A15;;B or A15;;T will solve this problem and give us a report that prints like this:

```
Alexander, Joseph  
Donaldson, James  
Green, Sally
```

I didn't reformat the field in the *panel*. Rather, I did that in the Edit Report Form screen itself by cursoring to the field in question, hitting **F6**, and then altering the field format. Note also that Print Mode Indicators use double *semi-colons*, in contrast to the Display Mode Indicators (e.g., ::N), which use the double *colon*. I discuss all Print Mode Indicators in detail in *Print Mode Indicators* in my **Reports: Fields** chapter.

Getting the Report to Look Good

This can simply be an issue of deciding between things like Subreports and Two-Level Reports on the one hand, or as complex as coming up with complicated printer control strings that move the laser printer head up and down the paper, while the report prints with exotic proportional fonts. If your report requires a lot of proportional font work, DataPerfect's report facility is going to frustrate you when contrasted with the Windows database managers out there. It just doesn't have the formatting power these other have in this regard.

But if your report requires grabbing data from many linked data files, DataPerfect's report facility will offer you power and flexibility lacking in other database managers. So there's a trade-off here. DataPerfect's report facility's power rests with its data access flexibility, and, as you'll see later, its Iteration Control options. I'm not going to take up space on this topic. It's best served by your experimenting, first defining the proposed report with it set to print to the screen. Then, if it looks good, set it to print to a file. If that file looks good in your file viewer, then set it to print to the printer and start examining hard copy.

Getting the Report to Complete in a Reasonable Amount of Time

First, use the right index. And if available, use an index with an appropriate Exception List on it (see *Speeding Reports with Exception List Indexes* in the *Exception Lists* section of my **Indexes** chapter). This can make a big difference. If you use an index with an Exception List on it that limits records to only those you want for this report, great. For instance, in an Accounts Panel you might have a field showing Account Balance. If you want to make a report print Monthly Statements for all and only those Accounts with a positive balance, you don't want the report to have to search the entire Accounts Panel data file. If you have an index with an Exception

List on it that excludes all records without a positive balance, you'll greatly speed up this report.

When combined with the appropriate index, Iteration Control options can greatly speed reports. I devote an entire chapter to this topic (see the **Iteration Control** chapter). If you understand Iteration Control, and how to integrate it with appropriate indexes, you frequently can take a report that previously took a hour to finish on a large database, and turn it into a report that finishes in less than a minute.

Knowing Your Place

When constructing a Report Definition in the Edit Report Form screen, it's crucial you know where you are in your panel hierarchy, and what record your report would be working on or have access to if active at this moment. In any section of your Edit Report Form screen you can see what panel is being accessed by that section by hitting **F4**. Hitting **F7** then takes you back to where you were before taking a peek at the panel. But this doesn't tell you what record that report will be accessing at that moment. Let's explain.

In the Main Report

In the First Page Header of the main report, the report has access to the first record in the active index and no others. In fact, the main report's First Page Header is printed only once—at the beginning of the whole report—and, at that time, the first record in the active index (the index assigned to the report in the Initial Report Definition Screen) is the one up for processing. Selecting fields with **F4** in the main report's First Page Header will be selecting fields from that record.

The main report's Other Page Header is processed only when the report definition senses the end of a page while in the main report (as opposed to when it's in subreport). At that time, it issues a page eject and prints the Other Page Header at the beginning of the next page. The record the Other Page Header has access to at that moment will be the first record in the index processed on that new page. Selecting fields with **F4** in the main report's Other Page Header will be selecting fields from that record.

The main report's Report Body, of course, processes every record in the active index, barring certain Iteration Control options that may be active.

In a Subreport

But let's talk about subreports here. In the above statements, I was very careful to refer to the *main report's* First Page Header, Other Page Header and Report Body. Things get a little more complicated when trying to keep track of where you are in terms of what records are being processed in subreports. First, always remember that a subreport is based on a single record in its parent report. That isn't to say that it has access to only a single record. Rather, it lives in a world of records that are

determined by the parent record it just left in its parent report, consistent with the index and field list that creates the subreport.

For instance, if the main report is based on the Invoice Panel, and the subreport is based on the Transaction Panel, you're most likely going to define this report in such a way that the subreport is based on the Invoice Number of the record in the main report. This might be done by creating the subreport using a panel link in the Invoice Panel that takes you to the Transaction Panel, where this link has a field list consisting solely of the Invoice Number field in the Invoice Panel. The index for this panel link probably will be a Transaction Panel index with Invoice Number and Date as its first two fields, so that records accessed via that link will be in chronological order for each given Invoice.

In the above example, the subreport will, at any moment in time, have access only to a well-defined subset of the Transaction Panel records: those records that have the same Invoice Number as the parent record just left in the Invoice Panel in the main report. So, the *subreport's* First Page Header will have access to the first record it comes to in the subset of records tied to the current parent record. In our Invoice/Transaction example, that would be the earliest Transaction Panel record tied to the Invoice Panel record the report just left when entering the subreport.

Likewise, if the report comes to the end of a page *when in a subreport*, it will process the Other Page Header of that subreport and *not the Other Page Header of the main report*. This means you need to put an Other Page Header in all subreports that might be active at the end of a page, if you want an Other Page Header to print properly. Further, it means that the record the Other Page Header will have access to at that moment in time will be the first record it sees in the subset on the next page, so you can't just Block and Copy the Other Page Header from the main report to the Other Page Header of the subreport. If you do that, you'll be selecting fields from the wrong panel and can actually cause the report to kick the user to DOS! See my caveat about this in the *In Report Definition Mode* section of my **The Clipboard** chapter.

The subreport's Report Body, of course, will process all record in its subset, in the order its index dictates.

So, in complex reports, you must constantly think about where you are in the panel hierarchy as well as in the record stream. This gets tricky with subreports, and even more trickier with the Subreport Using Virtual Link option. With the latter, you're allowed to create a subreport that is not only free of being tied to an existing link, but also free of having any field list at all. It provides unfiltered access to all records in a panel while in a subreport. A common mistake developers make in using this very powerful Report Definition option is to forget that the main report runs once per record in the main panel, causing the Subreport Using Virtual Link to run on all its records over and over again—one full cycle per main report record. Take a look at some of the reports *Dummy Report Examples* in the **Subreports** chapter for examples of reports that take this into account (especially the *Report That Branches to Other Reports*).

Reports: General Structure

Though I mainly target the beginner here, I do cover topics that will benefit the experienced DataPerfect application developer.

The Basic Report

Let's do a simple report, starting from the beginning. Say I have a Class Panel that looks like this:

CLASS PANEL	
Year	Semester
.....
Name	Schedule
.....

N9999, G9
A15, A10

To the right of the panel, I indicated the format for each field. Further, I have a single record for each Class that occurs in each Semester for each Year. So, assuming no two Classes have the same Name, my indexes for this panel include *all* the fields you see above.

Let's say I want a report to list all Classes in order of their being taught, and, when taught in the same Semester, list them in alphabetical order by Name. That is, I want a report to list Classes sorted primarily by time taught, and secondarily by Name. Then I'll need to base the report on the following index on my Initial Report Definition Screen (item 3):

Year, Semester, Name, Schedule

After choosing Edit Report Form on the Initial Report Definition Screen (item 8), I see this:

```
-----FIRST PAGE HEADER-----  
--Empty--  
-----OTHER PAGE HEADER-----  
--Empty--  
-----TWO-LEVEL REPORT HEADER-----  
--Empty--  
-----REPORT BODY-----  
--Empty--  
-----TWO-LEVEL FOOTER-----  
--Empty--  
-----PAGE FOOTER-----  
--Empty--  
-----FINAL FOOTER-----  
--Empty--
```


Then I put in some text in the report, leaving x 's to make space for fields to be inserted later:

```

-----FIRST PAGE HEADER-----
CLASSES TAUGHT AT FALLBROOK HIGH

Year/Semester   Class Name           Schedule
=====
-----OTHER PAGE HEADER-----
                                   Page xx
                                   Classes, cont'd.

Year/Semester   Class Name           Schedule
=====
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
xxxxx/x         xxxxxxxxxxxxxxxx   xxxxxxxxxxxx
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

I'll leave it all flush left, allowing the Print Margins setting on the Initial Report Definition Screen to handle centering later, at which time I'll give it a big Left margin.

So, with the above layout, DataPerfect will first print the First Page Header and then start printing the Report Body once per record. Each time it comes to the end of a page, it will issue a page eject and print the Other Page Header at the top of the next page, and then start printing the Report Body once per record, starting where it left off on the previous page. The report ends when there are no more records in that index. It won't print the First Page Header more than that once. If I had a Page Footer defined, it would have printed that at the end of each page. If I had a Final Footer defined, it would print that once, at the very end.

So, with my x 's still in place, my report's first page would look like this:

[illegible]

And each subsequent page would look like this:

[illegible]

Let's figure out what I need for my Left margin setting. With 80 possible columns and 45 actual columns printed above, if I subtract 45 from 80 I get 35. Half that is about 18, which is a good start for my Left margin setting. If I want to add a half-inch for binder holes, I would increase that figure by 5 (figuring 10 characters per inch with courier print), for a Left margin of 23. That should get me pretty close to centered text on three-hole punched paper. Of course I'd have to actually print it out to see, and possibly reset that Left margin.

Of course I don't want all those x 's in my report. So let's put actual fields in my Edit Report Form screen. In the Other Page Header I have *Page xx*. To put a Page Number field in there instead of those x 's, I cursor to the two x 's and delete them. And then I **Ctrl-F7** for Report Options available to me in the Other Page Header (the Report Options menu you get with **Ctrl-F7** depends on what section of the Edit Report Form screen you're in). I now see this menu:

```

-----Report Options for Other Page Header-----
      1 - Select Report Field           4 - Skip if Start of Two Level
      2 - Eliminate Line if Blank
      3 - Include Before First Record

-----Selection: 0[OK-Enter]-----

```

I want a field that keeps track of the current page number and prints it. I hit **1**, looking for such a special field and get this menu (the Select Report Field menu you get by hitting **1** also depends on the section of the Edit Report Form screen you're in):

```

--Report Fields and Variables--
1 - Date              7 - Turn Print Off
2 - Time              8 - Turn Print On
3 - Page Number       9 - Turn File Off
4 - Store Value in Report Variable
5 - Print Report Variable
6 - Set Page Number   A - Turn File On
                     B - Printer Control
--Selection: 0[OK-Enter]--

```

Take a look at 3 above. That's what I want. When I hit **3**, DataPerfect inserts a **GZZZZZ9** field in my Edit Report Form screen, wherever I had my cursor at that moment. If I move the cursor to the left, landing it on that new field, I'll see its properties in the upper left portion of the Edit Report Form screen:

```

-----Column 1-----
Type the text to be included in the repo
press Select (F4), move to the field (po
press Select. While the cursor is on a
F6 to edit the report format. To includ
special control instructions, press Repo
-----
Page
-----
Field Format: GZZZZZ9
-----FIRST PAGE HEADER-----

```

Notice that the screen now displays *Page* and *Field Format: GZZZZZ9*, telling me this is a Page Number field with a format of **GZZZZZ9**. While on that field with my cursor, I can change its format to have less digits by hitting **F6** and typing it, say, **GZ9**. This field will now print the current Page Number in **GZ9** format, with the Edit Report Form screen looking like this now:

```

-----FIRST PAGE HEADER-----
CLASSES TAUGHT AT FALLBROOK HIGH
-----
Year/Semester  Class Name          Schedule
=====
-----OTHER PAGE HEADER-----
Page 1
Classes, cont'd.
-----
Year/Semester  Class Name          Schedule
=====
-----TWO-LEVEL REPORT HEADER-----
--Empty--
REPORT BODY-----
xxxx/x         xxxxxxxxxxxxxxxx    xxxxxxxxxxxx
-----TWO-LEVEL FOOTER-----
--Empty--
PAGE FOOTER-----
--Empty--
FINAL FOOTER-----
--Empty--

```

Now I replace the *x*'s in the Report Body. First I cursor the first set of *x*'s, representing the Year/Semester combination, and delete them. Then I hit **F4**, which throws me into the Class Panel. I then **Tab** to the Year field and **F4**. This places an **N9999** Year field in my Edit Report Form screen. I then type in the **'** if I previously deleted it and then **F4** again. I then **Tab** to the Semester field and **F4**. Now I have both fields in my Edit Report Form screen. The same process is done for the Name and Schedule fields, giving me the following Edit Report Form screen:

This will produce a first page like this:

And subsequent pages will look like this:

Two-Level Reports

Let's say you'd like this report organized a little differently. Say you'd like it to not repeat the same Year value over and over again. Rather, you'd like it to group records by Year like this:

Reports: General Structure 181

	2	Art 2	MWF 10-11
	2	Biology 2	TuTh 9-11
1996			
	1	Art 1	MWF 11-12
	1	Biology 2	TuTh 9-11
	2	Art 2	MWF 10-11
	2	History 1	MWF 8-9
etc.			

You can easily produce the above with a Two-Level Report version of the report I was defining previously. To accomplish the above grouping in my report, I cursor to the First Page Header and **Ctrl-F7**, giving me the following menu:

```

-----Report Options for First Page Header-----
1 - Select Report Field          6 - Prompt for Report Variable
2 - Eliminate Line if Blank     7 - Do Report in Subgroups
3 - Skip to Bottom of Page     8 - Create Record From Panel List
4 - Page Eject                 9 - Create Secondary Merge Report
5 - Two-Level Report           A - Iteration Control (Skip, etc.)
-----Selection: 0-----

```

Note item 5: Two-Level Report. That's the one I need to select in the First Page Header now. When I do, DataPerfect presents me with the panel on which this report is based and asks me for the Two-Level Sort Field for this Two-Level Report. The Two-Level Sort Field is the field DataPerfect will use to group records for this report, and this should be the first field of the index assigned to the report. In this case, I want records grouped by Year, so I'll choose the Year field as the Two-Level Sort Field by **Tabbing** to it and hitting **F4**. Assuming the Year field is the first field in that panel, this is what the First Page Header now looks like (I also took out the '/' between 'Year' and 'Semester'):

```

-----FIRST PAGE HEADER-----
-----Two-Level Report Sorted by Field: 1 -----
CLASSES TAUGHT AT FALLBROOK HIGH

Year Semester  Class Name          Schedule
=====
-----OTHER PAGE HEADER-----
Page 00
Classes, cont'd.

Year Semester  Class Name          Schedule
=====
-----TWO-LEVEL REPORT HEADER-----
REPORT BODY
-----TWO-LEVEL FOOTER-----
--Empty--
PAGE FOOTER
--Empty--
FINAL FOOTER
--Empty--

```

Note also I took the Year field out of the Report Body section and put it in the Two-Level Report Header section instead. I also got rid of the '/' that preceded the Semester field. Everything else remains the same as before.

What does the above Edit Report Form screen really say? It says DataPerfect will first group all records in that panel by Year. Then the Report Body will print those records as before, but will print the Year field (the Two-Level Report Header) at the beginning of each group a header to that group. Like this:

CLASSES TAUGHT AT FALLBROOK HIGH

Year	Semester	Class Name	Schedule
=====			
1995			
	1	Art 1	MWF 10-11
	1	Art 1	TuTh 9-11
	1	Biology 1	MWF 9-10
	1	Design 2	MWF 9-10
	2	Art 2	MWF 10-11
	2	Biology 2	TuTh 9-11
1996			
	1	Art 1	MWF 11-12
	1	Biology 2	TuTh 9-11
	2	Art 2	MWF 10-11
	2	History 1	MWF 8-9
etc.			

Subgroup Reports

But suppose you want that report by Semester, not just Year. That is, you want it to group by Year/Semester, printing each Year/Semester once, with all its Class records below it. This is easily done with a different option, very similar to the Two-Level Report option.

In the First Page Header, **Ctrl-F7, 7** is *Do Report in Subgroups*. This is similar to, but different from, **Ctrl-F7, 5**, which is a *Two-Level Report*. Whereas a Two-Level Report groups records by the first field in the active index, a Subgroup Report groups records by the combination of the first *and second* fields of the active index. Further, it prints the First Page Header more than once. It prints it at the beginning of each Subgroup, ignoring the Two-Level Report Header section.

So I change my Edit Report Form screen to look like this:

```

-----FIRST PAGE HEADER-----
-----Do Report in Subgroups-----
Year/Semester  Class Name      Schedule
=====
[Pattern]
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
[Pattern]
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

And it prints like this:

Year/Semester	Class Name	Schedule
=====		
1995/1		
	Art 1	MWF 10-11
	Art 1	TuTh 9-11
	Biology 1	MWF 9-10
	Design 2	MWF 9-10
1995/2		

	Art 2	MWF 10-11
	Biology 2	TuTh 9-11
1996/1		
	Art 1	MWF 11-12
	Biology 2	TuTh 9-11
1996/2		
	Art 2	MWF 10-11
	History 1	MWF 8-9

The Subgroup Report does the following: It first groups the records by the first-second field combination in the index, like you see above. Then it prints the First Page Header at the beginning of each subgroup. This differs from the Two-Level Report not only in that it groups on a different set of fields in the index (the Two-Level Report groups on the first field, not the first-second combination), but it takes the First Page Header as its group header, not the Two-Level Report Header.

This has a limiting consequence. Because the Subgroup Report uses the First Page Header as its group header, you have no *real* First Page Header for your Edit Report Form screen when doing a Subgroup Report. The Other Page Header works as usual in the Subgroup Report, however.

The Subreport

Note: For more on the ins and outs of Subreports, see my **Subreports** chapter.

Many times your database application will have a panel that is a sort of hybrid of two or more panels. In my Fallbrook High application, I might not only have a Class Panel, but also a Teacher Panel. The Teacher Panel might look something like this:

TEACHER PANEL	
Last Name	First Name
.....

Or, because two Teachers might have the same First and Last Names, something like this:

TEACHER PANEL	
ID	
.....	
Last Name	First Name
.....

In the second version of the Teacher Panel, I put in an ID field. It's a G9999::I field that automatically assigns a unique four-digit number to each Teacher Panel record (i.e., a unique four-digit number for each Teacher). If you're unsure of the ::I modifier, see *Display Mode Indicators* in my **Fields: Introduction** chapter.

In fact, as a general rule, it's a good idea to have an ID field in the Class Panel too. In my Fallbrook High example, there probably won't ever be two Classes with the same values in the Year, Semester, Name and Schedule fields, but it's still good database practice to give every record in a panel a unique ID value that will not only fail to appear in any other record in that panel, but will never change after the record is saved. So if Art History 101, being taught in 1996 in Semester 1 on MWF 10-11, is assigned ID 0032, then even if I later change its Name to History Of Art 101, any reference to that Class Panel record anywhere else in the database, in any other panel, will still refer to that very same Class Panel record, because it's still referring to Class Panel record 0032. If this isn't all that clear right now, it will be after you read the **Links** chapter.

So let's change our Class Panel to look something like this:

CLASS PANEL

ID

Year

Semester

Name

Schedule

The ID field above is a G9999::I field.

Now, if I put the ID field in the Class Panel's various indexes, I actually allow for more than one Class to have the same Name and be taught at the precisely the same time (Year, Semester, and Schedule). This could certainly happen if two different Teachers taught that Class at the same time.

My Class Panel indexes could now look like this:

Class Panel Indexes	
No.	Field List
1	Year, Semester, Name, Schedule, ID
2	Name, Year, Semester, Schedule, ID
3	ID, Year, Semester, Name, Schedule
4	ID, Name, Year, Semester, Schedule

I can then assign Classes to Teachers in a third panel called the Class/Teacher Panel:

CLASS/TEACHER PANEL			
Class			
Year		Semester	
Name		Schedule	
Teacher			
Last Name		First Name	

Now, suppose I wanted to create a report that would print out a list of Teachers showing, under each Teacher's name, all the Classes that Teacher ever taught. I'd like the Teachers to be listed in alphabetical order by Last Name, and their Classes listed chronologically. It would look something like this:

FALLBROOK HIGH TEACHERS

Adams, Joyce

1995/1 Physics 1
 1995/1 Math 2
 1995/2 Physics 2
 1995/2 Math 1
 1996/1 Math 2
 1996/1 Physics 1

Appleton, Sam

1995/1 Art History 1
 1995/1 Life Drawing 2
 1995/2 Life Drawing 1
 1995/2 Art History 2
 1996/1 Art History 1
 1996/1 Life Drawing 2

etc.

Though there are tricks I can use to do the above as a Two-Level Report based on the Class/Teacher Panel, it's difficult. Let's see why.

To define a Two-Level Report on the Class/Teacher Panel in the way that will produce a report like I just outlined above, I first need to find an index in that panel that sorts by Last Name. If I don't have one, I can certainly create one. Let's take a look at these possibilities:

Class/Teacher Panel Indexes	
No.	Field List
1	Last Name, First Name, Teacher ID, Year, Semester, Class Name, Schedule
2	Last Name, Teacher ID, First Name, Year, Semester, Class Name, Schedule
3	Last Name, Teacher ID, Year, Semester, Class Name, Schedule
4	Last Name, Year, Semester, Class Name, Schedule, First Name, Teacher ID

The first makes the most sense because it sorts first by Last Name, then by First Name, then by the time the Class took place (Year/Semester). That clearly represents the sorting of the report as outlined by hand above. But if I were to do a Two-Level Report using that index, I would get a report that groups by Last Name only. Don't forget that Two-Level Reports group by the first field of the index. This means that if I have two Teachers with the same Last Name, they'll share the same group in the report, and the header for that group will be the first name of those two Teachers in the index. So, if both Joyce Adams and Steve Adams teach at Fallbrook High, this Two-Level Report will print all the Classes of *both* Joyce *and* Steve in the same group, headed only by the name *Joyce Adams*. This isn't what I want.

Okay, then what about a Subgroup Report? That at least groups records by the first *two* fields of the index, taken as a unit. That would work if no two Teachers had the same Last and First Names. If that were the case, I could get rid of the Teacher ID field and use this index:

Last Name, First Name, Year, Semester, Class Name, Schedule

A Subgroup Report using that index would group the records by the combination of the Last and First Name fields, which means each Teacher gets his or her own group, even if they have the same Last Name.

But I'm allowing for the very common possibility that two Teachers have the same Last and First Name, so I need a Teacher ID field. A Subgroup Report that uses an index that begins with those two fields will, again, group by those two fields. But that means that if I have, say, two Teachers with the name *Steve Adams*, then all the Classes of those two Teachers will be in the same group in this Subgroup Report, sharing the same header, *Steve Adams*.

If I had a Subgroup Report option to group by the first *three* fields, I could use it here. Then I could use this index:

Last Name, First Name, Teacher ID, Year, Semester, Class Name, Schedule

But no such option exists as of this writing. I need a different approach: the Subreport.

Whereas you use the Report Options (**Ctrl-F7**) in the *First Page Header* to tell DataPerfect to run a report as Two-Level Report or a Subgroup Report, you use

the Report Options in the *Report Body* to tell DataPerfect to run a Subreport. You access the Subreport options with **Ctrl-F7, 6**, which gives you this menu:

```

Subreports & Record Creation
1 - Include Subreport
2 - Create Record Through Link
3 - Create Record From Panel List
4 - Subreport Using Virtual Link
0 - Return to Edit
Selection: 0

```

The two options relevant to my current problem are *Include Subreport* and *Subreport Using Virtual Link*. Understanding either of these requires that you already understand what a link is, and how to create one. If you don't, it's time to read my **Links** chapter.

Let's assume I have a panel link in the Class Panel and one in the Teacher Panel, and that each of these takes the user to the Class/Teacher Panel. The Class Panel panel link will link on the Class ID field, and the Teacher Panel panel link will link on the Teacher ID field. That is, each panel link has only one field in its field list, and that's the ID field in that panel.

Note: ID fields like these are ideal for panel link field lists because their values typically never change after the record is saved. So linkage is never broken.

My Class and Teacher Panels might now look like this:

CLASS PANEL	TEACHER PANEL
Class ID []	Teacher ID []
Year Semester [] []	Last Name First Name [] []
Name Schedule [] []	
To Class/Teacher Panel []	To Class/Teacher Panel []

The panel link in the Class Panel has the following properties:

Class Panel Panel Link	
Field List	Class ID
Index	Class ID, Year, Semester, Name, Schedule

When the user penetrates that link, he lands in the Class/Teacher Panel with access only to records with the same Class ID value found in the record he just left in the Class Panel. That is, penetrating that panel link gives him access to all and only Class/Teacher records that belong to the Class he's Browsing in the Class Panel.

Likewise, the panel link in the Teacher Panel has the following properties:

Teacher Panel Panel Link	
Field List	Teacher ID
Index	Teacher ID, Year, Semester, Name, Schedule

That gives the user access to all and only Class/Teacher Panel records that belong to the Teacher he's Browsing in the Teacher Panel.

Now I have what I need for the report at hand. First, I'll base the report on the Teacher Panel, not the Class/Teacher Panel. So I load the Teacher Panel and **Shift-F7** to call the Report List. In the Report List, I hit **Insert** on *Built-In Short Reports*. Now the Edit Report Form screen is based on the Teacher Panel.

Next, still on the Initial Report Definition Screen, I pick **3** (Index Number) and choose an index that sorts by the Last Name field in the Teacher Panel.

Now I hit **8** (Edit Report Form) and put in my First Page Header and Other Page Header. Then I cursor to the Report Body and **F4** to select the Teacher's Last Name and First Name fields, separating them with a comma, and reformatting (**F6**) the Last Name field to have the ;;T Print Mode Indicator (to strip off any trailing spaces in the Last Name field before printing the comma and First Name).

So far I have this:

```

-----FIRST PAGE HEADER-----
FALLBROOK HIGH TEACHERS
-----OTHER PAGE HEADER-----
FALLBROOK HIGH TEACHERS, continued
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
, <A15;;T and A15>
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

That Edit Report Form screen will, of course, only print a list of Teachers. For each Teacher's list of Classes, I need my Subreport. In the Report Body, I cursor to just past the First Name field (the second field in the report) and hit **Enter** twice so the list of Classes begins two lines after the Teacher's name. I then hit **Ctrl-F7, 6** to see my Subreport options menu:

```

-----Subreports & Record Creation-----
1 - Include Subreport
2 - Create Record Through Link
3 - Create Record From Panel List
4 - Subreport Using Virtual Link
0 - Return to Edit
-----Selection: 0-----

```

I want my Subreport to use the panel link I just put in the Teacher Panel, so I choose **1**. The difference between choice **1** and **4** is that **1** uses an existing panel link and **4** has the Subreport create a temporary panel link on the fly, and then remove it when the Subreport is done. Option **1** is easier to understand at this stage, but **4** is better to

use once you understand what you're doing (that is, after you read my **Subreports** chapter).

Okay, I choose **1**. DataPerfect now puts me in the panel on which the report is based (the Teacher Panel), and asks me to *Move to the desired panel link or data link, then press Select (F4)*. This is asking me what link the report should use in creating its Subreport. The Subreport will have precisely the same access the user has when penetrating that link with **Down Arrow** in Browse mode. That is, it will have access to all and only those records in the Class/Teacher Panel that relate to the Teacher Panel record that is currently being processed by the Report Body.

```

-----FIRST PAGE HEADER-----
FALLBROOK HIGH TEACHERS
-----OTHER PAGE HEADER-----
FALLBROOK HIGH TEACHERS, continued
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
....., .....
=====SUBREPORT LINK/PANEL: 4 3=====
-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
--Empty--
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
=====END OF SUBREPORT=====
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

In the above Edit Report Form screen, notice that a new section has been inserted into the Report Body. It starts with *SUBREPORT LINK/PANEL: 4 3* and ends with *END OF SUBREPORT*. That's the Subreport I just created. The first line of it says that the Subreport is using a link whose field number is 4 (DataPerfect considers panel links to be fields, and this panel was the fourth field I created in this panel) and whose target is Panel 3, which is the Class/Teacher Panel. So the Subreport takes the report into the Class/Teacher Panel, doing this for each record it comes to in the Report Body of the main report.

Notice also that a Subreport has all the same sections available to you the main report has. It's a complete report in and of itself. It can even have Subreports in its own Report Body.

In this Subreport, I'll put in another Other Page Header and a few fields. The Subreport's Other Page Header will print just in case the report comes to the end of a page before finishing the current Subreport. I'll select the Year, Semester and Class Name fields with **F4**, also putting in a '/' between the Year and Semester fields. And finally, I'll add some needed spacing by putting carriage returns in the First Page Header and Other Page Headers, and one in the Final Footer. This gives me a Edit Report Form screen something like this:

```

-----FIRST PAGE HEADER-----
FALLBROOK HIGH TEACHERS

-----OTHER PAGE HEADER-----
FALLBROOK HIGH TEACHERS, continued

-----TWO-LEVEL REPORT HEADER-----
--Empty--
REPORT BODY
-----
-----SUBREPORT LINK/PANEL: 4 3-----
-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
FALLBROOK HIGH TEACHERS, continued

-----TWO-LEVEL REPORT HEADER-----
--Empty--
REPORT BODY
-----
-----TWO-LEVEL FOOTER-----
--Empty--
PAGE FOOTER
--Empty--
FINAL FOOTER
--Empty--
-----END OF SUBREPORT-----

-----TWO-LEVEL FOOTER-----
--Empty--
PAGE FOOTER
--Empty--
FINAL FOOTER
--Empty--

```

That Edit Report Form screen produces a report something like this, without the annotations:

FALLBROOK HIGH TEACHERS	First Page Header
Adams, Joyce	Report Body, Record 1
1995/1 Physics 1	
1995/1 Math 2	
1995/2 Physics 2	Subreport for
1995/2 Math 1	Record 1
1996/1 Math 2	
1996/1 Physics 1	
Appleton, Sam	Report Body, Record 2
1995/1 Art History 1	
1995/1 Life Drawing 2	
1995/2 Life Drawing 1	Subreport for
1995/2 Art History 2	Record 2
1996/1 Art History 1	
1996/1 Life Drawing 2	
etc.	Report Body, Record 3

Two-Level Reports in Subreports

I can take this another step. I can put a Two-Level Report inside the Subreport:

```

-----FIRST PAGE HEADER-----
FALLBROOK HIGH TEACHERS

-----OTHER PAGE HEADER-----
FALLBROOK HIGH TEACHERS, continued

-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
....., .....

=====SUBREPORT LINK/PANEL: 4 3=====
-----FIRST PAGE HEADER-----
-----Two-Level Report Sorted by Field: 2-----

-----OTHER PAGE HEADER-----
FALLBROOK HIGH TEACHERS, continued

-----TWO-LEVEL REPORT HEADER-----
.....<Year>
-----REPORT BODY-----
S.....<Semester and Class Name>
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
=====END OF SUBREPORT=====

-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

In the above Edit Report Form screen, in the Subreport's First Page Header, I told DataPerfect I wanted a Two-Level Report (**Ctrl-F7, 5**), and, when prompted to do so, selected the Year field in the Class/Teacher Panel (I **Tabbed** to that field and then hit **F4**) to sort this Two-Level Report. I also changed the Report Body a little. I moved the Year field out of the Report Body and put it in the Subreport's Two-Level Report Header. I also put an *S* immediately before the Semester field.

Here's what the Edit Report Form screen will produce:

FALLBROOK HIGH TEACHERS	First Page Header
Adams, Joyce	Report Body, Record 1
1995	
S1 Physics 1	
S1 Math 2	
S2 Physics 2	
S2 Math 1	
1996	
S1 Math 2	
S1 Physics 1	
Appleton, Sam	Report Body, Record 2

***Subreport as Two-Level Report
for Record 1***

1995	S1 Art History 1	Subreport as Two-Level Report for Record 2
	S1 Life Drawing 2	
	S2 Life Drawing 1	
	S2 Art History 2	
1996	S1 Art History 1	Report Body, Record 3
	S1 Life Drawing 2	
etc.		

The Primary Sorting Field

This brings up an important point about Subreports, Two-Level Reports, and Subgroup Reports. Note that the panel link on which I based the above Subreport has the following properties:

Teacher Panel Panel Link	
Field List	Teacher ID
Index	Teacher ID, Year, Semester, Name, Schedule

Now review what I did when I created the Two-Level Report inside the Subreport. I told DataPerfect to sort that Two-Level Report on the Year field in the Class/Teacher Panel. And the index you see above is active when this report is processing that Two-Level Report. But the Year field is not the *first* field in the active index at that moment; whereas, I told you earlier that you must choose *the first field of the index assigned to the report*. Well, that's not completely accurate. What you're supposed to do is make sure the Two-Level Report sorting field is the primary sorting field on which DataPerfect will be sorting in that part of the report. If you're not in a Subreport—that is, you're in the main report, then the primary sorting field *will* be the first field in the report index. But if you're in a Subreport, the primary sorting field will be the first field in the active index that follows the fields that make up the field list of the link that takes you to that Subreport. This sound much more complicated than it is. Let's explain.

In the main report of this subreport, DataPerfect is sorting on the Teacher's Last Name and First Name fields. That's the first field of the active index. In this case the active index is the index assigned to the report as a whole in the Initial Report Definition Screen. But in the Subreport, the active index (the index on the link used to create the Subreport) starts with the Teacher ID field, followed by the Class Year field. But when you or the report penetrates that link, you land in the Class/Teacher Panel with access only to records of a single Teacher (the Teacher you just left in the Teacher Panel). This is because the field list of that link consists of the Teacher ID field. That effectively filters out all other Class/Teacher Panel records from view.

So when the main report, which is in the Teacher Panel, penetrates the panel link on its way to the Subreport, it has access only to those Class/Teacher Panel

records that belong to the Teacher it was processing just before penetrating the link. So it wouldn't make sense for DataPerfect to be sorting on the *first* field of the active index (which is the Teacher ID field), because all and only the records with that single Teacher ID value are in its world at this moment. So DataPerfect sorts at this point on the field that immediately follows the fields in the link's field list, which, in this case, is the Year field. The Year field is the primary sorting field in the active index of this Subreport.

If you want to create a Two-Level Report in this Subreport (as I just did), then you must choose the *second* field in the active index as the field on which to sort the Two-Level Report. That's because the *second* field in this case is the primary sorting field. Again, in the main report, the *first* field is *always* the primary sorting field. Also note that the primary sorting field in a Subreport need not be the *second* field in the active index. If there's more than one field on the field list of the link used to create the Subreport, the primary sorting field will be later in the index field list than the second field. *The primary sorting field is the field in the index that immediately follows the link field list*, no matter how many fields are in the link field list.

Subgroup Reports in Subreports

Just as you can put a Two-Level Report in a Subreport, you can put its cousin, the Subgroup Report, in a Subreport. Instead of using **Ctrl-F7, 5** in the Subreport's First Page Header, I would use **Ctrl-7, 7**. That inserts the *Do Report in Subgroups* code there. I would also move the Semester field into the Subreport's First Page Header, just after the *Do Report in Subgroups* code. Here's what that Edit Report Form screen would look like:

```

-----FIRST PAGE HEADER-----
FALLBROOK HIGH TEACHERS

-----OTHER PAGE HEADER-----
FALLBROOK HIGH TEACHERS, continued

-----TWO-LEVEL REPORT HEADER-----
--Empty--
REPORT BODY
-----
-----SUBREPORT LINK/PANEL: 4 3-----
-----FIRST PAGE HEADER-----
-----Do Report in Subgroups-----
-----
-----OTHER PAGE HEADER-----
FALLBROOK HIGH TEACHERS, continued
-----
-----TWO-LEVEL REPORT HEADER-----
--Empty--
REPORT BODY
-----
-----TWO-LEVEL FOOTER-----
--Empty--
PAGE FOOTER
--Empty--
FINAL FOOTER
--Empty--
-----END OF SUBREPORT-----
-----
-----TWO-LEVEL FOOTER-----
--Empty--
PAGE FOOTER
--Empty--
FINAL FOOTER
--Empty--

```

And here's what it produces:

FALLBROOK HIGH TEACHERS		First Page Header
Adams, Joyce		Report Body, Record 1
1995/1		
Physics 1		
Math 2		
1995/2		
Physics 2		
Math 1		
1996/1		
Math 2		
Physics 1		
Appleton, Sam		Report Body, Record 2
1995/1		
Art History 1		
Life Drawing 2		
1995/2		
Life Drawing 1		
Art History 2		
1996/1		
Art History 1		
Life Drawing 2		
etc.		Report Body, Record 3

Though Subgroup Reports are supposed to group records by the first-second field combination in the index, note that, in this case, it groups them by the second-third field combination. Again, just like the Two-Level Report I earlier inserted in a Subreport, this Subgroup Report considers the primary sorting field in that part of the report (the Subreport) to be the *second* field of the active index, not the *first* field. Unlike the Two-Level Report, which groups records by the solitary primary sorting field in the active index, the Subgroup Report sorts on the first-second field combination, *starting with the primary sorting field in the active index*. That combination, in this case, is the second-third field combination of that active index.

Fields in Report Definitions have different rules than those found in panels. Again, though my main focus here is the beginner, experienced DataPerfect application developers will get something out of this chapter.

F Fields

These are floating decimal fields, available only in reports. This field type begins either with *FN* or *FZ*, followed by a number from 3 to 16. The FZ field strips leading zeroes, otherwise they're the same. The number in this field type determines how many character spaces, including the decimal point, will be allotted. The need for this field type arises in reports where you don't have enough room to print the largest possible value that may be found in the accessed numerical field.

For instance, suppose your report is to print the value found in a panel's G999.9999 field. That field contains eight characters, counting the decimal. You only have room for six characters on that line in the report. You could use a G999.99 field in the report (which would round the number to the nearest two decimals), but you want this report to show as much of the decimal part of the number as possible. If you format this report field FN6, the report will always print all the digits to the left of the decimal, coupled with as many of those to the right as possible, for a maximum of six, including the decimal point.

Here are examples I took from the DataPerfect manual to clear this up for you:

Number	FN6	FN10	FZ6	FZ10
12.66666	12.667	0012.66666	12.667	12.66666
23,987.3456	023987	23987.3456	23987	23987.3456
-34.5	-34.5	-0000034.5	-34.5	-34.5
7,376,191.4	*****	07376191.4	*****	7376191.4

Note that the F field displays the negative sign if the accessed panel field has a negative value. Also note the asterisks where the field format was too small to accommodate all the digits to the left of the decimal. Asterisks display in *any* DataPerfect numerical field when the value in that field doesn't fit that field's format. This goes for N, G, and H fields as well as F fields.

Print Mode Indicators

As I've mentioned previously (in my **Fields: Introduction** chapter), there are special field modifiers that may be used in reports, while having no effect on field display in a panel. These Print Mode Indicators, as the DataPerfect manual calls them, all follow a double *semi*-colon, as opposed to a double colon. They can be placed on any field selected in a Report Definition or used in formatting Report Variables. If the field in the Edit Report Form screen was selected with the **F4** key, to alter its format in the report, you just cursor to that field, hit **F6**, and then fill in the new format.

Print Mode Indicators That Alter Field Output Spacing

These Print Mode Indicators tighten field output by stripping it of wasted space:

;;T (Truncate Trailing Blanks)

Say you select an A15 field with **F4** in the Edit Report Form screen, and then change its format to A10;;T. This field will now print no more than the first ten characters in the field, and will remove all trailing spaces from the output. By truncating the trailing spaces, the space taken up by the actual field in the report adjusts downward (from the right) to fit the actual output.

So, say these two fields appear in your Edit Report Form screen:

```
.....
```

If the first field's output is *Sally*, and the second's is *Adams*, then if the two fields are each formatted A10, the output for that line is

```
Sally      Adams
```

To remove the trailing spaces in the first field's output, we change its format to A10;;T to get this output:

```
Sally Adams
```

;;S (Suppress Leading Blanks)

This removes the leading spaces from a field's output, instead of the trailing spaces. It would have no effect on the example above unless the user entered, say, two spaces in the first field before typing *Sally*. More commonly, it would be used with G fields that have their leading zeros suppressed with the Z placeholder.

For instance, look at the following line from an Edit Report Form screen:

```
It's been ..... days since your last  checkup.
```

The above field is a GZZZ9 field (perhaps a Report Variable that's being printed, or an actual field from a panel—it's not important for this example which it is). If the field in the Edit Report Form screen remains GZZZ9, without a Print Mode Indicator

added to it, we'll get output like this for someone who hasn't had a checkup in eighty-five days :

```
It's been    85 days since your last checkup.
```

If we reformat that field to GZZZ9;;S we get the following output:

```
It's been 85 days since your last checkup.
```

;;B (Truncate Both Leading and Trailing Blanks)

Very straightforward. It works just like using both ;;S and ;;T on the same string.

;;1-9 (Truncate Leading and Trailing Blanks and Leave n Spaces)

This option works just like ;;B with one added feature. After removing leading and trailing spaces, it then adds the number of spaces indicated to the end of the output. This means that the output might actually take up more space than the original field. For instance, if the field is formatted A15;;9, and it contains Sally Adams, the final output will occupy 20 character spaces (11 for *Sally Adams* plus 9).

;;E (Delete Zero Subfields from the End)

This is intended for numeric fields only. If that field is divided into subgroups of contiguous digits, it drops the final subgroup if it's zero, and truncates the resulting trailing spaces. Its classical use is with 9-digit Zip Code fields that are formatted N99999-9999. If the final four digits in that field are all zero, and the report field is formatted

```
N99999-9999;;E
```

then the output will lack the hyphen and the trailing zeroes. Also, if there's no data in a numerical field modified by the ;;E Print Mode Indicator, it won't print anything. In either case the indicator removes any resulting trailing spaces.

So, in the case of only the final subgroup having all zeroes, the output will decrease by five character spaces (the hyphen and the four zeroes). In the second case, where there's no data at all, there won't be any output.

Print Mode Indicators That Don't Alter Field Output Spacing

;;D (Delete All Blanks)

The DataPerfect manual misleads you here:

```
This indicator removes all blanks from
fields that have G, H, or N formats. Data
is shifted from the right of the blanks to
the left, and the trailing spaces are not
truncated. This option will maintain the
amount of space specified by the field
format in the report.
```

Yes, this indicator *removes all blanks from fields that have G, H, or N formats*, but it also does this with alpha fields (A and U). All data is shifted to left after spaces are deleted, but the end result still occupies the same amount of space (This latter point is why I didn't include this in the ***Print Mode Indicators That Alter Field Output Spacing*** section above.). So, essentially, all those spaces are thrown to the end of the string.

;;R (Right Adjust Characters)

;;L (Left Adjust Characters)

;;C (Center Characters)

The output here always maintains its original space. ;;R shifts all data to the rightmost edge of that space. ;;L shifts it to the left. ;;C centers it.

;;P (Postal Bar Code)

This Print Mode Indicator works only if DPPrint is active. If attached to a Zip Code field (N99999-9999;;P), DataPerfect will print the Zip Code out using bar codes the Post Office recognizes. The bar codes will show up under the Zip Code.

;;Q (Enclose Alphanumeric Fields in Double Quotes)

This Print Mode Indicator only works on alpha fields (A and U). It encloses the field output with quotes. What the DataPerfect manual doesn't tell you is that it removes trailing (not leading) spaces first. So it acts like putting quotes around a ;;T field. When using the ;;Q Print Mode Indicator, all other indicators attached to that field will be ignored. You might want to use this format for comma delimited exports, but don't forget to manually insert the commas between fields. Also don't forget it fails to do anything with numerical fields, so you'll have to manually enter quotes for them.

;;N (New Occurrence of Field)

Read my the *Variable-Length Text Fields in Reports* section, later in this chapter, for an explanation of the ;;N Print Mode Indicator.

Summary Table

Here are three fields that might be found in an Edit Report Form screen. I put the format of each field above it:

A20	GSZZZ,ZZ9.99	N999 999 9999
.....

And here are the printed results using Print Mode Indicators. Each line uses a different Print Mode Indicator, as noted on the left. The block below each string shows the field placement and how much space the field actually occupies on paper (I reformatted the example found in the DataPerfect manual for this):

Indicator	Print Result		
None	Lynda A. Warner	\$485.27	012 345 6789
;;T	Lynda A. Warner	\$485.27	012 345 6789
;;S	Lynda A. Warner	\$485.27	012 345 6789
;;B	Lynda A. Warner	\$485.27	012 345 6789
;;1	Lynda A. Warner	\$485.27	012 345 6789
;;6	Lynda A. Warner	\$485.27	012 345 6789
;;E	\$485.27	012 345 6789	
;;D	LyndaA.Warner	\$485.27	012 345 6789
;;R	Lynda A. Warner	\$485.27	012 345 6789
;;L	Lynda A. Warner	\$485.27	012 345 6789
;;C	Lynda A. Warner	\$485.27	012 345 6789

The last four Print Mode Indicators (D, R, L, C) are grouped together because they don't alter field output spacing. The other do, so, when you use them, columns might not line up in the report if you don't use tabs between fields. That said, say you place fields like this in your Edit Report Form screen:

	1	2	3
Columns	123456789012345678901234567890123456789		
Fields

In the above example, the first two fields are formatted A10, and the third GZZZ,ZZ9.99. The periods represent spaces. The definer intends for these fields to start at columns 1, 12 and 25, respectively, yielding output like this:

	1	2	3
Columns	123456789012345678901234567890123456789		
	Sally Adams		234.00
	John Smith		2188.00
	Sam Yardley		5.00

Leaving the fields formatted A10, A10, and GZZZ,ZZ9.99, you get the following output:

	1	2	3
Columns	123456789012345678901234567890123456789		
	Sally	Adams	234.00
	John	Smith	2188.00
	Sam	Yardley	5.00

Adding the ;;T Print Mode Indicator to only the first field yields this:

	1	2	3
Columns	123456789012345678901234567890123456789		
	Sally Adams	234.00	
	John Smith	2188.00	
	Sam Yardley	5.00	

And adding the ;;T Print Mode Indicator to both the first and second fields yields this:

	1	2	3
Columns	123456789012345678901234567890123456789		
	Sally Adams	234.00	
	John Smith	2188.00	
	Sam Yardley	5.00	

And finally, taking the ;;T Print Mode Indicator off the second field (but leaving it on the first field), and then replacing the spaces between the second and third fields with a Tab to Column 25 (when you hit **Tab** in the Edit Report Form screen DataPerfect prompts you for a column number), we have an Edit Report Form screen that looks like this (with the period representing a space and angle brackets indicating the Tab to Column code):

```
.....<tab to col 25>.....
```

That yields output like this:

	1	2	3
Columns	123456789012345678901234567890123456789		
	Sally Adams	234.00	
	John Smith	2188.00	
	Sam Yardley	5.00	

That was the originally intended result.

Sneaking Print Mode Indicators into Panel Fields

Now wait a minute. I began this discussion of Print Mode Indicators saying they are *special field modifiers that may be used in reports, while having no effect on field display in a panel*. Well, I lied. That's what the DataPerfect manual wants you to think, and it's probably how they really are supposed to be used, but there's a sneaky way to get them to affect panel fields. I give examples of this in more detail in the **APPLY.FORMAT** section of my **Formulas** chapter.

You do this with the APPLY.FORMAT function. If you don't understand how to use that function, wait until you get to my discussion of it referenced in the above paragraph, otherwise I doubt you'll understand what I'm about to say.

First off, though DataPerfect lets you put a Print Mode Indicator in a panel field's format (e.g., A20;;R), it won't have any effect. To sneak a Print Mode Indicator into a panel field's display formatting, in a way that actually impacts the display, that field will have to be one that displays data via a field formula. Second, that formula must use the `APPLY.FORMAT` function to convert the data using the appropriate Print Mode Indicator.

For instance, suppose P1F1 and P1F2 are the First Name and Last Name fields in an application, and both are formatted A15. Because they're both A fields, they're both left-aligned. A third field, formatted A30::C, can display a *right*-aligned concatenation of the *left*-aligned strings found in P1F1 and P1F2 with this formula:

```
apply.format["A30;;R";cat.t[P1F1;" "P1F2]]
```

I give other examples of using Print Mode Indicators in panel field formulas in the *APPLY.FORMAT* section of my **Formulas** chapter.

Variable-Length Text Fields in Reports

DataPerfect Report Definitions offer you a variety of ways to control variable-length text field output, so I must devote an entire section to that topic now.

When in an Edit Report Form screen, if you simply select a variable-length text field with the **F4** key and leave it in default format, you'll notice that DataPerfect inserted a field with a format of `AxA0`, where *x* is the width assigned to that field in its panel. So if the variable-length text field in the panel is A25A5, selecting it with **F4** in an Edit Report Form screen will insert a single-line field of the format A25A0, *not* A25A5. It will appear no different on the Edit Report Form screen than an A25 field. The only way you would know it's a variable-length text field is by cursoring to the field and watching field format indicator, just above the First Page Header.

The *x* component of an `AxAy` report field tells DataPerfect how many characters to print per line. You can change that value by just editing the report field with **F6**. If the panel field is A25A5, hitting **F4** to select it in the Report Definition inserts an A25A0 field. If you edit it with the **F6**, and change the 25 to a 20, DataPerfect will print 20 characters per line for that field, wrapping where necessary. It won't truncate. It'll wrap. So you still print out the entire field's contents, even though you decreased the *x* component in the report field. Likewise, if you increase the *x* component, DataPerfect will simply wrap lines not ending with a carriage return further to the right.

But what's that *y* component? Well, the manual is misleading (actually, incorrect) in its explanation of it. If the *y* component is greater than 0, DataPerfect will print the first *y* lines of the remaining lines of that field. By *remaining lines*, I mean all those lines that have yet to be printed from that field for that record. So, let's say we reformat that field to A25A1. If this is the first time that variable-length text field has been accessed by this Report Body, when DataPerfect comes to that A25A1 field, it will print the first line of the variable-length text field, and then set the

report's internal pointer for that field to its next line, waiting for another call to print data from that field for that record.

Now, if you put another A25Ay field in that Report Body, somewhere after the first A25Ay field, when DataPerfect comes to that second A25Ay field, it will begin printing where it left off with the previous A25Ay field. So if the first field that accesses that variable-length text field is A25A1, and the second one is A25A3, the first will print the first line and the second will print the next three lines (the second, third and fourth lines).

If the *y* component is 0, DataPerfect prints all the remaining lines in that variable-length text field, starting where the last AxAy access of that field left off for that record.

So, to summarize this with a rule, for any given record, an AxAy report field will print *x* characters per line. If *y* is 0, it prints *all* of the remaining lines of that field. If *y* is greater than 0, it prints *y* of the remaining lines. When all the lines of that field are printed, the report's internal pointer for that field will be reset to the beginning of the field, waiting for the next AxAy access of that field.

Here's the mistake the manual made:

If you want to use the same text field twice in a report, you have two options.

In the first use of the text field, the format must end with something other than A1 (for example, A0, A2, A3, etc.). DataPerfect then resets its internal pointer back to the beginning of the field. When the text field is selected for the second use, DataPerfect prints the field from its beginning.

To reset the DataPerfect internal pointer to the beginning of the field, you must change the text field format in the second use to include the ;N print mode indicator (see Field Format, Print Mode Indicators in Reference). This indicator tells DataPerfect that it has a new occurrence of the field, and the field will be printed from the beginning.

That second paragraph is false. DataPerfect *resets its internal pointer back to the beginning of the field* only when the last line has printed. This is true no matter what *y* is. So setting *y* to some value other than 1 makes no difference.

For any given record, there's only one way of assuring the report's internal pointer for that field is reset to the beginning of that field after the current AxAy field prints. That's to set *y* to a value that assures this particular AxAy field prints all the remaining lines in that field. Setting *y* to 0 will certainly assure this, but you could also set *y* to some other value that will assure it as well (if you know that at this stage of the Report Body there will never be more than 2 lines remaining, setting *y* to 2 will work).

The ;;N Print Mode Indicator (New Occurrence of Field)

Alternatively, you can forget about getting the *current* AxAy field to reset the report's internal pointer to the beginning after that field prints. Instead, you can attach the ;;N Print Mode Indicator to the *next* AxAy field. The ;;N Print Mode Indicator tells DataPerfect to start at the beginning of the variable-length text field. So an A25A3;;N field will print the first three lines of the field, even if the report's internal pointer for that field wasn't at the beginning of the field at that time. The ;;N Print Mode Indicator resets that pointer to beginning of the field *before* counting *x* lines. When done, the pointer is left at the end of line *x*.

Examples

Consider the following panel display of a record that has three fixed-length fields and one variable-length text field (field formats shown on the right):

Name:	Sally Adams	A20
Home Phone:	310/555-1414	N999/999-9999
Work Phone:	310/555-1515	N999/999-9999
Address:	123 Elm Street	A20A5
	Apt 213	
	Anywhere, CA 90024	
	USA	

What follows is an extensive list of examples of different variations of the AxAy report field. In each case, the Edit Report Form screen layout in the Report Body is indicated, followed by the output it would produce:

Report Body layout

Name	Name
A20A0	A20A0

Output

Sally Adams	Sally Adams
123 Elm Street	123 Elm Street
Apt 213	Apt 213
Anywhere, CA 90024	Anywhere, CA 90024
USA	USA

Report Body layout

Name	A20A0
------	-------

Output

Sally Adams	123 Elm Street Apt 213 Anywhere, CA 90024 USA
-------------	--

Report Body layout

A20A0	Name
-------	------

Output

123 Elm Street Apt 213 Anywhere, CA 90024 USA	Sally Adams
--	-------------

Report Body layout

Name	A20A0
Home	

Output

Sally Adams	123 Elm Street Apt 213 Anywhere, CA 90024 USA
310/555-1414	

Report Body layout

A20A0	Name
	Home

Output

123 Elm Street Apt 213 Anywhere, CA 90024 USA	Sally Adams
	310/555-1414

Report Body layout

Name	A20A1
Home	A20A0

Output

Sally Adams
310/555-1414

123 Elm Street
Apt 213
Anywhere, CA 90024
USA

Report Body layout

A20A1	Name
A20A0	Home

Output

123 Elm Street	Sally Adams
Apt 213	310/555-1414
Anywhere, CA 90024	
USA	

Report Body layout

Name	A20A1
Home	A20A0
Work	

Output

Sally Adams	123 Elm Street
310/555-1414	Apt 213
	Anywhere, CA 90024
	USA
310/555-1515	

Report Body layout

A20A1	Name
A20A0	Home
	Work

Output

123 Elm Street	Sally Adams
Apt 213	310/555-1414
Anywhere, CA 90024	
USA	
	310/555-1515

Report Body layout

Name	A20A1
Home	A20A1
Work	A20A0

Output

Sally Adams	123 Elm Street
310/555-1414	Apt 213
310/555-1515	Anywhere, CA 90024
	USA

Report Body layout

A20A1	Name
A20A1	Home
A20A0	Work

Output

123 Elm Street	Sally Adams
Apt 213	310/555-1414
Anywhere, CA 90024	310/555-1515
USA	

Report Body layout (includes the ;;N indicator)

Name	A20A1
Home	A20A1;;N
Work	A20A0;;N

Output

Sally Adams	123 Elm Street
310/555-1414	123 Elm Street
310/555-1515	123 Elm Street
	Apt 213
	Anywhere, CA 90024
	USA

Report Body layout (includes the ;;N indicator)

A20A1	Name
A20A1;;N	Home
A20A0;;N	Work

Output

123 Elm Street	Sally Adams
123 Elm Street	310/555-1414
123 Elm Street	310/555-1515
Anywhere, CA 90024	
USA	

Here I discuss the various Report Options accessed via **Ctrl-F7** in the Edit Report Form screen. This is mainly for beginners, with a few topics that will offer new information to many experienced DataPerfect application developers.

The Report Options Menus

Here are Report Options menus called by **Ctrl-F7** in the various sections of your Edit Report Form screen:

Report Options for First Page Header

1 - Select Report Field
2 - Eliminate Line if Blank
3 - Skip to Bottom of Page
4 - Page Eject
5 - Two-Level Report

6 - Prompt for Report Variable
7 - Do Report in Subgroups
8 - Create Record From Panel List
9 - Create Secondary Merge Report
A - Iteration Control (Skip, etc.)

Selection: 0

Report Options for Other Page Header

1 - Select Report Field
2 - Eliminate Line if Blank
3 - Include Before First Record

4 - Skip if Start of Two Level

Selection: 0

Report Options for Two-Level Header

1 - Select Report Field
2 - Eliminate Line if Blank
3 - Conditional Page Eject

4 - Page Eject

Selection: 0

Report Options for Report Body

1 - Select Report Field
2 - Eliminate Line if Blank
3 - Conditional Page Eject
4 - Skip to Bottom of Page
5 - Labels

6 - Subreports
7 - Record Number
8 - Store Report Variable into Field
9 - Iteration Control (Skip, etc.)
A - Delete Record

Selection: 0

Report Options for Two-Level Footer

1 - Select Report Field
2 - Eliminate Line if Blank
3 - Skip to Bottom of Page

4 - Page Eject
5 - Number of Records in Section

Selection: 0

Report Options in Page Footer

1 - Select Report Field
2 - Eliminate Line if Blank
3 - Skip to Bottom of Page

4 - Include After Last Record
5 - Number of Records on Page

Selection: 0

Report Options in Final Footer

1 - Select Report Field
2 - Eliminate Line if Blank
3 - Page Eject

4 - Number of Records in Report
5 - Create Record From Panel List
6 - Skip to Bottom of Page

Selection: 0

Global Report Options

Though the Report Options menu in each section varies somewhat from the Report Options menus in other sections, there are some options that appear on all such Report Options menus. Let's go over these first.

Select Report Field

Ctrl-F7, 1 (Select Report Field) calls the Report Fields and Variables menu:

Report Fields and Variables	
1 - Date	7 - Turn Print Off
2 - Time	8 - Turn Print On
3 - Page Number	9 - Turn File Off
4 - Store Value in Report Variable	A - Turn File On
5 - Print Report Variable	B - Printer Control
6 - Set Page Number	C - Open Filename in RV
Selection: 0	

This is the same in every Edit Report Form screen section, and is very straightforward.

Options 1, 2, and 3 (Date, Time and Page Number)

These options allow you to insert Date, Time and Page Number fields respectively. The first prints today's date in D99/99/99 format. The second prints the current time in TZ9:99 format. And the third prints the current page number in GZZZZZ9 format. You can reformat any of these three fields with **F6**.

Option 4 (Store Value in Report Variable)

This option allows you to create a Report Variable. It first prompts you for the Report Variable's number, then calls a Specify Formula screen, where you give that Report Variable a formula. See my **Formulas** chapter for more on formulas.

Option 5 (Print Report Variable)

This option allows you to print the value found in a selected Report Variable. I go over this in my **Report Variables** chapter, in the *Printing Data Not Already in Fields* section.

Option 6 (Set Page Number)

This options allows you to reset the page number counter, which prints with option 3. Sometimes you don't want the page number counter to start at the beginning of the report. Sometimes you want it to start over with each new section. This option meets these sorts of needs.

Options 7, 8, 9, and A (Turn Print/File On/Off)

I use these options throughout this book when outlining report examples. They allow you to place text in a report that displays via the screen to the user, without going to the printer or disk file. Examine many examples of this in the **Subreports** chapter. Here's a small piece from that chapter:

-----FIRST PAGE HEADER-----	
-----Turn Print Off-----	Main Report First Page Header
-----Turn File Off-----	
You just chose to print a series of Day Sheets, one per Doctor. Please fill in the desired Date at the prompt and hit ENTER; otherwise, hit F1 now to cancel this procedure.	
-----Prompt for Value of Report Variable 1 -----	Explanatory text for the user to read
-----Turn Print On-----	
-----Turn File On-----	
-----OTHER PAGE HEADER-----	

The above *explanatory text* is seen by the user when the report runs, but never finds itself going to the printer or a disk file.

Option B (Printer Control)

This option allows you to insert printer control strings that end up taking no visible space in the various report sections of the Edit Report Form screen. When you insert a printer control string in the field called with **B**, the string can only be seen again when the cursor sits on the place where the Printer Code was inserted. With the cursor there, the Edit Report Form screen displays the printer control string in the area just under the help box at top of the Edit Report Form screen.

One thing about the Printer Control option. Any attempt to insert an ASCII 27 (the leftward arrow that starts laser printer control strings) will be replaced with

<27>

You can either enter

<27>

directly or enter an ASCII 27 and let DataPerfect convert it automatically to a

<27>

Option C (Open Filename in Report Variable)

Introduced with DataPerfect 2.3c, this option lets you direct disk file output to the value found in a Report Variable. This can be conveniently used to allow the user to decide the filename of the output disk file, or you can use a Report Variable formula to determine the filename. I discuss this option in detail in the *Open Filename in RV* section of my **Printer Control** chapter.

Eliminate Line if Blank

Ctrl-F7, 2 places the Eliminate Line if Blank code in any Edit Report Form screen section. Its purpose is straightforward. You may have fields that are occasionally blank, and you don't want them printed when blank. This would be typical of series of fixed-length fields that make up an address. You might allow the user to enter information in more than one line before tabbing to the City field in the following panel:

Acct	Last Name	First Name
Address 1		
Address 2		
City, ST, Zip		

In the above panel, not all records will have data in both Address fields. Here's a simple report that lists people and their addresses:

-----FIRST PAGE HEADER-----	
--Empty--	
-----OTHER PAGE HEADER-----	
--Empty--	
-----TWO-LEVEL REPORT HEADER-----	
--Empty--	
-----REPORT BODY-----	
-----, -----	Name fields
-----	Address 1 field
-----	Address 2 field
-----	City, ST, Zip fields
-----TWO-LEVEL FOOTER-----	
--Empty--	
-----PAGE FOOTER-----	
--Empty--	
-----FINAL FOOTER-----	
--Empty--	

Without the Eliminate Line if Blank code available to us, that Report Definition could easily print a list like this:

```
Adams, Sally
123 Elm St.

Los Angeles, CA 90024

Jackson, Sam
Jackson and Associates, Inc.
1245 Oak St.
Freeware, CA 90002

Smith, John

2345 Woodley Ave.
Perfection, CA 90034
```

By putting the Eliminate Line if Blank code to the left of each Address field in the Report Body, that same report prints like this:

Adams, Sally
123 Elm St.
Los Angeles, CA 90024

Jackson, Sam
Jackson and Associates, Inc.
1245 Oak St.
Freeware, CA 90002

Smith, John
2345 Woodley Ave.
Perfection, CA 90034

When you put the Eliminate Line if Blank code in a report, you won't see it where you inserted it. You only know it's there when you cursor to that spot and watch the screen, just below the help box. You'll see the *Eliminate Line if Blank* phrase appear.

The Eliminate Line if Blank code will keep its line from printing as long as that code is all the way to the left on that line, all the fields on that line are blank, and there's no text typed in the Edit Report Form screen on that line.

Page Eject and Skip to Bottom of Page

With the exception of the Other Page Header section, you'll find either the Page Eject code or the Skip to Bottom of Page code available to you everywhere in the Edit Report Form screen. So I'll include them here in the Global Report Options section.

Though these two codes appear very similar, they're different. The Page Eject code issues a form feed to the printer or disk file. You'll find Page Ejects in a disk file as ASCII 12 characters (♀). The Skip to Bottom of Page code, however, fills the remainder of the page with enough blank lines to cause the printing to go to the next page.

This difference in behavior has consequences. If you use Page Eject, DataPerfect issues a form feed to the printer or disk file *without sending a footer* to the printer or disk file. Whereas, because Skip to Bottom of Page fills the remainder of the page with blank lines, DataPerfect processes the appropriate footer, if present. The Page Eject code was originally offered to allow the definer to send a form feed to a laser printer, since they don't automatically eject a page at the end of a print job.

Prompt for Report Variable

Though the Prompt for Report Variable code isn't available anywhere but the First Page Header's Report Options menu (**Ctrl-F7, 6**), there are frequently reasons to Block and Move it to other sections in the Edit Report Form screen. So I'll include it in the Global Report Options section.

After hitting **Ctrl-F7, 6**, you're prompted for three things:

- The Report Variable number
- The Prompt (single line, up to 78 characters)
- The Report Variable format

The Report Variable number is the number of the Report Variable that will get the value the user enters at the prompt. The Prompt is the language the user will see when the prompt is active, like

Start Date:

The Report Variable format is no different than a normal field format. It's the format of the field the user will see and use when they type in their response.

Iteration Control

Though this option shows up in only two Edit Report Form screen sections (First Page Header and Report Body), you can Block and Move its various options to other sections. I offer an extensive discussion of this in my **Iteration Control** chapter.

Section-Specific Report Options

This section deals with Report Options that never occur in more than one Edit Report Form screen section, and should never be Blocked and Moved to sections other than where they can be called from the Report Options menu.

Section-Specific Report Options for First Page Header

Two-Level Report, Do Report in Subgroups

I discuss these in the **Reports: General Structure** chapter.

Create Record From Panel List

If you choose Create Record From Panel List, DataPerfect offers you the Panel List. After you choose a panel with **Enter**, DataPerfect returns you to the Edit Report Form screen, where you'll find a new Report Body waiting for you to store Report Variable values in various fields to create the new record. This option is also offered in the Final Footer (**Ctrl-F7, 5**) section, and as a submenu option in the Report Body section (**Ctrl-F7, 6, 3**).

Create Secondary Merge Report

This is an interesting enhancement introduced with version 2.3. It allows you to quickly create a WordPerfect Merge file from the current panel. When you choose it, DataPerfect asks you for the first field in the merge file. You may choose one from the current panel by hitting **1**, or insert a Report Variable by hitting **2**. If you choose to insert a Report Variable, DataPerfect will ask for the Report Variable's number and format. In either case (choosing a field or a Report Variable), DataPerfect allows you to give the resulting merge field a field name. You do this with as many fields as you want. When you exit this menu, you have a report that will produce a WordPerfect Merge file.

This is best learned by playing with it. It's quite convenient and simple. Try both with field names and without them. Examine the difference in the resulting First Page Header in each case. Note also that you can exit and come back later to add more fields by choosing this option again in the First Page Header. In that case, DataPerfect will simply add those to the end of the line in the Report Body. You can always edit this report later, moving fields around, checking Report Variable formulas, and reformatting fields.

Section-Specific Report Options for Other Page Header

Include Before First Record

Use this when you want the Other Page Header to show up on the first page as well as subsequent pages. Without this code, the Other Page Header won't print on the first page. This is usually used when, after thinking it over a little, you realize that your First Page Header and Other Page Header should be identical. In that case, you don't need a First Page Header. Just put the Include Before First Record code in the Other Page Header, at the very end of that section (following its contents). When you do, you'll see *Include Header Before Data* inserted in the Other Page Header.

Skip if Start of Two Level

You use this if, when running a Two-Level Report, you don't want the Other Page Header to appear at the top of the page when the Two-Level Header is about to print. That is, use this when you don't want both the Other Page Header and the Two-Level Header to both appear on the top of a page.

Section-Specific Report Options for Two-Level Header

Conditional Page Eject

DataPerfect also offers this in the Report Body. This code works slightly differently in each section. When used in the Two-Level Header section, this code makes sure that a group of records in a Two-Level Report doesn't split across two pages. When used in the Report Body, it makes sure that a *record* doesn't split across two pages.

Section-Specific Report Options for Report Body

Conditional Page Eject

DataPerfect also offers this in the Two-Level Header. This code works slightly differently in each section. When used in the Report Body, this code makes sure a record doesn't split across two pages. When used in the Two-Level Header section, it makes sure a *group of records* in a Two-Level Report doesn't split across two pages.

Labels

This is pretty straightforward and easy to use. When choosing this option in the Report Body, DataPerfect asks for the number of labels across each page, the width of each label in characters, and the number of lines per label. After supplying

DataPerfect with this data, DataPerfect inserts a code in the Report Body, informing you of the data just supplied. The code, which should be inserted at the beginning of the Report Body, before any selected fields, will look something like this:

```

-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----3 Records per Line. Record Width & Depth: 25 6 -----
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

The number of lines per label (in this case, six) includes lines that will be blank. The above report will print four lines of text per label, but the labels accommodate six lines.

One important note. For labels, set your Bottom margin (on the Initial Report Definition Screen) to 99 to avoid page eject problems. Though setting that to 0 will work on laser printers (because they work with physical pages), setting it to 0 may give you problems if you're using a continuous feed of labels on a dot matrix printer. Though a Bottom margin setting of 0 keeps DataPerfect from issuing a page eject, it stills lets the printer issue its internal page eject, which, by default, occurs after 60 lines with a laser, and after 66 lines with a dot matrix. A Bottom margin setting of 99 keeps the printer from issuing any page eject at all.

Subreports

This option produces the following submenu:

```

-----Subreports & Record Creation-----
1 - Include Subreport
2 - Create Record Through Link
3 - Create Record From Panel List
4 - Subreport Using Virtual Link
0 - Return to Edit
-----Selection: 0-----

```

Include Subreport and Subreport Using Virtual Link are covered quite extensively in my **Subreports** chapter. If you choose Create Record Through Link, DataPerfect asks you for a link in the current panel. After you select it with **F4**, DataPerfect returns you to the Edit Report Form screen, where you'll see a new Report Body waiting for your input. Alternatively, if you choose Create Record From Panel List, DataPerfect offers you the Panel List. After you choose a panel with **Enter**, DataPerfect returns you to the Edit Report Form screen, where you'll find a new Report Body waiting for your input. In either case, you can store Report Variable values in various fields to create the new record. This option is also offered in the First Page Header section (**Ctrl-F7, 8**) and Final Footer section (**Ctrl-F7, 5**).

Record Number

This option inserts a GZZZZZ9 Record Number field in the Report Body. It simply prints the current record number for that Report Body section of the report definition (it starts over with each Report Body section). You can reformat the field with **F6**.

Delete Record

This is simple and dangerous. Use it with caution. Choosing this option in the Report Variable will cause each record the Report Body sees to disappear from the database. Each deletion has the same effect as manually deleting the record in Browse mode, so any Keep A Total codes in the panel will trigger and update their target panel records. See the **Keep A Total** chapter if you don't know what this means.

Section-Specific Report Options for Two-Level Footer

Number of Records in Section

Very simple. It prints the number of records processed in that Two-Level group of records. It's a GZZZZZ9 field you can reformat with **F6**.

Section-Specific Report Options in Page Footer

Include After Last Record

Normally, the Page Footer isn't printed on the last page. That's the job of the Final Footer. If you choose this code, *both* will print on the last page.

Number of Records on Page

Straightforward. It's a GZZZZZ9 field you can reformat with **F6**.

Section-Specific Report Options in Final Footer

Number of Records in Report

This code prints the number of records processed by that report or subreport. It's a GZZZZZ9 field you can reformat with **F6**.

Create Record From Panel List

If you choose Create Record From Panel List, DataPerfect offers you the Panel List. After you choose a panel with **Enter**, DataPerfect returns you to the Edit Report Form screen, where you'll find a new Report Body waiting for you to store Report Variable values in various fields to create the new record. This option is also offered in the First Page Header (**Ctrl-F7, 8**) section, and as a submenu option in the Report Body section (**Ctrl-F7, 6, 3**).

Before long, you'll use Report Variables routinely when defining your reports. There's almost no way around this, so let's discuss them. This chapter is for both beginners and the experienced DataPerfect application developer.

Introduction

A Report Variable is a report entity that takes on a value temporarily during the life of a report. DataPerfect allows up to 255 Report Variables per Report Form. You can create a Report Variable in one of two ways in an Edit Report Form screen.

One way is to prompt the user for it (**Ctrl-F7, 6**) in the First Page Header. When you do this, DataPerfect asks you for the Report Variable number, the prompt (up to 78 characters), and the field format (any format at all). When the user sees this prompt, and fills in the prompt field, DataPerfect will then stuff the assigned Report Variable with the user's reply. Though DataPerfect only offers the Prompt For code in the First Page Header, you can actually place it in other sections. To do this, just create it in the First Page Header and move it elsewhere with Block, Cut and Paste (**Alt-F4, Arrow, Ctrl-F4** to Cut it, and **Ctrl-F4** to Paste it).

The second way to create a Report Variable is to define it directly. You can do this in any Edit Report Form screen section with **Ctrl-F4, 1, 4**. In every section, the steps for creating a Report Variable are the same after **Ctrl-F7, 1, 4**. You're asked for a number to assign this Report Variable, and then offered a Specify Formula Screen that's blank, except for the following at the top:

Specify Formula—

Operands can be numbers, character strings in quotes (" or '), or fields selected with Select (F4). To select a field from another panel, move to a link and press !.

Operators are: + - * / // ^ < > <= >= <> = NOT AND OR.

Parentheses () should be used to group items.

When finished, press Save (F10) or Cancel (F1).

That Specify Formula Screen is the same one you see when defining field formulas in panels. It works the same in each case. Here we're telling DataPerfect what value to assign the Report Variable we just named with a number from 1 to 255. Via the Specify Formula Screen, we can assign our Report Variable any legal value or well-formed formula. All the same rules about defining field formulas apply here, so read up on that in the **Formulas** chapter if you're unclear about how to define a formula, and what constitutes a well-formed formula. Let's discuss some basic reasons you might need to use a Report Variable.

Printing Data Not Already in Fields

One reason you might need to use a Report Variable is to print data that isn't in any of the available fields in the database. For instance, let's say your Teacher Panel has a field for the Teacher's sex. It's a U1 field, taking the values *F* or *M* for Female or Male. You'd like to print a series of letters, one per Teacher, wishing them a happy new year, and you'd like each to start with *Dear Ms.* or *Dear Mr.* Well, you don't need a field to select with **F4** in this case. Just create a Report Variable that looks at the Sex field in the Teacher Panel and takes on a value of *F* or *M* depending on what it sees in the Sex field.

Let's say our Teacher Panel, which was Panel 2, now looks like this:

TEACHER PANEL	
Teacher ID	
.....	
Last Name	First Name
.....
Sex	
.....	
To Class/Teacher Panel	
.....	

If the Sex field above is P2F4, and we're setting Report Variable 1 to hold either 'Ms.' or 'Mr.', then, Report Variable 1's formula could look like this:

```
if P2F4="F" then "Ms." else "Mr." endif
```

Provided P2F4 only holds *F* or *M*, this will work. The above formula sets Report Variable 1 to either 'Ms.' or 'Mr.', depending on what it sees in P2F4.

Note: A way to make sure DataPerfect forces the user to decide between *F* or *M* in P2F4 is to format it *::M* and give it a field formula like this:

```
if P2F4="F" or P2F4="M" then P2F4 else "" endif
```

This way the field **formula** forces either *F*, *M*, or blank, while the field **format** prohibits the blank, forcing the user to decide between *F* or *M*.

We want our Report Definition to reset Report Variable 1 each time it sees a record, so it should be in the Edit Report Form screen's *Report Body*. If we set Report Variable 1 in the First Page Header, it will only be set once (when DataPerfect sees the first record in the report's index). Likewise, if we set Report Variable 1 in the Other Page Header or Two Level Report Header, it will only be reset each time

DataPerfect sees the top of a page or the beginning of the a Two Level group. But, again, we need Report Variable 1 to be reset whenever DataPerfect sees a new record in the current index. So, while in the Report Body, we **Ctrl-F7, 1, 4** to tell DataPerfect we want to *Store Value in Report Variable*. After giving that Report Variable a number (say, 1), we give it this formula:

```
if P2F4="F" then "Ms." else "Mr." endif
```

After exiting the Specify Formula Screen, we see this:

```
-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Store Value in Report Variable 1 -----
-----
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
```

That extra blank line after the *Store Value* code is always there, so don't bother trying to delete it. It won't print an extra line.

But that isn't enough. All we've done so far is store a value in a Report Variable. This assignment is just sitting in memory, doing nothing interesting. We must now tell DataPerfect we want the report to *print* that Report Variable for each record processed. We do that with DataPerfect's Print Report Variable option (**Ctrl-F7, 1, 5**). When we **Ctrl-F7, 1, 5**, DataPerfect asks us for the number of the Report Variable to print (in this case, 1) and the format of the field to be printed. When asked for a format, put in what makes sense. You're not affecting any field in any panel. Rather, you're telling the report to make up a temporary field on the fly for the purposes of printing this report. In this case we want an A3 field to accommodate the three characters in 'Ms.' or 'Mr.'. Answering the Print Report Variable prompts, our Edit Report Form screen now looks like this:

```
-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Store Value in Report Variable 1 -----
-----
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
```

The cursor above is immediately to the right of the field just inserted. If you move the cursor to the left, into the field itself, the screen changes, allowing you to see that the field isn't one selected with **F4**. Rather, it's Report Variable 1 to be printed in A3 format. The screen should change to look like this:

```

Report Variable 1

Field Format: A3
-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Store Value in Report Variable 1 -----
Dear _____,
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

We continue defining our letter writing report by selecting the Last Name field in the Teacher Panel (again, with **F4**), and adding text:

```

Report Variable 1

Field Format: A3
-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Store Value in Report Variable 1 -----
Dear _____,
We are pleased to ....

[rest of letter goes here]

-----Skip to Bottom of Page-----
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

After I selected the Last Name field above with **F4**, I censored to it and reformatted it to A15;;T (using **F6**) to remove trailing spaces that might come between the Last Name and the comma. I also used **Ctrl-F7, 4** to put in a *Skip to Bottom of Page* code at the end of the Report Body, after the whole letter is typed in.

So, in general, what will the above report do? Each time it comes to a Teacher Panel record in the report's index, it first checks to see what value is in that record's Sex field and stores *Ms.* in Report Variable 1 if it sees an *F* there, or *Mr.* if it sees an *M*. It hasn't printed anything yet, however. Next, it prints *Dear* followed by whatever is stored in Report Variable 1. Then it prints the Teacher's Last Name. If there are spaces after the Last Name in that A15 field, it removes them, and then prints a comma. Then it prints the rest of the letter. When done with that letter, it skips to the bottom of the page and starts the next page with the next record in the index, if more records exist.

So, as you can see, a Report Variable stores a value for later use. In this case, it was stored so it then could be printed. Just as common, a Report Variable is used

to store a value or a statement that will later be used to determine what records should be processed, and what should be skipped. I'll cover that next.

Skipping Certain Records

What if you'd like to send letters to all female teachers in the database to see if they'd be interested in forming a women's group in the school to discuss various issues related to women. Though there are more efficient ways to construct this report, let's do use a simple way that employs Report Variables.

First, we need a Report Variable to examine each record the Report Body sees, with an eye to the value in its Sex field. We also need to use one of the options found in the Iteration Control menu, accessed with **Ctrl-F7, 9** in the Report Body or **Ctrl-F7, A** in the First Page Header:

```
Report Iteration Control
1 - Skip Record if REPORT VARIABLE is False
2 - Stop [Sub]Report if REPORT VARIABLE is False
3 - Skip To Record At REPORT VARIABLE
4 - Choose Next Record Using LookUp
5 - Repeat Record if REPORT VARIABLE is True (not 0)

Selection: 0
```

Take a look at option 1: *Skip Record if REPORT VARIABLE is False*. If we can come up with a Report Variable that's false for just those records we want to skip (or equivalently, true for just those records we want to include), we're set. But we first must create such a Report Variable that will serve as the condition to be evaluated for each record DataPerfect comes to. That is, simply choosing **1** from the above menu won't help here because it's going to ask you what Report Variable it's supposed to evaluate, and you haven't created one yet.

So we go the beginning of the Report Body and **Ctrl-F7, 1, 4** to set Report Variable 1. Given that we're going to use Report Variable alongside Iteration Control Option 1 (Skip Record if REPORT VARIABLE is False), we need to come up with a formula for Report Variable 1 that will be true for all and only those records we want the report to consider for printing out letter. The easiest way to do this is like this:

```
P2F4="F"
```

Again, we do that in the Specify Formula screen by first hitting **F4** (which puts us in the Teacher Panel), then cursoring to the Sex field (which is field 5 in this case), and hitting **F4** again. Then we type in the rest of the formula. This formula says the value in the Sex field is *F*. That statement is true for all and only female Teachers in the Teacher Panel database. Now we have an Edit Report Form screen like this:

```

-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Store Value in Report Variable 1 -----
-----
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

Next we want to tell the report to skip those records that make Report Variable 1 false. Again, we do this with **Ctrl-F7, 9, 1**, which prompts us for the number of the Report Variable it's supposed to check. We tell it *1* and now have this Edit Report Form screen:

```

-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Store Value in Report Variable 1 -----
-----Skip Record if 0 (False) Is in Report Variable 1 -----
-----
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

This Edit Report Form screen says, for each record in the index, check to see if the Sex field has *F* in it, and skip the record if not. Note that the precise wording of the code that's actually placed in the Edit Report Form screen is slightly different than it was in the menu:

```

Iteration Control Menu:
    Skip Record if REPORT VARIABLE is False

```

```

Edit Report Form screen:
    Skip Record if 0 (False) Is in Report Variable 1

```

In DataPerfect lingo, they mean the same thing. A Report Variable is considered false as long as it's either a false statement or it's a numerical value of 0. I discuss DataPerfect's notion of truth in great length elsewhere (see *A Note about DataPerfect's Notion of Truth* in my **Iteration Control** chapter). With this in mind, I could have written the formula less elegantly, but just as correctly, as follows:

```

if P2F4="F" then 1 else 0 endif

```

If you understand the above formula is equivalent in DataPerfect to

```

P2F4="F "

```

you'll go a long way in understanding DataPerfect's logic. For reasons that elude me, many DataPerfect application developers prefer the less elegant formula above. To each his own. They both work fine.

If you use the less elegant formula, then, for each record, Report Variable 1 will be a *numerical value* of either 0 or 1, instead of a *statement* that's either true or false. A Report Variable that's a numerical value of *anything other than 0* is considered true by DataPerfect.

The rest of the Edit Report Form screen would be filled in *after* the two codes we have in there now. If you put the text in *before* the two codes, DataPerfect won't examine the current record for its Sex field value until after already printing the letter. We need DataPerfect to examine the Skip Record condition first.

As before, we would put in a *Skip to Bottom of Page* code in the Report Body, immediately after all the text:

```
-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Store Value in Report Variable 1 -----
-----Skip Record if 0 (False) Is in Report Variable 1 -----

        .....

        [letter goes here]

        .....

-----Skip to Bottom of Page-----
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
```

The above report will print letters only to females in the Teacher Panel database.

Self-Referencing Report Variables

Counters

Using Report Variables as counters usually involves using Report Variables that refer to themselves, so I want to take this up now to clear up what might seem confusing. Let's say you wanted a report that would tell you how many teachers in the database are male and how many are female. To do this, you would need two Report Variables: one that will count the males, and one the females.

In the First Page Header of this report you set each Report Variable to 0, getting each ready to count. To set Report Variable 1 to 0 you **Ctrl-F7, 1, 4** and tell DataPerfect you're interested in Report Variable 1. When you see the Specify Formula screen you simply type *0* and then exit with **F7**. That sets Report Variable 1 to 0 in the First Page Header, before the report even sees the first record in the

report's index. You do the same thing for Report Variable 2. Now the Edit Report Form screen looks like this:

-----FIRST PAGE HEADER-----] Report Variables set to 0
-----Store Value in Report Variable 1 -----	
-----Store Value in Report Variable 2 -----	
-----OTHER PAGE HEADER-----	
--Empty--	
-----TWO-LEVEL REPORT HEADER-----	
--Empty--	
-----REPORT BODY-----	
--Empty--	
-----TWO-LEVEL FOOTER-----	
--Empty--	
-----PAGE FOOTER-----	
--Empty--	
-----FINAL FOOTER-----	
--Empty--	

Now we need Report Variable 1 to add 1 to itself with each male record it sees in the index, and Report Variable 2 with each female. This must take place in the Report Body, since no other part of the Report Definition will see each and every record in the database. By the time the report finishes the last record, Report Variable 1 will hold the total number of males, and Report Variable 2 the females.

To accomplish this, we must reset each variable in the Report Body. If, as before, P2F4 is the Sex field, then we can set Report Variable 1 to this formula in the Report Body:

```
if P2F4="M" then rv1+1 else rv1 endif
```

The above formula, when being stored in a Report Variable I the Report Body, says this:

```
If the Sex field has M in it, then add 1 to Report  
Variable 1, otherwise leave Report Variable 1 alone.
```

Note that Report Variable 1's formula refers to Report Variable 1. What it's doing is referring to the value that is stored in Report Variable 1 at that moment, and then deciding what the value of Report Variable 1 will be next. It's taking the value it is at the moment and adding 1 to it if the current record has *M* in the Sex field, and then making that the new value for Report Variable 1. If the Sex field doesn't have *M* in it, this formula still stores a value in Report Variable 1, but the value it stores is what it was before examining the Sex field's value. That is, it stores back into Report Variable 1 the value it finds there, unchanged.

The reason I spend time on this is that I see some beginning developers getting confused when they see a Report Variable's formula referencing the same Report Variable. Just think of this as referencing the *value* that Report Variable currently has. That's all.

Now, of course, Report Variable 2 will have this formula in the Report Body:

```
if P2F4="F" then rv2+1 else rv2 endif
```

That formula says

If the Sex field has *F* in it, then add 1 to Report Variable 2, otherwise leave Report Variable 2 alone.

This Edit Report Form screen now looks like this:

```
-----FIRST PAGE HEADER-----
-----Store Value in Report Variable 1 -----
-----Store Value in Report Variable 2 -----
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Store Value in Report Variable 1 -----
-----Store Value in Report Variable 2 -----
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
```

RVs set to 0

RVs increment based on Sex field value

Now all we have to do is print the results in the Final Footer, which is the section that prints after all records in the index are processed. That would be done with **Ctrl-F7, 1, 5** twice in the Final Footer—once for Report Variable 1 and once for Report Variable 2. Each time you tell DataPerfect an appropriate field format. If you know there are more than 99 teachers in the database, but less than 1000, GZZ9 would make sense. So now our Edit Report Form screen looks like this:

```
-----FIRST PAGE HEADER-----
-----Store Value in Report Variable 1 -----
-----Store Value in Report Variable 2 -----
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Store Value in Report Variable 1 -----
-----Store Value in Report Variable 2 -----
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
Total female teachers: 
Total male teachers: 
```

RVs set to 0

RVs increment based on Sex field value

RVs printed

Recycled Report Variables

There are basically two reasons you might want to recycle a Report Variable. One would be because you're butting up against the 255 Report Variable limit per Report Form. This would certainly be rare, though I've seen at least one DataPerfect application developer come up against that limit frequently, as hard as that is to believe. The other reason would be to keep the Report Form's assignment of Report Variables easier to follow.

Here's what I mean by recycling Report Variables. Say you just prompted the user for a value for Report Variable 1. Further, let's say it's to hold a value for the Date field in the Transaction Panel. What you'd like the report to do is default Report

Variable 1 to today's date if the user leaves the prompt field blank and hits **Enter**. The prompt for Report Variable 1 might look like this to the user:

```
Enter date for this Day Sheet (hit Enter for today's date):
```

To test to see if the user simply left the prompt field blank before hitting **Enter**, you would typically use a new Report Variable, say Report Variable 2, with a formula like this:

```
if rv1=0 then today else rv1 endif
```

The report would then proceed to use Report Variable 2 instead of Report Variable 1 for the value of the Date field.

But using Report Variable 2 in the above example isn't necessary. We could just as easily use Report Variable 1 again, assigning that formula to Report Variable 1 and not bother creating Report Variable 2. So Report Variable 1 would have the following formula, referencing itself:

```
if rv1=0 then today else rv1 endif
```

The First Page Header of this report, then, would look like this:

-----FIRST PAGE HEADER-----	
-----Prompt for Value of Report Variable 1 -----	Date field prompt Convert RV1 with above formula
-----Store Value in Report Variable 1 -----	
-----OTHER PAGE HEADER-----	
--Empty--	

By recycling Report Variable 1 here we keep our Date field value in the same Report Variable, not having to remember what Report Variable we passed it to.

This chapter is for both beginners and the experienced. If you're not using subreports, you're not tapping the power within DataPerfect's report facility. Much of this chapter will lose the beginner.

Introduction

This can be very simple in many respects, very complex in others. When I previously outlined what a subreport is, I used a simple (and typical) report. It listed teachers and their class assignments. Under each teacher's name was listed their class assignments, one class per line. A subreport here is simply a Report Definition facility that allows you to print a report that lists children records for each parent. This might be a report that lists all classes taught by each teacher in a school, or all items purchased by each customer in a business. Typically you use a subreport to show all records in a subpanel that are attached to the current record.

Let's take a look at our Subreport Options menu, accessed with **Ctrl-F7, 6** in the Report Body:

```
-----Subreports & Record Creation-----
1 - Include Subreport
2 - Create Record Through Link
3 - Create Record From Panel List
4 - Subreport Using Virtual Link
0 - Return to Edit
-----Selection: 0-----
```

I'm interested here in options 1 and 4.

Option 1 - Include Subreport

This option depends on the existence of a link in the current panel, and will typically be used with a panel link, not a data link. When you choose that option, you'll be asked one thing: the link upon which to base the subreport. By choosing that link, you've chosen the target panel, the index that will sort records in that subreport, and the field list that will filter records upon entry into that subreport. All three of these characteristics are part of the definition of the link you already defined when in Panel Define mode. If you don't have a link that has just the characteristics you want for the subreport, and still want to use option 1 instead of option 4, you'll need to exit Report Definition mode and create such a link in Panel Define mode. Then you can re-enter Report Definition mode and start creating a subreport again.

So if you're creating a Report Definition that prints a list of Transactions for each Invoice, you'll base the report on the Invoice Panel and then create the

appropriate subreport in the Report Body of the Main Report. If you want to use option 1 in your Subreport Options menu, you'll need a panel link in the Invoice Panel that takes you to the Transaction Panel. Now let's say the only panel link in that panel takes you to the Transaction Panel using a Transaction Panel index that sorts backwards by date. That is, you designed this application to allow the user to always land on the most recent Transaction in the Transaction Panel when using the panel link in the Invoice Panel.

But you want the report to list Transactions in order of occurrence, from first to last. Well, if you still want to use option 1 in your Subreport Options menu, you'll need to exit Report Definition mode and get into Panel Define mode in the Invoice Panel and create a panel link that uses a Transaction Panel index that sorts Transactions forward by date. If you don't want the user to ever penetrate that link during data entry or while browsing records, you'll have to hide it by choosing option 5 in your Define Panel Link menu (**Shift-F8**):

```

Define Panel Link
Link to Panel:3 Field2 Index5 Field List to Build Key:1
1 - Edit Target Field/Target Index/Field List
2 - Define Related Records Window      5 - Display/Hide Link
3 - Create/Edit Window Field List      6 - Define Lookup List
4 - Delete Window                      7 - Cascade Off
Window Off
Selection: 0

```

Subreports: Going From Version 2.2 to 2.3

A subreport that used to work fine under DataPerfect 2.2 may, under 2.3, seem to produce its first printed line on the same line as the last line of its parent's Report Body. For instance, consider a DataPerfect 2.2 report printout that used to look like this:

MONTHLY STATEMENT			Report Body of Main Report
11/30/96			
Alex's Office Supplies			
17 Elm St.			
Anywhere, CA 90024			
Jim Jacobson			Acct 1234
234 Oak Ave.			
Somewhere, CA 90212			
Date	Transaction	Amount	
=====	=====	=====	
11/21/96	Backup tape	27.00	Report Body of Subreport
11/21/96	Copier paper	12.50	
11/28/96	Manilla files	19.99	
11/28/96	Printer toner	29.50	

Under 2.3, the same Report Definition might print something like this:

Date	Transaction	Amount	
=====	=====	=====	
11/21/96	Copier paper	12.50	2
11/28/96	Manilla files	19.99	
11/28/96	Printer toner	29.50	

This is the one area of your DataPerfect 2.2 applications that had to be slightly rewritten after upgrading to 2.3. What you see in the second printout above is the result of an enhancement. Starting with version 2.3, DataPerfect subreports don't automatically initialize with a carriage return and line feed. They *did* with earlier versions of DataPerfect, but no longer with 2.3.

If you want the subreport to begin on the line following the last line printed by its parent record's Report Body, you have to manually insert a carriage return by hitting **Enter** in the Edit Report Form screen, just before the subreport. You won't see anything happen when you hit **Enter** there, because the Edit Report Form screen doesn't show that carriage return. The only way to tell if it's there is to position the cursor in the parent Report Body, just before the subreport. Then hit the **Del** key. If you just deleted a carriage return, you won't notice anything happen on the screen. If you just attempted to delete the subreport itself, you'll be prompted as to whether or not you really want to do that.

Here's what that report looks like, with a reference as to where to put the cursor before hitting the **Del** key:

```
-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
MONTHLY STATEMENT

[REDACTED]

[REDACTED]

[REDACTED] Acct [REDACTED]

Date      Transaction      Amount
=====
SUBREPORT LINK/PANEL: 4 3
-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
[REDACTED]
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
-----END OF SUBREPORT-----

-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
```

This change in subreport behavior with 2.3 is actually an enhancement. It allows you to use subreports to produce reports whose subreports begin on the last line of the Report Body that calls them. Such a report would look something like this:

```

Jim Samuels          Art Appreciation 101
                    Art History 112
                    Art History 113

Sally Adams          Math 111
                    Math 201

etc.

```

Note that the subreport above begins printing on the same line here as its parent record in the main report leaves off. You achieve that in 2.3 by just making sure no hidden carriage return exists between the Report Body and its subreport.

The Edit Report Form screen that produces the above report could look no different than the one that produces this one:

```

Jim Samuels          Art Appreciation 101
                    Art History 112
                    Art History 113

Sally Adams          Math 111
                    Math 201

etc.

```

The only difference could be the hidden carriage return.

Subreports as Subroutines

The above examples are typical ones, where the subreport is used to list items attached to a parent record. But you greatly increase your possibilities here if you consider a subreport to really be a *subroutine*. Let's explain.

Suppose you'd like a report that prints Invoices to not only list all Transactions attached to an Invoice, but you'd also like it to show the total at the *top* of each Invoice. That is, you'd like to see each Invoice look something like this:

Jim Alexander 1234 Elm St. Anywhere, CA 90024			
Customer No.: 1544 Invoice No. : 0134 Total Due : \$345.00			
01/02/96	Secretarial chair	145.00	
02/05/96	Bookcase	100.00	
02/08/96	Small File Cabinet	100.00	

If you want the total to appear at the top of each Invoice, you'll have to compute it *before* listing the items. DataPerfect makes it easy to put a total at the *end* of each Invoice, but putting it in the *beginning* takes a little more thought.

Let's first see how to put a total at the end, since I said that's the easiest way to do it. When you select (**F4**) a numerical field *in the Final Footer* of a report or subreport, DataPerfect offers you some special options:

```

Numeric Field Options in a Footer
1 - Total (Sum)
2 - Average
3 - Maximum Value
4 - Minimum Value
5 - Standard Deviation
0 - Field As Is From Current Record
Selection: 0

```

So, if you select the Amount field in the Transaction Panel in your subreport's Final Footer, DataPerfect will show you the above intervening menu. If you choose option 1, DataPerfect will insert that field into your Report Form as usual, but it will now print the total for those records in the subreport for that parent record (the total of Transactions for that Invoice). A field chosen this way looks no different than one chosen any other way with **F4**, so to tell the difference between them, cursor to the field in the Final Footer (this sort of field only occurs in a Final Footer) and look at the area just under the Help screen. You'll see something like this:

```
Path to field: PlF1
  Amount
Total Field Value for entire report
Field Format: GZZ9.99::N
```

This tells you the field is printing the total for the Amount field (if that's what you named it), as gathered for records (Transactions) in that subreport for that particular parent record (Invoice).

Here's what such a report would look like, assuming the Invoice Panel is Panel 1 and the Transaction Panel is Panel 2, and the panel link that takes you from the Invoice Panel to the Transaction Panel is field 4:

<pre> FIRST PAGE HEADER --Empty-- OTHER PAGE HEADER --Empty-- TWO-LEVEL REPORT HEADER --Empty-- REPORT BODY XXXXXXXXXX XXXXXXXX XXXXXXXXXX XXXXXXXXXX, XX XXXXXXXX Customer No.: XXXX Invoice No. : =====Subreport LINK/PANEL: 4 2 ===== FIRST PAGE HEADER --Empty-- OTHER PAGE HEADER --Empty-- TWO-LEVEL REPORT HEADER --Empty-- REPORT BODY XXXXXX XXXXXXXX XXXXXXXX TWO-LEVEL FOOTER --Empty-- PAGE FOOTER --Empty-- FINAL FOOTER Total: XXXXXX =====END OF Subreport===== TWO-LEVEL FOOTER --Empty-- PAGE FOOTER --Empty-- FINAL FOOTER --Empty-- </pre>	<div style="margin-bottom: 10px;">Main Report (Invoice Panel)</div> <div style="margin-bottom: 10px;">Customer and Invoice info, which prints once per Invoice</div> <div style="margin-bottom: 10px;">Subreport (Transaction Panel)</div> <div style="margin-bottom: 10px;">Select Amount field Choose Total option</div> <div>Main Report (Invoice Panel)</div>
---	---

The above Report Form produces Invoices like this:

Jim Alexander			
1234 Elm St.			
Anywhere, CA 90024			
Customer No.: 1544			
Invoice No. : 0134			
01/02/96	Secretarial chair	145.00	
02/05/96	Bookcase	100.00	
02/08/96	Small File Cabinet	100.00	
Total:		345.00	

In the early days of DataPerfect, the above procedure wasn't reliable. Many of us still use a method we used to work around that problem. I still do. This alternative method uses a *counter*. This is a Report Variable that counts the money for each Invoice. Let's call it Report Variable 1. First, as with all counters, set it to 0 in the First Page Header of the subreport. Then, in the Report Body of the subreport (the subreport associated with the Transaction Panel) I'll set Report Variable 1 to

```
rv1 + P1F1
```

where P1F1 was selected with **F4** (it's the Amount field in the Transaction Panel). We would then print Report Variable 1 (**Ctrl-F7, 1, 5**) in the Final Footer of the subreport.

This Report Form looks something like this:

these entities: Schools, Teachers, and Students. The School Panel is linked to the Teacher Panel and is also linked to the Student Panel. From the School Panel you can go through a panel link to see what Teachers work there, and you can go through a different panel link in the School Panel to see what Students go there.

Suppose you want a report that prints all Teachers and Students connected with each School, like this:

School 1
Teachers
Teacher 101
Teacher 102
Teacher 103
etc.
Students
Student 101
Student 102
Student 103
Student 104
etc.
School 2
Teachers
Teacher 201
Teacher 202
etc.
Students
Student 201
Student 202
Student 203
etc.

To create such a report we base the report on the School Panel and the create a subreport to the Teacher Panel. After creating the Teacher Panel subreport, cursor just past it, but remain in the Report Body of the Main Report (just before the Two Level Footer of the Main Report). That's where you create the second *parallel* subreport, tied to the Student Panel. This Report Definition looks like this:

	FIRST PAGE HEADER	
	--Empty--	
	OTHER PAGE HEADER	
	--Empty--	
	TWO-LEVEL REPORT HEADER	
	--Empty--	
	REPORT BODY	
		School Name field
Subreport	LINK/PANEL: 4 2	
	FIRST PAGE HEADER	
Teachers		Subreport Teacher Panel
	OTHER PAGE HEADER	
	--Empty--	
	TWO-LEVEL REPORT HEADER	
	--Empty--	
	REPORT BODY	
		Teacher Name fields
	TWO-LEVEL FOOTER	
	--Empty--	
	PAGE FOOTER	
	--Empty--	
	FINAL FOOTER	
	--Empty--	
	END OF Subreport	
		Report Body of Main Report
Subreport	LINK/PANEL: 5 3	
	FIRST PAGE HEADER	
Students		Subreport (Student Panel)
	OTHER PAGE HEADER	
	--Empty--	
	TWO-LEVEL REPORT HEADER	
	--Empty--	
	REPORT BODY	
		Student Name fields
	TWO-LEVEL FOOTER	
	--Empty--	
	PAGE FOOTER	
	--Empty--	
	FINAL FOOTER	
	--Empty--	
	END OF Subreport	
		Main Report: School Panel.
	TWO-LEVEL FOOTER	
	--Empty--	
	PAGE FOOTER	
	--Empty--	
	FINAL FOOTER	
	--Empty--	

Now suppose your application has a fourth panel that joins Teachers and Students together. That is, each record in this Teacher/Student Panel has the name of a Teacher and a name of a Student assigned to that Teacher. A panel link in the Teacher Panel gives the user access to all and only records in the Teacher/Student Panel tied to the currently displayed Teacher in the Teacher Panel. Likewise, a panel link in the Student Panel gives the user access to all and only records in the Teacher/Student Panel tied to the currently displayed Student in the Student Panel.

In such an application you can easily create a report that prints a list showing Schools, with Teachers and their assigned Students. This report looks like this:

School 1
Teacher 111
Student 111
Student 112
Student 113
Student 114
etc.
Teacher 121
Student 121
Student 122
Student 123
etc.
School 2
Teacher 131

And the Report Definition that produces the above report looks like this:

=====	FIRST PAGE HEADER	Main Report (School Panel)
=====	OTHER PAGE HEADER	
=====	TWO-LEVEL REPORT HEADER	
=====	REPORT BODY	
=====		School Name field
=====	Subreport LINK/PANEL: 4 2	Subreport (Teacher Panel)
=====	FIRST PAGE HEADER	
=====	OTHER PAGE HEADER	
=====	TWO-LEVEL REPORT HEADER	
=====	REPORT BODY	
=====	=====, =====	Teacher Name fields
=====	Subreport LINK/PANEL: 5 3	Subreport (Student Panel) Placed in the Report Body of Subreport above, <i>not</i> in Report Body of Main Report
=====	FIRST PAGE HEADER	
=====	--Empty--	
=====	OTHER PAGE HEADER	
=====	--Empty--	
=====	TWO-LEVEL REPORT HEADER	
=====	--Empty--	
=====	REPORT BODY	
=====	=====, =====	Student Name fields
=====	TWO-LEVEL FOOTER	
=====	PAGE FOOTER	
=====	FINAL FOOTER	
=====	END OF Subreport	
=====	TWO-LEVEL FOOTER	Main Report (School Panel)
=====	PAGE FOOTER	
=====	FINAL FOOTER	

The second subreport in the Report Definition above was placed in the Report Body of the first subreport, not the Report Body of the Main Report. Thus, these are *nested* subreports.

Now, we started this section talking about considering subreports to be subroutines. I wanted to exemplify this with a report that prints the totals at the top of each Invoice instead of the bottom. Let's get into that now.

Again, as mentioned earlier, we need two parallel subreports to accomplish this. The first will compute the total for that particular Invoice and the second will print the actual invoice.

-----FIRST PAGE HEADER-----	Main Report
-----OTHER PAGE HEADER-----	(Invoice Panel)
-----TWO-LEVEL REPORT HEADER-----	
-----REPORT BODY-----	
=====Subreport LINK/PANEL: 4 2=====	
-----FIRST PAGE HEADER-----	Parallel Subreport 1
-----Store Value in Report Variable 1 -----	(Transaction Panel)
-----OTHER PAGE HEADER-----	
-----Empty-----	This Subreport
-----TWO-LEVEL REPORT HEADER-----	computes the total
-----Empty-----	
-----REPORT BODY-----	Set RV1 to 0 in 1st
-----Store Value in Report Variable 1 -----	Page Header to start
	counting for this
	Invoice. Set it to
	RV1+P1F1 in Report
	Body
-----TWO-LEVEL FOOTER-----	
-----Empty-----	
-----PAGE FOOTER-----	RV1 now holds total
-----Empty-----	for this Invoice
-----FINAL FOOTER-----	
=====END OF Subreport=====	
.....,	Back to Main Report
.....,	Report Body, just
.....,	before inserting
.....,	Parallel Subreport 2
Customer No.:	
Invoice No. :	
Total Due :	Print RV1
=====Subreport LINK/PANEL: 4 2=====	
-----FIRST PAGE HEADER-----	Parallel Subreport 2
-----Empty-----	(Transaction Panel)
-----OTHER PAGE HEADER-----	
-----Empty-----	This Subreport
-----TWO-LEVEL REPORT HEADER-----	prints the Invoice
-----REPORT BODY-----	
.....,	Detail fields
-----TWO-LEVEL FOOTER-----	
-----PAGE FOOTER-----	
-----FINAL FOOTER-----	
=====END OF Subreport=====	
-----TWO-LEVEL FOOTER-----	Remainder of Main
-----PAGE FOOTER-----	Invoice Panel
-----FINAL FOOTER-----	

Do you see why I call a subreport a subroutine? The first parallel subreport is a subroutine that computes a Report Variable that will be used by the second parallel subreport, and the second parallel subreport is a subroutine that sends data to the printer. I'm trying to get you to not limit yourself by thinking of subreports as simply sections of a Main Report that print child records. You can use a subreport to go off and compute a Report Variable, delete or create some records in another panel, etc., all before (or after) printing something.

Subreport Using Virtual Link

Let's take this a step further and take a look at a subreport option that was introduced with version 2.3 of DataPerfect. That's option 4 in your subreports menu (**Ctrl-F7, 6**):

```

-----Subreports & Record Creation-----
1 - Include Subreport
2 - Create Record Through Link
3 - Create Record From Panel List
4 - Subreport Using Virtual Link
0 - Return to Edit
-----Selection: 0-----

```

A Subreport Using Virtual Link is a subreport you create without basing it on an existing link. In the process of creating this sort of subreport, you're asked for the following, in this order:

- Target Panel
- Index
- Field List

Does that list look familiar? It should. Those are three of the four properties that characterize a link. All that's missing is the Target Field. When you create a Subreport Using Virtual Link, you're creating a temporary link and a subreport that will use that link. The link will disappear after the subreport ends.

This means you don't need a link to create a subreport. I suggest you use this option instead of option 1 (Include Subreport). This way, when considering whether or not to remove what may appear to be an unnecessary panel link, you no longer have to wonder if a subreport uses it.

When using Subreport Using Virtual Link instead of Include Subreport, you no longer have to wonder if there is an appropriate link for the contemplated subreport. Just create it on the fly when you create the Subreport using Subreport Using Virtual Link.

When creating a Subreport Using Virtual Link, you may create a kind of field list that can never be part of an existing link. Namely, you may create a field list that contains one or more Report Variables. At first, it may not be all that obvious to you as to why this is a powerful possibility. I'll bring this out with an example.

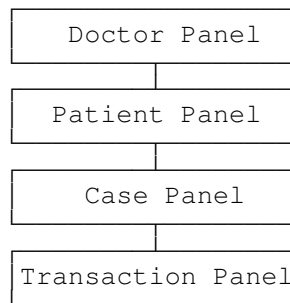
So, to review, Subreport Using Virtual Link option has the following advantages over the Include Subreport option:

- It doesn't need an existing panel link.
- Its field list accepts Report Variables.

Let's go over some examples that expose the power of Subreport Using Virtual Link. Suppose you have an application that keeps track of all Transactions for a multi-doctor office. In this application there exist at least the following three panels:

- Panel 1 - Doctor Panel
- Panel 2 - Patient Panel
- Panel 3 - Case Panel
- Panel 4 - Transaction Panel

Panel 1 has one record for each Doctor. Panel 2 has one record for each Patient. Panel 3 has a record for each Case for each Patient (a Case would be, say, a particular Patient's slip and fall in the shower in 1994, and then another record would be created when they have an auto accident in 1996). Panel 4 is obvious. Here's our panel hierarchy:



We want a Day Sheet report that will be based on the Doctor Panel and print all Transactions in the Transaction Panel for each Doctor, where all such Transaction Panel records have today's date in the Date field. Though there are other ways to construct this Report Definition, let's see how to use the Subreport Using Virtual Link to do it.

First, I'll load the Doctor Panel and hit **Shift-F7** and hit **Insert** on Built-In Short Reports. I now have a Main Report based on the Doctor Panel.

Second, I want to set Report Variable 1 to *today*, which is the date I'm going to use in the subreport in the Transaction Panel.

Third, I'm going to create a Subreport Using Virtual Link in the Report Body of the Main Report. The three properties of this Subreport Using Virtual Link will be the following:

Day Sheet Subreport Using Virtual Link	
Target Panel	Transaction Panel
Index	Doctor Code, Date, ...
Field List	Doctor Code, RV1

Even though I don't have a panel link in the Doctor Panel that leads to the Transaction Panel, I can still create this subreport linkage. Also, even if I *did* have such a panel link in the Doctor Panel, it probably doesn't have *today* on its field list.

Note: How would I put *today* on a panel link field list? Well, starting with the September 1993 version of DataPerfect 2.3 you can put computed fields (::C) on panel link field lists. So if you have a hidden computed date field that updates to *today* in that panel, you can just put that field on the panel link's field list, along with the Doctor Code field. This would give you access to all records in the Transaction Panel that have today's date in their Date field.

Here's what we have so far for this Report Definition:

-----FIRST PAGE HEADER-----	
-----Turn Print Off-----	Main Report
-----Turn File Off-----	First Page Header
You just chose to print a series of Day Sheets, one per Doctor. Please fill in the desired Date at the prompt and hit ENTER; otherwise, hit F1 now to cancel this procedure.	Explanatory text for the user to read
-----Prompt for Value of Report Variable 1 -----	
-----Turn Print On-----	
-----Turn File On-----	
-----OTHER PAGE HEADER-----	

In the above Report Definition, which only shows its First Page Header, the user is allowed to choose, via the Prompt for Value of Report Variable 1, the Date on which to run the report. So it's not committed to today's date.

Also note the use of four other codes:

```
Turn Print Off
Turn File Off
Turn Print On
Turn File On
```

Each of these four codes is accessed in the Select Report Field section of your Report Options menu (**Ctrl-F7, 1**). Their function is straightforward. The first two stop the report from sending data to the printer or disk file from that point on, until the report runs into the other two codes. I suggest you always insert *both* versions of each code (one that turns off the printer as well as one that turns off the file) even if, say, you plan for that report to be used to print and never go to disk file. You never know when you may need that print job to be temporarily routed to a disk file, to be printed later, or with someone else's printer.

The Dummy Report

Note: It'll be very difficult to understand much of this section if you don't understand the issues discussed in my **Iteration Control** chapter. References to Iteration Control codes appear throughout this section.

Though you may have occasionally used what I call *dummy reports* with versions of DataPerfect prior to 2.3, it's very likely you'll use them extensively with version 2.3 after you read this section. A *dummy report* is a Report Definition that's based on a panel whose records won't be processed, but whose Report Body will be used as an arena for the placement of parallel subreports which *will* process records. These parallel subreports will most likely be of the Subreport Using Virtual Link variety. This will have the effect of running many different reports in a single Report Definition.

The trick here is to keep in mind that DataPerfect sees the dummy report as the Main Report, and will run all the parallel subreports again and again, once for every record in the dummy report's panel. You don't want this. Rather, you want each the subreport that's supposed to run, to run to completion, processing all the records *its* Report Body is supposed to process, then print *its* Final Footer and then, when the last subreport that's supposed to run does all this, the whole report stops. So you need

to put a Stop If code at the end of the Report Body of the *dummy* report, after all the subreports, so the entire process doesn't start over again with the next record in the dummy report. This is easier demonstrated than explained, so let's do that.

When you start your Report Definition that involves a dummy report, place a series of carriage returns in its Report Body, followed by codes that will stop the report after but a single record:

```

-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

] Preliminary
carriage returns
inserted

] Only one record

The pair of codes that follow the carriage returns above consist of the Store Value code setting Report Variable 200 to 0, followed by a Stop If code on Report Variable 200. That effectively makes the report you see above run on precisely one record in the panel on which it's based. That panel can be *any* panel that will always have at least one record. The report will never access that record. For the sake of discussion, I'll call that pair codes the *stop routine*.

Note: As a general rule, I use Report Variable 200 to stop a report. This way, when I see Report Variable 200 on an Edit Report Form screen, I know immediately why it's there.

With the above setup, we now construct our various subreports in the Report Body, above the stop routine. Again, the above report will run on one record in the panel on which its based and then stop. But that means that the subreports we put in the Report Body, above the stop routine, will run completely, on all *their* records. We can now put a series of parallel subreports in that space, which will have the effect of allowing the user to run a series of reports off a single Report Definition, and then stopping.

If you've forgotten, a series of *parallel* subreports is to be contrasted with a series of *nested* subreports. Two subreports are *parallel* when the second is *not* in the Report Body of the first. That is, the second subreport begins sometime after the Final Footer of the first subreport, both being within the Report Body of the main (in this case, dummy) report. Whereas, two subreports are *nested* if the second is a subreport of the first (i.e., it's in the Report Body of the first).

There are three caveats for defining dummy reports, however. *First*, if we don't put the stop routine right after the final parallel subreport, the entire process will start over again with each record in the dummy report's panel. This has the effect of

running the entire series of subreports over and over again. This the most common mistake developers make in attempting this technique, so put the stop codes in there first.

Second, you must have at least one record in the dummy report's panel. If you don't, it won't run at all. Instead, it'll display the *No Data* error message and stop.

Third, make sure that even if you have a lot of records in that panel, you don't assign an Exception List Index to that report, such that that index excludes all records in the panel. This will have the same effect as running the report on a panel with no records. With these three caveats in mind, it makes no difference what panel you choose for the dummy report, or what index is assigned to it.

I'll now layout a few examples of using dummy reports to your advantage.

Dummy Report Examples

A Report That Branches to Other Reports

[For example of a report that branches to other reports, load UD.STR.
Find *Date Range Report that branches (in Master Panel series)* on the Report List.]

Here we define a report that offers the user a menu of possible reports to run. The user chooses one, and only that report runs. This allows you, the developer, to consolidate your reports on the Report List or menus.

The main principle here, again, is using a dummy report as an arena in which to place the *real* reports. We'll use the Report Body of this dummy report to construct a series of parallel subreports, where each such subreport is what I call a *real* report. Let's see how this is done.

First I choose any panel in which I know there will always be at least one record, and then start my Report Definition process with **Shift-F7**. In the First Page Header of this dummy report, I place a prompt and a menu:

<pre>-----FIRST PAGE HEADER----- -----Turn Print Off----- -----Turn File Off----- Choose the report you want to run: [1] Itemized Ledger [2] Day Sheet [3] Monthly Performance Report -----Prompt for Value of Report Variable 1 ----- -----Turn Print On----- -----Turn File On----- -----OTHER PAGE HEADER-----</pre>	<pre>Dummy Report: any panel User offered a menu while printing and any disk file is turned off.</pre>
--	---

The above Prompt for Report Variable 1 would be formatted G9, and be something like

Your choice?:

Now the report knows the user's choice, which is stored in Report Variable 1. The next thing I do is put in a bunch of carriage returns in the Report Body of this dummy report to remind me that that's where the real reports will live. Then I insert

a couple of codes that will guarantee this dummy report will stop running after one record in the panel on which it's based. Otherwise, the report the user selects will run over and over again, once per record in the panel on which the dummy report is based. Here's what I now have after doing this:

```

-----FIRST PAGE HEADER-----
-----Turn Print Off-----
-----Turn File Off-----
Choose the report you want to run by typing its number and
hitting ENTER.

    [1] Itemized Ledger
    [2] Day Sheet
    [3] Monthly Performance Report
-----Prompt for Value of Report Variable 1 -----
-----Turn Print On-----
-----Turn File On-----
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
                                                                    Preliminary
                                                                    carriage returns
                                                                    inserted

-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
                                                                    Stop routine

```

In the end of the Report Body above, I set Report Variable 200 to 0, immediately following it with the Stop If code on Report Variable 200. This effectively stops the dummy report after one record in its panel is processed. That means its Report Body is run only once, which is exactly what we want. Don't forget that our *real* report is going to be a subreport in this Report Body, so we really do want this particular Report Body to run only once.

Let's put our *real* reports in the dummy report's Report Body. Suppose our first real report (Itemized Ledger) should be based on the Account Panel. Then go to the beginning of the dummy report's Report Body and create a Subreport Using Virtual Link (**Ctrl-F7, 6, 4**), telling DataPerfect you want the Account Panel as its target. Tell it what index you want this report to sort on, but make sure you do *not* give it a field list (**F10** when asked for a field list).

Note I'm using a Subreport Using Virtual Link here. The Subreports menu (**Ctrl-F7, 6**) offers four possibilities:

```

-----Subreports & Record Creation-----
    1 - Include Subreport
    2 - Create Record Through Link
    3 - Create Record From Panel List
    4 - Subreport Using Virtual Link
    0 - Return to Edit
-----Selection: 0-----

```

Option 4 offers me the most flexibility, allowing me to choose an index for this link, as well as a field list, if any. By choosing *no* field list here, I'm essentially creating

a subreport that, in effect, starts right from the Panel List. It would be just as if I created this subreport as a *main* report from scratch by choosing the target panel on the Panel List, hitting **Shift-F7**, and then hitting **Insert** on Built-In Short Reports. What makes this just like creating a main report that starts from the Panel List is my refraining from assigning it a field list. By doing that, I'm telling DataPerfect to access the target panel without filtering its records.

If the Account Panel is Panel 8, then we now have something like this:

```

-----FIRST PAGE HEADER-----
-----Turn Print Off-----
-----Turn File Off-----
Choose the report you want:

    [1] Itemized Ledger
    [2] Day Sheet
    [3] Monthly Performance Report
-----Prompt for Value of Report Variable 1 -----
-----Turn Print On-----
-----Turn File On-----
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
=====SUBREPORT LINK/PANEL: 0 8 =====
-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
--Empty--
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
=====END OF SUBREPORT=====

[This is where we're going to put the next subreport.]

-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

Subreport inserted into dummy report's Report Body

What's left of the dummy report's Report Body, followed by its various footers

You can tell the above subreport is a Subreport Using Virtual Link by noting its first line:

```
SUBREPORT LINK/PANEL: 0 8
```

The 0 tells you this is a Subreport Using Virtual Link. If that number was higher than zero, it would be telling you the field number of the link being used to target Panel 8. To make sure you didn't assign this Subreport Using Virtual Link a field list, you can just cursor to that line and look at the area just under the Report Definition help screen. It should change to something like this:

```
Virtual Link Index/Field List:1 - 0
```

The 1 and 0 tell you the Subreport Using Virtual Link is using index 1 with no field list.

Okay, now let's put in one more Subreport Using Virtual Link (again, our subreports are the *real* reports—that is, the reports the user chooses off the menu presented in the First Page Header). The second report is the Day Sheet, so I assume it runs on something like a Day Panel (a panel that holds a record for each day). It really doesn't matter what panel it runs on for our purposes here. I just want to show you how to put various *real* reports in the *dummy* report's Report Body, even though each may run on a different panel.

Back to our second subreport, the Day Sheet. I want to insert a Subreport Using Virtual Link in the Report Body of the *dummy* report, not the Report Body of the other subreport. That is, I want these two subreports to be *parallel* subreports, not *nested* subreports. So I cursor to what's left of the dummy report's Report Body, in the space in my Report Definition where I put

[This is where we're going to put the next subreport.]

There I create a Subreport Using Virtual Link again, but this time I target the Day Panel. Again, I choose the index I want to sort this subreport's records and avoid creating a field list by hitting **F10** when asked for one.

If the Day Panel is Panel 13, I now have something like this:

-----FIRST PAGE HEADER-----		
-----Turn Print Off-----		
-----Turn File Off-----		
Choose the report you want:		
[1] Itemized Ledger		
[2] Day Sheet		
[3] Monthly Performance Report		
-----Prompt for Value of Report Variable 1 -----		
-----Turn Print On-----		
-----Turn File On-----		
-----OTHER PAGE HEADER-----		
--Empty--		
-----TWO-LEVEL REPORT HEADER-----		
--Empty--		
-----REPORT BODY-----		
=====SUBREPORT LINK/PANEL: 0 8 =====		
-----FIRST PAGE HEADER-----		Subreport 1 inserted into dummy report's Report Body
--Empty--		
-----OTHER PAGE HEADER-----		
--Empty--		
-----TWO-LEVEL REPORT HEADER-----		
--Empty--		
-----REPORT BODY-----		
--Empty--		
-----TWO-LEVEL FOOTER-----		
--Empty--		
-----PAGE FOOTER-----		
--Empty--		
-----FINAL FOOTER-----		
--Empty--		
=====END OF SUBREPORT=====		
[This space is here to make it clear where one subreport ends and the other begins. I can delete this, if I want to, by deleting carriage returns.]		Part of dummy report's Report Body
=====SUBREPORT LINK/PANEL: 0 13 =====		
-----FIRST PAGE HEADER-----		Subreport 2 inserted into dummy report's Report Body
--Empty--		
-----OTHER PAGE HEADER-----		
--Empty--		
-----TWO-LEVEL REPORT HEADER-----		
--Empty--		
-----REPORT BODY-----		
--Empty--		
-----TWO-LEVEL FOOTER-----		
--Empty--		
-----PAGE FOOTER-----		
--Empty--		
-----FINAL FOOTER-----		
--Empty--		
=====END OF SUBREPORT=====		
[This is where we're going to put the next subreport.]		What's left of the dummy report's Report Body, followed by its various footers
-----Store Value in Report Variable 200 -----		
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----		
-----TWO-LEVEL FOOTER-----		
--Empty--		
-----PAGE FOOTER-----		
--Empty--		
-----FINAL FOOTER-----		
--Empty--		

If, say, the panel for subreport 3 is Panel 17, we continue this procedure and get the following:


```

-----FIRST PAGE HEADER-----
-----Turn Print Off-----
-----Turn File Off-----
Choose the report you want:

    [1] Itemized Ledger
    [2] Day Sheet
    [3] Monthly Performance Report
-----Prompt for Value of Report Variable 1 -----
-----Turn Print On-----
-----Turn File On-----
-----OTHER PAGE HEADER-----
-----Empty-----
-----TWO-LEVEL REPORT HEADER-----
-----Empty-----
-----REPORT BODY-----
=====SUBREPORT LINK/PANEL: 0 8 =====
-----FIRST PAGE HEADER-----
-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----
-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----
=====END OF SUBREPORT=====
=====SUBREPORT LINK/PANEL: 0 13 =====
-----FIRST PAGE HEADER-----
-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----
-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----
=====END OF SUBREPORT=====
=====SUBREPORT LINK/PANEL: 0 17 =====
-----FIRST PAGE HEADER-----
-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----
-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----
=====END OF SUBREPORT=====
-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----
-----TWO-LEVEL FOOTER-----
-----Empty-----
-----PAGE FOOTER-----
-----Empty-----
-----FINAL FOOTER-----
-----Empty-----

```

Subreport 1
inserted into
dummy report's
Report Body

Subreport 2
inserted into
dummy report's
Report Body

Subreport 3
inserted into
dummy report's
Report Body

What's left of the
dummy report's
ReportBody,
followed by its
various footers.

As it stands now, the above report will run all three subreports once, and then quit. But we want it to run only the subreport the user selected in the menu presented in the First Page Header. To do that we need to add just a couple of codes to each subreport's First Page Header.

We insert the following codes in the First Page Header of Subreport 1:

```

Store Value in Report Variable 200
Stop [Sub]Report if 0 Is in Report Variable 200

```

where the formula for the first code would be

```
rv1=1
```

This will stop execution of Subreport 1 unless the user chose 1 in the First Page Header menu (which was stuffed into Report Variable 1). This doesn't result in the entire report stopping, however. The Stop If code here only causes its particular report (Subreport 1) to stop.

Note: Again, I used Report Variable 200 to stop something. This time I set it to

rv=1

instead of setting it to

0

Following this logic, I put the same two codes in the First Page Headers of Subreports 2 and 3, but change the value of Report Variable 200 in each to

rv1=2

for Subreport 2, and

rv1=3

for Subreport 3.

Now our Report Definition looks like this:

```

-----FIRST PAGE HEADER-----
-----Turn Print Off-----
-----Turn File Off-----
Choose the report you want:

[1] Itemized Ledger
[2] Day Sheet
[3] Monthly Performance Report
-----Prompt for Value of Report Variable 1 -----
-----Turn Print On-----
-----Turn File On-----
-----OTHER PAGE HEADER-----
-----Empty-----
-----TWO-LEVEL REPORT HEADER-----
-----Empty-----
-----REPORT BODY-----
=====SUBREPORT LINK/PANEL: 0 8 =====
-----FIRST PAGE HEADER-----
-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----
-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----
-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----
=====END OF SUBREPORT=====
=====SUBREPORT LINK/PANEL: 0 13 =====
-----FIRST PAGE HEADER-----
-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----
-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----
-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----
=====END OF SUBREPORT=====
=====SUBREPORT LINK/PANEL: 0 17 =====
-----FIRST PAGE HEADER-----
-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----
-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----
-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----
=====END OF SUBREPORT=====
-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----
-----TWO-LEVEL FOOTER-----
-----Empty-----
-----PAGE FOOTER-----
-----Empty-----
-----FINAL FOOTER-----
-----Empty-----

```

Subreport 1
RV200 set to RV1=1

Subreport 2
RV200 set to RV1=2

Subreport 3
RV200 set to RV1=3

RV200 set to 0

I would now put whatever I want in each subreport, as they are my *real* reports. I leave the *dummy* report as is. The above report will execute the subreport the user chooses, and no other.

Here are the major principles of such a Report Definition:

- I choose any panel that has at least one record in it, and base the report on that panel. Likewise, I assign it an index that doesn't have an Exception List on it that excludes all records in the panel. This will be the *dummy* panel, as its records will never be used.
- I present the user with a menu of possible subreports to run, allowing a single choice. I also use codes to make sure that menu doesn't print or go to disk file.
- In order for each *subreport* to act like a *main* report, I make each subreport a Subreport Using Virtual Link *without a field list*. This makes each such subreport act just like it started from the Panel List.

- In order to make only the chosen subreport run, in each subreport's First Page Header I put a stop routine that examines what number the user chose. It stops its particular subreport from running if the number the user chose isn't the one assigned to that subreport, otherwise it allows its subreport to run.
- To keep the report from causing the chosen subreport to run over and over again, once for each record in the main (dummy) report's panel, I put a stop routine in the main report's Report Body, just after the last parallel subreport. This will make the main report run on only one of its panel's records, but will cause the chosen subreport to run on all of *its* panel's records.

Gathering Preliminary Information from Various Panels Before Starting the "Real" Report

[For a report that offers a lookup in one subreport, before printing in another subreport, load UD.STR. Find *Date Range branching report that offers lookup for preliminary data* on the Report List .]

This is done with parallel subreports, as in the branching technique. Each parallel subreport gathers the data needed and enters it into Report Variables. It may do this via lookups, where what the user selects in the lookup is placed in a Report Variable.

Iteration Control enters into this in two ways. The lookup just described is one. But after the user selects something via the lookup and the parallel subreport stuffs one or more values in one or more Report Variables, that parallel subreport has fulfilled its function, so must stop, not offering the lookup again. To accomplish the latter, we do the usual: we insert two codes to stop the report after a single record.

Let's build this by starting with the generic dummy report:

```

-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
                                                                    Preliminary
                                                                    carriage returns
                                                                    inserted
-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
                                                                    Only one record

```

Now we add our first parallel subreport:

```

-----FIRST PAGE HEADER-----
--Empty--
PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
=====SUBREPORT LINK/PANEL: 0 1=====
-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----User Chooses Next Record By LookUp-----
-----Store Value in Report Variable 1-----
-----Store Value in Report Variable 200-----
-----Stop [Sub]Report if 0 Is in Report Variable 200-----
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
=====END OF SUBREPORT=====

Parallel Subreport 1

Give user a lookup
Put choice in RV1

Limit to one lookup

Carriage returns
where I'll place
more parallel
subreports

-----Store Value in Report Variable 200-----
-----Stop [Sub]Report if 0 Is in Report Variable 200-----
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

The above parallel subreport is a Subreport Using Virtual Link with no field list. Because it lacks a field list, its lookup offers the user a lookup with full access to its targeted panel's records. Whatever the user chooses from the lookup is stuffed into Report Variable 1 (perhaps an Part Number). To do that, all you do here is **Ctrl-F7, 1, 4, 1, Enter** to begin the Report Variable 1 definition process the usual way. Then **F4, Tab** to the Part Number field of that panel, and **F4**. This will stuff the Part Number of the user-selected record into Report Variable 1 (that is, you set Report Variable 1 to that selected field). Other fields can be stuffed into other Report Variables if needed here. These Report Variables can then be used later by the parallel subreport that actually prints.

Why do this? Well, consider a report that prompts the user for a Start Date and End Date, then asks for a Part Number via a lookup like we mentioned above, then asks the user for a state via another lookup, and finally prints a report that lists all sales of the selected part in the selected state in the prompted-for date range.

Such a report would be a dummy report with three parallel subreports. In its First Page Header, it would prompt for Start and End Dates, stuffing them into Report Variables 1 and 2 respectively.

The First Parallel Subreport: Parts Panel

The first parallel subreport would be a Subreport Using Virtual Link that targets the Parts Panel and has no field list. It would look just like the parallel subreport we just created above. It would offer the user a lookup, stuff the user's choice into a Report Variable, then stop. Again, it stops because of the Report Variable 200 stop routine

that appears right after the Report Variable is set with the user's choice. This makes sure the user sees that lookup only once.

The Second Parallel Subreport: States Panel

The second parallel subreport would be just like the first one, but would target the States Panel. It would be a Subreport Using Virtual Link, again with no field list. It would offer the user a lookup, stuff a Report Variable with the State Code the user chooses, then stops because of another instance of the Report Variable 200 stop routine.

The Third Parallel Subreport: The Printout

Now, as the report exits the second parallel subreport, preparing to enter the third parallel subreport, the following four Report Variables are set:

- Report Variable 1 holds the Start Date
- Report Variable 2 holds the End Date
- Report Variable 3 holds the Part Number
- Report Variable 4 holds the State Code

The third parallel subreport prints the report. It targets the Transaction Panel and uses an index with Part Number, State Code and Date, in that order. This Subreport Using Virtual Link, however, has a field list:

RV3 (Part Number) and RV4 (State Code)

That field list is in the order you see above. In its First Page Header, it has a Skip To Record at Report Variable 1 code, which will guaranty it starts with the first record equal to or later than the date stored in Report Variable 1. There will also be a pair of codes in its Report Body that will stop the report as soon as it comes to the first record after the date stored in Report Variable 2.

Here's what we have:

```

-----FIRST PAGE HEADER-----
-----Prompt for Value of Report Variable 1 -----
-----Prompt for Value of Report Variable 2 -----
-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----
=====SUBREPORT LINK/PANEL: 0 1=====
-----FIRST PAGE HEADER-----
-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----
-----User Chooses Next Record By LookUp-----
-----Store Value in Report Variable 3 -----
-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----
-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----
=====END OF SUBREPORT=====
=====SUBREPORT LINK/PANEL: 0 2=====
-----FIRST PAGE HEADER-----
-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----
-----User Chooses Next Record By LookUp-----
-----Store Value in Report Variable 4 -----
-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----
-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----
=====END OF SUBREPORT=====
=====SUBREPORT LINK/PANEL: 0 3=====
-----FIRST PAGE HEADER-----
-----Skip To Record at Report Variable 1 -----
Date      Description                      Amount
=====
-----OTHER PAGE HEADER-----
(cont'd.)                               Page
Date      Description                      Amount
=====
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----
-----Store Value in Report Variable 5 -----
-----Stop [Sub]Report if 0 Is in Report Variable 5 -----
-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----
=====END OF SUBREPORT=====
-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----
-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----

```

Dummy Report
Date range prompts

|Parallel Subreport 1
(Parts Panel)
(No field list)

Lookup of Parts
Put choice in RV3
Limit to one lookup

Parallel Subreport 2
(States Panel)
(No field list)

Lookup of States
Put choice in RV4
Limit to one lookup

Parallel Subreport 3
(Transaction Panel)
(F/List: RV3,RV4)
Skip to Start Date

RV5: P1F1 <= RV2
Stop if RV5 is false
Fields to print
(P1F1 is Date field)

Limit Dummy Report
to one record

This report will print only detail lines conforming to the Date Range entered at the first two prompts, and the Part Number and State Code picked on the lookups.

Prompting the User with the Number of Hits

[For an example of this, load UD.STR.
On the Report List, find *Showing the Number of Hits.*]

Here we want a report to prompt the user for a search condition—like a date range—and then tell the user how many records satisfy that search. The user then is asked if he wants to continue on to print the results or not.

Let's first start with a typical report that prints out all Transactions that occur in a date range in the Transaction Panel. It would look something like this, assuming it's based on the Transaction Panel (this isn't the Report Definition that shows the user the number of hits):

```

Panel: Transaction Panel
Index: Date field
-----FIRST PAGE HEADER-----
-----Prompt for Value of Report Variable 1 -----
-----Prompt for Value of Report Variable 2 -----
-----Skip To Record at Report Variable 1 -----
Transactions
to
Date Patient Name Description Amount
-----OTHER PAGE HEADER-----
Date Patient Name Description Amount
-----TWO-LEVEL REPORT HEADER-----
REPORT BODY
-----Store Value in Report Variable 3 -----
-----Stop [Sub]Report if 0 Is in Report Variable 3 -----
to
-----TWO-LEVEL FOOTER-----
PAGE FOOTER
FINAL FOOTER

```

Prompt: Start Date
Prompt: End Date
Skip to Start Date
Print RVs to show date range
RV3: P1F1<=RV2
Stop if outside date range
(P1F1 is Date field)

The above will prompt the user for Start and End Dates (Report Variables 1 and 2), print the date fields chosen in the First Page Header, and then skips to the first record it sees with a Date field value equal to or greater than the Start Date. Report Variable 3 is used to stop the report as soon as the Date field is found to contain a date beyond the date range. Don't forget that this sort of Iteration Control won't work unless the report is running on an index that sorts on the Date field.

Okay, now let's let the user choose to print the report only after being told how many Transactions satisfy his chosen date range. To do this, we use a Dummy Report again, basing it on any panel with at least one record. We *count* our Transaction Panel hits in the *first* parallel Subreport Using Virtual Link, and *print* the results with the *second* parallel Subreport Using Virtual Link. In between the two parallel Subreports Using Virtual Link, still in the Report Body of the Dummy Report, we show the user the number of hits and ask if he wants to continue on to the print job:


```

-----FIRST PAGE HEADER-----
-----Turn Print Off-----
-----Turn File Off-----
-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----
=====SUBREPORT LINK/PANEL: 0 4=====
-----FIRST PAGE HEADER-----
-----Prompt for Value of Report Variable 1 -----
-----Prompt for Value of Report Variable 2 -----
-----Skip To Record at Report Variable 1 -----
-----Store Value in Report Variable 4 -----
-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----
-----Store Value in Report Variable 3 -----
-----Stop [Sub]Report if 0 Is in Report Variable 3 -----
-----Store Value in Report Variable 4 -----
-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----
=====END OF SUBREPORT=====

Number of Transactions in that date range: #####

-----Prompt for Value of Report Variable 5 -----
-----Store Value in Report Variable 6 -----
-----Stop [Sub]Report if 0 Is in Report Variable 6 -----

=====SUBREPORT LINK/PANEL: 0 4=====
-----FIRST PAGE HEADER-----
-----Turn Print On-----
-----Turn File On-----
-----Skip To Record at Report Variable 1 -----
Transactions
##### to #####
      Date      Patient Name      Description      Amount
-----
      Date      Patient Name      Description      Amount
-----
-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----
-----Store Value in Report Variable 3 -----
-----Stop [Sub]Report if 0 Is in Report Variable 3 -----
#####
-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----
=====END OF SUBREPORT=====
-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----

-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----

```

Dummy Report
(any panel)
Turn printing off

Subreport 1
(Transaction Panel)

RV4 counter: 0

Stop as before
RV4: RV4+1

Dummy Report
Report Body
Print RV3, then ask
if user wants to
continue. See later
explanation.

Subreport 2
(Transaction Panel)
Turn printing on
The rest is as
before

Dummy Report again
Only one record

Between the two parallel subreports, still in the Dummy Report's Report Body, there's a section that looks like this:

```

Number of Transactions in that date range: #####

-----Prompt for Value of Report Variable 5 -----
-----Store Value in Report Variable 6 -----
-----Stop [Sub]Report if 0 Is in Report Variable 6 -----

```

Here the report prints to screen Report Variable 3 (the number of hits that satisfy the date range), and then produces what I call a *continuation prompt*:

Shall we continue? (Y/N):

Like before, it formats Report Variable 5 as a U1 field. Report Variable 6 then is coded

```
rv5="Y"
```

This way the user may choose to not print the report (i.e., run Subreport 2) after seeing the number of hits that will print. If the user answers anything other than Y to that prompt, Report Variable 6 will end up false, stopping the second parallel subreport (the one that prints).

Reports That Double-Sort Records

This is a common problem. For example, you need a report that prints records alphabetically by Last Name within a date range. Say you're an attorney and want a report that prints a list of all Cases with an Opening Date between two prompted-for dates. Further, all this information is in a single panel. What index do you choose for this?

Since more than one Client can have the same Last Name, you'll need the First Name in indexes in that panel. And since two Clients can have the same Last and First Names, you'll also have a Client ID Number in all indexes in that panel. If you want records in this report to print alphabetically by Last Name, you'll need an index that starts with these fields:

```
Last Name, First Name, Client ID Number
```

We need to do some sorting by Opening Date, so that field needs to be in there too. This index looks like a good candidate for our report:

```
Last Name, First Name, Client ID Number, Opening Date
```

If you run the report on that index, records will indeed print alphabetically by Last Name. Getting the report to print only records within the prompted-for date range will require the use of Iteration Control option Skip If. Unfortunately, no other Iteration Control option will work here. Let's discuss this.

Note that I can't use the Skip To option here, where I want to have the report Skip To the earliest Opening Date based on the prompted-for date range. Why? Because the Skip To option works on the primary sorting field of the current index, and that's the Last Name field in this case. So the Skip If option is the only option that will exclude records outside the prompted-for date range. If the Opening Date field is P1F4, and Report Variables 1 and 2 hold the Start and End Dates respectively, then here's what we have:

```

Report Index:
  Last Name, First Name, Client ID, Opening Date
-----FIRST PAGE HEADER-----
-----Prompt for Value of Report Variable 1 -----] Start Date
-----Prompt for Value of Report Variable 2 -----] End Date
Page
Date:
Attorney:
Client Opening
=====
OTHER PAGE HEADER
(continued) Page
Date:
Attorney:
Client Opening
=====
TWO-LEVEL REPORT HEADER
--Empty--
REPORT BODY
-----Store Value in Report Variable 3 -----] Skip If
-----Skip Record if 0 (False) Is in Report Variable 1 -----]
TWO-LEVEL FOOTER
--Empty--
PAGE FOOTER
--Empty--
FINAL FOOTER
--Empty--

```

In the above Report Body, Report Variable 3 is given the formula

P1F4>=rv1 and P1F4<=rv2

where, again, P1F4 is the Opening Date field.

This report works and accomplishes what we want. All records it prints will fall within the prompted-for date range, and they'll print alphabetically by Last Name. But this is a slow report to run on a large database. It examines each and every record outside the prompted-for date range, while printing only those within that range. It would be nice to be able to use the Skip To and Stop If codes here. Let's see how we might make that happen.

The Report Records Panel

To do this, we create a special Report Records Panel. Our report will send copies of records to that panel, to be sorted a second time before printing. Our Report Records Panel can be used by other reports that need to double-sort records, and looks something like this:

```

REPORT RECORDS PANEL

Report ID
D99/99/9999
G9999999999
N9999999999
A50
A50
A50
A50

```

As other reports are created that need additional fields, you can add more fields to the Report Records Panel. The fields can be extra large since the report that prints from this panel can have its fields reformatted on the Edit Report Form screen to truncate trailing spaces. Think of this panel as merely a receptacle for records copied to it for further sorting before printing. Let's outline how to use this sort of panel:

```

-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
=====SUBREPORT LINK/PANEL: 0 1=====
-----FIRST PAGE HEADER-----
-----Prompt for Value of Report Variable 1 -----
-----Prompt for Value of Report Variable 2 -----
-----Store Value in Report Variable 3 -----
-----Skip To Record at Report Variable 1 -----
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Store Value in Report Variable 4 -----
-----Stop [Sub]Report if 0 Is in Report Variable 4 -----
=====CREATE RECORD LINK/PANEL: 0 10=====
-----REPORT BODY-----
--Empty--
=====SAVE RECORD=====

-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
=====END OF SUBREPORT=====

-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----

-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

Dummy Report
(any panel)

Subreport 1
(Date index)
(No field list)
Start Date
End Date
Report ID
Skip To RV1

RV4: P1F1<=RV2
Stop if RV4 false

Create record in
Report Records Panel

Dummy Report
Report Body with
carriage returns
waiting for
Subreport 2

Only one record

In the above Report Definition, I made the following changes over the previous report:

- I created a Dummy Report (based on any panel that will always have at least one record), and then created a subreport (Subreport 1) that looks a lot like the prior report. This subreport will put copies of all relevant records in the Report Records Panel (Panel 10).
- The index for Subreport 1 sorts on Opening Date instead of Last Name. This allows me to use the Skip To code in its First Page Header, which causes the subreport to go right to the first record in the date range without inspecting earlier ones. This index also allows me to use the Stop If code in the Report Body, causing the subreport to stop as soon as it finds records beyond the date range.

- I added Report Variable 3, which is the Report ID value. I can use any formula I want for that, as long as it produces a unique number every time the user runs this report. I suggest just using my Moment function, which produces a unique ten-digit number every second:

`(86400*today)+now`

Read up on this ad hoc function in the *An Alternative Solution: Using the Concepts of MOMENT and MODULO* section of my the **Fields: Issues** chapter.

- I inserted a Create Record From Panel List in Subreport 1's Report Body, but have yet to fill it in. That's where each record found in the date range will be copied to the Report Records Panel. To fill out each record, however, I'll need to stuff some Report Variables earlier in the report. Let's do that now, and store those Report Variables in the proper fields in the Report Records Panel:

```

-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
=====SUBREPORT LINK/PANEL: 0 1=====
-----FIRST PAGE HEADER-----
-----Prompt for Value of Report Variable 1 -----
-----Prompt for Value of Report Variable 2 -----
-----Store Value in Report Variable 3 -----
-----Skip To Record at Report Variable 1 -----
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Store Value in Report Variable 4 -----
-----Stop [Sub]Report if 0 Is in Report Variable 4 -----
-----Store Value in Report Variable 5 -----
-----Store Value in Report Variable 6 -----
-----Store Value in Report Variable 7 -----
-----Store Value in Report Variable 8 -----
=====CREATE RECORD LINK/PANEL: 0 10=====
-----REPORT BODY-----
-----Store Report Variable 3 in Field 1 -----
-----Store Report Variable 5 in Field 2 -----
-----Store Report Variable 6 in Field 5 -----
-----Store Report Variable 7 in Field 6 -----
-----Store Report Variable 8 in Field 3 -----
-----SAVE RECORD-----
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
=====END OF SUBREPORT=====

-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----

-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

Grab Opening Date
Grab Last Name
Grab First Name
Grab Client ID Num

(Create in Panel 10)
Insert Report ID
Insert Opening Date
Insert Last Name
Insert First Name
Insert Client ID Num

Dummy Report
Report Body with
carriage returns
waiting for
Subreport 2

Now, let's create Subreport 2, which will do the printing:

-----FIRST PAGE HEADER-----	} Dummy Report
-----OTHER PAGE HEADER-----	
-----TWO-LEVEL REPORT HEADER-----	
-----REPORT BODY-----	
=====SUBREPORT LINK/PANEL: 0 1=====	
-----FIRST PAGE HEADER-----	} Subreport 1 (Cases Panel) (Date index) (No field list) (Copies records to Rpt Records Panel)
-----Prompt for Value of Report Variable 1 -----	
-----Prompt for Value of Report Variable 2 -----	
-----Store Value in Report Variable 3 -----	
-----Skip To Record at Report Variable 1 -----	
-----OTHER PAGE HEADER-----	
-----TWO-LEVEL REPORT HEADER-----	
-----REPORT BODY-----	
-----Store Value in Report Variable 4 -----	
-----Stop [Sub]Report if 0 Is in Report Variable 4 -----	
-----Store Value in Report Variable 5 -----	
-----Store Value in Report Variable 6 -----	
-----Store Value in Report Variable 7 -----	
-----Store Value in Report Variable 8 -----	
=====CREATE RECORD LINK/PANEL: 0 10=====	
-----REPORT BODY-----	
-----Store Report Variable 3 in Field 1 -----	
-----Store Report Variable 5 in Field 2 -----	
-----Store Report Variable 6 in Field 5 -----	
-----Store Report Variable 7 in Field 6 -----	
-----Store Report Variable 8 in Field 3 -----	
=====SAVE RECORD=====	
-----TWO-LEVEL FOOTER-----	
-----PAGE FOOTER-----	
-----FINAL FOOTER-----	
=====END OF SUBREPORT=====	
=====SUBREPORT LINK/PANEL: 0 10=====	
-----FIRST PAGE HEADER-----	} Subreport 2 (Rpt Records Panel) (Index: Rpt ID, Last Name, ...) (F/List: RV3) (Prints records)
Page []	
Date: []	
Attorney: []	
Client Opening	
=====	
-----OTHER PAGE HEADER-----	
(continued) Page []	
Date: []	
Attorney: []	
Client Opening	
=====	
-----TWO-LEVEL REPORT HEADER-----	
-----REPORT BODY-----	
[] []	
-----TWO-LEVEL FOOTER-----	
-----PAGE FOOTER-----	
-----FINAL FOOTER-----	
=====END OF SUBREPORT=====	
-----Store Value in Report Variable 200 -----	
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----	
-----TWO-LEVEL FOOTER-----	} Dummy Report limited to one record
-----PAGE FOOTER-----	
-----FINAL FOOTER-----	

Note that Subreport 2 uses an index that begins with Report ID Number and Last Name, in that order, and is created with a field list consisting of Report Variable 3. Report Variable 3, remember, is the Report ID Number for this report. This way Subreport 2 prints only records just processed by Subreport 1 on this run, not a previous run.

Of course our new Reports Records Panel will keep accumulating records with every report run like this. You can have the same report delete all records in the Report Records Panel it prints:

```

-----FIRST PAGE HEADER-----
-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----
=====SUBREPORT LINK/PANEL: 0 1=====
-----FIRST PAGE HEADER-----
-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----
=====CREATE RECORD LINK/PANEL: 0 10=====
-----REPORT BODY-----
=====SAVE RECORD=====
-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----
=====END OF SUBREPORT=====
=====SUBREPORT LINK/PANEL: 0 10=====
-----FIRST PAGE HEADER-----
-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----
-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----
=====END OF SUBREPORT=====
=====SUBREPORT LINK/PANEL: 0 10=====
-----FIRST PAGE HEADER-----
-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----
-----Delete Record-----
-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----
=====END OF SUBREPORT=====
-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----
-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----

```

Dummy Report

Subreport 1
(Cases Panel)
(Copies records to
Rpt Records Panel)

Subreport 2
(Rpt Records Panel)
(Prints Records)

Subreport 3
(Rpt Records Panel)
(Same index and
field list)
(Deletes records
from last run)

Dummy Report limited
to one record

You could do that, but it makes more sense to just create a new report that allows the application Supervisor to periodically clean out that panel. So forget about putting in parallel Subreport 3 in the above report. Just let the records accumulate in the Report Records Panel, and have the Supervisor run this report once a month:

```

Panel: Report Records Panel
Index: An index that has no Exception List
-----FIRST PAGE HEADER-----
-----Turn Print Off-----
-----Turn File Off-----

```

You just chose to delete all records in the Report Records Panel!

```

-----Prompt for Value of Report Variable 1 -----
-----Store Value in Report Variable 2 -----
-----Stop [Sub]Report if 0 Is in Report Variable 2 -----
-----Turn Print On-----
-----Turn File On-----
-----OTHER PAGE HEADER-----
-----Empty-----
-----TWO-LEVEL REPORT HEADER-----
-----Empty-----
-----REPORT BODY-----
-----Delete Record-----
-----TWO-LEVEL FOOTER-----
-----Empty-----
-----PAGE FOOTER-----
-----Empty-----
-----FINAL FOOTER-----
-----Empty-----

```

Warning message

(See below for
explanation of
this)

Deletes all records
in Rpt Records Panel

The above report is based on the Report Records Panel, and runs on any index that has no Exception List so as to guaranty is sees all records in that panel. The Prompt For Report Variable 1 code is our usual continuation prompt:

```
Shall we continue? (Y/N) :
```

Report Variable 1 is formatted U1 and Report Variable 2 is coded with this formula:

```
rv1="Y"
```

This chapter is for the experience DataPerfect application developer. I suggest beginners take only a cursory look at it.

Open Filename in Report Variable

This *Report Fields and Variables* option (**Ctrl-F7, 1, C**) was introduced with DataPerfect 2.3c. As of this writing, it's currently going through changes, but I'll outline its implementation as of the 11/07/96 version of DataPerfect 2.3c.

The Open Filename report option allows the developer to direct disk file output to a disk file whose filename is held by a Report Variable. To use this option, a filename must first be placed in a Report Variable in one of four ways:

- The definer puts the filename in the Report Variable directly, using the **Ctrl-F7, 1, 4** Specify Formula screen, surrounding it with quotes. For instance,

```
"myfile.txt"
```

or

```
"d:\apps\myfile.txt"
```

- The definer codes the Report Variable to yield a filename based on a formula. For instance,

```
apply.format["DYMD9999/99/99";today]".rpt"
```

If you run the report on November 21, 1996, that would yield this filename:

```
19961121.RPT
```

- The definer has the Report Variable grab a filename found in an A or U field.
- The definer uses a Prompt For code to allow the user to enter their own choice of filename. The Prompt For field must be formatted A or U.

When DataPerfect sees the Open Filename code in a Report Definition, it opens a file by that name, preparing it for report output. The default behavior of this code is to *append* output to the opened file. To force DataPerfect to *overwrite* the

file, precede the Open Filename code with the Turn File Off code, and follow it with the Turn File On code (see the last example below for *overwriting*).

Each of these produces identical output, opening and *appending* to the file named chosen by the user:

```

-----FIRST PAGE HEADER-----
-----Prompt for Value of Report Variable 1 -----
-----Begin Writing to Filename in Report Variable 1 -----

-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
[REDACTED]
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

-----FIRST PAGE HEADER-----
-----Prompt for Value of Report Variable 1 -----

-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Begin Writing to Filename in Report Variable 1 -----
[REDACTED]
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

This one does almost the same thing, but starts writing to file starting with the second record in the index (that is, it doesn't start writing until after the first record):

```

-----FIRST PAGE HEADER-----
-----Prompt for Value of Report Variable 1 -----

-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
[REDACTED]
-----Begin Writing to Filename in Report Variable 1 -----

-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

This one sends all records in the index to the user-chosen file, *overwriting* the file it opens:

```

-----FIRST PAGE HEADER-----
-----Prompt for Value of Report Variable 1 -----
-----Turn File Off-----
-----Begin Writing to Filename in Report Variable 1 -----
-----Turn File On-----

-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
[Patterned Area]
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

Further, if you assign the null string ("") to the Report Variable, that closes the file and sets the value of that Report Variable to 0. If the Report Variable has value improperly formatted for the Open Filename code (e.g., variable-length text or numerical), DataPerfect will ignore the Open Filename code that references that Report Variable and set that Report Variable to 0. If the Report Variable is properly formatted for the Open Filename code, the file is opened and the Report Variable is set to 1.

Overwrite Mode from the Report List

If the Open Filename code is used in Overwrite mode in a report that runs the *from Report List*, and its output file already exists, when the user runs the report he'll be asked if he wants to overwrite the existing file. If he answers *N* to the *Y/N* prompt, the Report Variable is set to 0. If he answers *Y*, the file is opened and the Report Variable is set to 1. This last feature allows you to abort the remainder of the report if the use replies *N*, like this:

```

-----FIRST PAGE HEADER-----
-----Prompt for Value of Report Variable 1 -----
-----Turn File Off-----
-----Begin Writing to Filename in Report Variable 1 -----
-----Stop [Sub]Report if 0 Is in Report Variable 1 -----
-----Turn File On-----

-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
[Patterned Area]
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

In the above Report Definition, the Stop If code will abort the report if, when asked if he wants to overwrite an existing file, the user answered *N*. This is because the Open Filename code changes Report Variable 1 to 0 after the user replies *N* to the Overwrite prompt DataPerfect issues when it sees the file exists.

Note: DataPerfect never warns of overwriting an existing file if the report is run from a menu. This only happens if run from the Report List. When run from a menu, the user isn't given a choice here (so as not to confuse him). The file will simply be overwritten.

Overwrite Mode Caveat

Please be aware that this feature allows the overwriting of just about any file, so you should use it cautiously. This is especially worrisome if this option receives its input from a Prompt For code, allowing the user to choose just about any filename he wants. In such cases you should consider following the Prompt For code with a pair of codes that will stop the report if a dangerous filename was entered at the prompt. For instance, you can follow a Prompt For code with a Store Value code like this.

```
cases substring[rv1;0;4]
  case cv = ".EXE"           of 0 endof
  case cv = ".COM"           of 0 endof
  case cv = ".BAT"           of 0 endof
  case cv = ".STR"           of 0 endof
  case cv = ".IND"           of 0 endof
  case cv = ".TXX"           of 0 endof
  case cv = ".DAT"           of 0 endof
  case rv1 = "[some data file]" of 0 endof
  case rv1 = "[another data file]" of 0 endof
  case rv1 = "c:\\"          of 0 endof
default rv1
endcases
```

In the above Report Variable formula, the last CASE line keeps the user from entering a string that begins with the root directory of drive C. The others are obvious. Now we have this:

```
-----FIRST PAGE HEADER-----
-----Prompt for Value of Report Variable 1 -----
-----Store Value in Report Variable 1 -----
-----Turn File Off-----
-----Begin Writing to Filename in Report Variable 1 -----
-----Stop [Sub]Report if 0 Is in Report Variable 1 -----
-----Turn File On-----

-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
#####
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
```

The above Report Definition examines the user's choice of filename (Report Variable 1). If the filename is on the illegal list found in the Store Value formula above, Report Variable 1 is set to 0. This causes DataPerfect to ignore the Open Filename code. The Stop If code then aborts the report. On the other hand, if the user enters a filename that's *not* on the illegal list found in the Store Value formula above,

Report Variable 1 retains the value entered by the user at the initial Prompt. The Open Filename code then opens the user-chosen file, overwriting it if it exists.

Printer Control Panel

The concept of a Printer Control Panel is an interesting one. Here you want to be able to plan ahead for all possible printers your enduser might use when running the application you're building. The Printer Control Panel is simply a panel that contains a record for each possible printer type—perhaps one for dot matrix printers, one for HP compatible lasers, etc. A Printer Control Panel record will contain fields to accommodate printer control strings for various printer needs. For instance, it will usually contain a field for a printer control string that turns Bold on, a field for a printer control string that turns Bold off, one that starts 12 characters per inch, and one that returns to 10 characters per inch.

So there will be a record for dot matrix printers that shows which printer control strings do all this, and another record for HP compatible laser printers that shows an entirely different set of printer control strings for the same printer actions. One field per printer control code, clearly labelled with panel text. Here's a very simple Printer Control Panel:

PRINTER CONTROL PANEL

Printer ID#/Name:

Reset Printer:

Bold On/Off:

Underline On/Off:

Italics On/Off:

10, 12, 16.67 cpi:

Each record in that panel will get a unique ID Number for identification purposes later, and a Name, like *Dot Matrix*, *HP Laser Compatible*, or *IBM Proprinter*.

Now, what does this sort of panel get us? Well, if we can somehow tell a report which of those Printer Control Panel records is the correct one (the one that corresponds to the printer currently in use), we can access printer control strings simply by selecting fields from the Printer Control Panel. When defining a report in the Edit Report Form screen, if I want Bold On, or Italics Off, I simply hit **F4** and, in one way or another which I'll soon discuss, I select the correct field from the Printer Control Panel. So when this report is run, if we somehow get the application to know which of the Printer Control Panel records is the correct one, we know the currently running report will be issuing the correct Bold On and Italics Off printer control strings. Tell the application we now have a different Printer Control Panel record active (we just replaced our dot matrix with a laser), and the same reports issue printer control strings appropriate for *that* printer.

So let's talk about how to use the Printer Control Panel—that is, how to get reports to see the correct Printer Control Panel record, and consequently issue the

correct printer control codes. There are a few ways to accomplish this, each with its strengths and weaknesses.

Direct Approach 1

Here we create a way to make the desired Printer Control Panel record be the first one in some index in that panel. One way to do this is to use the ad hoc moment function I discuss in *An Alternative Solution: Using the Concepts of MOMENT and MODULO* in my **Fields: Issues** chapter. The current moment is defined as

$$(86400 * \text{today}) + \text{now}$$

which is the number of seconds since March 1, 1900. That number requires a 10-digit field. What we need here is the negative of that value, so our index sorts backwards based on that field. Our G-9999999999 field, then, will have the following formula:

$$- ((86400 * \text{today}) + \text{now})$$

Or we can give it the simpler equivalent formula:

$$(-86400 * \text{today}) - \text{now}$$

This Reverse Moment field will be hidden, update on any change, and be positioned as the first field in some index in the Printer Control Panel.

Now, any time the Supervisor goes into that panel and selects the record he wants, he just hits **F6** to cause the record to update the Reverse Moment field, and then **F10** to save it. Now that record will be at the head of the Printer Control Panel's Reverse Moment index.

Next, put a hidden panel link in each and every panel, such that that link targets the Printer Control Panel, using the Printer Control Panel's Reverse Moment index without a field list. This way that hidden link always lands on the last Printer Control Panel record put in Edit mode.

Finally, whenever you're in an Edit Report Form screen and want to select the appropriate printer control string field in the Printer Control Panel, simply do this:

- **F4.**
- **Tab** to the hidden panel link we just created.
- **Down Arrow.**
- **Tab** to the desired Printer Control Panel's printer control string field.
- **F4.**

This places the selected Printer Control Panel field in the Report Form, just like any other field. The difference here is that that field will send the printer a printer control string. And it will always issue the printer control string that's found in that Printer Control Panel field for the last Printer Control Panel record that was in Edit mode.

But this is dangerous. This scheme assumes the user can not only get into the Printer Control Panel, but can put it in Edit mode. All those fields with valuable

printer control strings in them can now be edited carelessly by the user. These are the sorts of fields you probably don't want the user to be able to edit. If you make them all ::N fields, then whenever you, the definer, have to go into that panel and change some of the printer control strings, you'll have to go into Define Mode and remove the ::N modifier from the field in question, and then remember to put that ::N modifier back after you're done editing the value in that field for some record. Or you can use a special report for this, where that report is on a menu no user ever sees.

Direct Approach 2

To get around this complication, keep the user out of the Printer Control Panel entirely, and remove the formula from the Reverse Moment field. Keep it a G-9999999999::H field, however. Then create a report that offers the user a lookup of possible Printer Control Panel records, sorted by Printer Name. Have that report's First Page Header set Report Variable 1 to

```
(-86400*today)-now
```

Then have its Report Body offer the user a lookup of Printer Control Panel records. Still in the Report Body, Store Report Variable 1 into the hidden Reverse Moment field of the record the user selected, and then stop:

-----FIRST PAGE HEADER-----	
-----Store Value in Report Variable 1 -----	RV1 set to
-----OTHER PAGE HEADER-----	Negative Moment
--Empty--	
-----TWO-LEVEL REPORT HEADER-----	
--Empty--	
-----REPORT BODY-----	
-----User Chooses Next Record By LookUp-----	User chooses
-----Store Report Variable 1 in Field 100 -----	Stuff -Momemt field
-----Store Value in Report Variable 200 -----	Limit to one lookup
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----	
-----TWO-LEVEL FOOTER-----	
--Empty--	
-----PAGE FOOTER-----	
--Empty--	
-----FINAL FOOTER-----	
--Empty--	

Now the user can choose the printer control record without having access to the Printer Control Panel. After the user selects the appropriate Printer Control Panel record to be used by reports, that's the record all those hidden panels links will see during report processing (the links that have no field list, and are tied to the Reverse Moment index in the Printer Control Panel).

Indirect Approach 1

This offers much more flexibility. I recommend it over either Direct Approach outlined above. Here you use an intermediary panel to access the Printer Control Panel during report definition. One way to do this is to use the User ID Panel. Don't even bother reading this section unless you're familiar with the User ID facility

introduced with DataPerfect 2.3's second official release (September 1993). See *User ID Panel* in my **Securing the Application** for details on this.

Create a Printer ID field in the User ID Panel, where that field matches the format of the Printer ID field in the Printer Control Panel. Let's say it's a G99 field. On this new User ID Panel field, which we'll call P10F12, attach a data link that targets the Printer Control Panel, using a Printer Control Panel index that begins with the Printer ID field. Put only the Printer ID field on its field list. Now the application's Supervisor, who, I assume, is the only one you're letting into the User ID Panel, can easily assign each user a printer from the Printer Control Panel. You might also consider formulating the Printer ID field in the User ID Panel to initialize to some particular printer in the Printer Control Panel, just in case the Supervisor forgets to assign a printer to each user.

Next, give the user a way to change his printer when he, say, changes work stations. To do this, simply define a report that offers him a lookup of possible printers from which to choose. This report will already know who the user is, since he logged into the application with his User ID and Password, so it knows exactly which User ID Panel record to alter. If you want, you can have it ask who the user is, just to make sure no one else is attempting to change their default printer. We'll go over the mechanics of this Printer Selection report in a minute. Let's move on to the other aspects of this Indirect Approach to our problem.

So now we have a mechanism of assigning a printer from the Printer Control Panel to each and every user of the database. Next we must figure out how to access the appropriate Printer Control Panel record when inserting printer control strings in an Edit Report Form screen. To do this we need those panel links we hid in each panel (in the *Direct Approaches* outlined above), but we must change their target panel. They now target the User ID Panel, and do so with the user's User ID field on its field list (and only that field on its field list). But this means we need a hidden User ID field in each of these panels. Such a field will have to be a computed field with the formula

```
user.field[1]
```

In each panel, if we put that field on the field list of the hidden panel link that targets the Printer Control Panel, that link will land on the current user's User ID Panel record. Note that this hidden field *must* be computed (::C) because we need it to update whenever the panel link is penetrated by our report.

With each panel having a hidden panel link that goes directly to the current user's User ID Panel record, we can now, while in the Edit Report Form screen, grab a printer control string from the current user's selected printer. We would do this:

- **F4.**
- **Tab** to that panel's hidden panel link that takes us to the current user's User ID Panel record.
- **Down Arrow.**
- **Tab** to the User ID Panel's data link that takes us to that user's selected Printer Control Panel record.
- **Tab** to the desired Printer Control Panel's printer control string field.
- **F4.**

That puts the printer control string field in the Edit Report Form screen. This is more cumbersome than either Direct Approach, but it provides each user with a different Printer Control Panel record selection. Here's the report that allows the user to change that selection without you giving them access to the User ID Panel:

```
[Destination: Screen Only]
[Based on Printer Control Panel]
-----FIRST PAGE HEADER-----

This routine will offer you a lookup of available printers.
After you highlight your choice and hit Enter, that printer
will be the one that works under your User ID until you run
this routine again.

-----Prompt for Value of Report Variable 1 -----
-----Store Value in Report Variable 1 -----
-----Stop [Sub]Report if 0 Is in Report Variable 1 -----

-----Prompt for Value of Report Variable 2 -----
-----Prompt for Value of Report Variable 3 -----
-----Store Value in Report Variable 4 -----
-----Stop [Sub]Report if 0 Is in Report Variable 4 -----

-----OTHER PAGE HEADER-----
-----Empty-----
-----TWO-LEVEL REPORT HEADER-----
-----Empty-----
-----REPORT BODY-----
-----User Chooses Next Record By LookUp-----
-----Store Value in Report Variable 5 -----

=====SUBREPORT LINK/PANEL: 0 10 =====
-----FIRST PAGE HEADER-----
-----Empty-----
-----OTHER PAGE HEADER-----
-----Empty-----
-----TWO-LEVEL REPORT HEADER-----
-----Empty-----
-----REPORT BODY-----
-----Store Report Variable 5 in Field 12 -----
-----TWO-LEVEL FOOTER-----
-----Empty-----
-----PAGE FOOTER-----
-----Empty-----
-----FINAL FOOTER-----
-----Empty-----
=====END OF SUBREPORT=====

-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----
-----TWO-LEVEL FOOTER-----
```

Printer Selection:
Indirect Approach 1

See below for details

Lookup of printers
Put choice in RV5

Virtual subreport to
User ID Panel

Its field list has
only RV2, which is
the user's User ID

Store chosen Printer
ID in current user's
User ID record

Set RV200 to 0 so
this stops after one
lookup

In the First Page Header above, after the user is told what's about to happen, he's prompted for Report Variable 1 with a continuation prompt:

Shall we continue? (Y/N):

The *Store Value in Report Variable 1* code that follows, has this formula:

```
rv1="Y"
```

The report then stops if Report Variable 1 is false, effectively stopping the report cold if the user answers anything other than Y.

Next, still in the First Page Header, the user is prompted for his User ID and Password. The first goes into Report Variable 2 and the second in Report Variable 3. The *Store Value in Report Variable 4* code examines the user's answers here by comparing them with what DataPerfect already knows about the current user from his initial application logon procedure. That is, it compares Report Variables 2 and 3 with *user.field[1]* and *user.field[2]*, respectively. If they match, the report continues. If not, it stops. So the *Store Value in Report Variable 4* code's formula would look like this:

```
rv2=user.field[1] and rv3=user.field[2]
```

The Stop code that immediately follows Report Variable 4 being set prevents a user from changing *another* user's printer selection.

Indirect Approach 2

A slightly different twist to this gives the Supervisor control over the default value found in the User ID Panel Printer ID field. If you look back at Indirect Approach 1, I have you, the definer, determining the default Printer ID for records in the User ID Panel. You put it in that field as its Initial Formula or Initial Value. Instead, you might want the Supervisor of your application to determine this, taking you out of the picture altogether, at least in terms of deciding what the default printer is going to be.

Here you have a separate panel, which we might call the System Control Panel (as opposed to the *Printer* Control Panel). The System Control Panel must have exactly one record, so to guarantee that, we'll put in a hidden G9 field (with no formula, just blank) that sits as the only field in a single-field index in this panel. There are other ways to do this, but this is a simple way to guarantee a panel has no more than one record.

This System Control Panel may contain many things that apply to various panels in the application, but in the case at hand, it has a very important field: the Default Printer ID field. That field has the same format as the Printer ID field in the User ID Panel and the Printer Control Panel (in our example, G99). And, as in the User ID Panel, that field has a data link that targets the Printer Control Panel, using a Printer Control Panel index that begins with the Printer ID field, while having only the Printer ID field on its field list.

Now go back to the User ID Panel and put in a hidden panel link that targets the System Control Panel. Since there's only one record in the System Control Panel, this hidden panel link uses any index in the System Control Panel and has no field list. It will always land on the single record in the System Control Panel.

Next, in the User ID Panel, reformulate the Initial Formula (or Initial Value) of that panel's Printer ID field. Instead of hardcoding a particular Printer ID in there (like we did in Indirect Approach 1), blank out any formula there and do this:



- **F4.**
- **Tab** to the new hidden panel link.
- **Down Arrow.**
- **Tab** to the Default Printer ID field in the System Control Panel.
- **F4.**
- Save that formula with initialization on record creation.

Each record creation in the User ID Panel will now initialize the Printer ID field to the value found in the System Control Panel's Default Printer ID field. This gives the Supervisor control over the default printer for your application. You might want to read *Controlling Data Entry: The Basic Default Panel* in my **Securing Data Entry**. This is a version of the Basic Default Panel scheme discussed there, where the Supervisor here is a special user being given control over initial values of fields in selected panels.

What These Reports Look Like

Now that we have the Printer Control Panel set up, to be used with an Indirect or Direct Method, reports will look something like this when you use the Printer Control Panel to insert printer control strings:

```

-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
Dear Fellow DataPerfect User,
I'm happy to announce publication of my new book on DataPerfect
application development. Here's a summary of what it contains:
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

] Italics On/Off codes

In the above Edit Report Form, note two selected fields surrounding the word *DataPerfect*. Those fields were obtained in one of the ways described above, depending on whether the application is set up with a Direct or Indirect Approach to using the Printer Control Panel. The field on the left is the Italics On field in the Printer Control Panel, and the one on the right, Italics Off.

Direct Report Variable Approach

There's another interesting approach to using the Printer Control Panel, but it will involve a major rewrite of all reports that use it. Here you base all reports on the Printer Control Panel. Each report starts by finding the appropriate Printer Control Panel record and stuffing the values found in all its fields into Report Variables. Then, when a printer control string is needed, you simply print the appropriate Report

Variable instead of searching through links for the appropriate Printer Control Panel field to select.

If the Printer Control Panel is set up with Direct Approach 1 or 2, this is pretty straightforward. Just make sure the report is based on the index that has the active Printer Control Panel record up front: the Reverse Moment field index. If you make the report stop after processing just that single record, you can have the intended report be a Virtual Subreport (Subreport Using Virtual Link), like this:

```
[Based on Printer Control Panel]
[Index: Reverse Moment field index]
-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Store Value in Report Variable 221 -----
-----Store Value in Report Variable 223 -----
-----Store Value in Report Variable 224 -----
-----Store Value in Report Variable 225 -----
-----Store Value in Report Variable 226 -----
-----Store Value in Report Variable 227 -----
-----Store Value in Report Variable 228 -----
=====SUBREPORT LINK/PANEL: 0 1=====
-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----

Dear Fellow DataPerfect User,

I'm happy to announce publication of my new book on DataPerfect
application development. Here's a summary of what it contains:

-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
=====END OF SUBREPORT=====
-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
```

Stuff RVs with printer control values found in current Printer Control Panel record

Virtual Subreport
This is the real report. In this case it targets Panel 1. Has no field list.

Print two RVs here instead of selecting fields

Stop after one Printer Control Panel record

In the above Edit Report Form screen, the First Page Header stuffs Report Variables with all the values it can find in the current Printer Control Panel record, one field per Report Variable. So Report Variable 221 might hold the Printer Reset field, Report Variable 222 the Bold On field, 223 the Bold Off field, etc. Once you've constructed this in one Edit Report Form screen, you can block and copy it to others in the same application (you must use **Alt-F4** for this, not Shell's screen capture; see *In Report Definition Mode* in my **The Clipboard** chapter). You might even consider creating a Shell macro to insert all that in a Edit Report Form screen. Once those Report Variables store the printer control strings, you can then just **Ctrl-F7, 1, 5** to print the appropriate Report Variable, as shown above. This is a lot easier than going through panel links to select the appropriate field in the Printer Control Panel.

To remember what printer control string each Report Variable holds, you might consider creating a Shell macro you can use to insert their mappings in the Edit Report Form screen, so it looks like this:

```
[Based on Printer Control Panel]
[Index: Reverse Moment field index]
-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----

Printer Code      RV
-----
Reset Printer      221
Bold On/Off        222/223
Underline On/Off   224/225
Italics On/Off     226/227
10, 12, 16.67 cps 228/229/230

-----REPORT BODY-----
-----Store Value in Report Variable 221 -----
-----Store Value in Report Variable 223 -----
-----Store Value in Report Variable 224 -----
-----Store Value in Report Variable 225 -----
-----Store Value in Report Variable 226 -----
-----Store Value in Report Variable 227 -----
-----Store Value in Report Variable 228 -----

=====SUBREPORT LINK/PANEL: 0 1 =====
-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----

Dear Fellow DataPerfect User,

I'm happy to announce publication of my new book on DataPerfect
application development. Here's a summary of what it contains:

-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
=====END OF SUBREPORT=====
-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----

-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--
```

RV map for printer control strings. This won't print because it's in a Two-Level section when no Two-Level Report is active. A good place for notes.

Don't forget, however, that when using **Ctrl-F7, 1, 5** to print a Report Variable, you need to tell DataPerfect its field format. If you make all the Printer Control Panel fields the same format, this isn't much of a problem (say, A10 for all of them, to accommodate those that need a lot of characters). But if you want to preserve space, you might consider using different field formats for the various Printer Control Panel fields, using smaller fields for those that don't need many characters. If you opt for the latter, you should insert each Report Variable's format in the Report Variable map above.

Indirect Report Variable Approach

We'll have to modify this Report Variable Approach somewhat if we're setting up our Printer Control Panel with the Indirect Approach. In that case, our report can't rely on the appropriate Printer Control Panel record being at the beginning of the report's index. The report, which will still be based on the Printer Control Panel, must Skip To the appropriate Printer Control Panel record, process the Virtual Subreport, and then not go to the next Printer Control Panel record.

There are a couple ways to do this. I'll just outline the simplest. Put a hidden computed field in the Printer Control Panel that updates to

```
user.field[1]
```

and a hidden panel link in that panel that has this hidden computed field as the only field on its field list. Have the link target the User ID Panel. This hidden panel link will now go from the current Printer Control Panel record to the current user's User ID Panel record.

We now base the report on a different index: the Printer Control Panel index that sorts by the Printer ID field (we were using the Reverse Moment field index before). Next, in the First Page Header, we create a Report Variable whose formula penetrates our new Printer Control Panel panel link in order to select the Printer ID field in the User ID Panel. And now we have the report, still in the First Page Header, Skip To the first record it sees in that active index that has this Report Variable as its first field's value. That is, we have the report Skip To the Printer Control Panel record that has the same Printer ID field value as found in the current user's User ID Panel record's Printer ID field.

All that said, the *only* differences between this report and the prior report is the index it's based on, and the First Page Header, which now looks like this:

```
[Based on Printer Control Panel]
[Index: Printer ID field]
-----FIRST PAGE HEADER-----
-----Store Value in Report Variable 1 -----
-----Skip To Record at Report Variable 1 -----
-----OTHER PAGE HEADER-----
```

The above *Store Value in Report Variable 1* code has a formula obtained by hitting **F4**, **Tabbing** to our new hidden Printer Control Panel panel link, hitting **Down Arrow**, **Tabbing** to Printer ID field in the User ID Panel, and hitting **F4**. That assigns Report Variable 1 the current user's Printer ID value. The Skip To code causes DataPerfect to start with the first record it sees in the current Printer Control Panel index with the Printer ID value found in Report Variable 1. And, as before, the Stop If code that follows the subreport (not shown above; see the prior Report Definition) keeps the report from processing more than one Printer Control Panel record.

This chapter is for both beginners and the experienced. The fully commented complex formulas, however, will probably lose the beginner.

Introduction

There are two places where the developer can find a type of programming language in DataPerfect. One is the .STE file, which you create when you run DPEXP.COM on an .STR file. The second place, which is the topic of this chapter, is the Specify Formula screen. This screen comes into play in three areas of an application: Report Variables, Fields, and Searches. The principles for defining a formula in all three places are the same.

Some quick notes on terms used in forming formulas. An *operator* is a mathematical or logical symbol like the plus sign (+), the minus sign (-), the greater than (>) or less than (<) signs. An *operand* is what an operator acts upon. For instance, in

$$1 + 3 > 2$$

the + sign is an operator that's taking the operands 1 and 3. The > sign also is taking two operands:

$$1 + 3$$

and

$$2$$

An *expression* is what you get when you combine an operator with the appropriate number of operands. In

$$1 + 3 > 2$$

there are two expressions:

$$1 + 3$$

and

$$1 + 3 > 2$$

A *formula* is what this chapter is all about. It's a series of expressions. What counts, however, is that it's a *well-formed* formula. We'll cover this shortly.

Formula Error Messages: A Warning

Be forewarned about something. If you create or edit a formula in the Specify Formula screen, and then exit with **F7** or **F10**, DataPerfect will stop you with an error message if it sees something wrong with the formula's syntax (i.e., if it's not well-formed). But if you just entered the Specify Formula screen to *browse* the formula without editing anything, DataPerfect won't examine the formula's syntax. So let's say you just finished creating or editing a formula in the Specify Formula screen and then hit **F7**. You were then greeted with an error message, telling you something about a syntax error. You then hit **F7** again, ignoring the error for now, figuring you'll handle it later. DataPerfect now politely lets you exit the Specify Formula screen without another error message.

Tomorrow you go back into that formula (in the Specify Formula screen) to look for the error. You don't see it right away, so hit **F7**, to handle it after lunch when you have more time. But now DataPerfect doesn't complain with an error message. Perhaps the formula is okay after all. Right?

No. DataPerfect only issues formula syntax error messages when exiting a Specify Formula screen after creating or editing a formula during that Specify Formula session. If you come back to the *ill-formed* formula later, and want to see that message again, signal DataPerfect you're editing the formula (like entering and deleting a bogus character). Then hit **F7** to see the message again.

Legal Values and Well-Formed Formulas

A *legal value* can be a character string surrounded by quotes, like one of these (parenthetical numbers in these examples aren't part of the formulas):

- (1) "A"
- (2) "DataPerfect is wonderful."
- (3) "1"
- (4) "21"

Or it can be a numerical value:

- (5) 1
- (6) 21

But it *cannot* be one of these:

- (7) A

(8) DataPerfect is wonderful.

Please note a few things about the above groups. Each formula in the first group, (1)-(4), refers to a character string, as signified by the quotes. Each formula in the second, (5) and (6), refers to numerical values. The third group, (7) and (8), is a group of illegal formulas.

Also, it's important to understand the difference between (3) and (4) on the one hand, and (5) and (6) on the other. (3) and (4) refer to character strings, not numerical values. That is, (3) and (4) aren't categorically different from (1) and (2). They just refer to strings of alphanumeric characters found between the quotes.

However, (5) and (6) are formulas that refer to things that can be added, subtracted, etc. They don't refer to character strings. They're what the character strings between the quotes in (3) and (4) *refer to*. In classical analytic philosophy circles, this is called the Use-Mention distinction. In (3), the definer is *using* the string

"1"

to *mention* the *string*

1

Whereas (5) is *using* the string

1

to *mention* the *numerical value*

1

A Report Variable can also be given a *well-formed formula*, like one of these:

- (a) 1 + 1 = 2
- (b) 1 + 1 = 3
- (c) if P1F1 > P1F2 then "Goodbye" else "Hello" endif

But the following are *not* well-formed:

- (d) if P1F1 > P1F2 then Goodbye else Hello endif
- (e) if P1F1 > P1F2 "then Goodbye else Hello endif"

Note that (b) is false even though well-formed. (d) is *not* well-formed because

Goodbye

and

Hello

have no meaning in a DataPerfect formula. However,

"Goodbye"

and

"Hello"

do have meaning in a DataPerfect formula. The latter pair of expressions refer to character strings, and are used properly in (c). In (e), these strings are used properly, but too much of the formula is surrounded by double quotes, thus causing (e) to fail to have a *then* complement to the *if* portion of the formula. Even the *then* itself, has been changed to a character string. In traditional logic, an *if-then* statement is called a *conditional* statement or proposition; the portion between the *if* and the *then* is called the *antecedent*; and the portion that follows the *then* is called the *consequent*.

Note also that, in (c)-(e),

P1F1

and

P1F2

are obtained by hitting **F4** in the Specify Formula screen and then **Tabbing** to either field P1F1 or P1F2 and hitting **F4** again. They are *not* obtained by merely typing

P1F1

or

P1F2

If I had done that, even (c) wouldn't be well-formed. Simply typed in,

P1F1

has no meaning in DataPerfect formulas. If you *do* type that in there, upon exiting the Specify Formula screen DataPerfect will tell you there's an *unrecognized word* in the formula.

About The Rest of This Chapter

I'm not covering all available DataPerfect functions and operators in this chapter. Rather, I'd like to cover a few of them that I feel many find confusing.

String Identity

This section will save you a lot of grief. The identity operator works as expected when it comes to numerical values. The following are all true:

```
1    = 1
2    = 1+1
1+1  = 2
1    < 2
1+2  > 2
1    <> 2
```

And all these are false:

```
1 = 2
1 > 2
2 > 1+1
```

Nothing odd about this so far. But these are true:

```
"abc" = "aB"
"a"   = "AbC"
```

Now, that's odd. Well, both those statements are true because DataPerfect considers an identity statement between two strings to be true just in case *there's an ordered case-insensitive match on all characters contained in the shortest operand*. So the first string identity statement above is true because, independent of case, the shortest of the two operands (*aB*) begins the longest of two operands (*abc*). Don't forget I said *independent of case*. And the second string identity statement is true because, independent of case, the shortest operand (*a*) begins the longest (*AbC*).

What does this odd way of doing things get you? Say you want a formula that tests whether or not the value in a field begins with *p*. Even though that field, which we'll call P1F1, is an A25 field, the following formula will do:

```
P1F1 = "p"
```

That formula will be true for all and only records where the string in field P1F1 begins with *p* or *P*, no matter how long that actual string is.

But this can have some unintended results. If you were trying to use a formula like

```
P1F1 = "sam"
```

to find all records in that panel with *Sam* in field P1F1, you might not realize such a formula will also be true for records with *Samantha* in that field. Again, that formula will be true for any record with a string in field P1F1 that starts with *sam*.

Perfect Matches and the Identity Operator

So, what if you want a formula to test for a *perfect match*, not just strings that begin with some string or another? Say you have a report that's supposed to process only records where the value in a U5 field, P1F1, is precisely *AB*, not *ABC*, *ABX*, etc. Well, let's say you set Report Variable 1 to

P1F1 = "AB"

in the beginning of this report's Report Body, and then insert a Skip If Report Variable 1 Is False code right after it.

You should realize by now that that won't work. This report will process all records where *AB* starts the string found in P1F1. That will include records with *ABC*, *ABX*, etc, in that field.

To get a *perfect match* here, you can set Report Variable 1 to one of these:

- (1) `cat.t[P1F1;"#%"] = "AB#%"`
- (2) `P1F1"#%" = "AB #%"`

where

#%

is a string the user will never put in P1F1. I could have used any such string here.

In example (1), I use the CAT.T function to strip trailing spaces in the value found in P1F1. So, let's say P1F1 is *ABC*. Then the CAT.T expression in (1) will produce

#%ABC

yielding this identity statement:

"ABC#%" = "AB#%"

Look carefully at the above string identity statement. Neither string begins the other. So it's false. But if P1F1 was *AB*, then the CAT.T expression in (1) would produce

AB#%

yielding this identity statement:

"AB#%" = "AB#%"

Now that's true.

Example (2) avoids the CAT.T function. To do that, however, I need to know exactly how long field P1F1 is. In this case, P1F1 is a U5 field. If the user enters *AB* in field P1F1, DataPerfect actually stores a five-character string: *AB* with three spaces after it. So if you don't strip these spaces with the CAT.T function, you'll need to pad the string identity expression on the right with enough spaces to total five, before the dummy string. To make this more clear, suppose the user entered *AB* in field P1F1. Then the lefthand expression in example (2) would produce

AB # \$ %

not

AB# \$ %

So, in forming the righthand expression in (2), I must take this into account and add three spaces following the two-character expression *AB*:

AB # \$ %

That's the reasoning behind expression (2) looking like this:

P1F1"# \$ %" = "AB # \$ %"

Prompting for a Report Variable

Now, the above discussion concerned forming Report Variables that test records for a perfect match on a character string. Alternatively, a report might *prompt* the user for Report Variable's value. If that's the case, the Report Variable will carry a field format with it (don't forget that you must tell the report definition what field format the Report Variable has). This changes things. Let's explain.

Let's say the report's First Page Header prompts the user for Report Variable 1, which is formatted U5. Now I need a Report Variable 2 in the Report Body to test each record with a formula like one of the following:

- (1) P1F1"# \$ %" = RV1"# \$ %"
- (2) cat.t[P1F1; "# \$ %"] = cat.t[RV1; "# \$ %"]

What's going on here that makes this so different from the previous cases that didn't involve a Prompt For code? Well, since both P1F1 and Report Variable 1 have the same field format, you must either use CAT.T on *both* of them, or *neither* of them. Using CAT.T on only one of them, as we did before, fails. To see why it would fail, let's first see what happens if we use (1) and (2) when the user enters *AB* for Report Variable 1 and P1F1 is indeed *AB*. (1) and (2) will produce the following *true* statements respectively:

- (1a) "AB # \$ %" = "AB # \$ %"
- (2a) "AB# \$ %" = "AB# \$ %"

But now consider the two possibilities of applying CAT.T to only one side but not the other:

```
(3)   cat.t[P1F1;"#$$%"] = RV1"#$$%"
(4)   P1F1"#$$%" = cat.t[RV1;"#$$%"]
```

In the same situation, (3) and (4) produce the following *false* statements respectively:

```
(3a)  "AB#$$%" = "AB    #$$%"
(4a)  "AB    #$$%" = "AB#$$%"
```

On the other hand, if the user enters *ABC* for Report Variable 1, and P1F1 is still *AB*, we get the following results, all of which are false:

```
(1a)  "AB    #$$%" = "ABC    #$$%"
(2a)  "AB#$$%" = "ABC#$$%"
(3a)  "AB#$$%" = "ABC    #$$%"
(4a)  "AB    #$$%" = "ABC#$$%"
```

As you may have surmised by now, the only way (3) and (4) will ever be true is when the value in P1F1 fills that field (it's five characters long) and the user enters that precise five-character string for Report Variable 1. Otherwise, it will yield a false statement. On the other and, (1) and (2) will always be true or false correctly.

So, instead of choosing between

```
(3)   cat.t[P1F1;"#$$%"] = RV1"#$$%"
```

and

```
(4)   P1F1"#$$%" = cat.t[RV1;"#$$%"]
```

we should choose between these:

```
(5)   P1F1"#$$%" = RV1"#$$%"
(6)   cat.t[P1F1;"#$$%"] = cat.t[RV1;"#$$%"]
```

Again, the difference here is that prompted-for Report Variables *are formatted*. Before we entertained using prompted-for Report Variables, we were dealing with *unformatted* Report Variables. This makes a big difference that has confused many developers.

One final note. I made this discussion easy by choosing a *U5* format. If I wanted a perfect match on an *A5* field, I couldn't do it. DataPerfect has no way to perform a *case-sensitive* test on a string with the identity operator.

CASES Statements vs. IF-THEN Statements

Of all the formula manipulation questions I get, I think the most common involves showing how to use CASES statements in place of IF-THEN statements. You'll feel at ease with CASES statements after reading this.

You use a CASES or IF-THEN statement when you need conditional logic. Let's say you're a loan broker and have an application that tracks the history of various loans you manage. You want a field in the Loan Panel that displays *MATURED* if the displayed loan matured. You format it U7::C (it's a computed field so it's always current), and you assign it this formula:

```
if P1F1 >= today then "MATURED" else "" endif
```

Assuming P1F1 is the loan's Maturity Date, this is very straightforward. There's no reason to use a CASES statement here. The conditions to be analyzed are very simple.

But suppose you have more than just one condition to analyze. Say, in this same panel, you want a different field showing you the day of the week the displayed loan matures. Whether a loan matures on a Sunday or a Friday makes a big difference in the way you run your business. You could use an IF-THEN statement, but it's not as readable as a CASES statement. Here are both:

IF-THEN statement

```
if day.of.week[P1F1]=1 then "Monday"   else
if day.of.week[P1F1]=2 then "Tuesday"  else
if day.of.week[P1F1]=3 then "Wednesday" else
if day.of.week[P1F1]=4 then "Thursday" else
if day.of.week[P1F1]=5 then "Friday"   else
if day.of.week[P1F1]=6 then "Saturday" else
if day.of.week[P1F1]=7 then "Sunday"   else
endif endif endif endif endif endif endif
```

CASES statement

```
day.of.week[P1F1] cases
  case cv=1 of "Monday"   endof
  case cv=2 of "Tuesday"  endof
  case cv=3 of "Wednesday" endof
  case cv=4 of "Thursday" endof
  case cv=5 of "Friday"   endof
  case cv=6 of "Saturday" endof
  case cv=7 of "Sunday"   endof
endcases
```

A few points about CASES statements. First, as soon as a Case condition is true, the statement logic terminates and grabs the consequent to the right of the *of*. This is similar to the series of IF-THEN lines above, where the logic ends with the first condition it finds true, and then grabs the consequent to the right of the *then*.

Second, if you want to reference the Case Variable in the CASES statement header, all you need to use is the string *cv*. This is quite different from an IF-THEN

statement, where you must reference that object over and over again, as we did above. In both the CASES and the IF-THEN statements, we were continuously referencing

```
day.of.week[P1F1]
```

But I only had to write that string (and select P1F1) once in the CASE statement.

Third, in a CASES statement, you don't need to use *cv* at all. A common misunderstanding about CASES statements is that you must use *cv* on every Case line, like I did above. Consider the following:

```
day.of.week[P1F1] cases
  case P1F1<=today of "MATURED"   endof
  case cv=1         of "Monday"   endof
  case cv=2         of "Tuesday"  endof
  case cv=3         of "Wednesday" endof
  case cv=4         of "Thursday" endof
  case cv=5         of "Friday"   endof
  case cv=6         of "Saturday" endof
  case cv=7         of "Sunday"   endof
endcases
```

You might consider what day of the week the loan matures to be irrelevant if it *already* matured. So here the user is told the loan has matured if it has, otherwise they're told the day of the week it matures. Note that the first Case line does *not* use the Case Variable.

In fact, though a Case Variable must be defined, I could write a CASES statement that *never* references it. The logic of the CASES statement is still the same. It stops as soon as it finds a true condition on the left of an *of*, and then yields the consequent on the right. It cares not one bit if you ever reference the Case Variable. So you can just make up a useless Case Variable header line, in order to make use of the conditional logic in the subsequent lines. For instance, the following is a well-formed CASES statement:

```
P1F1 cases
  case P1F2 > P1F3 of "Goodbye" endof
  case today < P1F4 of "Hello"   endof
  case P1F4 = "Sally" of "Thanks" endof
  default "Greetings"
endcases
```

About the above, note two things. First, the Case Variable in the Case Header line is never referenced (that is, this CASES formula never uses the Case Variable *cv*). Second, unlike the other CASES statements in this section, this one has a *default* line. This expresses what the CASES statement will yield if none of its Case lines yields a result. If no Case line yields a result and there's no *default* line, then the CASES statement yields a blank if it's part of an alphanumeric field or Report Variable, or 0 if it's part of a numerical field or Report Variable.

The beauty of the CASES statement, however, is availability of using the Case Variable over and over again, line by line. It not only makes it easier to read than its IF-THEN counterpart, but it makes it easier to change the formula later when

you decide you were referencing the wrong field (or want to change it later for whatever your reason). Whereas you'd have to change every occurrence of that field in an IF-THEN formula, you'll only have to change one occurrence of that field in the CASES statement, provided it's the Case Variable.

APPLY.FORMAT and CONVERT

Once you understand APPLY.FORMAT, you'll use it all the time. You'll also use CONVERT, but nowhere near as often as APPLY.FORMAT.

APPLY.FORMAT converts an entity into *text*. Contrary to what many DataPerfect users think, the entity being converted by APPLY.FORMAT need not be a numerical value. It can itself be another text string. Whereas you'll most often use APPLY.FORMAT to convert numerical field's value to a text string (like a G9999 formatted numerical value to an A4 formatted text string), you may sometimes need to, say, convert an A field's contents to an upper case string (like an A8 formatted string to a U8 formatted string). The entity being converted can be referenced by a field name like *P1F1*, or a report variable like *RVI*, or some other expression, like a CAT.T expression. What's important here is what the end result is: *a text string*.

On the other hand, CONVERT converts an entity to a *numerical value*. Again, like APPLY.FORMAT, the entity being converted by CONVERT can be either text or numerical value. What's different is the result: *a numerical value*.

APPLY.FORMAT

Let's demonstrate these two functions with examples. Consider the following typical use of APPLY.FORMAT, which also shows a common use of a CASE statement I frequently use in panels:

```
(day.of.week[P1F1] cases
  case cv = 1 of "Monday"      endof
  case cv = 2 of "Tuesday"     endof
  case cv = 3 of "Wednesday"   endof
  case cv = 4 of "Thursday"    endof
  case cv = 5 of "Friday"      endof
  case cv = 6 of "Saturday"    endof
  case cv = 7 of "Sunday"      endof
endcases)
", "
(month[P1F1] cases
  case cv = 1 of "January"      endof
  case cv = 2 of "February"     endof
  case cv = 3 of "March"        endof
  case cv = 4 of "April"        endof
  case cv = 5 of "May"          endof
  case cv = 6 of "June"         endof
  case cv = 7 of "July"         endof
  case cv = 8 of "August"       endof
  case cv = 9 of "September"    endof
  case cv =10 of "October"      endof
  case cv =11 of "November"     endof
```

```

        case cv =12 of "December" endof
endcases)
" "
apply.format["GZ9;;S";day[P1F1]]
","
apply.format["N9999";year[P1F1]]

```

That formula looks at a D99/99/99 or D99/99/9999 field (P1F1) and then produces a *text string* like

```
Monday, June 13, 1994
```

for an A29 field. This is tricky because it has *numbers* in it. At first, you might think the lines

```

apply.format["GZ9;;S";day[P1F1]]
apply.format["N9999";year[P1F1]]

```

could have been

```

day[P1F1]
year[P1F1]

```

But the latter don't yield *text strings*. They yield *numerical values*. We convert these numerical values to text strings with APPLY.FORMAT.

The second APPLY.FORMAT line above (the *year* line) is pretty straightforward. You simply put its numerical format in quotes, to its left, followed by a semi-colon. That tells DataPerfect that you want its numerical value converted to a text string in the format that would typically be displayed by an N9999 field.

But the first of these two lines (the *day* line) has something else interesting in it. Notice I used a modifier (;S) that's supposed to be used in reports (it's a Print Mode Indicator, not a Display Modifier), but I use this entire formula in *panels* as well as reports. As I mentioned in *Sneaking Print Mode Indicators into Panel Fields* in my **Reports: Fields** chapter, though the DataPerfect manual fails to mention this, you can also use Print Mode Indicators with the APPLY.FORMAT function in *panel* field formulas. I used it here to strip the leading space of a GZ9 field. Otherwise the formula would convert 01/01/94 to

```
Saturday, January 1, 1994
```

instead of the more desirable

```
Saturday, January 1, 1994
```

This trick comes in handy when you want to display text in a panel in a more readable fashion than that in which it's stored in some other place. For instance, you might want text right-aligned in the upper right corner of a panel:

```

John Doe
1234 Elm St., Apt. 2

```

```
Pleasant Town, CA 99390
213/555-5678 213/555-3565
```

Each of the first two lines arise from formulas like these:

```
apply.format["A30;;R";P1F2P12F1]
```

and

```
apply.format["A30;;R";P1F12P2F2]
```

where P1F12P2F1 and P1F12P2F2 are A30 fields in another panel, selected via panel link P1F12. The first holds the person's name and the second, his street address.

The third line arises from a formula like this:

```
if P1F12P2F2="" then "" else
  apply.format
  [
    "A40;;R";
    cat.t
    [
      P2F12P2F3;
      ", "P2F12P2F4;
      "apply.format["N99999";P1F12P2F5]
    ]
  ]
endif
```

It looks first to see if there's a street address for this person (first line). If not, the formula yields a blank, otherwise it yields a right-aligned A40 field. Note the use of APPLY.FORMAT in the concatenation statement (CAT.T) in that formula.

CONVERT

Again, APPLY.FORMAT converts an entity into *text*. Let's discuss CONVERT, which converts an entity to a *numerical value*. A simple CONVERT example would be

```
convert["GZZ9";P1F1]
```

Here the definer wants to take the value found in P1F1 and turn it into to a numerical value formatted as GZZ9. P1F1 might be a U3 field, thus holding numerical *text*. But it could also be a N999 field, and the user wants to convert it to a GZZ9 format. Again, CONVERT allows you to convert the contents of *any* sort of field (text *or* numerical) into a numerical value. It's not just for converting *text* into numerical value. A more involved use of CONVERT is discussed in the next section of this chapter.

SUBSTRING and SUBFIELD

SUBFIELD

I find the SUBFIELD function particularly useful when importing data, when that data isn't in the format I'd like it to be in my database. For instance, consider dates in a WordPerfect secondary merge document that look like this:

January 21, 1993

If you want a date like that to be placed in a D99/99/9999 field in your database during an import, use the SUBFIELD function.

Just create a field large enough to hold the date text string you see above. An A18 field will accommodate any such date *text string*. That's the field that will grab the incoming date text. Now we need a D99/99/9999 field that will hold the converted date *numerical value*. Don't forget that a date field holds a *numerical value* (the number of days since March 1, 1900) not a *text string*. The formula we use for this date field is a nice one, in that it demonstrates many concepts all in one formula. In constructing it, we'll consider our A18 field to be P1F1.

First, because we're changing a *text string* to a *numerical value*, we need the CONVERT function. So we start like this:

```
convert
[
  "D99/99/9999";
]
```

Now we need to come up with the formula for the text string that needs converting, and put it between the semi-colon and the second right-angle bracket.

Let's start with the month portion of the text string. For this we can use the following CASES statement followed by a forward slash:

```
subfield[P1F1;" ";1] cases
  case cv = "January"    of "01" endof
  case cv = "February"  of "02" endof
  case cv = "March"     of "03" endof
  case cv = "April"     of "04" endof
  case cv = "May"       of "05" endof
  case cv = "June"      of "06" endof
  case cv = "July"      of "07" endof
  case cv = "August"    of "08" endof
  case cv = "September" of "09" endof
  case cv = "October"   of "10" endof
  case cv = "November"  of "11" endof
  case cv = "December"  of "12" endof
endcases
"/"
```

Let's discuss this. The CASES statement's header line is built on the SUBFIELD function. It reads, in essence,

Consider P1F1 a series of subgroups delineated by the space, and take the second subgroup.

The first argument in the SUBFIELD function is the field being considered. The second, surrounded by quotes, is the mask or masks that delineates the subfields (the space). The third tells us what subgroup is picked, counting from the left.

Two things about that second argument. *First*, it accepts multiple masks. Let's say you wanted the SUBFIELD function to delineate subfields on more than one mask. Say you wanted it to delineate subgroups based on either the space or the comma, or both. Then, in our above example, the first line of the CASES statement would look like

```
subfield[P1F1;" ,";1] cases
```

or

```
subfield[P1F1;" , ";1] cases
```

Second, the mask used in this function is surrounded by quotes except for one special situation: when you want the mask to be the End Of Line character (carriage return). DataPerfect's author reserved the number 0 for that. So suppose P1F1 is a variable-length text field with the following value:

```
Mary had a little lamb.  
I like lambs.
```

In this case,

```
subfield[P1F1;0;2]
```

yields

```
I like lambs.
```

Back to using only the space as the mask. The first P1F1 subgroup delineated by the space is the month text string. Now, the next twelve lines say to use the *numerical text string* in place of the *alpha text string* for that month. Note I surrounded the numerical text string on each of these twelve lines with quotes. Don't forget that, at this stage in the formula, we're simply changing one text string to another text string. Later, the CONVERT function will convert this stuff to a numerical value formatted as D99/99/9999. But we're still dealing with text strings here. That is, we're still just putting text between the semi-colon and the second right-angle bracket in this formula:

```
convert  
[  
  "D99/99/9999";  
]
```

One last note about that CASES statement. I followed it with a quoted forward slash. That's for the slash that separates the month from the day, in expressions like

01/23/1993

If it wasn't surrounded by quotes, it would be evaluated as a division operator.

So far, we have the text version of

01/

in our formula partially completed as this:

```
convert
[
  "D99/99/9999";
  subfield[P1F1;" ";1] cases
    case cv = "January" of "01" endof
    case cv = "February" of "02" endof
    case cv = "March" of "03" endof
    case cv = "April" of "04" endof
    case cv = "May" of "05" endof
    case cv = "June" of "06" endof
    case cv = "July" of "07" endof
    case cv = "August" of "08" endof
    case cv = "September" of "09" endof
    case cv = "October" of "10" endof
    case cv = "November" of "11" endof
    case cv = "December" of "12" endof
  endcases
  "/"
]
```

Now, the day portion of the date. That's much simpler, because the day portion of the string is already in *numerical text format*. The day portion of the formula, followed by its forward slash, would look like this:

```
subfield[P1F1;" ",2]
"/"
```

That's read

Consider P1F1 a series of subgroups delineated by the space or the comma, and take the second subgroup.

In the string

January 23, 1993

there are three such groups, the second being

23

Though we *should* use the *or the comma* part of that formula when we extract the numerical month portion of the date, we don't need to. If we leave out the *or the comma* part of this formula, and use

```
subfield[P1F1;" ";2]
```

instead of

```
subfield[P1F1;" ",";2]
```

we would technically be extracting

```
23,
```

instead of

```
23
```

As it turns out, though, that would still work fine. The D99/99/9999 conversion would have interpreted the former as a 23 and ignored the comma. But let's stick with technically more correct

```
subfield[P1F1;" ",";2]
```

Okay, now the year portion. That's simply

```
subfield[P1F1;" ";3]
```

Our completed formula would now be the following:

```
convert
[
  "D99/99/9999";
  subfield[P1F1;" ";1] cases
    case cv="January"    of "01" endof
    case cv="February"  of "02" endof
    case cv="March"      of "03" endof
    case cv="April"      of "04" endof
    case cv="May"         of "05" endof
    case cv="June"        of "06" endof
    case cv="July"        of "07" endof
    case cv="August"      of "08" endof
    case cv="September"  of "09" endof
    case cv="October"     of "10" endof
    case cv="November"   of "11" endof
    case cv="December"   of "12" endof
  endcases
  "/"
  subfield[P1F1;" ",";2]
  "/"
  subfield[P1F1;" ";3]
]
```

month

date as

numerical

text

date as

numerical

value

SUBSTRING

That's how I would write that formula with the SUBFIELD function used for all parts of the date string. In the case of the month portion of the date string, it looks like I need to use the SUBFIELD instead of the SUBSTRING function because the months were denoted by words of varying lengths. But since each month is uniquely determined by its first three characters (actually, the first two will do, but that's less readable in my opinion), I could have used the SUBSTRING function. Take a look at the SUBSTRING function help screen:

```
substring[text;start;lnth]
```

Produces "lnth" characters beginning at "start"
If start is 0, produces last "lnth" characters

As you can see, the SUBSTRING function relies on counting characters from a fixed position in a string. So I could have constructed the formula like this:

```
convert
[
  "D99/99/9999";
  substring[P1F1;1;3] cases
    case cv="Jan" of "01" endof
    case cv="Feb" of "02" endof
    case cv="Mar" of "03" endof
    case cv="Apr" of "04" endof
    case cv="May" of "05" endof
    case cv="Jun" of "06" endof
    case cv="Jul" of "07" endof
    case cv="Aug" of "08" endof
    case cv="Sep" of "09" endof
    case cv="Oct" of "10" endof
    case cv="Nov" of "11" endof
    case cv="Dec" of "12" endof
  endcases
  "/"
  subfield[P1F1;" ,";2]
  "/"
  substring[P1F1;0;4]
]
```

month

date as
numerical
text

date as
numerical
value

-slash
-day
-slash
-year

Note also that I used SUBSTRING for the year as well the month in the above formula, though I opted to retain my use of SUBFIELD for the day.

A String Identity Variation

Let's go a step further. Because of what I mentioned in *String Identity* in this chapter, I can actually do away with the SUBSTRING and SUBFIELD functions in the *month* part of this formula. Consider the following:

```
convert
[
  "D99/99/9999";
  P1F1 cases
    case cv="Jan" of "01" endof
  ]
```

case cv="Feb" of "02" endof			
case cv="Mar" of "03" endof			
case cv="Apr" of "04" endof			
case cv="May" of "05" endof			
case cv="Jun" of "06" endof	month		
case cv="Jul" of "07" endof		date as	date as
case cv="Aug" of "08" endof		numerical	numerical
case cv="Sep" of "09" endof		text	value
case cv="Oct" of "10" endof			
case cv="Nov" of "11" endof			
case cv="Dec" of "12" endof			
endcases			
"/"	-slash		
subfield[P1F1;" ,";2]	-day		
"/"	-slash		
substring [P1F1;0;4]	-year		
]			

In the above formula, I can use the first three characters of a month name because of two things. First, it only takes the first three characters to uniquely identify a month name. Second, an identity statement is true just in case the shortest string (the three-character month name string) begins the longer one (*cv*, or P1F1). Re-read *String Identity* earlier in this chapter if this isn't clear.

CONTAINS

The CONTAINS function can be a little confusing unless you realize it's best thought of as a *match* function (thanks to Paul Durban for describing the CONTAINS function this way in my 1994 Atlanta seminar). The CONTAINS function is used to test a fixed-length or variable-length text field for the presence of a character string. For instance, you might want to define a report that prints all records with

Sally Adams

in variable-length text field P1F1, formatted A30A4. You might think this will work:

contains[P1F1;"Sally Adams"]

But that fails unless P1F1 is exactly eleven characters long (an A11 field). That is, the above formula is true only for a *perfect match* on the entire contents of P1F1. It will be false, for instance, if P1F1 is an A20 field containing

Sally Adams is nice

Here's the formula that works:

contains[P1F1;"*Sally Adams*"]

That too is true if the argument to the right of the semicolon is a perfect match on the entire contents of P1F1. But with the wildcard characters in there (the asterisks), it allows for text to precede or follow

Sally Adams

This chart should clear all this up for you:

The CONTAINS Function Delineated	
Formula	Truth Conditions
<code>contains[P1F1;"Sally Adams"]</code>	<i>sally adams</i> completely fills P1F1 (so P1F1 must be 11 characters long).
<code>contains[P1F1;"*Sally Adams"]</code>	<i>sally adams</i> is at the end of the text field P1F1 (not followed by anything, even a space).
<code>contains[P1F1;"Sally Adams*"]</code>	<i>sally adams</i> is at the beginning of text field P1F1.
<code>contains[P1F1;"*Sally Adams*"]</code>	<i>sally adams</i> is anywhere in text field P1F1.

Spaces and Carriage Returns in Formulas

Now, some comments on how I use spaces and carriage returns in developing the above formula. I use them to make the formula more readable. In analyzing a formula, DataPerfect ignores spaces and carriage returns outside quotes. So, for instance,

```
apply.format["G999";P1F1]
```

can just as well be written as

```
apply.format
["G999";P1F1
]
```

or

```
apply.format
[
"G999";P1F1
]
```

or

```
apply.format
[
"G999";
P1F1
]
```

or

```

apply.format
[
  "G999"
;
P1F1
]

```

Likewise,

```
P1F1="A"
```

is equivalent to

```
P1F1                                ="A"
```

and

```
P1F1                                =          "A"
```

and

```
P1F1                                =
                                "A"
```

but not

```
P1F1=" A"
```

or

```
P1F1="A "
```

or

```
P1F1=" A "
```

Troubleshooting Formulas

Problem

You have a field formula on a numerical field that occasionally causes the field to update to a string of asterisks (****) instead of a number.

Solution

Your field formula is attempting to yield a number too big for the field format. Increase the field format size.

Problem

A field formula that uses a link to get its data doesn't update properly, even though the link is defined properly.

Solution

The field list of the involved link must precede the link in the Edit Order for this field formula to work.

Problem

When attempting a Search on a formula with a string identity, you get too many matches.

Solution

Read *Perfect Matches and the Identity Operator* under *String Identity* above. You're not understanding how string identities work.

Problem

When attempting a Search on a formula with a CONTAINS function, you get too few matches.

Solution

Read the CONTAINS section above. You're not treating this function as a *match* function.

Beginners should peruse this chapter but not expect to grasp a lot of it. This is really for the experienced DataPerfect application developer.

Introduction

To *iterate* is to do something repetitively. In the world of DataPerfect, Iteration Control is a set of possible Report Definition options that, in general, determine whether or not the current or next record will be processed or skipped. For instance, you might want a report to produce an Itemized Ledger for one specific Account, and do so by printing only services rendered (i.e., excluding payments) within a date range. If all charges and payments are in the same Transaction Panel, such a report must skip payments, as well as skip all records with dates outside the date range. This involves using some sort of Iteration Control.

When in the Edit Report Form screen, you access the Iteration Control menu with **Ctrl-F7, A** in the First Page Header, or **Ctrl-F7, 9** in the Report Body. In either location, the displayed menu is the same:

```
Report Iteration Control
1 - Skip Record if RV is False
2 - Stop [Sub]Report if RV is False
3 - Skip To Record At RV
4 - Choose Next Record Using LookUp
5 - Repeat Record if RV is True (not 0)

Selection: 0
```

With the exception of Iteration Control option 4 (Choose Next Record Using LookUp), Iteration Control involves Report Variables, so you're going to need to know how to store formulas in Report Variable to understand this section. Read up on Report Variables in the DataPerfect manual or my **Report Variables** chapter if that's new to you. Let's start with the first Iteration Control option.

Skip Record if RV is False

Before version 2.3, this was the only Iteration Control option DataPerfect application developers had. Though you can use it in the First Page Header, you're almost always going to be use it in the Report Body. When DataPerfect sees that code in the Report Body, it uses the Report Variable referenced by that code to evaluate each record in the report's index. It will skip any record that makes that Report Variable false. Or better, in terms of figuring out how to construct the Report Variable's formula, DataPerfect will include all and only records that make that Report Variable true, skipping all others.

Let's say you want a report to print a letter to each female teacher in the database, skipping all males. Assuming the Sex field in the Teacher Panel is P2F5, we would probably want to work out a way to use that field with the Skip If code. The Skip If code will work if we can come up with a Report Variable that will be true of all and only Teacher Panel records that reference female teachers.

After calling the Specify Formula screen for Report Variable 1 with **Ctrl-F7, 1, 4, 1, Enter**, we can insert this formula:

P2F5="F"

P2F5, of course, was obtained using the **F4** key and selecting the Sex field in the Teacher Panel. Now *that* formula is true of each record that has an *F* in the Sex field, and false of all others. This is the sort of Report Variable we want the Skip If code to evaluate here. After saving that formula for Report Variable 1 by hitting **F7** on the Specify Formula screen, our Edit Report Form screen looks like this:

```

-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Store Value in Report Variable 1 -----
-----TWO-LEVEL FOOTER-----
--Empty--

```

Note that I put that Store Value code in the Report Body. That's the only logical place to put it, since that's the only place in the Report Definition that processes or evaluates each record in the index, one at a time.

Next, I **Ctrl-F7, 9, 1** to insert the Skip If code:

```

-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Store Value in Report Variable 1 -----
-----Skip Record if 0 (False) Is in Report Variable 1 -----
-----TWO-LEVEL FOOTER-----
--Empty--

```

The above report will run the Report Body on all and only those Teacher Panel records with an *F* in the Sex field. So if I put some fields and text in that Report Body, along with a Skip to Bottom of Page code (**Ctrl-F7, 4**) after all that, I can print letters to all and only females in the database, starting with a new page for each female:

```

-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Store Value in Report Variable 1 -----
-----Skip Record if 0 (False) Is in Report Variable 1 -----

Dear Ms. [REDACTED],
I'm sending you this letter ...

[etc.]

-----Skip to Bottom of Page-----

-----TWO-LEVEL FOOTER-----
--Empty--

```

Rest of Report Body
(the actual letter)

Each time the above report sees a female, it processes the rest of the Report Body (everything that follows the Skip If code), including sending a Page Eject to the printer. Each time it sees a male, it passes to the next record without processing the rest of the Report Body.

The Skip If code is powerful, and we got along with it being our only Iteration Control option for a long time. But it can be frustrating to use with large databases in certain types of reports. Consider a database that holds records of all voters registered in California. Among other things, it holds the date they first registered as a voter in California. Consider a report that prompts the user for Start and End Dates, and then proceeds to print a list of citizens whose First Time Registration Date in California falls within that date range. Assume the database has all voters whose First Time Registration Date is within the last forty years, and you want a list of those whose First Time Registration Date is between June 22, 1995 and January 15, 1996.

If the only Iteration Control option you have for this is the Skip If code, that report will physically examine each and every record in the database while printing out information on a relatively small percentage of them. Don't forget that the Skip If code doesn't limit how many records that report *inspects*. It only limits the number it *prints*. What would be ideal is if we can get this report, when indexed by First Time Registration Date, to cause DataPerfect to actually start with the first record in the desired date range and stop as soon as it sees a record beyond that date range, without having to examine all the records outside the desired date range.

Consider a different, slightly more complex, but common, report: one that prompts for Account Number, Start Date, and End Date, and then generates a ledger of transactions that fall within that date range for that Account. Limited only to using the Skip If code (again, which is all we had before version 2.3), we were limited to defining a report that did this:

- * 1. Starting at the beginning of an Account Panel index, it inspects and, using the Skip If code, it skips records until finding the one with the chosen Account Number.
- 2. It drops to a subreport targeting the Transaction Panel.
- * 3. It inspects and, using the Skip If code, skips all Transactions that precede the Start Date.
- 4. It processes all Transactions that are within the date range.

- * 5. It inspects and, using the Skip If code, skips all Transactions that follow the End Date.
- 6. It returns to the Account Panel section of the report.
- * 7. It inspects and, using the Skip If code, skips all Account Numbers that follow the desired Account Number.
- 8. It ends.

The steps shown with asterisks (1, 3, 5, and 7) all slow the report terribly when applied to large databases. But, if, instead of the Skip If code, we use version 2.3's Skip To Record At RV and Stop [Sub]Report if RV is False codes (which I'll hitherto call *Skip To* and *Stop If*) in coordination with each other, we can define a report that does this:

- A. Using the Skip To code, it goes directly to the first Account Panel record with the chosen Account Number.
- B. It drops to a subreport that targets the Transaction Panel.
- C. Using the Skip To code, it goes directly to the first Transaction Panel record with the chosen Start Date.
- D. It processes all Transactions that are within the date range.
- E. Using the Stop If code, it stops processing Transactions as soon as it comes to one after the chosen End Date.
- F. It returns to the Account Panel.
- G. Using the Stop If code, it stops the entire report as soon as it's done processing the chosen Account Panel record.

The above report skipped all the following time consuming steps of the other report definition that relied completely on the old Skip If code:

- * 3. It inspects and, using the Skip If code, skips all Transactions that precede the Start Date.
- * 5. It inspects and, using the Skip If code, skips all Transactions that follow the End Date.
- * 7. It inspects and, using the Skip If code, skips all Account Numbers that follow the desired Account Number.

Let's explain how to implement the Skip To and Stop If codes.

Skip To and Stop If

I find it easier to discuss Skip To and Stop If codes together, since that's how you'll usually use them. Again, I use *Skip To* to refer to what DataPerfect calls *Skip To Record At RV*, and *Stop If* to refer to what DataPerfect calls *Stop [Sub]Report if RV is False*.

Though the menu that gives you access to these Edit Report Form screen codes (the Iteration Control menu) is found only in the First Page Header or the

Report Body, you can always move these codes to any place you want on the Edit Report Form screen. This may come in handy in some rare cases, but you'll usually be putting them in either the First Page Header or the Report Body. To Copy a code from one location of an Edit Report Form screen to another (even to another Report Definition in the same application), just position the cursor to its immediate left and Block it with **Alt-F4, Down Arrow, F10**. To Move instead of Copy the code, use **Ctrl-F4** instead of **F10** after Blocking it. In either case, you Paste with **Ctrl-F4** in the new location.

The Variable Entity in Skip To Operations

Value vs. Statement

The Report Variable you use in defining Skip To code should hold a *value of* a field, not a *statement about* a field. A *value of* a field might be, say, the Account Number

1009

or the field

P1F1

Whereas, a *statement about* a field might be

P1F1=1009

or

P2F1>=RV2

In the report that concerns us here (the Itemized Ledger report), such a Report Variable will hold a numerical value: Account Number or Start Date. That is, we're going to define one part of this report to Skip To the Account Number chosen (in the Account Panel), then Skip To the Start Date chosen (in the Transaction Panel).

This contrasts with the other three Iteration Control options that also use Report Variables (Skip If, Stop If, and Repeat If). Each of these three uses a Report Variable that holds a *statement*, and consequently looks to see if that statement is *true of* the current record. Because the Skip To option uses a Report Variable holding a *value*, it looks to see if that value *exists in* the primary sorting field of the current record.

The Primary Sorting Field

When using Skip To, the primary sorting field of the active index must be compatible with the value assigned to the Report Variable in the Skip To definition. If the Skip To code isn't in a subreport, then the primary sorting field is the first field in the main report's index. If the Skip To code *is* in a subreport, then the primary sorting field is the field in the *subreport's* index that immediately follows the matching fields in the subreport's link field list.

So, in the latter case (where the Skip To code is in a *subreport*), let's say the main report is based on the Account Panel. Further, the link that governs the *subreport* targets the Transaction Panel, has a link field list consisting only of the Account Number field, and uses Index 1 of the Transaction Panel. In this case, Index 1 of the Transaction Panel looks like this:

Account Number, Transaction Date, Transaction Number

The primary sorting field of that subreport is the field that immediately follows the Account Number field in Index 1 of the Transaction Panel: Transaction Date.

Strategic Placement of the Skip To Code

Where you place the Skip To code is crucial. If you place it in the First Page Header, DataPerfect goes immediately to the first record whose primary sorting field has a value equal to *or greater than* the Report Variable assigned that Skip To code, and processes all records in that index *from that point forward*. If you place it in the Report Body, DataPerfect goes immediately to the first record whose primary sorting field is *equal to* the Report Variable, and processes that perfect match *repeatedly* (as opposed to processing all records in that index from that point forward). In the former case (placing the Skip To code in the First Page Header), if DataPerfect fails to find a perfect match, it settles for next record in the index (following the nonexistent perfect match) and starts processing the index from there. In the latter case (placing the Skip To code in the Report Body), if DataPerfect fails to find a perfect match, the report (or subreport) stops. You'll usually put the Skip To code in the First Page Header. As I outline our Itemized Ledger example here, I'll show how to successfully use it in both the First Page Header and the Report Body.

The Internal Logic of the Skip To Code

Whether you put the Skip To code in the First Page Header or the Report Body, DataPerfect processes it with the logic of a lookup type-to-search, jumping almost instantaneously to the desired record, though in the Report Body case it aborts the process if it doesn't see a perfect match. So a Skip To code in the First Page Header is a little more like a lookup than a Skip To code in the Report Body, as lookups in Browse mode don't abort just because they don't find a perfect match. But the way DataPerfect moves to the desired record is just like a lookup. This contrasts with the Skip If code, which acts more like a Search. Under the command of the Skip If code, DataPerfect examines each and every record in the index on its way to the desired record, like running the report with Search Conditions, or doing an **F2** Search in Browse mode. On the other hand, under the command of the Skip To code, DataPerfect never sees the records on its way to the desired one (like performing a lookup in Browse mode).

Combining the Skip To Code with the Stop If Code

Just as the Skip To code works in concert with a previously created Report Variable, checking to see if that Report Variable's assignment *exists in* the current record's primary sorting field, the Stop If code works in concert with a previously created Report Variable, checking to see if its assignment *is true or false*.

Let's put these two codes into practice with our Itemized Ledger report. Look at these two steps in our proposed report:

- A. Using the Skip To code, it goes directly to the first Account Panel record with the chosen Account Number.
- G. Using the Stop If code, it stops the entire report as soon as it's done processing the chosen Account Panel record.

Those are the two that concern the Account Panel. We want the report to jump to the correct Account Panel record and stop as soon as it's done processing that Account. To use a Skip To code here, we need to first create an Report Variable that holds the Account Number the user is interested in. Though the User Chooses code (**Ctrl-F7, 4**) is more effective here, I haven't talked about that yet. Let's use the Prompt For code for this (**Ctrl-F7, 6**):

```
-----FIRST PAGE HEADER-----
-----Prompt for Value of Report Variable 1 -----] Prompt for Acct Num
-----OTHER PAGE HEADER-----
--Empty--
```

Now we need to put in a Skip To code to get DataPerfect to skip to the Account Panel record with the Account Number held by Report Variable 1 in its Account Number field. We also want the report to abort if it doesn't find a perfect match on this Report Variable. That is, we don't want DataPerfect to just settle for the first Account Number it finds equal to or greater than the one the user stuffed into Report Variable 1.

To do this, we go down our check list. First, is the Skip To code using a Report Variable that holds a *value of* a field, not a *statement about* a field? Yes. It holds an Account Number, like

1009

not a *statement about* the Account Number field, like

P1F1=1009

Second, is the Skip To code using a Report Variable whose value is compatible with the primary sorting field of the active index? Yes, as long as the report's index (Initial Report Definition Screen, Item 3) is one that has the Account Number field as its first field. And third, is the Skip To code in the appropriate Report Form section? Let's discuss this last concern.

Deciding where to put the Skip To code in this case has a few options. Here we're attempting to make our report Skip To a perfect match on the Account Number furnished by the user. The report prompts the user for this number in the First Page Header. At first, you might consider putting the Skip To code in the Report Body because it aborts if a perfect match isn't found; whereas, if we put it in the First Page Header, DataPerfect will settle for the first Account Number its sees equal to *or greater than* the chosen one. But putting the Skip To code in the Report Body causes the report to run over and over again on that perfect match. So let's see how to work around this, as long as we're using the Prompt For code to stuff our Report Variable. Again, we could have used the User Chooses code instead of this pair of Prompt For and Skip To codes. I discuss this later. This would have been a much better option all around, but I want to use the Prompt For and Skip To codes here to expose some subtleties.

If we put the Skip To code in the First Page Header, just after the Prompt For code, then we should put a Stop If code in the Report Body to make sure the Report Body fails to run if the match isn't perfect. This is done by first evaluating the record to which the Skip To code directs DataPerfect. Since that record might be *greater than* the chosen Account Number, we need to prevent that. Our Prompt For code stuffs Report Variable 1, so we'll evaluate the current record in the Report Body with a Store Value code for Report Variable 2, and then follow that with a Stop If code (**Ctrl-F7, 9, 2**) on Report Variable 2:

-----FIRST PAGE HEADER-----	
-----Prompt for Value of Report Variable 1 -----	Prompt for Acct Num
-----Skip To Record at Report Variable 1 -----	Skip to Acct Num
-----OTHER PAGE HEADER-----	
--Empty--	
-----TWO-LEVEL REPORT HEADER-----	
--Empty--	
-----REPORT BODY-----	
-----Store Value in Report Variable 2 -----	Put P1F1=RV1 in RV2
-----Stop [Sub]Report if 0 Is in Report Variable 2 -----	Stop if P1F1 not RV1
[Rest of report goes here, including the subreport(s).]	
-----TWO-LEVEL FOOTER-----	
--Empty--	

The above Report Definition stops if the Account Number (P1F1) DataPerfect Skipped To isn't a perfect match on Report Variable 1.

A different approach would be to put the Skip To code in the Report Body. But, again, though this easily solves the imperfect match problem posed by the First Page Header approach, if we're not careful this will result the in the Report Body processing the same record over and over again. That's simply the way the Skip To code works in the Report Body. So we make sure the Report Body runs only on one record:

```

-----FIRST PAGE HEADER-----
-----Prompt for Value of Report Variable 1 -----] Prompt for Acct Num
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Skip To Record at Report Variable 1 -----] Skip to Acct Num

      [Rest of report goes here, including the subreport(s).]

-----Store Value in Report Variable 200 -----] Store 0 in RV200
-----Stop [Sub]Report if 0 Is in Report Variable 2 -----] Stop if RV200 is 0
-----TWO-LEVEL FOOTER-----
--Empty--

```

In the above Report Form, the very end of the Report Body has a pair of codes that stop the report before it goes on to process the Report Body again on the same perfect match. The first code in this pair stores 0 in Report Variable 200, and the second stops the report if it sees 0 in Report Variable 200 (that's the same as saying Report Variable 200 is false). As mentioned elsewhere in this book, I like to use Report Variable 200 for this purpose in *any* report, so whenever I see Report Variable 200 on a Edit Report Form screen, I know why it's there (to stop the report or subreport).

Also note that either of these two reports processes only one Account Panel record. The first report stops after the first one is processed because its Stop If code stops the report as soon as it sees an imperfect match on the chosen Account Number. The second report stops after the perfect match is processed once.

We just finished these two steps:

- A. Using the Skip To code, it goes directly to the first Account Panel record with the chosen Account Number.
- G. Using the Stop If code, it stops the entire report as soon as it's done processing the chosen Account Panel record.

How we handle the Transaction Panel subreport will differ in logic from the way we just handled the Account Panel main report. In this subreport we need the report to Skip To the first record *equal to or greater than* the chosen Start Date. So we do *not* want to limit this Skip To operation to a perfect match for the starting record there. This is handled in a much simpler fashion: just put the Skip To code in the subreport's First Page Header. That handles the Start Date. To get the subreport to stop as soon as it leaves the date range (that is, as soon as it goes past the End Date), we use the Store Value and Stop If codes, having the first code evaluate the current record to see if it's still equal to or less than the End Date, and having the second code stop the subreport if the first is false. Here's what we now have:

```

-----FIRST PAGE HEADER-----
-----Prompt for Value of Report Variable 1 -----
-----Prompt for Value of Report Variable 11 -----
-----Prompt for Value of Report Variable 12 -----
-----Skip To Record at Report Variable 1 -----
Prompt: Acct Num
Prompt: Start Date
Prompt: End Date
Skip to Acct Num

-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Store Value in Report Variable 2 -----
-----Stop [Sub]Report if 0 Is in Report Variable 2 -----
Put P1F1=RV1 in RV2
Stop if P1F1 not RV1

=====SUBREPORT LINK/PANEL: 1 2=====
-----FIRST PAGE HEADER-----
-----Skip To Record at Report Variable 11 -----
Skip to Start Date

ITEMIZED LEDGER

Date of Report: 
Date Range: - 
Account Number: 
Name: 

Ctrl-F7, 1, 1 (Date)
RV11 and RV12
P2F1
P2F2 and P2F3

Date      Description      Amount
=====
-----OTHER PAGE HEADER-----

ITEMIZED LEDGER (continued)      Page 
Date of Report: 
Date Range: - 
Account Number: 
Name: 

Ctrl-F7, 1, 3 (Pg #)
Ctrl-F7, 1, 1 (Date)
RV11 and RV12
P2F1
P2F2 and P2F3

Date      Description      Amount
=====
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Store Value in Report Variable 13 -----
-----Stop [Sub]Report if 0 Is in Report Variable 13 -----
P2F4<=RV12 in RV13
Stop if P2F4 > RV12
P2F4, P2F5, and P2F6

-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
=====
Total: 
P2F6 Final Ftr Total

-----END OF SUBREPORT-----

-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

Let's talk about what I just did. The index in the subreport must have the Date field (P2F4) as its primary sorting field. Otherwise, the Skip To in its First Page Header won't work properly. So, assuming the link used to create this subreport has Account Number as the only field on its field list, the index assigned this link must begin with Account Number, followed by the Date field in the Transaction Panel. That will make the Transaction Panel's Date field the primary sorting field in the subreport.

Given the above caveat, the Skip To code in the subreport's First Page Header will make the subreport skip to the first record it finds in the date range (equal to or greater than the Start Date), and the two codes in the Report Body will stop the subreport as soon as it sees a record outside that range (greater than the End Date). Note that though we place the Skip To code in the First Page Header, we place the Stop If code in the Report Body. Again, this is because the Stop If code must

continuously receive an updated Report Variable 13, which is the job of the Store Value code just above it.

User Chooses Next Record By LookUp

We can make our report much easier for the user to run if we replace the Prompt For code with a User Chooses code (I'll use *User Chooses code* to refer to DataPerfect's *User Chooses Next Record By Lookup code*). The Prompt For code approach is only going to work if the user knows exactly what Account Panel record they're looking for. If the Account Number they feed that prompt is off at all, the report fails. It may print the wrong report, or no report at all. On the other hand, the User Chooses code presents the user with a lookup of records in the Account Panel, letting them browse and choose the one they want. Using the User Chooses code actually replaces two codes here: the Prompt For code and the Skip To code. That is, the User Chooses code presents the user with a lookup of records in the current panel. The User Chooses code positions the report on the record the user chooses, so no Skip To code is needed here.

Here's how we'd do this with the current report project:


```

-----FIRST PAGE HEADER-----
-----Prompt for Value of Report Variable 11 -----
-----Prompt for Value of Report Variable 12 -----
Prompt: Start Date
Prompt: End Date

-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----User Chooses Next Record By LookUp-----
User Chooses Acct

=====SUBREPORT LINK/PANEL: 1 2=====
-----FIRST PAGE HEADER-----
-----Skip To Record at Report Variable 11 -----
Skip to Start Date

ITEMIZED LEDGER

Date of Report: 
Date Range: 
Account Number: 
Name: 
Ctrl-F7, 1, 1 (Date)
RV11 and RV12
P2F1
P2F2 and P2F3

Date      Description      Amount
=====
-----OTHER PAGE HEADER-----

ITEMIZED LEDGER (continued)      Page 
Ctrl-F7, 1, 3 (Pg #)

Date of Report: 
Date Range: 
Account Number: 
Name: 
Ctrl-F7, 1, 1 (Date)
RV11 and RV12
P2F1
P2F2 and P2F3

Date      Description      Amount
=====
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
-----Store Value in Report Variable 13 -----
-----Stop [Sub]Report if 0 Is in Report Variable 13 -----
P2F4<=RV12 in RV13
Stop if P2F4 > RV12
P2F4, P2F5, and P2F6

-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
=====
Total: 
P2F6 Final Ftr Total

=====END OF SUBREPORT=====
-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----
Usual Stop routine

-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

In the above report, note how much simpler the main report's First Page Header and beginning of the Report Body is (just before the subreport). Also note the two codes that follow the subreport, in the main report's Report Body. There, Report Variable 200 is set to 0 and then the report stops, keeping the user from seeing the lookup again. If you want this report to allow the user to print as many Itemized Ledgers as he wants, for as many Accounts as he chooses from the lookup, you'd make the following changes:

```

-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
=====
-----Page Eject-----Total: =====
-----END OF SUBREPORT-----
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

Ctrl-F7, 3
Stop routine removed

The Page Eject makes sure each Account's Itemized Ledger starts on a fresh page, and removing the Stop routine allows the user to see the lookup again and again, allowing him to print as many Itemized Ledgers as needed in one run of the report. This process ends when the user hits **F7**.

When using the User Chooses code, you'll either put it in the First Page Header or the Report Body. If you put it in the First Page Header, DataPerfect will position the report on the record the user selects. Any fields selected in the First Page Header will get their data from that particular record. The Report Body will process all records in the report's index from the selected record to the end of the index.

On the other hand, if you put the User Chooses code in the *Report Body*, DataPerfect will offer the user a new lookup after each selection is processed by the Report Body, stopping only after the user selects the last record in the lookup (DataPerfect will process that record and then quit the report or subreport that has the User Chooses code), or the user hits **F7**. Let's go over each possible placement of the User Chooses code.

Placing the User Chooses code in the First Page Header

I see no way to make this attractive without combining it with other Iteration Control codes. Think about this. When you put the User Chooses code in the First Page Header, you're telling the report to skip to the record the user chooses, and then go from there until it reaches the end of the index. This is a cross between putting the Skip To code in the First Page Header and in the Report Body. Like a Skip To code in the First Page Header, it works only once, repositioning the report to start at some designated place in the index. Like a Skip To code in the Report Body, it works on a perfect match. So it's really like the Skip To code, which is also almost always going to need other Iteration Control options to be practical. Usually we're talking about complementing either code with the Stop If code.

That said, you're still not going to put the User Chooses code in the First Page Header all that much. Here's an example where it might come in handy, however: Your client is an attorney using an application you wrote for him. He'd like to occasionally print a list of all clients living in a particular city. You could define a Two Level Report built on the Client Panel that sorts on the City field, but that produces a report covering *all* cities in the Client Panel. Or you could define a report that prompts for the City, using a Prompt For code in the First Page Header, but that will require the user know exactly what City he's interested in, and exactly how to spell it. If you'd like the user to be able to peruse the Cities in the Client Panel before

generating the list, you can put the User Chooses code in the First Page Header. Just make sure the report's index sorts by City.

Configure the report to stop when it comes to the last occurrence of the selected City. One way to do this is to place the value of the selected City field in a Report Variable immediately after the User Chooses code in the First Page Header. If the City field is P1F5, then, in the First Page Header, store the value P1F5 in Report Variable 1. Now go to the Report Body and store

```
P1F5=rv1
```

in Report Variable 2, which makes Report Variable 2 true only if the current record has the user-selected value in the City field. Next, use a Stop If code to stop the report if Report Variable 2 is false. Now select the fields to print. This will stop the report as soon as it sees a record with a City field value other than the selected one.

Here's what that Edit Report Form screen looks like:

[Report Index: City sort]		
-----FIRST PAGE HEADER-----		
-----User Chooses Next Record By LookUp-----		} User chooses by City Put P1F5 in RV1
-----Store Value in Report Variable 1 -----		
-----OTHER PAGE HEADER-----		
Clients living in	Page	} Select P1F5 Ctrl-F7, 1, 3 (Pg #) Ctrl-F7, 3
-----Include Header Before Data-----		
-----TWO-LEVEL REPORT HEADER-----		
--Empty--		
REPORT BODY		
-----Store Value in Report Variable 2 -----		} Put P1F5=RV1 in RV2 Stop when next City Select Name fields
-----Stop [Sub]Report if 0 Is in Report Variable 2 -----		
-----TWO-LEVEL FOOTER-----		
--Empty--		

The only code here that might not be obvious if you haven't used it before is the Include Header code in the Other Page Header section. When placed after the Other Page Header contents, it makes that Other Page Header show up on the first page as well as all the other pages. Since I wanted the same header on every page, I did it this way.

When running this report, as long as the user selects the first record in the lookup that has desired value in the City field, all and only clients living in that city will be processed by this report. It allows the user to peruse values in the City field before deciding on the city desired. This is much more lenient than a Prompt For code. He just needs to make sure that the record selected from the lookup is the first record with that City field value. If so, all and only records with that value will be printed. Note that this avoids forcing DataPerfect to search the entire database of clients, from beginning to end, before ending the report. The report simply starts with the first desired record and ends with the last.

Placing the User Chooses code in the Report Body

Say your attorney client wants a report that generates a list of selected clients, including their addresses and phone numbers. He wants to be able to pick each client from a lookup. He might want to use this list later to call these people, or send them a note.

This time the User Chooses code should be in the Report Body. It's pretty straightforward, and would look like this:

```
-----FIRST PAGE HEADER-----
                  Clients To Contact Later
-----OTHER PAGE HEADER-----
                  --Empty--
-----TWO-LEVEL REPORT HEADER-----
                  --Empty--
-----REPORT BODY-----
-----User Chooses Next Record By LookUp-----
                  .
                  .
                  .
Home: . Work: . Fax: .
-----TWO-LEVEL FOOTER-----
                  --Empty--
```

The above report offers the user a new lookup after each record is processed. It stops offering lookups after the user hits **F7** or selects the last record in the lookup. This contrasts with the previous report, where the user is offered only one lookup because the User Chooses code is in the First Page Header.

Single Record Report From Lookup off a Menu

There's one more place an option similar to the User Chooses code shows up, and it's not on the Iteration Control menu. It shows up when attaching a report to a menu item (assuming your application has menus). When you choose the Run Report option on your Define Menu screen and select a report from the Report List, you're then presented with an Initial Report Definition Screen that has a few options not found on the Initial Report Definition Screen when creating or editing a Report Definition on the Report List. They're at the bottom of the Initial Report Definition Screen, following option 9:

```
A - Report Mode - Normal/Lookup      Normal Report Mode
B - Password:
C - User Set-Up                      No
```

Note option A. It toggles between *Normal Report Mode* (the default) and *Single Record Report From Lookup*. If the Report Definition is set for *Single Record Report From Lookup*, the report will begin by offering the user a lookup. Each time the user selects a record from the lookup, DataPerfect will run the report *in its entirety*, processing only the selected record, and then offer the lookup again.

Don't confuse the behavior of Single Record Report From Lookup with that of putting the User Chooses code in the Report Body. They're similar, but different. A report set for Single Record Report From Lookup processes the *entire* report on each record chosen by the user from the lookup; whereas, a report with the User

Chooses code in the Report Body processes *only the Report Body* on each record. So the Single Record Report From Lookup report is going to produce a series of reports (one per record chosen from the lookup), each with a First Page Header and Final Footer. On the other hand, a report with the User Chooses code in the Report Body will produce a *single* report that has one First Page Header and one Final Footer, but with a Report Body that will be run once per record selected from the lookup.

How Report Lookups Display

How DataPerfect displays a report lookup when it sees the User Chooses code depends on the version of DataPerfect you're using. Report lookups began with DataPerfect 2.3, but changed a few times after its initial release.

The First Two DataPerfect 2.3 Releases

The 02/01/93 and 09/01/93 releases of DataPerfect 2.3 worked identically in terms of record lookup display. DataPerfect displayed all and only the fields of the index chosen in the Report Definition, leaving out those fields in the beginning of the index that are involved in the link's field list if this is a subreport. So you had only one way to control the way such a lookup displays: the index you assigned to that part of the report.

The First DataPerfect 2.3b Release

Starting with the 08/19/94 version (2.3b), DataPerfect offered more control over how a report lookup displays. If the first field in the report index that follows the linking fields (if any) had a Lookup Definition on it (in its panel), the report lookup displayed according to whatever it found in that Lookup Definition. If that Lookup Definition had both a field list and an index, the report lookup used both. If that Lookup Definition had only a field list, the report lookup used that field list, but used the active index in that part of the Report Definition. If such a field had no Lookup Definition, the report lookup displayed the way it did with the 1993 versions of DataPerfect.

Later Versions of DataPerfect

Though the initial release of DataPerfect 2.3b offered more control over report lookups, by allowing us to control them with our Lookup Definitions in the panel itself, a serious problem arose. Suppose you wanted your report to display a lookup based on an Exception List Index? Well, if the first field in that Exception List Index had a Lookup Definition on it (or, if this is a subreport, the first field following the linking fields had a Lookup Definition on it), and that Lookup Definition assigned an index to the lookup, the report would use *that* index and *not* the Exception List Index found in the Report Definition. This was terrible. Imagine defining a report that's supposed to expose only records on an Exception List Index to the user, for possible deletion by the report, and it ends up exposing all records in the panel for this deletion routine!

The January 1995 version of 2.3b resolves this. Starting with that version, the report lookup uses the lookup *field list* in the Lookup Definition of the first field that

follows the linking fields (if that field has a Lookup Definition on it), and ignores the *index* assigned to that field's lookup. That is, the index found in the Report Definition prevails, no matter what. If the report sees a Lookup Definition on the index's first field that follows the linking fields, it uses the field list it finds there, otherwise it uses only the fields it finds in the report index. Much, much better.

This means you might need to assign a lookup definition (at least a lookup field list, anyway) to a hidden field to make this work. I discuss this in *Reasons for Assigning a Lookup to a Hidden Field* in my **Lookups** chapter.

How to Work Around the August 1994 Version

If you have the August 1994 release of 2.3b, upgrade to a later release soon. Until then, use these rules in designing your database applications (it makes use of the Smart Lookups algorithm):

- Make sure any Exception List Index has a counterpart index that matches it perfectly, except that its counterpart lacks an Exception List Index.
- Make sure each Exception List Index counterpart is a lower numbered index than any of its sisters that have Exception Lists assigned.

So, if you have an Exception List Index that sorts by Last Name, First Name, and Account Number, then make sure you also have a lower numbered index that sorts by Last Name, First Name, and Account Number, and lacks an Exception List.

Now the final rule:

- If you put a User Chooses code in a report, and the index that lookup will use is an Exception List Index, then make sure that a lookup defined for the first field following the linking fields in that Exception List Index has *no* index assigned to its Lookup Definition (unless, of course, you actually *want* the Exception List Index assigned to the field's Lookup Definition).

This final rule takes the Smart Lookups algorithm into consideration. If the Exception List Index's first field following the linking fields lacks an index in its Lookup Definition, the Smart Lookups algorithm will temporarily assign it the lowest numbered index that activates that field during a Browse mode lookup. If every Exception List Index is a higher numbered index than its counterpart that has no Exception List, you're safe because you're assuring the Smart Lookups algorithm will always choose an index without an Exception List. This way, Browse mode lookups will grab the correct index. And because no index is assigned in the Lookup Definition for that field, the report's User Chooses code will use the index assigned it in the Report Definition. Again, this precaution is unnecessary with versions of DataPerfect later than 1994.

Repeat If

The Repeat If code makes the report run the current record again if the selected Report Variable is true or not 0. This can be tricky, however. Consider the following report. You might think it will print two copies per record by storing 2 in Report Variable 1 in the First Page Header, then decreasing it by 1 in the Report Body after each record is printed:

```
-----FIRST PAGE HEADER-----  
-----Store Value in Report Variable 1 -----  
-----OTHER PAGE HEADER-----  
-----Empty-----  
-----TWO-LEVEL REPORT HEADER-----  
-----Empty-----  
-----REPORT BODY-----  
  
[printed stuff goes here]  
  
-----Store Value in Report Variable 1 -----  
-----Repeat Record if True Is In Report Variable 1 -----  
-----TWO-LEVEL FOOTER-----  
-----Empty-----
```

Put 2 in RV1

Put RV1-1 in RV1

But this doesn't work. It prints two copies of the first record, and then *one* of each of the following records in the index. Why? Because Report Variable 1 is 0 after the first record is printed twice, never to increase again. That is, after the first record is printed twice, the Repeat If code is continuously encountering a false Report Variable 1 (or an Report Variable 1 with a value of 0), so it never causes a repeat of a record again. It causes a repeat only if Report Variable 1 is true (or nonzero).

To get a particular number of copies of each record, you can do something like this:

```
-----FIRST PAGE HEADER-----  
-----Prompt for Value of Report Variable 1 -----  
-----Stop [Sub]Report if 0 Is in Report Variable 1 -----  
-----Store Value in Report Variable 2 -----  
-----OTHER PAGE HEADER-----  
-----Empty-----  
-----TWO-LEVEL REPORT HEADER-----  
-----Empty-----  
-----REPORT BODY-----  
  
[printed stuff goes here]  
  
-----Store Value in Report Variable 2 -----  
-----Store Value in Report Variable 3 -----  
-----Repeat Record if True Is In Report Variable 3 -----  
-----TWO-LEVEL FOOTER-----  
-----Empty-----
```

Copies per record
Stop if entered 0
Put RV1 in RV2

Put RV2+1 in RV2
Put RV2//RV1 in RV3
Repeat until RV3=0

The above report prompts the user for the number of copies he wants to print for each record. It stores that number in Report Variable 1. It then abruptly stops if the user enters 0 for that prompt. If the report didn't stop (the user entered a number higher than 0), Report Variable 1 is stored in Report Variable 2. After each record is printed, Report Variable 2 increases by 1, and DataPerfect stores, in Report Variable 3, the remainder of dividing Report Variable 2 by Report Variable 1. That's what the modulo operator does. Each record repeats until Report Variable 3 (the modulo RV2//RV1) is 0.

For example, if the user enters 3 at the prompt, Report Variable 2 is set to 3. After the first record is printed, Report Variable 2 increases to 4. Then 4//3 is evaluated. It isn't 0 (it's 1), so the record is printed again. After the second printing, Report Variable 2 increases to 5. Then 5//3 is evaluated. It's still not 0 (it's 2), so the record is printed again. So far the report has printed that same record three times. Now Report Variable 2 increases to 6. Then 6//3 is evaluated. Now it's 0, so the report won't repeat that record. Three printings is all we get for that record.

Now the report goes to the next record. At that time, Report Variable 2 is 6. The report prints that record and increase Report Variable 2 to 7. Then 7//3 is evaluated. It isn't 0 (it's 1), so the record will be printed again. After the second printing, Report Variable 2 increases to 8. Since 8//3 isn't 0 (it's 2), it prints a third time. After the third printing, Report Variable 2 increases to 9. Since 9//3 is 0, the report no longer repeats that record. It now goes to the next record and continues with the same logic.

A Note about DataPerfect's Notion of Truth

Just what DataPerfect considers *true* and what it doesn't might be confusing. This confusion is compounded when you notice that what DataPerfect calls a code in the Iteration Control menu doesn't always seem to match well with what it inserts in the Edit Report Form screen. The three Iteration Control codes of concern here look like this in the Iteration Control menu:

```
Skip Record if RV is False
Stop [Sub]Report if RV is False
Repeat Record if RV is True (not 0)
```

But, assuming we assign Report Variable 1 to each of the above, here's how DataPerfect inserts them, respectively, into the Edit Report Form screen when selected:

```
Skip Record if 0 (False) Is in Report Variable 1
Stop [Sub]Report if 0 Is in Report Variable 1
Repeat Record if True Is In Report Variable 1
```

Each of the above Iteration Control codes is examining the truth or falsehood of a given Report Variable. The model I proposed before was that each of these types of Report Variables examines the truth or falsehood of a Report Variable's *statement* with respect to the current record, whereas the Skip To code examines the existence of a Report Variable's *value* in the current record. But the above three Iteration Control codes, as they're expressed in the Edit Report Form screen, are talking about *true* or 0 being in an Report Variable.

So what's going on here? Just this. What a Report Variable stores is considered *true* under any of these conditions:

- It's a statement, and it's true.
- It's a character string, and it isn't blank.
- It's a numerical value, and it isn't 0.

Put another way, what a Report Variable stores is considered *false* under any of these conditions:

- It's a statement, and it's false.
- It's a character string, and it's blank.
- It's a numerical value, and it's 0.

This makes all the following Report Variables equivalent, at least in terms of Iteration Control codes that might be evaluating them:

```
(1) P1F1 > P1F2
(2) if P1F1 > P1F2 then 1 else 0 endif
(3) if P1F1 > P1F2 then "A" else 0 endif
(4) if P1F1 > P1F2 then "A" else "" endif
(5) if P1F1 <= P1F2 then 0 else 1 endif
(6) if P1F1 <= P1F2 then "" else 1 endif
(7) if P1F1 <= P1F2 then "" else "A" endif
```

(1) is a *statement*, and says P1F1 is greater than P1F2. If this is tied to, say, the Skip If code, then if the current record is such that P1F1 is *not* greater than P1F2, that Report Body will skip it. (2), though formed as a statement, is really a value. That value is either 0 or 1, depending on the current record. Again, if we're talking about the Skip If code, if that value is 0, the Report Body skips the current record. If it's 1, it processes that record. Statement (3) is similar to (2), only it yields the character A (nonblank character string) or the numerical value 0, processing the record if the Report Variable yields A. (4) is also similar to (2) and (3), only it yields character A or the blank character string. (5), (6), and (7) share the reverse logic of (2), (3), and (4), respectively.

The point of the above is that if, say, Report Variable 1 is being used by the Skip If code to evaluate records in Panel 1, the same records will be printed no matter which of the seven formulas above were used for Report Variable 1. In terms of the Skip If code, formulas (2) through (7) are each equivalent to formula (1).

Here's a list of examples to clarify this rather confusing topic:

Report Variables and Truth Values		
<i>Report Variable Contents</i>	<i>Truth Value</i>	<i>Reasoning</i>
1	True	Numerical value, not zero.
0	False	Numerical value, zero.
-1	True	Numerical value, not zero.
"1"	True	String, not blank. It's surrounded by quote marks, so it's not a numerical value.
"0"	True	String, not blank. See above.
"A"	True	String, not blank. Surrounded by quote marks.
"1=2"	True	String, not blank. See above.
"This is False"	True	String, not blank. See above.
""	False	String, blank.
P1F1=P1F2	Depends on fields	Statement.
if P1F1=P1F2 then 1 else 0 endif	Depends on fields	Numerical value. Though it's a statement, we're evaluating what it yields.
1=1	True	Statement, true. An identity of two numerical values.
0=0	True	Statement, true. See above.
1=2	False	Statement, false. See above.
"1"="1"	True	Statement, true. An identity of two strings.
if 1=1 then "" else 0 endif	False	String, blank. Though it's a statement, we're evaluating what it yields.
if 1=2 then "" else 0 endif	False	Numerical value, zero. See above.
if 1=1 then 0 else "" endif	False	Numerical value, zero. See above.
if 1=2 then 0 else "" endif	False	String, blank. See above.
if 1=1 then 1 else "" endif	True	Numerical value, not zero. See above.
if 1=2 then 1 else "" endif	False	String, blank. See above.
if 1=1 then "0" else "" endif	True	String, not blank. See above.
if 1=2 then "0" else "" endif	False	String, blank. See above.
if 1=1 then 1 else 0 endif	True	Numerical value, not zero. See above.
if 1=2 then 1 else 0 endif	False	Numerical value, zero. See above.

The most typical two ways of formulating a Report Variable to be used with a Skip If code, Stop If code, or a Repeat If code (again, these are the ones that use the notion of truth during record processing) are to either create a statement that is true for all and only the records desired, or a statement that yields the numerical value 1 for all and only the records desired, otherwise 0. So, for instance, if we want the Skip If code to cause the report to process all and only records where the value in P1F1 equals the value in P1F2, then we would usually choose

```
P1F1=P1F2
```

or

```
if P1F1=P1F2 then 1 else 0 endif
```

The first is a statement that will either be true or false. The second is a value that will be either 1 or 0. Put another way, for purposes of the three Iteration Control codes of concern here, the first Report Variable holds either truth or falsehood, and the second either 1 or 0. So, as you can now see, when I earlier said that

```
each of these three [Iteration Control  
codes] uses a Report Variable that holds a  
statement, and consequently looks to see if  
that statement is true of the current  
record
```

I wasn't telling the whole truth. To be more complete, I might have said that such an Iteration Control code looks to see if the Report Variable holds a true statement, a nonblank character string, or a nonzero numerical value. But you get what I mean.

Iteration Control Examples

Limiting a Report to One Record

[For an example of this, load UD.STR.
Find *Printing just one record* on the Report List.]

To limit a report to one record, use what I call a stop routine at the end of the Report Body, setting Report Variable 200 to 0, immediately following it with a Stop if Report Variable 200 is False code. Again, I always choose Report Variable 200 for this only so I can easily remember why it's there.

```

-----FIRST PAGE HEADER-----
-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----

[Selected fields, etc., here]

-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----
-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----

```

RV200: 0

Limiting a Report to a Particular Number of Records

[For an example of this, load UD.STR.
Find *Printing a particular number of records* on the Report List.]

To limit a report to a particular number of records, prompt the user for the number of records (Report Variable 1). Stop the report if the user enters 0. At the very end of the Report Body, decrement Report Variable 1 by one with this formula:

rv1-1

The report prints records until Report Variable 1 is 0.

```

-----FIRST PAGE HEADER-----
-----Prompt for Value of Report Variable 1 -----
-----Stop [Sub]Report if 0 Is in Report Variable 1 -----
-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----

[Selected fields, etc., here]

-----Store Value in Report Variable 1 -----
-----Stop [Sub]Report if 0 Is in Report Variable 1 -----
-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----

```

RV1: Num of records

RV1: RV1-1

A Report That Prints a Particular Number of Iterations per Record

[For an example of this, load UD.STR.
Find *Printing a particular number of iterations per record* on the Report List.]

As discussed in the *Repeat If* section previously, prompt the user for the number of copies per record (Report Variable 1). Stop the report if the user enters 0. Then set Report Variable 2 to Report Variable 1. After each record is printed, increase Report Variable 2 by 1, and set Report Variable 3 to the remainder of Report Variable 2 divided by Report Variable 1, using modulo (RV2//RV1). Each record will now repeat until Report Variable 3 is 0.

```

-----FIRST PAGE HEADER-----
-----Prompt for Value of Report Variable 1 -----
-----Stop [Sub]Report if 0 Is in Report Variable 1 -----
-----Store Value in Report Variable 2 -----
RV1: Num of copies
RV2: RV1

-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----

[Selected fields, etc., here]

-----Store Value in Report Variable 2 -----
-----Store Value in Report Variable 3 -----
-----Repeat Record if True Is In Report Variable 3 -----
RV2: RV2+1
RV3: RV2//RV1

-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----

```

A Date-Range Report

[For an example of this, load UD.STR.

Find *Date Range Report (Master Panel series)* on the Report List.]

Make the active index in that part of the report, which can be a subreport, one that sorts forward on the relevant date field. In that part of the report, in its First Page Header, prompt the user for the Start Date (Report Variable 100) and the End Date (Report Variable 101). Then, in the same First Page Header, insert a Skip to Record at Report Variable 100 code. As the first line of the Report Body of that part of the report, set Report Variable 103 to

```
P1F1<=rv101
```

where P1F1 is the relevant date field. Immediately after the above, but still in the same Report Body, insert a Stop Report if Report Variable 103 is False code. All other Report Body fields, text, etc., follow this code.

```

-----FIRST PAGE HEADER-----
-----Prompt for Value of Report Variable 100 -----
-----Prompt for Value of Report Variable 101 -----
-----Skip To Record at Report Variable 100 -----
RV100: Start Date
RV101: End Date

-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----
-----Store Value in Report Variable 103 -----
-----Stop [Sub]Report if 0 Is in Report Variable 103 -----
P1F1<=RV101
[P1F1 is date field]

[Selected fields, etc., here]

-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----

```

A Report That Prints Monthly Statements for All and Only Accounts with a Positive Balance

First off, it's best to do this with an Exception List Index that points only to Account Panel records with a Balance field value over 0. If you elect to do that, you'd create a hidden G9 field that updates to

```
if P1F1=0 then 1 else 0 endif
```

on any change, where P1F1 is the Balance field. Your Exception List Index would have that field alone on its Exception List. Read up on Exception List Indexes in *Exception Lists* in my **Indexes** chapter.

That said, you can do this with Iteration Control options instead of an Exception List Index. Run that part of the report, which may be a subreport, with an index that sorts forward by Balance, where the Balance field is the primary sorting field at that point in the report. In the First Page Header of that part of the report, set Report Variable 1 to 0.01, followed by a Skip To Report Variable 1 code. This will make the report start with the first record it sees with a Balance field value of one penny or greater. You don't need a Stop If code for this report.

```
-----FIRST PAGE HEADER-----
-----Store Value in Report Variable 1 -----
-----Skip To Record at Report Variable 1 -----
-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----
[Selected fields, etc., here]
-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----
```

RV1: 0.01

Getting a Report to Continue after the Last Record in the Lookup Is Selected

[For an example of this, load UD.STR.
Find *Report Lookup that doesn't end when choosing last record* on the Report List.]

The User Chooses code stops offering the user a lookup after they select the last record in the lookup. It sees that as the last record in the index, so its job is done. This might not be what you want, since you might want a particular report to allow the user to go up and down the various records in the lookup, selecting them without worrying that selecting the last one will end the lookup cycle. If you want the lookup to keep displaying, even after the user selects the last record, to end only when they hit **F7**, **F1**, or **ESC**, you can usually accomplish this by setting a Report Variable to the value of the primary sorting field, right after the lookup, and then have the report go to that Report Variable with a Skip To code.

```
-----FIRST PAGE HEADER-----
-----OTHER PAGE HEADER-----
-----TWO-LEVEL REPORT HEADER-----
-----REPORT BODY-----
-----User Chooses Next Record By LookUp-----
[Selected fields, etc., here]
-----Store Value in Report Variable 1 -----
-----Skip To Record at Report Variable 1 -----
-----TWO-LEVEL FOOTER-----
-----PAGE FOOTER-----
-----FINAL FOOTER-----
```

RV1: primary sorting field

Here Report Variable 1 takes the value of the primary sorting field of the user-selected record. If, say, the lookup sorted Client Panel records by Last Name, Report Variable 1 would end up holding the Last Name of the Client the user just

selected in the lookup. The report will then Skip To that Last Name in the next lookup. Conveniently, if there's no more than one Client with this Last Name, this actually results in the highlight bar landing on the *next* record in the lookup display. In most report definitions, this will be enough to keep the report from stopping after the last record in the index is selected.

This technique has a slight side effect. Consider the Last Name lookup case above. If there's more than one record with the Last Name the user just chose, the highlight bar will Skip To the first occurrence of that Last Name in the lookup, not the *next* one. So if the user chose Sally Adams, but there's more than one Adams in the lookup display, the highlight bar will return to the first Adams in the display, not the one that follows Sally Adams. I consider this a minor price to pay in order to keep the lookup displaying after the last record is selected.

Troubleshooting Iteration Control

Problem

Your report seems to be processing the same record repeatedly, never dealing with other records in the index.

Solution

You probably put a Skip To code in the Report Body instead of the First Page Header. Read *Strategic Placement of the Skip To Code* in the *Skip To and Stop If* section above.

Problem

Your Skip To code doesn't work. It just doesn't ever go the right record, whether you place it in the First Page Header or the Report Body.

Solution

Either the record doesn't exist in the current index, or the Report Variable being used for the Skip To code is incompatible with the current index.

Problem

Your User Chooses code keeps offering the user a lookup with negative numbers in the first column. You know that's the field you're sorting on, and need it to sort that way. You find no way to control the lookup field list for that User Chooses code.

Solution

Because that negative number field is hidden, you never thought to assign it a Lookup Definition. Give it a Lookup Definition that has at least a lookup field list. See *How Report Lookups Display* in *User Chooses Next Record By LookUp* above.

This is for both beginners and the experienced. Exporting and importing becomes useful very early in the life of an application.

Reasons for Exporting or Importing Data

There will definitely come a time when you'll need to export data out of a DataPerfect application. Exporting data from an application involves sending data from that application to a disk file, placing it in a format that can later be imported by that or some other application. Importing data into an application is similar to retrieving a document into a word processor. Data comes into the application and is then converted into a format it can read. DataPerfect can't read data in a file of exported data—even data exported by DataPerfect itself. For DataPerfect to read that data, it must be *imported* into a DataPerfect application.

So the reasons for *importing* data into a DataPerfect database are pretty obvious. One of the most common reasons is that the data might have originated from an application written with a different database management system. That system reads and writes data files DataPerfect can't read, so the data was exported to a file that, though DataPerfect still can't read it, DataPerfect can at least *import* it into a format it *can* read.

But why *export* data from a DataPerfect application? Why not leave it in the database for browsing and reporting? Here are the most common reasons:

Passing Data to Another Application

This is usually the first reason people usually think of for exporting data from a DataPerfect application. You want to import that data into another application, which may or may not be a DataPerfect application.

Merging Data with a Word Processing Document

Here you want to export some or all of the application's data to a disk file that can be used as a secondary merge file by your favorite word processor.

Altering an Application's Structure

There are many times you'll need to export data from a DataPerfect application in order to change the structure of the database (e.g., add or reformat a field). Most of the time, this will require removing data from at least one panel in order to make these changes. Before deleting all data in a panel, of course, you'll want to export it to a safe place for later import into the newer application.

Maintaining an Application

By this, I mean something different from altering an application's structure. This refers to running maintenance routines to optimize its data files, and to clean potential corruption from a database. I discuss this in *The Big Clean: Cleaning the Entire Application* in my **Application Maintenance Issues** chapter.

WordPerfect Merge Files

DataPerfect has always supported exporting to and importing from a WordPerfect Merge file. It currently supports WordPerfect Merge files in WordPerfect 4.2, 5.x and 6.x format. Before DataPerfect 2.3, this was the primary way users exported data from of a DataPerfect application, when that data was to be imported into the same or other DataPerfect application later. With DataPerfect 2.3 came the Transaction Log, which does all this much faster and more conveniently. The Built-In Short Reports WordPerfect Merge export, however, is more useful than the Transaction Log in circumstances I outline later. The Built-In Short Reports WordPerfect Merge report exports data from a single panel to a disk file that's in WordPerfect Merge format.

The Setup

To access the Built-In Short Reports WordPerfect Merge report facility, you must be in Browse mode in the panel whose data you want to export. **Shift-F7** to call the Report List. Highlight *Built-In Short Reports* and hit **Enter**. Here's the screen that confronts you:

BUILT-IN REPORT/EXPORT					
Destination:		Create Disk File			
1 - Printer On/Off					
2 - Disk File On/Off					
Filename: SCRATCH.REP					
3 - Index Number		1			
4 - Search Conditions		No Search			
5 - Sort Direction		Forward			
6 - Disk File Mode WP/DOS		WordPerfect			
7 - Print Margins		Top	Bottom	Left	Text Lines
		6	0	0	54
8 - Report/Export Format: WordPerfect Merge (Can be imported)					
Fields to be Included:		All Real Fields (Including Hidden)			
9 - Lookup Fields					
A - All Display Fields					
B - All Real Fields					
Selection: (Press Shift-F7 to begin the report) 0					

Use **2** to give the export file a name. Make sure the index seen by **3** has no Exception List on it, and sorts the way you want (if sort order is important in this case, as when using the export file as a secondary merge file from within WordPerfect). Use **4** if you want to limit the export to a subset of records that can be defined by a formula, template, or range. **5** allows you to further control the sort order

if that's important. **6** should be set for *WordPerfect*. **7** is irrelevant. **8** should be set for WordPerfect Merge. And last, the export should be set for *All Real Fields* if this is to be used to export all data out of that panel's data file.

The Nature of the Export File, Including Some Caveats

Text Editors and Carriage Returns

Running this report (hitting **Shift-F7** again) creates a file that can later be imported into the same or similar panel, or can be used as a secondary merge document from within WordPerfect. The data from each field in the resulting file will terminate with an ASCII 18 (↑) and a linefeed. Each record will terminate with an ASCII 5 (♣) and a linefeed. The file will be in pure WordPerfect 4.2 format, so each line (a single field's data) will terminate with a linefeed but *no carriage return*.

This latter point can be deceiving. If after creating such a WordPerfect Merge file, you load it in a text editor, you'll see each field's data on a single line terminating with a '↑', and each record terminating with a '♣' (some editors, like WordPerfect Corporation's Editor display these as ^R and ^E, respectively). Whether a file has linefeeds *with* carriage returns or linefeeds *without* carriage returns, it looks the same in a text editor. But most text editors, will, on saving the file, insert a carriage return before each linefeed that isn't preceded by a carriage return already. WordPerfect Corporation's Editor even does this as you move the cursor up and down the screen. If field data terminates with carriage returns as well as linefeeds, DataPerfect will not import that file properly as a WordPerfect Merge file.

So be careful if you inspect or edit the resulting WordPerfect Merge file export in a text editor. If that text editor supports loading files in *Binary* mode, choose that instead of Text mode. Binary mode doesn't insert a carriage return, even when you hit the **Enter** key, unless you consciously insert it as an ASCII character (ASCII 13) or as a control character (^M). If your text editor of choice is WordPerfect Corporation's Editor, load the file with the /B switch, like this:

```
ed.exe mydata.exp /b
```

Alternatively, on loading the same file in what WordPerfect Corporation's Editor calls DOS mode (what I'm calling Text mode), don't move the cursor until you hit **Ctrl-F5, 2**. That puts the editor in Binary mode. With this editor you should see *BIN* on the lower left corner of the screen if you're in Binary mode. Now you can safely edit that file.

Edit Order

The fields you find in a Built-In Short Reports WordPerfect Merge export file will be ordered according to the Edit Order that existed during the export of that panel's data. As you find later, importing a WordPerfect Merge file into a panel, unless done under the direction of an Import List, is done in the order found in the import file, and that order will be assumed to be the Edit Order of the current panel.

So this means that the first field's data you see in each record in the export file will be taken from the first real field in the Edit Order of the panel from which the

data was exported. The second field's data in each record in the export file will be from the second real field in the Edit Order of that panel, etc. On importing that file back into the same panel, there's no problem, as long as you didn't change the Edit Order some time after performing the export. But if you *did* change the Edit Order, you can run into problems during the import. Be aware that moving a field in a panel with Block (**Alt-F4**) and Move (**Ctrl-F4**) will change the Edit Order of that panel, placing the moved field to the end of the Edit Order. Using Transaction Logs gets around this problem, which I discuss later.

Troublesome ASCII Characters

Not all ASCII characters were created equal, and DataPerfect discriminates with respect to them. You'll have no problem importing a WordPerfect Merge file if it's confined to ASCII characters 32 to 127. That will cover all alphanumeric characters, including all possible punctuation. However, if your import file has characters below ASCII 32 or above ASCII 127, you may find some records may not import, or the import will abort midstream.

If you really want ASCII characters outside the 32-127 range to import, you can do that by first surrounding them with ASCII 225 (ß) characters. For instance, consider the following field data from a record in a teacher's math exam database:

$x \leq (y + z) \uparrow$

That operator you see, after the x is ASCII 243. The ASCII character at the end, of course, is the merge field delimiter (ASCII 18). Because of the occurrence of ASCII 243, attempts to import a record with that line as a field's data won't succeed.

But you can successfully import that field's complete record if you surround that ASCII character with a pair of ASCII 225 characters, like this:

12 ßß (21 + 34)↑

In fact, that's exactly what DataPerfect does when it *exports* such ASCII characters already found in a DataPerfect database. It surrounds each one with a pair of ASCII characters so they can be later imported without problem.

If you're not sure if your import file has any such characters, and you know how many records are in that file, just make sure the total number of records that imported is correct. DataPerfect shows you an import record count during the import. If they all got in, you probably didn't lose any data.

Importing a WordPerfect Merge File

If you're about to import a WordPerfect Merge file, it's probably one you just created with Built-In Short Reports facility discussed above. If so, and you didn't mess with the Edit Order of the panel whose data that export file holds, importing the data back in is easy. First, load the panel in question, which is probably empty of data at this point. You probably deleted all its data in order to do some work on it, like adding a field.

Note: Adding a field won't affect that panel's Edit Order; whereas, *moving* or *deleting* a field will.

Call the Import screen with **Ctrl-F5**:

IMPORT	
1 - Import Filename	
SCRATCH.REP	
Import Type	WordPerfect Merge
2 - WordPerfect Merge	
3 - DOS Delimited Text	
4 - Duplicate Records Action	
Copy All Duplicates	
5 - Copy Duplicate Records to Filename	
6 - Search Conditions	No Search
7 - Create/Edit Import List	
Do Import	
8 - Import Without Disk Space Checking	
9 - Import With Disk Space Checking	
Selection: 0	

Use **1** to name the file to import. Note that if you just ran an export with Built-In Short Reports WordPerfect Merge, the name you gave the export file is already filled in for you. Use **2** to make this a WordPerfect Merge file import. **4** determines what DataPerfect will do with duplicates during import. **5** lets you create the file DataPerfect will use to copy duplicates if **4** was set for that option. **6** lets you limit just what records will be imported from the import file. **7** lets you create an Import List to govern the import process. An Import List essentially maps the fields in the import file to those in the panel. You'll need to create an Import List if the Edit Order of the current panel differs from that in the import file, or if you choose something other than Copy All Duplicates for option **5**. Choose **8** instead of **9** if you know you have enough disk space to perform the entire import. It's a lot faster than **9**. Item **4** is the most difficult one here to understand, so I'll spend time on it.

Duplicate Records Action

Duplicate Records Action (item 4) tells DataPerfect what to do with records in the import file it finds to be duplicates. A duplicate will be any record that has fields that match those of another record with respect to some index field list in that panel. You have three options here, which are presented to you when you hit **4**:

- 1 - copy it to the duplicate record file,
- 2 - ignore it, or
- 3 - replace/merge the record in the database with it.

Copy All Duplicates

This is the default. You'll almost never use anything else. This tells DataPerfect to copy any duplicate it finds to the file specified in option 5. During the Import, you'll see a message at the top of the screen telling you how many records got in the panel, and how many didn't because they were considered duplicates. You can peruse the duplicates in the file specified by option 5.

Ignore Records Duplicated in Index n

If you choose this, DataPerfect will prompt you for the index to use for the comparison testing. Only the fields seen in that index field list will be compared. Records that turn out to be duplicates won't be imported, and there will be no record of which ones didn't get in. You must also set up an Import List for this option to be used.

Replace/Merge Records Duplicated in Index n

Like choice 2, if you choose this, DataPerfect will prompt you for the index to use for the comparison testing. Only the fields seen in that index field list will be compared. The difference here is that each time a duplicate is found, the values found in the duplicate about to be imported will replace those found in the record already in the panel. You must also set up an Import List for this option to be used. This option allows you to *replace* existing records with newer ones, instead of having the newer ones getting dumped into a duplicates file. Just make sure the import file has the *newer* data, otherwise you'll end up replacing newer data with older data.

What Happens During a WordPerfect Merge File Import

Data is fed to fields in Edit Order or Import List order.

This should be obvious by now. Unless an Import List governs the import, data in the import file will stream into the database in the order found in the import file, and will fill real fields in the panel in the order found in the panel's Edit Order. If an Import List governs the import, data will still stream into the database in the order found in the import file, but will fill real fields in the panel in the order found in the Import List.

Formulas update.

All fields not receiving data will have their formulas update, whether those formulas are set to update on any change or on record creation. All fields that have formulas that update on any change, whether or not they receive data, will update.

This last point can be deceiving, however. Many times, if a formula that updates on any change relies on other fields in the panel, you'll find it didn't update properly during an import if the formula is rather complex. You'll notice that, after the import, loading such a record and then hitting **F6** will update the field properly. So you know the formula is okay. Trying the import again won't change this. Changing the Edit Order of the field in the panel sometimes helps, but sometimes not. Watch out for such fields by running a sample import of just a few records first,

seeing whether or not the formulas that are supposed to update on any change during an import really did.

To do a sample import, copy the entire application to another directory and start the import there. After a few records, hit **F1** to Cancel the import. Now browse the records that got in. Check for fields with formulas that update on any change. Hit **F6** to see if they change. If they don't, all is probably well, so you can return to the original directory and do the whole import.

If you simply can't change the formula or the Edit Order before the import to accommodate this problem, you'll have to run a report after the import. Have that report store a value in some field in that panel. Each time it does that, it will put the record into Edit mode. This will successfully update the formulas that failed to update on import.

Totaling may trigger.

If DataPerfect detects a Keep A Total on any field in the current panel, before the import takes place DataPerfect will tell you it sees such code in the panel and will ask you if you want to trigger the totalling. If you say *1* for *Yes*, all Keep A Total operations in that panel will trigger during import.

How you answer this question depends on a few things. For instance, if, after exporting the data to this export file, you deleted all data in that panel and told DataPerfect *not* to undo totals, you don't want to trigger those totals now during import. When deleting all data from a panel with **Alt-F5, 1**, the default is *0* (don't undo totalling). Likewise, when importing data to a panel that has at least one Keep A Total, the default is also *0* (don't trigger totaling). You probably *want* to trigger totaling if this is an empty database and are importing data from another application. In such a case, all the rules that govern Keep A Total operations apply (parent records that need to be created in order to receive the totals in the foreign panels will be created, etc.).

Exporting and Importing Transaction Logs

Transaction Logs were introduced with DataPerfect 2.3. When you can use a Transaction Log instead of a Built-In Short Reports WordPerfect Merge file, do it. They're faster and more convenient.

To export data to a Transaction Log, choose *Export Data Files to Log* from the System and Recovery Operations menu (**Shift-F9, A**). There, DataPerfect allows you to *Export All Data Files to Log* or *Export Selected Data Files to Log*. The first exports all data from all panels to a single file. This is a big difference from the Built-In Short Reports WordPerfect Merge export, which can be done only on a one panel at a time. The second choice allows you to pick a single panel from which to export data to a single file. If you want more than one panel's data in the same log file, but don't want *all* panels' data, you can simply run the Transaction Log export again, choosing *Export Selected Data Files to Log* again, choosing a different panel this time, telling DataPerfect to *append*. If DataPerfect sees a log file by the same name, it always asks if you want to overwrite or append it.

Unlike the Built-In Short Reports WordPerfect Merge export, DataPerfect doesn't use an index when performing a Transaction Log export. It's a physical dump of all data in a single panel, or the entire database. This makes it faster than the Built-In Short Reports export.

Unlike the Built-In Short Reports WordPerfect Merge export file, the Transaction Log file contains references to the panel number and field number for each piece of data. This means you don't have to worry about Edit Order when importing a Transaction Log, nor do you need the proper panel loaded when importing a Transaction Log. The import process (**Shift-F9, 8**) will take care of all that for you.

Unlike importing a WordPerfect Merge file with the Import menu, importing a Transaction Log doesn't offer you a chance to decide what happens with duplicates during import. When importing a Transaction Log, all duplicate records are sent to a file named DP{LOG}.PRB.

Duplicates may show up in DP{LOG}.PRB more often than you would expect. The reason for this is that, as mentioned already, DataPerfect's Transaction Log export facility performs a physical dump of all data in the selected panel or database. Again, this amounts to not using an index to do the export. If there are duplicates in the panel before the export takes place, they'll end up in the log file. This contrasts with how the Built-In Short Reports WordPerfect Merge process works, which uses an index to do its exporting. Because it uses an index, no duplicates will end up in the merge file since an index never sees duplicates. For a more detailed discussion on the problem of duplicates in a panel, see *Removing Duplicates in a Panel* in my **Application Maintenance Issues** chapter.

Importing a Transaction Log also differs from importing a WordPerfect Merge file with respect to what it does with data as it fills the panel's fields. There are two significant differences here. First, a Transaction Log import doesn't trigger any formulas or Keep A Total codes. Second, it resets all auto-incrementing fields to their next highest value, no matter what they were set at before the import.

Strategies: Merge File vs. Transaction Log

Let's outline strategies that help you decide between exporting data with the Built-In Short Reports facility and exporting data with the Transaction Log facility. Here we're concerned with exporting data from a DataPerfect application that will later be imported into the same DataPerfect application.

Deleting and Creating Fields

- If you don't need fields to update their formulas or Keep A Total codes to trigger, export and import your data to and from a Transaction Log. When the database is empty, create new fields first, then delete what you want to delete. This way the fields you create don't take on the field numbers abandoned by the deleted fields. Otherwise your Transaction Log import will place data from the deleted fields in the new ones.

- If you need fields to update their formulas, but don't need Keep A Total codes to trigger, and you have a lot of fields in that panel, it still might be more useful to export and import your data to and from a Transaction Log. This way you don't have to worry about designing an Import List that will skip the deleted fields' data.

But since the field formulas won't update on import this way, you need to run a simple report afterwards. That report simply inserts some value in a single field that updates on any change, and does so for every record. This will trigger all fields in the panel to update, as long as they're coded to update on any change. The format of this report would simply be to set Report Variable 1 to, say, 1, in the First Page Header, and then Store Report Variable 1 in some field in that panel that updates on any change. Though Report Variable 1 holds a numerical value, the field you choose to Store Report Variable 1 in need not be numerical. The simple act of *attempting* to Store Report Variable 1 in any field of any format will throw the panel into Edit mode, causing all fields with formulas that update on any change to update.

However, this won't update the fields that didn't receive data from the Transaction Log, where those fields had formulas that were supposed to update *on creation*. For those, you'll have to make that report update them with a Report Variable that uses the same formula the field does.

- If you need Keep A Total codes to trigger, you're almost always better off exporting and importing to and from a WordPerfect Merge file. This can be done, however, with a Transaction Log. If you elect to use the Transaction Log method, you'll need to run a report later—one that updates all totals. This will probably be much slower than the WordPerfect Merge file method, even if you have to set up an Import List to govern the merge file import.

Moving Fields Without Deleting or Adding Fields

This is still easier with the Transaction Log method, since you don't need to worry about Edit Order. But, since no fields are being deleted or added, you could use the WordPerfect Merge file method without needing an Import List. You should certainly consider this if you need field formulas to update, or Keep A Total codes to trigger.

If you elect to use the WordPerfect Merge method, you'll need to make sure the Edit Order matches the import field order. Don't forget that moving a field with **Ctrl-F4** always places it at the end of the Edit Order. One way to handle this is to place it at the end of the Edit Order *before* doing the export. Then when you can move it later, it won't change its position in the Edit Order.

DOS Delimited Text

This is where you deal with exporting or importing data between a DataPerfect application and an application created by a different database management system. As of this writing, no other database management system reads DataPerfect data files, though some will import data in WordPerfect Merge format.

When someone speaks of DOS Delimited data, they're speaking of data laid out in a DOS text file with one record per line, terminating with a designated ASCII character, and fields delimited by a specific ASCII character. DataPerfect lets you export data to this format as well as import from it, though this can be done only one panel at a time.

Exporting to DOS Delimited Format

To export data from the current panel, **Shift-F7** in Browse mode to call the Report List, and press **Enter** on *Built-In Short Reports*. You get the following familiar screen:

BUILT-IN REPORT/EXPORT				
Destination:		Create Disk File		
1 - Printer On/Off				
2 - Disk File On/Off				
Filename: SCRATCH.REP				
3 - Index Number	1			
4 - Search Conditions	No Search			
5 - Sort Direction	Forward			
6 - Disk File Mode WP/DOS	WordPerfect			
7 - Print Margins	Top	Bottom	Left	Text Lines
	6	0	0	54
8 - Report/Export Format: WordPerfect Merge (Can be imported)				
Fields to be Included:		All Real Fields (Including Hidden)		
9 - Lookup Fields				
A - All Display Fields				
B - All Real Fields				
Selection: (Press Shift-F7 to begin the report) 0				

The item above to attend to first is item 8 (Report/Export Format). It defaults to whatever it was set at the last time that screen was loaded in that application. If this is the first time, you see Report/Export Format set for WordPerfect Merge. Hit **8** to get other choices:

BUILT-IN REPORT FORMAT SELECTION	
1 - Columns, Single Line	(All destinations OK; disk file both WORDPERFECT and DOS)
2 - Columns, Text Wrapped	(All destinations OK; disk file both WORDPERFECT and DOS)
3 - List	(All destinations OK; disk file DOS text only)
4 - WordPerfect List	(WordPerfect disk file destination only)
5 - WordPerfect Merge (Can be imported)	(WordPerfect disk file destination only)
6 - Export DOS Delimited Text (Can be imported)	(DOS text/comma delimited disk file destination only)
Selection: 0	

What interests us here is choice 6 (Export DOS Delimited Text). When you choose this option, you get the following menu:

```

                                BUILT-IN REPORT/EXPORT
Destination:                      Append to Disk File
 1 - Printer On/Off
 2 - Disk File On/Off
    Filename: SCRATCH.REP
 3 - Index Number                  1
 4 - Search Conditions             No Search
 5 - Sort Direction                Forward
 6 - Disk File Mode WP/DOS         DOS Text
 7 - Print Margins                 Top      Bottom    Left      Text Lines
                                6          0          0          54
 8 - Report/Export Format: Export DOS Delimited Text
    Field Delimiter: | Record Delimiter: ~<CR><LF>
Fields to be Included:            All Real Fields (Including Hidden)
 9 - Lookup Fields
A - All Display Fields
B - All Real Fields

Selection: (Press Shift-F7 to begin the report) 0

```

Note the defaults in option 8, where fields are delimited by the pipe symbol (|), and records by tilde (~), carriage return and linefeed. DataPerfect defaults to these delimiters because they rarely appear in data. Exporting data with this setup would produce a file of records like this:

```

Adams|Sally|123 Elm St.|Los Angeles|CA|90024~
Josephson|Abe|34556 Oak Avenue|Santa Monica|CA|90403~
Conrad|J.R.|1121 Yale Blvd.|Somewhere|CA|92345~

```

Though the fields from which the above data was taken were fixed-length fields (A, U, and N fields), the export process strips them of trailing spaces. If you need fixed length-fields in your export file, you'll need to do the export with a report you manually create. Such a report, mirroring the other export, would look something like this (I'll just show just the first three fields here, due to space limitations on the page):

```

-----FIRST PAGE HEADER-----
--Empty--
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
XXXXXXXXXX|XXXXXXXXXX|XXXXXXXXXX~
-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

Just make sure the above report prints to a DOS Text file. It creates a fixed-length export file that has records like this:

Adams	Sally	123 Elm St.	~
Josephson	Abe	34556 Oak Avenue	~
Conrad	J.R.	1121 Yale Blvd.	~

Let's return to the DOS Delimited export. DataPerfect supports exporting to and importing from a special DOS Delimited format: *Comma* Delimited format. To *export* to Comma Delimited format, insert comma for the field delimiter and space for the record delimiter. This produces a menu screen like this:

BUILT-IN REPORT/EXPORT				
Destination:		Append to Disk File		
1 - Printer On/Off				
2 - Disk File On/Off	Filename: SCRATCH.REP			
3 - Index Number	1			
4 - Search Conditions	No Search			
5 - Sort Direction	Forward			
6 - Disk File Mode WP/DOS	DOS Text			
7 - Print Margins	Top	Bottom	Left	Text Lines
	6	0	0	54
8 - Report/Export Format: Export DOS Delimited Text				
Field Delimiter: , Record Delimiter: <CR><LF>				
Fields to be Included:		All Real Fields (Including Hidden)		
9 - Lookup Fields				
A - All Display Fields				
B - All Real Fields				
Selection: (Press Shift-F7 to begin the report) 0				

If we export the same data with the above setup, DataPerfect gives us this:

```
"Adams","Sally","123 Elm St.,""Los Angeles","CA","90024"
"Josephson","Abe","34556 Oak Avenue","Santa Monica","CA","90403"
"Conrad","J.R.,""1121 Yale Blvd.,""Somewhere","CA","92345"
```

The above is in classic Comma Delimited format. You might wonder what will happen to character strings that, in the panel, contain quotes. In creating a Comma Delimited file, DataPerfect will convert all such quotes to

\ "

and all back slashes to

\\

So, consider the following phrase found in an A60 field:

```
I see the name "Ralph" used in file c:\docs\myfile.
```

When DataPerfect exports that string to Comma Delimited format, it ends up like this in the export file:

```
"I see the name \"Ralph\" used in file c:\\docs\\myfile."
```

Also, DataPerfect replaces each carriage return in a variable-length text field's data with

\n

So, consider this data found in an A50A5 field:

```
This is the first sentence.  
Here's the second sentence.
```

There are exactly two carriage returns in the above data, both following the first sentence, yielding a blank line between the two sentences. When Comma Delimited format is used, DataPerfect exports that data like this

```
"This is the first sentence.\n\nHere's the second sentence."
```

Importing From DOS Delimited Format

To *import* from DOS Delimited format, just tell the **Ctrl-F5** Import menu that comma is the field delimiter and space is the record delimiter, using option 3. This gives you this screen:

IMPORT	
1 - Import Filename	
SCRATCH.REP	
Import Type	DOS Delimited Text
2 - WordPerfect Merge	
3 - DOS Delimited Text	
	Field Delimiter: , Record Delimiter: <CR><LF>
4 - Duplicate Records Action	
	Copy All Duplicates
5 - Copy Duplicate Records to Filename	
6 - Search Conditions	No Search
7 - Create/Edit Import List	
Do Import	
8 - Import Without Disk Space Checking	
9 - Import With Disk Space Checking	
Selection: 0	

This chapter is for both beginners and the experienced.

Introduction

DataPerfect offers the definer an internal clipboard facility that allows him Block, Copy, Cut and Paste various database entities. The DataPerfect clipboard is not a *screen capture* facility like that offered by Shell 3.x or 4.x, DESQview, Windows, OS/2, to name a few. The difference here is very important

DataPerfect's internal clipboard facility allows the definer to Block text and fields by simply hitting **Alt-F4** to begin the Blocking operation, followed with using the **Arrow** keys to paint the Block. When the Block is completely painted, the definer may use **F10** to Copy the Block's contents to DataPerfect's internal clipboard, or **Ctrl-F4** to Cut it to the clipboard. In either case, following up with **Ctrl-F4** will Paste it at the new cursor position.

This is very different from using screen capture. A screen capture facility operates on *screen* entities, not *application* entities. I'll make this difference more clear in the following sections, and also point out when it's actually better to use screen capture, even when DataPerfect's clipboard can perform the same operation faster.

In Define Panel Mode

In Define Panel mode, DataPerfect's clipboard allows the definer to Block (**Alt-F4**), Copy (**F10**), Cut (**Ctrl-F4**) or Paste (**F10**) any panel entity, or bunch of entities. All these may be the objects of such clipboard operations:

- Text
- Bold and Underline codes
- Fields
- Links

When you Cut a field from one position on the screen and Paste it to another position in the *same* panel, its field number, name, format, and any formula attached to it, are preserved. However, its Edit Order position is *not* preserved. As soon as it's Cut, DataPerfect assigns it the end position in the Edit Order. Also, when you Cut a field this way, DataPerfect warns you that you *may* delete an index if that field participates in one. I've never seen such an index deleted by a Cut and Paste operation unless, of course, you fail to immediately Paste the field to its new position. If you

don't, you essentially delete the field. That will definitely delete any index in which it participated.

However, if you Cut a field from one position on the screen and Paste it to a *different* panel, only its field format is preserved. It loses its field name and any formula attached to it. And, of course, since you just deleted that field from the source panel, any index in which it participated is now deleted.

If you Copy (instead of Cut) a field from one position on the screen to another position (in *any* panel), only its field format is preserved. It loses its field name and any formula attached to it. Indexes are always preserved in a Copy operation.

Using Screen Capture

If, instead of using DataPerfect's clipboard, you attempt to *screen capture* Define Panel entities, you'll only end up capturing the characters you see on the screen. That's fine if all you want to do is capture text and put it somewhere else. But if you attempt to capture a field or any Bold and Underline codes, you won't be capturing anything significant. That is, all you'll be capturing are the onscreen characters you see, not fields or display codes like Bold or Underline. When you attempt to screen capture a field in Define Panel mode, at best, you'll just get a series of ASCII characters, not a unified field.

In a Specify Formula Screen

In the Specify Formula screen, DataPerfect's clipboard allows you to Block, Copy, Cut and Paste any Specify Formula screen entity (text, as well as field codes like P1F1).

Pasting a portion of the Specify Formula screen to *another Specify Formula screen in the same panel* should work with no special considerations to take into account, other than the format of the field to which it's being Pasted. I mention the latter consideration because you might be, say, Pasting a formula that yields a character string into a Specify Formula screen for a numerical field. For instance, a you might have the following formula in an A5 field:

```
if P1F1=0 then "Hello" else "" endif
```

If you Block and Copy that formula, and then Paste it into the Specify Formula screen for a G999 field, DataPerfect will allow it, but it won't make much sense. That formula will always produce a zero for that field.

Or, again on the field format issue, you need to be careful that the format of the field to which you're Pasting the formula is large enough to accommodate all possible outputs of the formula. The following formula might be attached to an A7 field:

```
cases P1F1
  case cv=0 of "Hello"   endof
  case cv=1 of "Goodbye" endof
  default      ""
```

endcases

But Pasting it into the Specify Formula screen of an A5 field won't work well, as it won't ever display more than the first five letters of *Goodbye*.

Pasting a portion of the Specify Formula screen to *another Specify Formula screen in a different panel* requires more thinking than when targeting a field in the *same* panel. Every field code in the original formula will be invalid in the target field, so you'll have to reselect all fields in the formula in its new home. For instance, take the above formula again. P1F1, when in a formula in Panel 1, is no problem. It refers directly to the Field 1 in Panel 1. But that field code in a field formula in Panel 2 is nonsense. It refers to nothing. DataPerfect will let you Block and Copy P1F1 from the Specify Formula screen in Panel 1, and Paste it into the Specify Formula screen associated with Field 1 of Panel 2, but it won't do anything because the formula won't find P1F1 while sitting in Panel 2. To get the formula to see P1F1 while in Panel 2, you'll need to create a link to Panel 1, and replace any field code in that formula referring to field P1F1 with one obtained by selecting P1F1 after penetrating the link (**F4** in the Specify Formula screen, **Tab** to the link, **Down Arrow** to penetrate the link, **Tab** to the field and hit **F4**). This will give you something like

P2F15P1F1

instead of

P1F1

where the link used is P2F15. P2F15 would either be a panel link, identified by DataPerfect as Field 15 in Panel 2, or a data link attached to Field 15 in Panel 2.

Using Screen Capture

If, instead of using DataPerfect's clipboard, you attempt to *screen capture* Specify Formula screen entities, you'll do fine with all but field codes. That is, all the alphanumeric (and other ASCII) characters will Block, Copy and Paste fine, but the field codes, like P1F1 won't transfer as field codes. Rather, they'll transfer as alphanumeric strings. For instance, if you screen capture the field code P1F1, you'll be capturing four alphanumeric entities, not a field code. In a Specify Formula screen, a field code is a single entity. Note that when you place the cursor on field code P1F1, hitting **Right Arrow** moves the cursor to the right of P1F1, not just to the right of the *P*.

In Report Definition Mode

DataPerfect's clipboard allows you to Block, Copy, Cut and Paste an entire section of a Report Definition at one time. That is, it allows you to Block all that exists in a First Page Header section, or all that exists in an Other Page Header section, or all that exists in a Two-Level Header section, etc. It won't allow you to paint the Block beyond the borders of a single Report Definition section. This allow you to quickly

Paste all or some of the text, fields, Report Variables, and Report Options from one section to another in the same Report Definition, or to another Report Definition in the same application. However, heed similar caveats here:

First, if you Paste some or all of the Specify Formula screen of a Report Variable into the Specify Formula screen of a different Report Variable, and the target Report Variable will later be printed (**Ctrl-F7, 1, 5**), remember that a *printed* Report Variable has a field format. Make sure the formula's output is compatible with the target Report Variable's format.

Second, be painfully aware of any Report Definition entities in the clipboard that may reference specific panel entities. In a Report Definition, such entities would be selected fields in the Edit Report Form screen (what you get when you **F4** to select a field in, say, the First Page Header or Report Body) and selected fields in Report Variable formulas (what you get when you **F4** to select a field in a Specify Formula screen).

Along these lines, don't think that just because you're Pasting within the same report that you're safe here. You may be Pasting between sections in that report that are tied to different panels (as is typically the case when Pasting between a main report and one of its subreports). For instance, consider the following Report Definition:

```

=====FIRST PAGE HEADER=====
--Empty--
=====OTHER PAGE HEADER=====
--Empty--
=====TWO-LEVEL REPORT HEADER=====
--Empty--
=====REPORT BODY=====
Name: 
Date: 

Date      Transaction      Amount
=====
=====SUBREPORT LINK/PANEL: 4 3=====
=====FIRST PAGE HEADER=====
=====OTHER PAGE HEADER=====
Date      Transaction      Amount
=====
(continued)
=====TWO-LEVEL REPORT HEADER=====
--Empty--
=====REPORT BODY=====
=====TWO-LEVEL FOOTER=====
--Empty--
=====PAGE FOOTER=====
--Empty--
=====FINAL FOOTER=====
=====
Total: 

Thanks for your patronage, !
-----Page Eject-----
=====END OF SUBREPORT=====
=====TWO-LEVEL FOOTER=====
--Empty--
=====PAGE FOOTER=====
--Empty--
=====FINAL FOOTER=====
--Empty--

```

Main report:
Client Panel

Subreport:
Transaction Panel

Main report:
Client Panel

Note the Final Footer of the subreport. The field that follows

Thanks for your patronage,

is supposed to point to the First Name field. But if that field was Blocked and Copied from the Report Body of the main report, it won't work. It must be reselected with **F4** in that Final Footer.

If you aren't careful to reselect fields in this sort of situation, running that report will dump the entire application out to DOS! And if you're stumped as to why this is happening, and decide to cursor to each such field and hit **Alt-F3** to see what it points to, it will show you the original field it pointed it to. For instance, hitting **Alt-F3** on the field in the Final Footer above, even if Blocked and Copied from the Report Body of the main report, will still show, incorrectly, that that report field points to the desired panel field. You'll only realize it's incorrect when you take a look at the field string noted by DataPerfect when the cursor sits on the field in the Edit Report Form screen. If that field was taken from a part of the Report Definition that's based on Panel 1, and it now sits in a section that's based on Panel 2, then this should tip you off that something's not right:

—Column 1—

Type the text to be included in the report. To include a data field, press Select (F4), move to the field (possibly through links,) and again press Select. While the cursor is on a report field mark, you can press F6 to edit the report format. To include variable fields, prompts or special control instructions, press Report Options (Ctrl-F7).

Path to field: P1F1

Field Format: G999

—FIRST PAGE HEADER—

Note the *Path to field* line above. That field is in Panel 1, but if the Report Form section in which its field codes sits is tied to Panel 2, you're in trouble. To reference that field while in a Report Form section tied to Panel 2, you'll need to delete the field and then reselect it with **F4**. This may involve going through a link to get it if a similar field doesn't exist in Panel 2. When you do that, the display should look more like this:

—Column 1—

Type the text to be included in the report. To include a data field, press Select (F4), move to the field (possibly through links,) and again press Select. While the cursor is on a report field mark, you can press F6 to edit the report format. To include variable fields, prompts or special control instructions, press Report Options (Ctrl-F7).

Path to field: P2F15P1F1

Field Format: G999

—FIRST PAGE HEADER—

In the above, the *Path to field* line now references

P2F15P1F1

not

P1F1

Hitting **Alt-F3** on that field in the Report Definition will still show you Field 1 of Panel 1, but the *Path to field* statement above is telling you the Report Definition is using link P1F15 to grab it.

Field Formulas and Help Screens: A Caveat

Some DataPerfect application developers like to put copies of field formulas in Help screens during debugging stages of development. They Block and Copy a complicated formula from the field's Specify Formula screen and Paste it in that field's Help screen. This way they can see the formula of a field without going into the Specify Formula screen. A useful practice, indeed, but don't use DataPerfect's clipboard for this.

First of all, doing this isn't going to Paste a Specify Formula screen field code (like P1F1) to the Help screen. But more importantly, there's a bug still deep in DataPerfect code that sometimes trashes the .STR file when you Paste strings from the Specify Formula screen to the Help screen, or vice versa. It has yet to be pinned down. But, since you can't get the Specify Formula screen field codes this way anyway, just use *screen capture* instead of DataPerfect's clipboard. It's slower to use, but will avoid both problems mentioned.

Securing the Application

Here I target both the beginner and the experienced, though the beginner will find much of this rather confusing.

Application Passwords

The most basic way to secure an application from user abuse is to use what DataPerfect has provided since its initial release: application passwords. These are accessed with **Shift-F9, 5** while in Browse mode, letting you define rights based on application password access to the database when it's initially loaded.

Here are the four possible levels given to a user via the application password facility:

- Definer
- Supervisor
- Read/Write
- Read-Only

For some odd reason, though, the Reference manual never exhaustively delineates what each application password grants or denies you. Here they are, from the most powerful to the least. Each password level grants its owner all the rights next to its name here, and all the rights next to the names below it:

Rights Granted by Application Passwords	
Level	Rights Granted
Definer	Create or change application passwords. Enter Define Panel mode. Enter Define Index mode. Select User ID Panel. Define the application banner.
Supervisor	Mass deletes with Alt-F5 Recover indexes. Start and stop a Transaction Log. Export data to a Transaction Log. Import a Transaction Log.
Read/Write	Create, Edit, and Delete records as permitted by menu access restrictions.
Read Only	Browse records.

And here's an outline of just when DataPerfect prompts the user for an application password (as opposed to their *User ID* Password, which I discuss later).

The user actions on the right cause DataPerfect to prompt for an application password, given the state of the application security on the left:

When DataPerfect Prompts for Application Passwords	
<i>Application Security Status</i>	<i>User Action Causing Prompt</i>
Application passwords defined. User ID Panel selected.	Leaving User ID or User Password blank. Define Panel (Alt-F8) . Define Index (Ctrl-F8) . System and Recovery Options (Shift-F9) .
Application passwords defined. User ID Panel <i>not</i> selected.	When loading the application. Define Panel (Alt-F8) . Define Index (Ctrl-F8) . System and Recovery Options (Shift-F9) .
Application passwords <i>not</i> defined.	Never. What did you <i>think</i> I'd say?

Menus

With version 2.3 came DataPerfect's menu facility. This gives the definer much more control over how the user will manipulate the database. With only application passwords defined, and no menus defined, whatever rights are assigned via the application passwords on entry into the application, carry forward to all panels on the Panel List. So if the user has Read Only rights on entry into the application, they can't create, edit or delete any records in any panel in the application. Likewise, if they have Read/Write rights, they can create, edit or delete any record in any panel on the Panel List.

But suppose you want the user to have Read/Write rights in the Transaction Panel of your application, Read Only rights in the Auto Parts Panel and Create Only (No Delete or Edit) rights in the Invoice Panel. Until version 2.3, this sort of security wasn't possible with native DataPerfect, though some of it was possible with the DPMouse© third-party utility (see the entire DPMouse© manual on the provided diskette). This sort of security is possible with DataPerfect 2.3's menu facility.

The Define Menu Screen

To call DataPerfect's menu facility, load the Panel List (or load a menu, if you already have a menu on this application). From there, hit **Alt-F8** to get the Define Menu screen:

Define Menu					
Options 1-7 Create New Menu Entries.					
1 - Go to Panel (Definer Keyword)	6 - Submenu				
2 - Go to Panel (User Selects Record)	7 - Launch Shell Macro				
3 - Run Report	8 - Edit an Existing Entry				
4 - Go to Panel List	9 - Delete an Existing Entry				
5 - Go to Report List	A - Create/Edit Menu Text				
Selection: 0					B - Move the Menu Prompt
01	18	35	52	69	86
02	19	36	53	70	87
03	20	37	54	71	88
04	21	38	55	72	89
05	22	39	56	73	90
06	23	40	57	74	91
07	24	41	58	75	92
08	25	42	59	76	93
09	26	43	60	77	94
10	27	44	61	78	95
11	28	45	62	79	96
12	29	46	63	80	97
13	30	47	64	81	98
14	31	48	65	82	99
15	32	49	66	83	
16	33	50	67	84	
17	34	51	68	85	

Let's go over the options you see above.

Create/Edit Menu Text

Let's start with option A (Create/Edit Menu Text). Hitting **A** gives you this screen:

This is a sample menu form. You may delete it entirely, change its form, or press the Ins key (for replace mode) and fill it in.

MENU

SELECTION:

Here's where you create the menu itself, as the user will see it. If this is the first time you got this screen for this application, this is the initial menu of the application. All other menus will be, directly or indirectly, submenus off this menu.

This is a simple text editor that supports all the character keys, along with **Enter**, **Delete**, **Backspace**, and **Insert**. After editing it to look the way you want (you can always change it later via option A), hit **F7**. At that point, your screen may look like this, with a movable Menu Prompt somewhere on the screen:

```

Auto Parts Database 1.0

Main Menu
[1] Enter Customer Panel
[2] Enter Auto Parts Panel
[3] Print Day Sheet
[4] End of Month Submenu

Selection:      (movable Menu Prompt)
  
```

That movable little block is the Menu Prompt. Use the cursor keys to put it where you want (in this case, to the right of the string *Selection:*). Then hit **F7** or **F10** to save and exit the screen.

That simple menu doesn't do any thing yet. Now we must code it so that when the user hits **1** he lands in the Customer Panel, when he hits **2**, he lands in the Auto Parts Panel; when he hits **3**, he prints a Day Sheet; and when he hits **4**, he sees yet another menu, this one offering him options for to perform at the end of the month.

Move the Menu Prompt

Option B just lets you move the Menu Prompt without having to go through editing the Menu Text first. If you choose option A instead, but then don't edit the Menu Text screen, hitting **F7** or **F10** won't prompt you to move the Menu Prompt.

Go to Panel

As you can see, there are two Go to Panel options (1 and 2). The first allows the definer to designate a Keyword that will control entry into the designated panel, the second allows the user to do this after choosing that menu item. A menu item that takes you to a panel acts just like a panel link, and the menu item's Keyword acts like the first field in a panel link's field list.

For example, if the definer defined menu item 1 above to go to the Customer Panel with the Keyword *Adams*, then every time the user chose menu item 1 on the main menu, he'd land in the Customer Panel with access only to Customers with the Last Name *Adams*. This, of course, would be true if the menu item was assigned a Customer Panel index that begins with the Last Name field. Most of the time you won't assign a Keyword to a Go to Panel option, which means you're giving the user full access to all records in its target panel.

When you choose Go to Panel from the Define Menu screen, DataPerfect first asks you what menu item this is supposed to attach to, then it presents you with the Panel List from which to choose the target panel. Next it offers you one of two possible menus, depending on whether you chose the Definer Keyword (1) or User Selects Record (2):

Go to Panel (Definer Keyword)			
Panel: TEST.NEW			
1 - Panel Access Rights:	Read/Write		
2 - Access Report List?	Yes		
3 - Restrict Modification to First Level	No		
4 - Edit Password	6 - Choose Index	0	
5 - Edit Key Word	7 - Subset	No	
Selection: 0			

Go to Panel (User Selects Record)			
Panel: TEST.NEW			
1 - Panel Access Rights:	Read/Write		
2 - Access Report List?	Yes		
3 - Restrict Modification to First Level	No		
4 - Edit Password			
5 - Choose Index	0	6 - Subset	No
Selection: 0			

They're identical Go to Panel options menus, except the first one includes the Edit Key Word option. Let's go over these options one at a time.

Panel Access Rights

This determines what the user can do in the target panel. It toggles between the following four options:

```

Read/Write
No Delete
Create OK - No Edit/Delete
Read-Only

```

They determine what the user can do in the target panel, and, if Restrict Modification to First Level is set to *No*, whatever you set for Panel Access Rights carries forth to all panels the user can access directly or indirectly from the target panel. If Restrict Modification to First Level is set to *Yes*, then Panel Access Rights applies only to the target panel, because Restrict Modification to First Level will force the user into Read Only mode in panels accessed from the target panel.

Access Report List

This is straightforward. If set to *Yes*, the user has access to the Report List while in Browse mode in the target panel. This is dangerous. A user with access to the Report List can create a report that prints out data from any panel in the database. It also lets the user create a report that deletes any data anywhere in the database. I suggest this is always set to *No*.

Restrict Modification to First Level

As mentioned already, if set to *Yes*, it overrides Panel Access Rights when the user penetrates a link in the target panel. From that point on, the user is in Read Only mode, until they return to the target panel.

Edit Password

This lets you place a password on entry in the target panel. If you do, DataPerfect prompts the user for a password when choosing to enter this panel from this menu item.

Edit Key Word

We already discussed this. It only shows up on the Go to Panel (Definer Keyword) option.

Choose Index

This is straightforward. Choosing that option allows you to tie entry into the target panel to the index of your choice. Otherwise, the user will enter that panel under the control of the lowest numbered index for that panel. This option is especially important if you choose Go to Panel (Definer Keyword) and do indeed define a Keyword for entry into the target panel, or choose Go to Panel (User Selects Record). In either case, the Keyword facility is activated, so the index is important. The index must be one whose first field is compatible with the Keyword.

Subset

This is important only if panel entry is, again, governed by a Keyword. If so, and Subset is set to *Yes*, then DataPerfect will honor subsets beginning with the Keyword. For instance, if panel entry here is governed by Go to Panel (User Selects Record), and the user enters Adams for the Keyword, then they'll have access to Adams, Adamson, etc.

Run Report

When you choose Run Report from the Define Menu screen, DataPerfect first asks you what menu item this is supposed to attach to, then it presents you with the Report List from which to choose a report. When you highlight a report and hit **Enter**, you're presented with the following slightly modified Initial Report Definition Screen:

REPORT: Day Sheet				
Destination:		Create Disk File		
1 - Printer On/Off				
2 - Disk File On/Off	Filename: SCRATCH.REP			
3 - Index Number	1			
4 - Search Conditions	No Search			
5 - Sort Direction	Forward			
6 - Disk File Mode WP/DOS	DOS Text			
7 - Print Margins	Top 6	Bottom 0	Left 0	Text Lines 54
8 - Edit Report Form				
9 - Edit Report Name				
A - Report Mode - Normal/Lookup		Normal Report Mode		
B - Password:				
C - User Set-Up		No		
Selection: 0				

The above Initial Report Definition Screen differs slightly from that found if you accessed the same report on the Report List when hitting **Shift-F7**. It's the same report (that is, the Edit Report Form screen is identical), but has options A, B, and C at the bottom. Hitting **A** toggles between Normal Report Mode (the default) and Single Record Report From Lookup.

Normal Report Mode

This results in the report running just like it does when run from the Report List. Single Record Report From Lookup mode causes the report to first offer the user a lookup of records in the report's index. Once the user selects a record from that lookup by hitting **Enter**, the report runs to completion on that single record and the presents the user with another lookup where they may choose another record or hit **F7** to terminate the report. I discuss this further, especially the way it differs from using the User Chooses Next Record From Lookup code, in the *Single Record Report from Lookup off a Menu* section of my **Iteration Control** chapter.

Password

This lets you setup a password for this report. If you do, DataPerfect prompts the user for a password when choosing to run this report from this menu item.

User Set-Up

This defaults to *No*. If toggled to *Yes*, then, upon choosing this report from this menu item, DataPerfect will initially present this abbreviated Initial Report Definition Screen to the user, instead of immediately running the report:

REPORT: Some Report				
Destination:		Screen Only		
1 - Printer On/Off				
2 - Disk File On/Off				
3 - Index Number	1			
4 - Search Conditions	No Search			
5 - Sort Direction	Forward			
6 - Disk File Mode WP/DOS	No Disk File			
7 - Print Margins	Top	Bottom	Left	Text Lines
	6	0	0	54
Selection: (Press Shift-F7 to begin the report) 0				

The above intervening menu lets the user change any of the key elements you see above before running the report. DataPerfect won't save any changes the user makes.

Run Report Option Caveats

Menu Reports as Copies

As you can see, you, the definer, can make changes to a Report Definition that's attached to a menu item. When first attaching it to the menu item, you have full access to all the Initial Report Definition Screen options of that report, including option 8 (Edit Report Form). And you can later gain access to that Initial Report Definition Screen by going back into Define Menu mode (**Alt-F8**) and choosing

Define Menu's option 8 (Edit an Existing Entry), then choosing the menu item to which that report is attached.

So you can make changes to the report you attach to a menu item, just as though you were accessing it from the Report List. Be aware of something crucial here. That report you assigned to that menu item is a *copy* of the one you selected from the Report List. It's not the same report. This has consequences.

When you edit the Menu version of that report, you don't affect its Report List version. Likewise, when you alter the Report List version of that report, you don't affect its Menu version. To update the Menu with the new version of the report you just changed on the Report List, you must go back to that menu in Define Menu mode, delete the old version from that menu item, and then assign the new version to that menu item. Keep this in mind when, after working hard on revising an old report on the Report List, it still seems to run the way it used to when run from a menu.

Menu Report Bug Alert

There's a very important DataPerfect bug here that may occasionally bite you. If you tell DataPerfect to attach a report to a menu item and then cancel with **F1**, you may corrupt your .STR so that it's unfixable with DPEXP.COM. That is, in Define Menu mode, if you hit **3** to attach a report to a menu item, and then choose a menu item number, you see the Report List from which to choose a report. If, while the Report List is on the screen, you decided to cancel this operation with **F1**, you may later find your .STR is corrupt. The corruption may not show up right away. If it's there, however, running DPEXP will expose it. The symptom will be that DPEXP never completes. If this happens, the .STR is not fixable.

So, if, while attaching a report to a menu item, you decide to cancel the operation, don't hit **F1** with the Report List displayed. Rather, complete the attachment with some report (any will do, even Built-In Short Reports). Then delete the menu item with **9** from the Define Menu later.

Go to Panel List

Option 4 (Go to Panel List) is straightforward and dangerous. Assigning that to a menu item gives the user full access to all panels in the database. I can't imagine ever giving any user this sort of access. Most developers assign this option to menu item 99, never displaying that option on the menu itself, while assigning it a password (DataPerfect allows this access to be password protected, as you find out when you attach it to a menu item). That way the developer can gain access to the Panel List when needed. Don't forget that the Panel List doesn't display when the application is loaded, as long as it has a menu. So this is the way most developers give themselves access to the Panel List when a menu exits.

There's another way for the developer to get to the Panel List in an application with a menu, but that only applies to applications that employ the User ID Panel facility. I discuss this later, in *The User ID Panel vis a vis Application Passwords* in the *User ID Panel* section of this chapter.

Go to Report List

Like option 4, option 5 (Go to Report List) is straightforward and dangerous. This option, if assigned to a menu item, gives the user full access to the DataPerfect report facility, allowing him to write reports that print all data in sensitive panels, or, worse, delete all data in any or all panels of the database. Many developers assign this option to menu item 98, never displaying that option on the menu itself, while assigning it a password (DataPerfect allows this access to be password protected, as you find out when you attach it to a menu item). That way the developer can gain access to the Report List when needed. Otherwise, you, the developer, can just rely on menu item 99 granting you access to the Panel List (see above section). From there you load a panel and hit **Shift-F7**.

Submenu

Option 6 (Submenu) is straightforward. It lets you create an entire system of menus that branch off other menus, starting with the main menu. When you assign this option to a menu item, you get a fresh menu to define with all the same possible options. However, when you later decide to return to edit a submenu, hitting **Alt-F8** always brings up the main menu, no matter what menu was displayed when you hit **Alt-F8**. So you must begin the Edit an Existing Entry process (option 8) in the *main* menu. From there you tell DataPerfect which of that menu's submenus to edit.

Launch Shell Macro

With option 7 (Launch Shell Macro), you can have a menu item do things that reports can't. Reports, of course, are the only automated processes a user can call from a menu, other than Shell macros. Any macro that can be run with Shell 3.x or 4.x with DataPerfect loaded can be used here. Such a macro might, for instance, run a particular DataPerfect report that offers the user a lookup of records from which to choose, where that report creates a WordPerfect Merge file from the user's choices. That same macro can then load WordPerfect and complete the merge from there, printing a series of letters.

Edit an Existing Entry

Option 8 (Edit an Existing Entry) is straightforward. After hitting **8**, DataPerfect prompts you for the menu item to edit. For instance, if the menu item is a Run Report option, then you get that report's Initial Report Definition Screen (see the *Run Report Option Caveats* section above for important notes about this). If the menu item is a Go to Panel option, you get the Go to Panel options menu for entry into that panel.

Delete an Existing Entry

Option 9 (Delete an Existing Entry) is straight forward. When you hit **9** from a Define Menu screen, DataPerfect prompts you for the item to delete.

User ID Panel

The User ID Panel facility, which was introduced with DataPerfect 2.3's second release (September 1993), offers another layer of database security to your application. It allows for 16 million unique User ID/Password combinations per database. With the User ID facility activated in a particular application, the user is prompted for his User ID and Password before being allowed into the database. Activating this optional User ID Panel facility offers two significant benefits:

- A way to track exactly who's entering data in any record at any moment in time
- Controlled access, based on assigned User IDs and User Passwords

Here's how the User ID facility works. First, you'll need a panel to serve as your User ID Panel. It must have at least three fields. Fields 1 and 2 must be A or U fields, up to 25 characters each. Field 3 must be a G or N field, up to two digits in length. When I say Fields 1, 2, and 3, I'm referring to their Field Numbers, not their Edit Order positions. Field Numbers are exposed in Browse or Define Panel mode by hitting **Alt-F3** twice. Hitting **Alt-F3** once more shows Edit Order. You'll probably want this panel to be a newly created panel. You can name it anything you want, but you might as well name it the User ID Panel. Also, though DataPerfect will allow you to do otherwise, index 1 of this panel should consist solely of Field 1 (more on this last point in a later section, *Tracking User Activity*).

After you create your User ID Panel, you must tell DataPerfect that it's the one it should consider your User ID Panel. That is, naming it the User ID Panel isn't enough. Like I said, you can actually name it anything you want. At this stage you must select it as the User ID Panel with **Shift-F9, B**. This will present you with the Panel List, where you must highlight the User ID Panel and hit **Enter**. After you select your User ID Panel from the Panel List, there's no indication on any screen that any panel at all has been selected as the User ID Panel. The only way to find out if one has been selected is to load the application and see watch to see if it asks you for your User ID and Password.

Once selected, DataPerfect will consider that the User ID Panel's Field 1 to be the User ID field, Field 2 to be the Password field, and Field 3 to be the Access Level field. All other fields are up to the Definer in terms of significance in the database. Again, when I say Field 1, I'm referring to its field *number*, not Edit Order position.

Selecting this panel as the User ID Panel has major consequences. Every user entry into the database itself will now be controlled by the User ID facility. If the User ID Panel is selected, and both Definer and Supervisor application passwords

defined (see the next section, *User ID Panel Caveats*, for more on the need for application passwords being defined), then when the user attempts to load the application, DataPerfect will ask him just who they are. This is done with a series of two prompts, one asking for his User ID and the other asking for his Password. After the user fills in both prompts, DataPerfect looks to see if there's a User ID Panel record with that precise combination. That is, say John enters *John* as his User ID and *SECRET* as his Password. Then DataPerfect will look to see if there's a User ID Panel record with *John* in Field 1 and *SECRET* in Field 2. If no such User ID Panel record exists, John is sent back to DOS.

If there *is* such a User ID Panel record, two important things happen. First, DataPerfect sets up a scheme to track the movements and actions of each user currently accessing the database. To do this, DataPerfect stores in memory the user's User ID. With that value in memory, which is the value found in Field 1 of the user's User ID Panel record, DataPerfect makes it possible to globally access *all* fields in that user's User ID Panel record. DataPerfect allows this access with the `USER.FIELD[n]` function. Second, DataPerfect sets up a scheme that allows for controlled access to the database. DataPerfect does this by activating the Access Level found in Field 3 of the user's User ID Panel record. Before spelling out each of these events in more detail, I must emphasize a few caveats.

User ID Panel Caveats

Three caveats. *First*, though the User ID Panel facility can be activated with **Shift-F9, B** without any Definer or Supervisor application passwords assigned to the application (**Shift-F9, 5**), it shouldn't be used that way. I'll explain this later. For now, make sure both Definer and Supervisor application passwords are defined for any application for which you want to activate the User ID facility. You don't need to have Definer and Supervisor application passwords defined for the application before selecting your User ID Panel. Just make sure you define them at some point before letting users use the application (assuming you want the application protected by the User ID Panel facility).

Second, when you export an .STR file DPEXP.COM, you create an .STE file that lacks application passwords. This is done to protect the application from users seeing application passwords in a stray .STE file on their hard drive (after, say, you did some maintenance on their application in their office and didn't delete an .STE you created with DPEXP.COM). So when you then create a new .STR by importing the new .STE with DPIMP.COM, the newly created .STR will have no application passwords defined anymore. Also, this newly created .STR won't have the User ID Panel selected. So, after a DPEXP/DPIMP session, make sure you redefine your application passwords and reselect your User ID Panel.

Third, as mentioned earlier, and explained soon, the first index of the User ID Panel must consist solely of the User ID field.

Tracking User Activity: the *USER.FIELD[n]* Function

If the User ID Panel facility is active, DataPerfect knows who's in the database. No matter what panel a user is in, or what report he's running, DataPerfect knows not only what his User ID and Password are (which he gave during the initial two prompts), but DataPerfect also indirectly knows the value found in each field of his User ID Panel record. I don't just mean the values found in the first three fields (User ID, Password, and Access Level fields) of the user's User ID Panel record, but, rather, the values found in *all* the fields in that record. These values follow the user around wherever he goes in the database, and they can be accessed with the *USER.FIELD[n]* function.

DataPerfect, via the *USER.FIELD[n]* function, has *direct* access to the user's User ID because it's sitting in memory, as mentioned before. DataPerfect, via the *USER.FIELD[n]* function, has *indirect* access to the *other* fields in the user's User ID Panel record by using the User ID value as a key field in a dynamically created virtual link to the User ID Panel. It's essentially acting like there's a link in the current panel that targets the User ID Panel, with the User ID value as its sole field list component. Using an index that starts with the User ID field, DataPerfect then has access to all fields in the User ID Panel, not just the User ID field (Field 1).

Given the way the *USER.FIELD[n]* function works, *the first index of the User ID Panel must consist solely of the User ID field*. This will allow DataPerfect, when called to do so by the *USER.FIELD[n]* function in a formula, to create a virtual link and access all the User ID Panel fields, using the User ID stored in memory to form the virtual link field list. The Reference manual doesn't talk about the User ID Panel facility (the manual wasn't updated with the second, September 1993 version of DataPerfect). But even the README that references the User ID Panel facility fails to mention the necessity of User ID Panel index 1 consisting solely of the User ID field.

The developer can access those User ID Panel fields in any formula in any field or Report Variable by using the *USER.FIELD[n]* function, where *n* is the field number of concern in the User ID Panel. So *USER.FIELD[1]* returns the current user's User ID, *USER.FIELD[2]* returns his Password, and *USER.FIELD[3]* returns his Access Level. And *USER.FIELD[4]* returns the value found in Field 4 of his User ID Panel record, *USER.FIELD[5]* returns the value found in Field 5, etc. Fields 4 through 125 (you're only allowed 125 fields in a panel) are left up to the developer for their significance in the database. Fields 1 through 3, of course, are rigidly defined.

The *USER.FIELD[n]* function, then, can be used to access user-specific data. For instance, you might want, in your Transaction Panel, an A25::N field to display show who created the current record by giving it

```
user.field[1]
```

as its field formula, to update on record creation. That would display the User ID of the user who created the record. You might also want a second A25::N field to

display the User ID of the user who most recently edited the current record. Again, you'd use

```
user.field[1]
```

but this time you'd have it update *on any change*.

Alternatively, you might want only certain users to run certain reports. One way to do this is to have a special field in the User ID Panel set up for this purpose. Say it's Field 4, and has the format G9. You have a special report that's to be run only by users with certain qualifications. Assuming the Supervisor of the application has access to the User ID Panel, he can enter the number 1 in Field 4 of the appropriate users' User ID Panel records, otherwise leaving other users' User ID Panel records with a 0 in Field 4. Now you have the sensitive report abort if it sees the current user's USER.FIELD[4] isn't equal to 1. Assuming you already know about reports and Iteration Control options, such a Report Definition would involve setting, say, Report Variable 1 to

```
user.field[4]=1
```

in the report's First Page Header, followed immediately by a Stop if Report Variable 1 is False code. Such a device would cause the report to stop as soon as it started, if the current user's Field 4 isn't equal to 1.

More on using the USER.FIELD[n] function later. The point to get right now is that, when the User ID Panel facility is active, any formula can access the value of any field in the current user's User ID Panel record by using the USER.FIELD[n] function. Also, a secondary consequence of DataPerfect storing the user's User ID in memory is that, if the Transaction Log facility is active, each entry in that log will contain the User ID of the user responsible for that entry. This way the Supervisor can see just who created, edited or deleted this or that record.

Controlling User Access

Whereas the first thing DataPerfect does when it finds out who the current user is, is to store their User ID in memory, making data in their User ID Panel record globally available to the application, the second thing it does is guide the user's entry into the database itself. Again, if DataPerfect doesn't see a User ID Panel record with the current user's User ID and Password in Fields 1 and 2 respectively, he's kicked back to DOS. But if it *does* see such a User ID Panel record, one of two events takes place. If a menu exists, DataPerfect executes the menu item on the initial menu that matches the value found in the Access Level field (Field 3) of that user's User ID Panel record. If no menu exists, DataPerfect places the user in the panel whose panel number corresponds to the value found in the Access Level field of that user's User ID Panel record.

When the application has menus, coupled with an active User ID Panel facility, the user never sees the initial menu. For instance, let's say the initial menu in the application has menu items 1 through 25 defined by the developer. Well, if the

value in Access Level field of the current user's ID Panel record is 5, then, assuming that user answered the User ID and Password prompts properly, DataPerfect will immediately execute the initial menu's item 5 *without the user ever seeing the initial menu*. So the User ID Panel facility puts the current user into a submenu off the initial menu.

You've already seen what security options are available to the definer when framing that initial menu. All those you read about in the *Menus* section above apply equally well to the initial access menu. As you can now see, you can hold the user's hand with the User ID Panel facility in concert with the application's initial menu, by having the initial menu branch to various submenu paths, each with different levels of security. What will determine which initial path the user falls into is the value found in the Access Level field of his User ID Panel record.

Okay, before I get into more detail on the User ID Panel facility, let me show you an example of how to set up the initial menu so it works with the User ID Panel facility, holding the users' hands as they initially enter the database. That may not be all that clear at this point to you. Here's a summary of how to do this:

- Create your initial menu, which I call the application's Access Menu. This must be the initial menu of the application.
- Configure the Access Menu in a way that provides different paths into the database. This usually means you define a submenu for each item on the Access Menu. Each submenu has certain rights, corresponding to the Access Menu option chosen. The Access Menu option chosen will be the user's Access level, as found in their User ID Panel record's Access Level field (again, that's Field 3). Take a look at this example of a simple Access Menu I wrote for a client:

ACCESS MENU	
[1] Read/Write and User ID Panel access	
[2] Read/Write without User ID Panel access	
[3] Read Only with Read-Only report generation	
[4] Read Only with no report generation	
	[98] Reports
	[99] Panels

The user never sees this menu. Each of the first four menu items calls a submenu with options for that particular Access level. Access to the Panel and Report List are there too, but the definer is the only one who ever gets to use those menu items (he's the only one who ever sees this menu)

- Create and select the User ID Panel. The User ID Panel for the above application looks like this:

USER ID PANEL	
User ID	XXXXXXXXXX
User Password	XXXXXXXXXX
Access Level	1
Access Levels explained	
Level 1 (Owner):	Full Access.
Level 2 (Manager):	Full Access, excluding the User ID Panel.
Level 3 (Operator):	Browse Only, excluding the User ID Panel. Can run reports that don't alter data.
Level 4 (Browser):	Browse Only, excluding the User ID Panel. No reports.

As you can see, I give only the Owner of this application rights to access the User ID Panel.

- Create a User ID Panel record for the application's Owner. Otherwise they won't be able to get into the application you ship to them. Tell them their User ID and Password, of course. They can change that later.
- Define both Definer and Supervisor application passwords.
- Create appropriate submenus, each called by a different Access Menu option. Each submenu will have different rights, corresponding to the Access level scheme implied by the Access Menu.

Don't feel you must be limited to the scheme outlined in the User ID Panel and Access Menu I just showed you. You can have many more Access Menu options, with varying rights. Perhaps you want to have a different access path for each of 50 workers at the job site. If so, you'll need at least 50 menu items in the Access Menu.

In the example I gave you above, each access path (at least Levels 3 and 4, anyway), might be used by more than one person. For instance, secretaries might be divided into Level 3 and Level 4 users on this scheme; Level 2 will be assigned only to the Day Manager and the Night Manager; and Level 1 assigned only to one person, the Owner. In this case, the Owner would be the one who assigns Access Level values to the various people in the company, doing this by editing Field 3 of each person's User ID Panel record. Note that on this scheme, only Level 1 (the Owner) has access to the User ID Panel.

[For an example of the above technique, load UD3.STR. When asked for a User ID, use JOE. When asked for a Password, use COMMON. Note the menu you land on. Now exit the application and re-enter, using SUE for the User ID and SPECIAL for the Password. Note you land on a different menu.]

The User ID Panel vis a vis Application Passwords

Now, remember I said that if the User ID Panel facility is active, and both Definer and Supervisor application passwords are defined, no one can get into the database without answering the User ID and Password prompts properly. Two important points come to mind around this, however.

First, what happens if the User ID Panel facility is active but either the Definer or the Supervisor application password isn't defined? Well, the user will still be prompted for his User ID and Password. And if he answers appropriately, he'll still be dropped into the submenu that corresponds with his Access Level. But if he fails to answer properly, either by entering a User ID and Password that don't correspond to a User ID Panel record, or by just hitting **Enter** at either prompt, leaving one or both blank, DataPerfect will place him in the initial menu instead of a submenu! This gives him a choice as to which submenu to access, which could spell disaster if this is a user who should only have limited rights (like Read Only).

Second, if the User ID Panel facility is active and both application passwords are defined, then, again, nobody gets into the database without properly answering the User ID and Password prompts. If they answer improperly, they're kicked out to DOS. That's what we want.

But what if the Supervisor has edited all the User ID Panel records and doesn't remember any of them. He calls you, telling you he can't get into the database. How do *you*, the *definer*, get in? Well, as long as both Definer and Supervisor application passwords are defined, the User ID facility offers the definer a backdoor into the database. (Don't forget that anybody can get in without those application passwords defined, so this issue only arises when they're both defined, which they *should* be.) With both Definer and Supervisor application passwords defined, you just hit **Enter** for either or both User ID and Password prompts (leaving one or both blank). DataPerfect will then prompt you for an application password (only Definer or Supervisor passwords will work at this point). After you give DataPerfect your Definer application password, DataPerfect drops you into the initial Access Menu. If you enter the Supervisor application password instead, you'll find yourself in the initial Access Menu as well, but you'll only get into the database with Supervisor privileges (you won't be able to enter Define Panel mode, for instance).

So again, *without both* application passwords defined, *anyone* can get into the Access Menu by simply hitting **Enter** for their User ID and Password. *With* both the Definer and Supervisor application passwords defined, only the definer or Supervisor can get into the application.

[For an example of this point, load UD2.STR. When asked for a User ID, just hit **Enter**. When asked for a Password, just hit **Enter**. Note you land on the hidden initial menu the user is never supposed to see. That's because I didn't define Definer and Supervisor application passwords for this application. Now exit the application and load UD3.STR. Again, just hit **Enter** when asked for User ID and Password. Note that you're now prompted for a password. That's an *application* password prompt. The Definer password is *PASSWORD*. Use that as the password, and you'll land on the hidden initial menu. .]

User-Stamping Records

Suppose you'd like to stamp every record entered in a given panel with the User ID of the user that created the record, and have another field that shows who last edited it. The following A45::N field formula, if set to update *on creation*, would show when a record was created, and by whom:

```
cat.t[
    apply.format["T99:99:99";now];
    " "apply.format["D99/99/9999";today];
    " "user.field[1]
]
```

The same formula, set to update *on any change*, would show who last edited the record, and when. I arrived at the size of the field (45 characters) by adding the space occupied by the following three fields, plus the two spaces between them:

Format	Formula	Size
T99:99:99	NOW	8 characters
D99/99/9999	TODAY	10 characters
A25	USER.FIELD[1]	25 characters

Controlling Access to Data Accessed from a Menu

Suppose you'd like to keep the wrong pair of eyes from seeing the contents of a particular Read Only panel (perhaps it's a panel with information you, the developer, filled in for the Supervisor—like application maintenance notes). Use the USER.FIELD[n] function for this. Say you reserve User ID Panel Field 4, formatted G9, for assigning a user the right to see data in, say, Panel 10. You can do this by giving the user a 1 in field 4 if they're allowed to view Panel 10 data, and a 0 if not. Then create a special hidden G9 field in Panel 10, where that field updates to 1 on record creation. Next, create an index in Panel 10 that has this new field as its first field.

Now grant Read Only access to that panel via a menu option tied to a User ID Panel Field 4. This is done by choosing the first Go To Panel option from the Define Menu menu below:

```

Define Menu
Options 1-7 Create New Menu Entries.
1 - Go to Panel (Definer Keyword)
2 - Go to Panel (User Selects Record)
3 - Run Report
4 - Go to Panel List
5 - Go to Report List
6 - Submenu
7 - Launch Shell Macro
8 - Edit an Existing Entry
9 - Delete an Existing Entry
A - Create/Edit Menu Text
B - Move the Menu Prompt
Selection: 0

```

When you choose 1, and tell DataPerfect which menu item and panel it involves, you get the following menu:

```

Go to Panel (Definer Keyword)
Panel:
1 - Panel Access Rights: Read/Write
2 - Access Report List? Yes
3 - Restrict Modification to First Level No
4 - Edit Password 6 - Choose Index 0
5 - Edit Key Word 7 - Subset No
Selection: 0

```

Note option 5 (Edit Keyword). Choosing that option gives you the following prompt:

Select a field from the User ID panel for the keyword (Y/N)?

Choosing *Y* presents you with the User ID Panel, where you select Field 4 with **F4**. That will be the Keyword for this menu option's entry into Panel 10.

What you just did is assign a Keyword to a Go To Panel menu item. Such a menu item will work just like a panel link with a single field on its field list. The Keyword on the Go To Panel menu item is equivalent to a panel link's single-field field list. So when the user chooses that Go To Panel menu item, he'll enter its target panel just as if he went through a panel link with that Keyword as its single-field field list.

If you assign this menu option Read Only access to Panel 10, then any attempt to enter Panel 10 with this option will turn the user away if they don't have a 1 in their User ID Panel record's Field 4. Why? Because every record created in Panel 10 has a 1 in the first field of the index assigned to our new menu option. If the user has anything other than a 1 in their User ID Panel record's Field 4, they don't see any record in Panel 10 when they choose this menu item. That's because this menu item looks at the current user's User ID Panel record's Field 4 and uses that as the Keyword to gain entry into Panel 10.

Controlled Panel Link Access to Subrecords

Similarly, the USER.FIELD[n] function can be placed on the field list of panel link that accesses Panel 10 described above, assigning the same index to that link. But how do we get that panel link to allow only certain users access to Panel 10 records, the way we did with the Go To Panel menu item that used a Keyword?

Well, don't forget that, as of DataPerfect 2.3's second release (September 1993), you can put computed fields on panel link field lists. There's no reason you can't use the USER.FIELD[n] function in such computed fields. In the panel that has the panel link (the panel link that accesses Panel 10), we'll put a hidden computed G9 field that updates to USER.FIELD[4]. We then put only that field on the field list of our panel link. Now our panel link gives access only to Panel 10 records that have the same value in their special field that is found in the current user's User ID Panel record's Field 4. But since all Panel 10 records have 1 in that field, this effectively gives the current user access to Panel 10 records only if the current user's User ID Panel's Field 4 is 1.

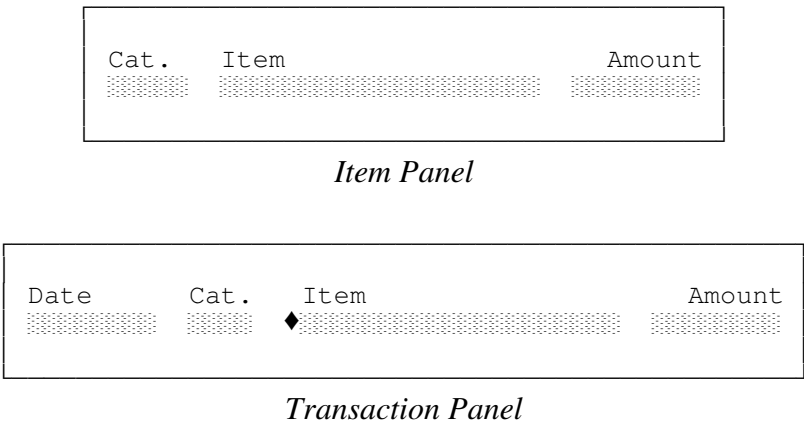
Note that *wrong* users (those with value 0 in their User ID Panel record's Field 4) won't even be able to *view* records in the subpanel via an **F8** dependent record Browse mode lookup at the panel link this way, because no subrecords exist with value 0 in the special field. Further, if there's a Read Only flag on the Go To Panel menu item that takes you to the panel with the panel link, then access to all of its subpanels (Panel 10, for instance) will be Read Only too.

As you'll later see, this is all much easier with DPMouse©, which allows the developer to keep users out of any field or panel link very simply, independent of any

rights assigned via a Go To Panel menu item. DPMouse© can be used to keep this or that user out of this or that field or panel link, using the USER.FIELD[n] function in a much more straightforward way (see *DPMouse© and Field Protection* and *Using DPMouse© To Conditionally Close A Panel Link* in my **Securing Data Entry** chapter).

Data Link Subgroup Lookups and the USER.FIELD[n] Function

The USER.FIELD[n] function offers the Data Link Subgroup Lookup a particularly powerful security option (I discuss the Data Link Subgroup Lookup in *Data Link Subgroup Lookups* in my **Links** chapter). Here are the two panels from the Software Store application discussed in my **Links** chapter (in the *Data Link Subgroup Lookups* section):



Suppose we want to restrict access to certain Items based on a value found in the user's User ID Panel record. Let's use User ID Panel Field 4 again for this, formatted G9. Our simple scheme for this will be that the user can see the *special* Items if the user has 1 in that field, otherwise they can't.

To accomplish this, we put a special G9 field in the Item Panel. Let's call it the Special Item field. Only the Supervisor will have access to that panel. For each item in the Item Panel, the Supervisor will create a record with a 1 in the Special Item G9 field. For every item that *all* users may access, the Supervisor will create yet another record, but this one will have a 0 in the Special Item G9 field; otherwise it's identical to it's sister record with a 1 in that field. So an item that *all* may access has *two* records in the Item Panel (one with a 0 in Special Item G9 field, and one with a 1 in that field), and an item that only *VIPs* may access has only *one* record in the Item Panel (with a 1 in the Special Item G9 field).

Now we put a hidden G9::C field in the Transaction Panel, coded with the following simple formula:

```
user.field[4]
```

Next we put the Item Panel's Special Item G9 field at the head of least one of the Item Panel indexes, where it's followed by Category and Item fields. That index will be

used by the data link in the Transaction Panel. That data link will have a field list consisting of the hidden G9::C field (the one that holds the current user's User ID Panel Field 4 value), followed by the Category field and the Item field.

During Create or Edit mode, a lookup on the Item field in the Transaction Panel will display Items in the targeted Item Panel consistent with the current user's User ID Panel Field 4 value. If that value is 0, they see only *nonspecial* items in the Item Panel to choose from. If it's 1, they see *all* items in the Item Panel to choose from. You'll need to read *Data Link Subgroup Lookups* in my **Links** chapter if this isn't clear.

[For an example of the above technique, load UD2.STR or UD3.STR. When prompted for User ID, use *JOE*. When prompted for Password, use *COMMON*. Note what Items Joe is allowed to pick from while in Create or Edit mode. Then exit the application and re-enter under the User ID *SUE* and Password *SPECIAL*. Note that Sue has more of a selection. The difference between UD2.STR and UD3.STR is discussed at the end of the prior section called *The User ID Panel vis a vis Application Passwords*.]

A Note on Deselecting the User ID Panel

It's not all that obvious how to deselect the User ID Panel. To do this, **Shift-F9, B**, followed by either **F1**, **F7**, or **F10**. Then reload the application to make sure you're not prompted for your User ID and Password. There's no onscreen notice that the User ID Panel facility is active or inactive.

Troubleshooting Menus

Problem

You just made many changes to a report, but when you run it from its menu assignment, it still looks just like the old report.

Solution

Either you made changes to a report assigned to a different menu item, or you made changes to a report on the Report List and forgot to replace the menu report with it.

Problem

You made some changes to your client's application and shipped it to them. They call you up, only to inform you that users occasionally find themselves in the Access Menu instead of their controlled submenu.

Solution

You probably ran DPEXP and DPIMP on that .STR before sending it to the client. If so, you forgot to define both Definer and Supervisor passwords. Application passwords aren't exported by DPEXP.

Problem

You made some changes to your client's application and shipped it to them. They call you up, only to inform you that users no longer are prompted for their User ID and Password.

Solution

Again, you probably ran DPEXP and DPIMP on that .STR before sending it to the client. This time you forgot to reselect the User ID Panel. That selection isn't exported by DPEXP.

This chapter is mainly for the experienced DataPerfect application developer. Beginners should peruse it, but not expect to grasp a lot of it.

Preventing Inadvertent Editing

The simplest way to guard against inadvertent editing of a record with DataPerfect is to call up the Panel Options menu while in Panel Define mode (**Alt-F8** to get to Panel Define mode and **Alt-F8** again to get the Panel Options menu:

```
Panel Options
1 - Edit Filename          5 - Change Edit Order
2 - Change Color          6 - Edit Panel Name
3 - Auto-Save (Y/N)       7 - Recompute Field Offsets
4 - Auto-Display Record (Y/N) 8 - Auto-Edit/Auto-Create/Menu
Browse Change => Auto-Edit
Display each record during Lookup.
Selection: 0
```

Note the line just under option 4:

```
Browse Change => Auto-Edit
```

That line is the default value for that menu's option 8. Hitting option 8 toggles between the following three possibilities:

```
Browse Change => Auto-Edit
Browse Change => Auto-Create
Browse Change => Menu
```

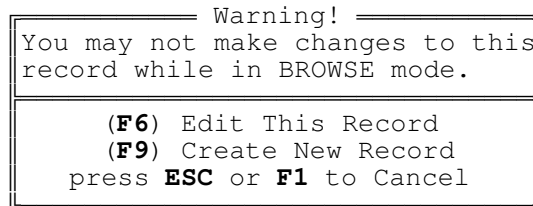
What this is doing here is determining how DataPerfect will respond to the user who, while in Browse mode, hits a key that would normally insert a character in a field if they were in Create or Edit mode. The first option is the default. With that active, DataPerfect will automatically put the user into Edit mode with the touch of a character key. With the second option active, they'll go right into Create mode when they do this. And with the third mode active, DataPerfect will present them with the following menu after they hit a character key (or a series of character keys) and hit **Enter**:

```
Note
You have changed a field in this record.

Do you want to
1 - Create a New Record
2 - Edit the Displayed Record
0 - Cancel the Change
Selection: 0
```

The DPMouse© Alternative

DPMouse© offers a better solution to preventing inadvertent editing. If DPMouse© is loaded with its **X** run-time flag, and the display is in Browse mode, hitting a key that would alter the value of a field produces the following popup:



I prefer this over the way native DataPerfect works in this situation because this popup appears as soon as the user hits a character key, instead of waiting for them to hit **Enter**. It also keeps them in the **F6 is Edit** and **F9 is Create** way of thinking. The developer can override the **X** flag in any given panel by placing an ASCII 240 symbol (≡) in the upper left corner of the panel. As long as that symbol is displayed (either via panel text or, say, a computed field) the application responds in the native DataPerfect manner when the user inadvertently edits a field in that panel in Browse mode.

Keeping a Saved Field from Being Edited

[Refer to UD.STR for this.

Find the *Keeping Saved Field From Being Edited* panel.

Notice that the Customer ID field won't change after saving its data.]

Here you want the user to be able to create a record, but after saving it, you want him not to be able to alter a particular field's value. For instance, this may be a key field, like an account number. Perhaps you wanted him to be able to choose an account number on record creation, but not alter it after it's saved.

This sort of protection is on the *field* level, not the *record* level. If you wanted to keep the user from editing a saved *record*, you could control entry into that panel by having the user get there via a menu item that has *Create OK - No Edit/Delete* as its Panel Access Rights. But you're interested here in protecting a *field* from editing, not a *record*. This can be done as follows.

Say the field you want to protect is P1F1. Create a new hidden field, say P1F2. Give it the same format as P1F1. Code P1F1 to update to P1F2 *on any change*. Code P1F2 to update to P1F1 *on create*.

Protecting P1F1 From Being Edited After Save			
Field	Format	Formula	Update Condition
P1F1	same as P1F2, but editable	P1F2	on any change
P1F2	same as P1F1, but hidden	P1F2	on creation

When the user goes into *Create* mode, both P1F1 and P1F2 are blank. When he places a value in P1F1, P1F2 stores it because he's still in Create mode. On Save, P1F1 updates to the value in P1F2, which is the value the user put in P1F1. In *Edit* mode, P1F1 will look at P1F2 for its value and ignores anything the user puts in P1F1 itself. P1F2 will never change from user input after creation because it's hidden, so P1F1 never will.

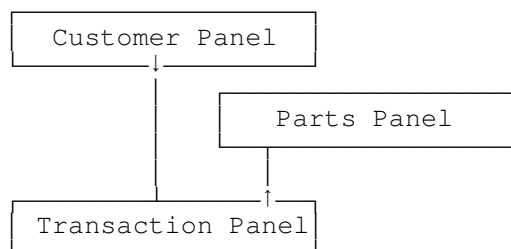
As you'll find out later, DPMouse© makes this much easier. With DPMouse© installed, the user can't edit a field in a saved record if an ASCII 250 (·) displays next to that field. So you can just put that character right next to the field, or have a computed field display it under certain conditions (perhaps depending on the value in a specific field in the user's User ID record). This is much easier than under native DataPerfect, because it doesn't require an additional field (hidden P1F2, above).

Field and Record Protection

Controlling Data Entry: The Pick List Field

Consider a typical way the data link is used: to turn a common data entry field—a field into which a user can manually enter data—into what I call a *pick list field*. Here we attach a data link to the field to get the user to select the field's value from a lookup display of records in a foreign (linked) panel during Edit or Create mode.

Now, though the user is confined to picking records from the linked panel, suppose you also want to keep him from *tampering* with those records. For instance, consider an application that tracks purchases in an auto parts store. In its Transaction Panel, you might want a Part Number field, formatted G9999, with a data link linking it to the Parts Panel. Of course you'd link it via the Part Number field in each of the two panels:



In the above diagram, the Down Arrow represents a panel link and the Up Arrow represents a data link.

Now, suppose you consider the Parts Panel records to be too important for anyone other than the application's Supervisor to touch. Let's discuss how to keep someone out of a panel in such a situation, preventing them from not only *modifying* records there, but also from *creating* them.

Closing Off a Data Link to Protect a Pick List Panel

DataPerfect 2.3 offers the definer the ability to keep the user from creating records in the linked panel after penetrating that data link. You do this by assigning the No Create option to the data-linked field while in Define Panel mode (**Alt-F8**). To do that in Define Panel mode, cursor to the field in data-linked field and hit **F5**. Toggle **4** until you see the

```
Do not allow user to create related record if not found
```

message in the menu, just under option 4:

```
Define Data Link for Field
Link to Panel:3 Field1 Index1 Field List to Build Key:1
 1 - Edit Target Field/Target Index/Field List
 2 - Remove Data Link          5 - Prompt-Create
 3 - Auto-Create              6 - Check During Data Entry Off
 4 - No Access                7 - Cascade Off
Do not allow user to create related record if not found.
Selection: 0
```

If you want to protect the linked panel further, toggle **4** until you see the

```
No Access
```

message just under option 4:

```
2 - Remove Data Link
3 - Auto-Create
4 - No-Create
No Access
Selection: 0
```

The No Access status prevents the user from even landing the linked panel via the data link.

Now we have a way to keep the user from accessing the Parts Panel via a data link. But, even if we take the added precaution of never displaying a panel link that takes the user to the Parts Panel, he still can access that panel from the Panel List. So let's see how to keep him out of that as well.

Closing off Access to the Panel List

Until version 2.3, DataPerfect application developers had to rely solely on DPMouse© (it was called DataMate then) to keep the user out of the Panel List,

using DPMouse's **N** command line switch. DataPerfect 2.3 offers two ways to keep the user out of the Panel List.

The /Z Command Line Switch

DataPerfect 2.3 introduced a command line switch that behaves similarly to the DPMouse© switch: **/Z=n**. It works a little differently than DPMouse's switch, however. If the .STR file for our Auto Parts application is AP.STR, then loading it with

```
dp app /z=1
```

will start the application by placing the user directly into Panel 1 (or, if a menu has been defined, it will automatically select menu item 1), skirting the Panel List. It will take the user back to DOS when he exits that panel (or menu) with **F7**, again skirting the Panel List. The user never sees the Panel List this way.

Of course if you use the **/Z=n** switch you'll need to design your application so that the user never need access the Panel List. This would amount to making sure that, for each panel you *want* the user to be able to access, there's some path to that panel via menus or links.

Menus

DataPerfect 2.3 introduced a much more important alternative to the **/Z** command line switch: menus. With a menu defined for an application, the user doesn't see the Panel List upon loading the application. And as long as the definer never gives the user access to the Panel List from the initial menu or any of its submenus, the user will never see the Panel List. I discuss the DataPerfect 2.3 menu facility in *Menus* in my **Securing the Application** chapter.

Controlling Data Entry: Initial Formula or Value

Besides the data link, another fundamental way DataPerfect allows you to develop an application with a certain level of data entry consistency is to let you attach an Initial Formula or Initial Value to a field. To do this with a panel displayed, you can either be in Browse mode or Define mode (**Alt-F8**). With the cursor on the field in question, **Shift-F8**. You'll get one of the following menu screens:

Options for Alphanumeric and Text Fields	
1 - Lookup Field List	5 - Range Check
2 - Initial Formula	6 - Validation Time (Edit/Save)
3 - Initial Value	7 - Define Search Field List
4 - Initialize at Create/Save/Change	0 - Exit

Selection: 0

Field Options for Computed Fields
1 - Lookup Field List
2 - Define Field Formula
0 - Exit

The formula value will be computed whenever displayed or accessed.

Selection: 0

The first menu screen above is what you get if you're on a real field. The second one, when you're on a computed field. As mentioned elsewhere, a real field is any field that isn't computed (::C). It's called real because it stores its data on disk, in either its panel's data file or, if it's a variable-length text field, the .TXX file. Computed fields don't store data anywhere. They're just screen entities that *display* data on the fly.

For our purposes, of course, we're concerned with the first menu screen above because we're attempting to control consistency of data *entry*. They can't do this in a computed field. So your choices in the first menu screen are 2 and 3:

```
2 - Initial Formula
3 - Initial Value
```

After defining the formula or value, **F7** back to the above menu and toggle the update conditions for this field by hitting **4** (Initialize at Create/Save/Change) enough times until the following displays at the bottom of the Field Options Menu screen:

```
Automatically computed when record is created.
```

That's easy enough. We all frequently use this feature in developing our DataPerfect applications. But suppose we want the *user* to be able to change the way a field will initialize, without letting the user touch that field's formula with **Shift-F8**, **2**. Let's discuss how to do this with a *default panel*.

Controlling Data Entry: The Basic Default Panel

[Refer to UD.STR for this.
Find the *Default Schemes*, *Hidden Panel*, & *Undelete* series.
Load the Master Panel.
Penetrate the panel link to the Client Panel.
Then penetrate the panel link to the Transaction Panel.
Examine the behavior of the Date field.]

In the billing application I use to run my practice, I don't want the user to be able to manually enter the date of a transaction each time he enters a record in the Transaction Panel. Rather, *immediately after loading the application*, I have the user set the Working Date for all transactions he might create in the Transaction Panel during this data entry session. After he does this (I'll show you how to set this up in a few moments), then, anytime he's in Create mode in the Transaction Panel, he finds the Date field has the user-defined Working Date already inserted. The Date field in the Transaction Panel is a non-updatable field (::N), so the user can't enter a value in it.

With this scheme, the user never touches the Date field in the Transaction Panel. It's assumed all Transaction Panel records he creates in this data entry session will have the same date. If he wants to change the Working Date during this data entry session, he must somehow go back to the panel that allows him to set it (perhaps via a menu option), and edit the Working Date field there. All in all, this

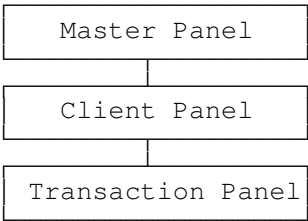
tends to severely limit the potential for typing in an incorrect date in the all important Transaction Panel, while speeding up data entry.

You might build this concept into a much simpler application by designing the first panel the user sees like this:

<u>MASTER PANEL</u>	
Working Date: 02/04/1993	D99/99/9999
==> THIS IS NOT TODAY'S DATE! <==	A33::C
To Clients: [icon]	Panel Link

Upon initial entry into the application, at the level of the Master Panel, the user sees the above displayed record. Note the bolded computed comment just below the Working Date, telling the user that the Working Date isn't current. If the Working Date *is* current, the bolded computed comment field will be blank. If the user desires, he can reset the Working Date by simply editing the Working Date field.

The Working Date comes into play in the Transaction Panel, deeper in the following fairly generic panel hierarchy:



At the level of the Transaction Panel, the Transaction Date field will initialize to the value found in the Working Date field in the Master Panel. It does this using a hidden panel link in the Transaction Panel. This protects that date field from sloppy data entry. I prefer this Transaction Date field in the Transaction Panel be non-updatable (::N), but you might consider letting it be editable. In any event, it initializes to the value found in the Master Panel's Working Date field.

This gives the user control over the initial value of a field without allowing them to touch that field's formula, even if that field is non-updatable.

Controlling Data Entry: The Complex Default Panel

[Refer to UD.STR for this.
Find the *Default Schemes, Hidden Panel, & Undelete* series.
Load the Master Panel.
Penetrate the panel link to the Client Panel.
Then penetrate the panel link to the Transaction Panel.
Examine the behavior of the Code, Description and Amount fields.]

Suppose the business for which you're designing a DataPerfect application is such that each client or customer typically enters into the same set of transactions on each visit, and you'd like this application to *remember* the default transactions for each client. This is a little more complicated than the Working Date scheme mentioned above because it involves having more than one default transaction per client, each to be automatically created in the right order. Let's explain with an example.

I suppose an accountant might want to set up a billing application which allows for different default transactions per client. For instance, because I only see my accountant once a year, he might want his billing application to *remember* that I typically incur three specific charges:

1. Annual Tax Consultation, 30 minutes
2. Preparation of Annual Tax Forms
3. Preparation of Annual Profit/Loss Statement

Alternatively, I assume other clients burden him with more significant requests, like getting financial advise quarterly. So my accountant might want his billing application to remember that, for instance, Joe Investor typically incurs the following charges:

1. Quarterly Consultation, 60 minutes
2. Preparation of Quarterly Profit/Loss Statement
3. Preparation of Quarterly Portfolio Analysis

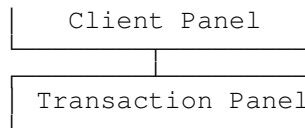
The object here is to get the Transaction Panel for the above application to do the following for each client, in this order:

- With no records with today's date, hitting **F9** will initialize to the client's *first* default Transaction.
- Hitting **F9** again will save the first default Transaction and initialize to the *second* default Transaction for that client.
- Hitting **F9** a third time will save the second default Transaction and initialize to the *third* default Transaction.
- Hitting **F10** or **F7** will save the third default Transaction (though hitting **F7** will also exit the Transaction Panel).

Briefly put, if the accountant merely wants to enter Steve Smith's three default Transactions, and he knows he hasn't entered any Transactions for Mr. Smith with today's date, I want him to be able to simply tap **F9** three times, then **F7**. That's it. No thinking and no room for sloppy data entry. Now on to his next client's data input.

That's what I want the accountancy application to do. Now let's see how to set this up. We start with the generic panel hierarchy again:

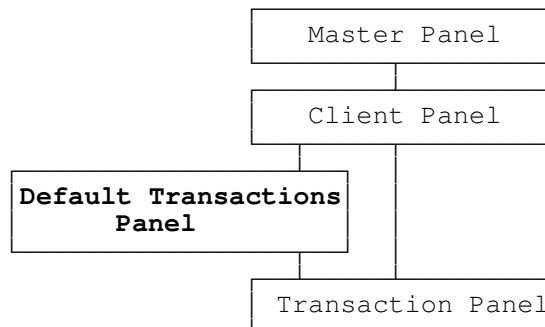




To deal with multiple default transactions per client, we need a Default Transactions Panel, which we'll set up as a subpanel under the Client Panel. I could place the fields for default transactions in the Client Panel itself, instead of as fields of a subpanel of the Client Panel, but want to keep this data from cluttering the Client Panel. Either way, the logic I express here will equally apply. It might look something like this, with formats displayed here in place of fields:

<u>DEFAULT TRANSACTIONS PANEL</u>		
Client No.		Client Name
G9999::N		A30::C
	Code	Description Charge
1.	U10	A30 GZZZ9.99
2.	U10	A30 GZZZ9.99
3.	U10	A30 GZZZ9.99
4.	U10	A30 GZZZ9.99

The above allows for up to four Default Transactions per client, and would fall in the panel hierarchy something like this:



The user fills in the Default Transactions as a subrecord of the Client Panel. Later, when it comes time to enter Transactions for this client, the user goes directly from the Client Panel to the Transaction Panel to do that. The only reason they'd ever go back to that Client's Default Transaction Panel would be to edit those defaults for future Transactions. If this is all set up correctly, data entry in the Transaction Panel will grab the defaults from that Client's Default Transaction Panel's record, probably using a hidden panel link for this.

Two questions come to mind:

1. How do we get the Transaction Panel to start over with Default Transaction #1 with each new date, and never repeat a Default Transaction on any given date? That is, how do we get the

Transaction Panel to know what's already been entered for that date and grab Default Transactions accordingly?

2. How do we get the Transactions Panel to grab these Default Transactions in the correct order?

To solve question 1, we need to add a field to the Transaction Panel: the Transaction Number field. This will be the number of the Transaction Panel record for a given Client on a given Date. Assuming no Client will have more than 99 transactions on a given date, we can format this field G99::N. Our Transaction Panel might look something like this:

TRANSACTION PANEL				
Client No.	Client Name			
G9999::N	A30			
Date	Tr#	TrCode	Description	Charge
D99/99/9999	G99::N	U5	A30	GZZZ9.99

Next, we need an index in the Transaction Panel that sorts records in reverse order with respect to the record's Transaction Number for that date. To do that, we add a Negative Transaction Number field to the panel. If the Transaction Number field is P4F2, then the Negative Transaction Number field would be formatted G-99::H, and formulated to update on any change to

-P2F2

We can now create the following two indexes for the Transaction Panel:

Forward Transaction Number Index:

Client No., Transaction Date, Transaction No.

Reverse Transaction Number Index:

Client No., Transaction Date, Negative Transaction No.

The Transaction Panel will have a recursive panel link (see *The Recursive Panel Link* in my **Links** chapter) that uses the Reverse Transaction Number index, and have the following field list:

Client Number, Transaction Date

Such a recursive panel link will have access to all and only those Transaction Panel records with the Client Number and Transaction Date of the currently displayed Transaction Panel record, and will access those records in reverse Transaction Number order (landing on the highest Transaction Number for that Client on that Date).

We now design a field formula for the Transaction Number field in the Transaction Panel. This formula will use the Transaction Panel recursive panel link, by looking through that link and adding 1 to the Transaction Number it sees there. If the Transaction Number field is P4F2 and the recursive Transaction Panel panel link is P4F4, then the formula for the Transaction Number field would be

P4F4P4F2 + 1

to update on creation. Now our Transaction Number field increments conditionally, starting over with 01 for each Client-Day.

Now question 2: How do we get the Transaction Panel to grab these Default Transactions in the correct order? To do this, we need the relevant field or fields in the Transaction Panel to initialize based on a routine that first looks at the Transaction Number, which will have just incremented to the proper number. Then, based on the Transaction Number it sees, it makes its decision as to which Default Transaction to grab in the Default Transactions Panel.

First, we need a new hidden panel link in the Transaction Panel. Let's say it's P4F5. It will access the Default Panel record for that particular Client Number. That's easy enough. Just make sure you have a Default Panel index that has only the Client Number field in it (you only want one set of defaults per Client Number). Then assign that index to this new hidden Transaction Panel panel link, and put only the Client Number field on that panel link's field list. This panel link takes the user to the current Client's single record in the Default Transactions Panel.

Now, take another look at our Default Transactions Panel:

DEFAULT TRANSACTIONS PANEL		
Client No. G9999::N	Client Name A30::C	
Code	Description	Charge
1. P3F1	A30	GZZZ9.99
2. P3F2	A30	GZZZ9.99
3. P3F3	A30	GZZZ9.99
4. P3F4	A30	GZZZ9.99

A CASES statement easily updates the Code field in the Transaction Panel:

```
P4F2 cases
  case cv=1 of P4F5P3F1 endof
  case cv=2 of P4F5P3F2 endof
  case cv=3 of P4F5P3F3 endof
  case cv=4 of P4F5P3F4 endof
endcases
```

The above CASES statement would update the Code field in the Transaction Panel on any change. Again, P4F2 is the Transaction Number field, and P4F5 is the new hidden panel link that accesses the Default Transactions Panel from the Transaction Panel.

Now our Transaction Panel will initialize an ordered series of default records as spelled out in the associated Default Transactions Panel for that Client, and it will do so with respect to that particular Client-Day.

Like our more basic default panel schema, this schema effectively allows the user to alter the output of a field-initializing formula without requiring the user touch that field's formula with **Shift-F8, 2**, even if that field is non-updatable. But this schema also allows the user to create *multiple defaults per client*, and to determine the order these defaults initialize.

Controlling Data Entry with ZipKey©

If you plan on creating an application that requires frequent data entry of Zip Codes, do yourself a favor. Forget about creating a separate panel of Zip Codes that other panels access via a data link. That approach works, but it's a tremendous waste of time, upkeep, and disk space.

Instead, take a serious look at the shareware copy of ZipKey© I included on your diskette (ZIPKEY.ZIP). ZipKey© is a TSR that allows your end user to simply type in a Zip Code and hit a definer-configured hotkey. What ZipKey© does at this point depends on what you, the definer, configured it to do. I have it configured to do this if the user enters a Zip Code in the City field (not the Zip Code field) and then hits **F11** (a key DataPerfect doesn't use):

- ZipKey© examines and then erases the Zip Code the user just typed.
- ZipKey© types in the appropriate City name.
- ZipKey© **Tabs** to the next field and fills in the appropriate two-character State Code.
- ZipKey© **Tabs** to the next field and fills in the Zip Code.

You can configure it to do all that differently (e.g., different field order, or insert the entire State instead of the State Code, etc.). It's very configurable. You save its configuration to a file, so you can have different configurations for different applications, or one for your DataPerfect applications and one for, say, your word processor.

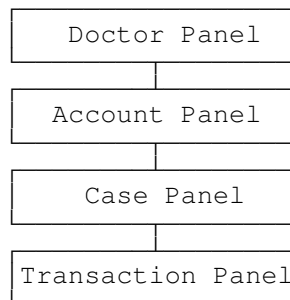
I use the same configuration with all my DataPerfect applications, so I just make sure I configure address fields the same way in every panel to conform to the four-step pattern you see above. That is, I always create a fixed-length City field first that doesn't have an ::E (Auto Entry) modifier, followed by a U2 State Code field that doesn't have an ::E modifier, followed by a N99999-9999 Zip Code field. When ZipKey© finishes the above four steps, the user's cursor sits in the N99999-9999 Zip Code field's sixth digit slot, waiting for the user to fill in the right four digits if he knows them.

ZipKey© does all this with a compressed database of Zip Codes that you can have shipped to you as often as you need to be current. I get a copy of the latest database once a year for \$25 each time. I can imagine some mailing list applications might need this updated more often. Because the ZipKey© database is compressed, it takes less space than the same database would be as a DataPerfect data file. And,

instead of importing that latest group of tens of thousands of Zip Codes into a DataPerfect panel each year, I'd much rather pay ZipKey's author a nominal fee for a file I simply copy to my hard drive and be done with it.

Pyramidal Design as a Data Integrity Strategy

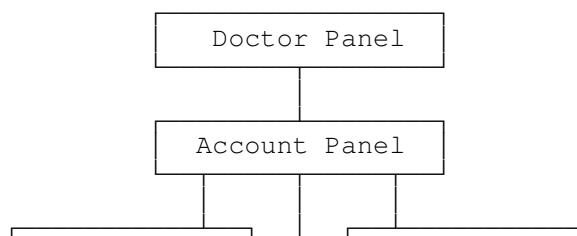
How you design your panel hierarchy can greatly determine how the user will enter data. The following is the axial structure of the panel hierarchy of the billing application I use to run my practice, upon which I place other secondary panels as limbs:

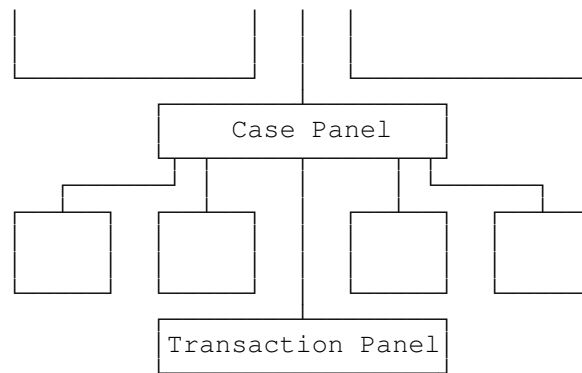


With the above panel hierarchy, I make sure the user picks the appropriate Doctor, then the appropriate Account, and finally the appropriate Case for that Account, all before entering even one Transaction for that patient. This way the user must make a conscious decision about which Doctor, Account, and Case the subsequent Transactions apply to. This schema also makes the user aware of certain information about that Account and Case before entering the Transaction Panel (like whether or not the patient is a minor, or hasn't given supplied some required personal information for their file).

I could have—as I had originally planned—allowed the user to initially enter the application via the Transaction Panel. Through the use of data-linked fields in the Transaction Panel, the user would choose the appropriate Doctor, Account, and Case for each Transaction entered. And once the first Transaction is entered for that Case, the user could repeatedly use **F4** to keep filling in the Doctor, Account, and Case fields for subsequent Transactions during that session for that Case.

But this fails to keep the user in a certain logical flow that I feel is needed in this type of office setting. A lot of data is entered in the formation of a new Account, or a new Case for an existing Account. Most of that information is found in multiple subpanel *extensions*, attached to the Account Panel and the Case Panel:





Also, if I set this application up so the user would routinely enter the database via the Transaction Panel, an unavoidable confusion would arise when a data link targets a panel that contains one or more panel links. Suppose the user is sitting in the Transaction Panel in Browse mode and sees a data link on the Case Number field. If he penetrates that data link with **Down Arrow**, he finds himself in the Case Panel with many panel links to tempt his curiosity or careless fingers. If he then hits **Down Arrow** on the panel link leading to the Transaction Panel, he'll find himself back in that panel; perhaps without realizing he's on a different linkage depth than he was on when he began this panel-hopping journey in the Transaction Panel.

There are ways around this, without forcing the user to enter the panel hierarchy at the top of the pyramid. One way is to put the No Access option on these Transaction Panel data links. That effectively keeps the user out of higher panels when in the Transaction Panel. But this means they won't be able to create a new Account or Case while entering data in the Transaction Panel. Rather, they'll have to enter the panel hierarchy from above to create a new Account or Case. This means they have one menu option that allows them to create an Account, one that allows them to create a Case, and another that allows them to create Transactions.

Also, this No Access data link workaround prevents the user from taking a look at relevant information about the person's Account or Case before entering a Transaction. Alternatively, requiring the user enter the application at the *top of the pyramid* forces him to see relevant information before entering a Transaction. And when the user finally lands in the Transaction Panel, DataPerfect automatically fills in the Doctor, Account, and Case fields. At that level of the hierarchy, these fields are non-updatable (:N), leaving no room for sloppy data entry there.

So, what I call the pyramidal panel hierarchy revolves around an axial structure that favors the panel link over the data link for data entry. This isn't to say that I don't use data links for data entry in that structure. Rather, I favor forcing the user to choose the parent *while in the parent panel* whenever design considerations allow it.

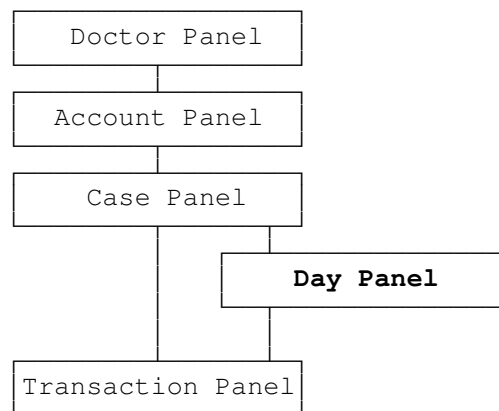
In general, the pyramidal hierarchy is especially suited to database situations where the user will, each session, typically enter multiple child records for a given parent—perhaps many transactions for a client, or songs on a record—and for situations where the user needs to see certain critical information before selecting the appropriate parent record.

Consider my billing application again. While still under treatment for an injury, a patient may, after getting off the phone with his ex-wife, fall down some stairs. Now, after the first treatment for the *new* injury, he has two Cases with positive balances. Next week he sends me a check. If I want to apply it to his oldest balance—or perhaps his smallest balance, or whatever—I want to see which Cases have balances and pick one from a lookup. After picking the appropriate Case, I can then see if there's anything I should know before penetrating a panel link to the Transaction Panel. Let's say the Case Panel memo field tells me his ex-wife is paying for this Case. Now I know not to apply this check to that Case and pick the other one with a positive balance. We're clearly better off handling this sort of scenario with a pyramidal panel hierarchy.

The Hidden Panel

Through the use of what I call *hidden panels*, the pyramidal panel hierarchy schema allows for further protection against aberrant record creation. I consider a panel *hidden* if the application design prevents the user from ever seeing it. Before outlining how to hide a panel, let's go over why you'd want to do that at all.

In the my billing application I have a hidden Day Panel that fits in the panel hierarchy like this:



The Day Panel holds totals in each Doctor-Account-Case-Day record, receiving those totals through the Keep A Total facility when records are entered in the Transaction Panel. That is, each record in the Day Panel shows the balance for a given Day for a given Case. So, on any given Day, a Case has an Account balance, a Case balance, and a Day balance, and all three can be different. The Account balance is the accumulated balance from all Transactions for all its Cases. The Case balance is the accumulated balance from all Transactions for that particular Case. And the Day balance is the balance from all Transactions that took place that Day for that Case.

I want the Day Panel in the panel hierarchy for various reasons, though the two major ones are the following:

1. It makes it easy to create reports that filter or group records by date.

2. It allows for, in the Transaction Panel, a dynamic computed field display of a particular day's total for a given Case.

Flowing from 1 above, I put many special panel links in the Day Panel—panel links I use in report definitions. Next to each link is panel text reminding me how reports use the link, like the following:

```
This panel link sees only payments in the
Transaction Panel, based on Exception List
Index 4
```

Regarding point 2, when the user enters a couple records in the Transaction Panel for a particular patient, he can see the real-time total of that Day's charges displayed in a computed field in the Transaction Panel without having to exit that panel to see that information in the panel holding the actual total (the Day Panel). This serves as a sort of calculator for the front desk at check-writing time, showing a real-time display of what the patient owes for today's services. Also displayed in the Transaction panel (via computed fields) are the totals for the Account and the Case.

Here's an abbreviated version of the Transaction Panel in that application:

TRANSACTION PANEL					
Jane Doe				Acc	300.00
				Cse	120.00
				Day	60.00
Acct 4300 Case 002					
Date	Day	Tr	Description	Charge	Payment
06/13/1996	Thu	1	Office Visit	60.00	

The record above is a single Transaction for Jane Doe. Note that her Account, Case, and Day balances display in the upper right corner. Those three fields are all computed fields, taking their values from the Account Panel, Case Panel, and Day panel, respectively. After each Save in this panel, the user gets to see those three figures dynamically update without having to exit back to any of those three linked panels.

[Refer to UD.STR for this.

Find the *Default Schemes*, *Hidden Panel*, & *Undelete* series.

Load the Master Panel.

Penetrate the panel link to the Client Panel.

Then penetrate the panel link to the Transaction Panel.

Note the fields in the upper right corner of the Transaction Panel.]

Though there are plenty of reasons the user will frequently be in the Account and Case Panels, there's really no good reason for the user to be in the Day Panel, or to even know it exists. So I hide it by making sure no link or menu option gives the user access to it. Also, don't forget that if the appropriate Day Panel record, into which DataPerfect is to carry the total, doesn't exist, DataPerfect automatically creates that record to receive the total (see *When There's No Parent Record* in my **Keep A Total** chapter if this confuses you).

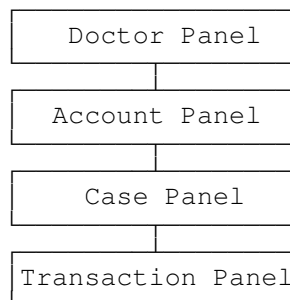
Controlling Record Creation with Indexes

One thing that may come up from time to time is needing to limit a particular panel to a single record, or a single record per parent record. In the case of limiting a panel to a single record, this usually arises when you need a panel to hold a record of system information or parameters, or perhaps application notes for you, the definer. One of the simplest ways to limit a panel to a single record is to hide a G9 field and then create an index that has only that single field on its field list. Now any attempt to create more than one record will be greeted with the

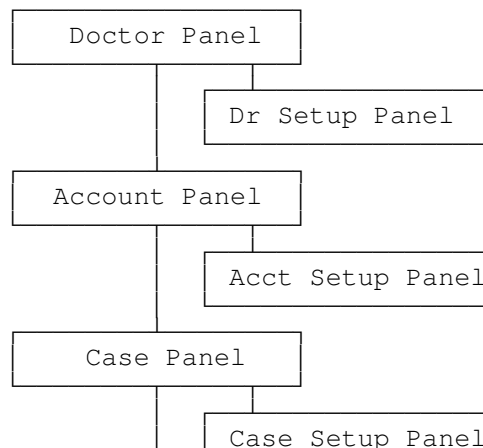
This record is not unique. At least one index key was found to be duplicated in the index(es).

error message because that hidden G9 field will be 0 in every record creation.

A more common reason for limiting a panel's records is to limit it to one subrecord per parent record. Here you typically have what some call an *extension* panel that should have only one subrecord in it. For instance, consider again the billing application I use in my practice, which has the following series of panels that compose its axial hierarchy:



I need some setup information for each of the top three panels. For instance, I have a subrecord for each Doctor in the Doctor Panel, where the user places the Doctor's home and office phones, tax identification number, etc. Likewise, the Account and Case Panels have setup subpanels as well:





Each of these three added subpanels is to be limited to one record per parent record—that is, one Doctor Setup Panel record per Doctor, one Account Setup Panel record per Account, and one Case Setup Panel record per Case. To do this, we can just make sure that at least one index in each subpanel perfectly matches the most limiting index of its parent panel.

For instance, if the Doctor Panel has an index consisting of merely the Doctor Code field, then there should be exactly the same index in the Doctor Setup Panel. If the Account Panel has an index consisting of the Doctor Code and Account Number fields (or maybe just the Account Number field), then there should be such an index in the Account Setup Panel. Likewise, if there's an index in the Case Panel consisting of Doctor Code, Account Number, and Case Number fields (or just Account Number and Case Number fields), there should be such an index in the Case Setup Panel. Whatever makes a record unique in the parent panel should be applied to the subpanel here. This will limit the subpanel to one record per parent record.

Controlling Record Creation with the ::M Field

Here we don't want the user to be able to create a record unless certain information is true or filled out. Perhaps we don't want him to be able to save the record he's attempting to create if his User ID is the wrong value, or he hasn't filled out certain fields in the panel properly. Don't forget that DataPerfect already offers you a way to make sure certain fields are filled out before saving a record. The ::M display modifier will keep the record from being saved if the field it's attached to is a blank text field or a numerical field with a value of zero. But suppose you don't want the record saved if certain fields aren't filled in properly.

One way to handle this, other than with reports which I discuss later in this chapter, is to use a ::M field. Place this field next to the field in question. Let's say we want the Payment Method field filled out if the Payment Amount field has a value other than zero in it. We don't really care about the Payment Method field if the Transaction is a Charge and not a Payment. We might have a panel like this:

TRANSACTION PANEL			
Acct	Description	Charge	Payment
Payment Method		◆	

Here, the Payment Method field has a data link that offers the user a choice of CASH, CHECK, VISA, AMEX, etc., during Create or Edit mode. We don't care if there's a value in that field if this Transaction is a Charge, but do care if it's a Payment.

We'll put our special ::M field to the immediate right of the Payment Method field. Let's use this layout:

<i>Field</i>	<i>Format</i>	<i>Name</i>
P1F1	G99999	Account
P1F2	A20	Description
P1F3	G-ZZZ9.99	Charge
P1F4	G-ZZZ9.99	Payment
P1F5	U5	Payment Method
P1F6	A1::MN	Special ::M field

And here's the field formula for the Special ::M field, to update on any change:

```
P1F4 cases
  case cv = 0 of "<ASCII 255>" endof
  case cv <> 0 and P1F5 <> "" of "<ASCII 255>" endof
  default ""
endcases
```

Of course, the first line above is testing the Payment field. The second line says that if the Payment field is zero, then the ::M field will be filled with the invisible character ASCII 255, thus allowing the record to be saved. By

"<ASCII 255>"

in the formula above, I mean enclosing in quotes the character you get by holding down the **Alt** key while tapping **2, 5, 5** on the Number Pad, in that order (don't use the white number keys across the top of your keyboard), then releasing the **Alt** key. This produces an invisible character, but a character nonetheless. In virtue of the third line, if the Payment field is other than zero and the Payment Method field is filled in, we allow the record to be saved by the same logic. And finally, in virtue of the fourth line, if the Payment field is zero and the Payment Method field is blank (this is the only remaining possibility), the ::M field is left blank, keeping the record from being saved.

So, the user never sees anything in the Special ::M field, but sometimes it's filled in and sometimes not. When it's filled in, it's filled in with the invisible ASCII 255 character. We wouldn't need to use the ASCII 255 character if we could hide an ::M field, but DataPerfect doesn't allow that. If we could hide it (::HM or ::MH), we could simplify the formula like this:

```
P1F4 cases
  case cv = 0 of "A" endof
  case cv <> 0 and P1F5 <> "" of "A" endof
  default ""
endcases
```

or use a G9 field like this:

```

P1F4 cases
  case cv = 0 of 1 endof
  case cv <> 0 and P1F5 <> "" of 1 endof
default 0
endcases

```

But this isn't an option available to us.

In the scheme we have in place now, when the user enters something in the Payment field, leaves the Payment Method field blank, and then hits **F10** to save, they're met with an error message:

```
You must enter a value in this field before saving.
```

This puts the cursor in the ::MN field, where the user can't edit it (because it's an ::N field). Thought it does place the user right next to the field that needs to be filled in (the Payment Method field), it can still be kind of confusing. But it won't let them save the record without entering a value in the Payment Method field. Personally, I feel this is better done with a report that holds the user's hand. Read on for that.

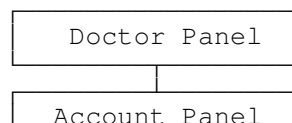
DPMouse© and Field Protection

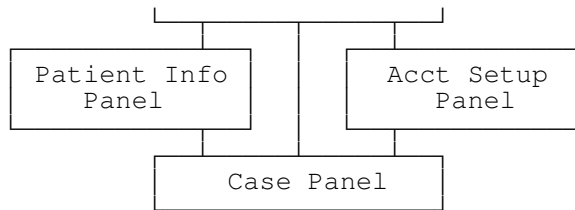
DPMouse© also offers special protection to an individual field. If DPMouse© sees an ASCII 249 (·) next to a field, it won't let the user's cursor land on that field. If it sees an ASCII 250 (•), it will let the cursor *land* on the field, but won't let the user *edit* the field if the displayed record was previously saved.

DPMouse©'s ASCII 249 code is especially powerful. It forces the user's cursor to skip past a designated field. Why not just format such a field as non-updatable (::N)? Well, you may have a field you want the user to avoid, given the value present in another field. Let's consider my billing application. It has both Debit and Credit fields in the same Transaction Panel. Depending on what type of transaction code the user enters in the Transaction Code field, DPMouse©, courtesy of its ASCII 249 code and a couple of strategically positioned computed fields, blocks either the Debit field or the Credit field from the cursor. I now don't worry about the user placing a value in the Debit field when a payment code is entered in the Transaction Code field.

Using DPMouse© To Conditionally Close A Panel Link

DPMouse© allows the developer to control access through a panel link in a way that's impossible with DataPerfect alone. A common scenario might be one where you don't want the user to create a record in a subpanel until certain information in a different subpanel is filled out first. For instance, one part of my office application can be diagrammed as follows:





I don't want the user to be able to penetrate the panel link that takes him from the Account Panel to the Case Panel without first filling out critical information in both the Patient Information Panel and the Account Setup Panel. The Case Panel needs to grab certain important information in those two higher panels, and needs to do this on record creation.

With DPMouse© installed, I merely place a properly formulated A1::C field next to the panel link I want to *conditionally* protect from penetration. That A1::C field will display the ASCII 249 character (·) under certain conditions. In this case it displays the ASCII 249 character when key fields in certain subpanels aren't filled out. If DPMouse© sees that character abutting a field, whether as panel text or displayed field data, it will keep the cursor out of the field (both DataPerfect and DPMouse© treat a panel link as a *field*).

Using DataPerfect 2.3's Menu Facility To Prevent Creation of Records via a Link

Whereas DPMouse© offers a very versatile way to prevent a user from creating a record on the other side of a panel link, DataPerfect 2.3 offers a menu facility to do this in a more general and, consequently, less versatile way.

When designing a DataPerfect menu (see *Menus* in my **Securing The Application** chapter), the definer can, with respect to a menu item that loads a particular panel, keep the user from modifying records through *any* of that panel's links. DataPerfect calls this menu option the *Restrict Modification to First Level* option. Choosing this protection for a menu item that loads a panel keeps the user from modifying and creating any records via the links in that panel, whether those links be data links or panel links. Very sweeping, but not very versatile like the our DPMouse© counterpart to this, which allows us to *conditionally* and *selectively* close off panel links.

Controlling Record Deletions with DPMouse©

DPMouse© offers the developer more flexibility in forming delete protection schemes than native DataPerfect. With DPMouse© installed with its *P* runtime flag active, the developer can not only keep all records from being deleted in a particular panel, but can also keep records from being deleted in a particular panel under certain conditions, even if the user enters that panel via a menu item that allows for deletions. For instance, with DPMouse© installed, a computed field placed in the upper left corner of a panel can keep a record from being deleted if it displays either ASCII 174 («) or 175 (»). The first warns the user they can't delete the current record

because it contains critical data, and the second warns the user they can't delete it because it has subrecords with critical data.

How you formulate this computed field determines which ASCII character displays, and under what conditions. So you might have the first one display for users with certain values in a particular User ID Panel field, and the second one display for everyone when it sees the current record has a subrecord. Such a field formula might look something this:

```
if P1F10P2F1 <> "" then "»"  
else if user.field[5] = 1 then "  
else "«"  
endif endif
```

The first line checks to see if the current record has any subrecords. It does this by penetrating the panel link P1F10, seeing if the first field of Panel 2 (P2F1) is blank. If it is, the computed field displays ASCII 175 (»). This causes DPMouse© to block the deletion and display a message about the current record having a subrecord with critical data.

If the current record does *not* have a subrecord, the *second* line looks at the value in the user's User ID Panel field 5. Assuming a 1 there allows for deletions in the current panel, and a 0 doesn't, if the second line sees a 1 there, the computed field displays nothing, allowing the deletion because it fails to trigger the DPMouse© delete protection mechanism.

Finally, if the current record doesn't have a subrecord and the user's User ID Panel field 5 isn't 1, the third line results in the computed field displaying ASCII 174 («), which causes DPMouse© to block the deletion while displaying a message about the current record containing critical data.

I discuss the User ID Panel facility in the *User ID Panel* section of my **Securing the Application** chapter.

Controlling Data Entry with Reports

This technique becomes important when you want to hold the user's hand in a sensitive panel. This may be a panel that has data you only want him to Browse when *in the panel*, but you'd like some way for him to create and edit records there *in a controlled fashion*. Or it may be a panel you don't want certain users to Browse, but in which you want them to be able to create a record without seeing the resulting completed record (certain fields that update automatically contain sensitive data, for instance). Reports work nicely here. Let's explain.

Reports That Control Record Deletions

This is a big issue, as you can imagine. Some applications typically don't, by convention, allow for *any* deleting of records. This would be typical of an accounting application, where you might prefer the user enter a record on a later date that counters the earlier record, as opposed to going back in time and deleting the older record. This is especially important in applications that have some sort of closeout procedure that locks figures in at the end of the day or month. Such an application is obviously going to make heavy use of DataPerfect's No Delete option in the menu system.

But you may have a need that falls somewhere in between the extremes of allowing unlimited delete rights and allowing none. Perhaps you'd like to allow only the Supervisor and Owner of the application to delete records. All others in the database won't be allowed to delete records. Again, you can simply give the Supervisor and Owner this sort of access to the database via a submenu off the Access Menu (I discuss the notion of an Access Menu in *Controlling User Access* in the *User ID Panel* section of my **Securing the Application** chapter).

But let's say you'd like a little more control over deletions. That is, you'd like to give the Supervisor and Owner of the application the right to delete records, but you don't want that to be easy for them to do. You want them to really think about it before they do it. Yes, DataPerfect temporarily interrupts an attempted record deletion during Browse mode with the *Confirm Deletion (Y/N)* message, but you want more of an obstacle than that. You want the obstacle to be more painful and informative.

A typical way to do this is to have the Supervisor and the Owner enter the database with a submenu path that, like other users, denies them Delete privileges. But unlike the submenu paths taken by other users of the database, their paths offer reports that delete records. Such a Delete Report uses the Delete Record code in the Report Body of the Edit Report Form (**Ctrl-F7, A**). Either of the following reports asks the user for a record to delete (via a lookup), and then deletes that record and stops. That is, each of these reports deletes only one record:

<pre>-----FIRST PAGE HEADER----- -----User Chooses Next Record By LookUp----- -----Store 1 in Report Variable 200 ----- -----OTHER PAGE HEADER----- --Empty-- -----TWO-LEVEL REPORT HEADER----- --Empty-- -----REPORT BODY----- -----Stop [Sub]Report if 0 Is in Report Variable 200 ----- -----Store 0 in Report Variable 200 ----- -----Delete Record----- -----TWO-LEVEL FOOTER----- --Empty-- -----PAGE FOOTER----- --Empty-- -----FINAL FOOTER----- --Empty--</pre>	<pre>Delete Report 1: One delete.</pre>
--	---

-----FIRST PAGE HEADER-----	Delete Report 2: One delete.
-----Store 1 in Report Variable 200 -----	
OTHER PAGE HEADER	
--Empty--	
TWO-LEVEL REPORT HEADER	
--Empty--	
REPORT BODY	
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----	
-----User Chooses Next Record By LookUp-----	
-----Store 0 in Report Variable 200 -----	
-----Delete Record-----	
TWO-LEVEL FOOTER	
--Empty--	
PAGE FOOTER	
--Empty--	
FINAL FOOTER	
--Empty--	

Whereas each of the above reports stops after deleting one record because Report Variable 200 is reset to 0 after the first delete, the following allows for unlimited deletes:

-----FIRST PAGE HEADER-----	Delete Report 3: Unlimited deletes.
--Empty--	
OTHER PAGE HEADER	
--Empty--	
TWO-LEVEL REPORT HEADER	
--Empty--	
REPORT BODY	
-----User Chooses Next Record By LookUp-----	
-----Delete Record-----	
TWO-LEVEL FOOTER	
--Empty--	
PAGE FOOTER	
--Empty--	
FINAL FOOTER	
--Empty--	

And here's another possibility. If you do this, it's probably a mistake in your report definition. Starting with the record the user chooses, it deletes that record and *all that follow it* in the active index:

-----FIRST PAGE HEADER-----	Delete Report 4: Probably a mistake.
-----User Chooses Next Record By LookUp-----	
OTHER PAGE HEADER	
--Empty--	
TWO-LEVEL REPORT HEADER	
--Empty--	
REPORT BODY	
-----Delete Record-----	
TWO-LEVEL FOOTER	
--Empty--	
PAGE FOOTER	
--Empty--	
FINAL FOOTER	
--Empty--	

I wouldn't settle for Delete Reports 1, 2, or 3, however. You really should put in some explanatory and warning text in the First Page Header, like this:

<pre> -----FIRST PAGE HEADER----- Warning: This routine deletes records permanently from the database! Please exercise caution when running it. It will present you with a lookup of possible records to delete. For each one you want to delete, highlight it and hit Enter. -----Prompt for Value of Report Variable 1 ----- -----Store Value in Report Variable 1 ----- -----Stop [Sub]Report if 0 Is in Report Variable 1 ----- -----OTHER PAGE HEADER----- --Empty-- -----TWO-LEVEL REPORT HEADER----- --Empty-- -----REPORT BODY----- -----User Chooses Next Record By LookUp----- -----Delete Record----- -----TWO-LEVEL FOOTER----- --Empty-- -----PAGE FOOTER----- --Empty-- -----FINAL FOOTER----- --Empty-- </pre>	<div style="border-left: 1px solid black; padding-left: 10px;"> Delete Report 3a: Unlimited deletes with explanatory text & a way out. See below for explanation of this part. Unlimited deletes by lookup. </div>
---	--

Make sure you run the above report with *Screen Only* as its Destination (in the Initial Report Definition Screen).

In its First Page Header, the Prompt for Value of Report Variable 1 is formatted U1, and has the following prompt:

```
Shall we continue with this routine? (Y/N):
```

When you type in the prompt above, hit the spacebar enough times before the first word so the prompt lines up with the text above it, like this:

```

Warning: This routine deletes records permanently from the database!
Please exercise caution when running it. It will present you with a
lookup of possible records to delete. For each one you want to delete,
highlight it and hit Enter.

Shall we continue with this routine? (Y/N):

```

Run the report to be sure the text is lined up with the prompt.

The next line in the First Page Header examines the user input for Report Variable 1 and changes Report Variable 1's value to 0 if they didn't answer with a *Y*. Here's the formula for the *Store Value for Report Variable 1* code:

```
if rv1 <> "Y" then 0 else 1 endif
```

This allows the subsequent *Stop [Sub]Report if 0 Is in Report Variable 1* code to stop the report cold if the user entered anything other than *Y* for Report Variable 1.

Providing an Undelete

All that said, I urge you to consider not letting users physically delete records. Instead, I suggest you consider letting them *mark* records as deleted. This way, mistakes can be corrected very easily, in that it allows the user to *unmark* a "deleted" record. This provides your application with an undelete mechanism. Let's explain.

In any panel in which you want to offer the user an undelete option, create a Delete field. Format it U1, and give it a field formula like this:

```
if P1F1<>" " then "D" else "" endif
```

where P1F1 is the Delete field itself. This formula makes sure that if the user enters anything other than a space in that field, DataPerfect will update it to *D*.

Immediately next to the U1 Delete field, create an A5::C field, and give it the following field formula:

```
if P1F1<>" " then "eleted" else "" endif
```

With those two fields next to each other, if the user puts any character other than the space in P1F1, he sees

```
Deleted
```

displayed on his screen. When he erases the character in P1F1, the word *Deleted* disappears from his screen.

Lookups and the Undelete Scheme

Now, let's take this undelete scheme a step further. I suggest that in any panel in which this scheme is active, you also create another field—a hidden A7::C field (i.e., A7::CH). If, again, the Delete field (the U1 field that displays *D* or nothing at all) is P1F1, then use the following field formula for P1F3 (our new A7::CH field):

```
If P1F1<>" " then "Deleted" else "" endif
```

Put that hidden field on every lookup field list in that panel. I'd make it the last field on each lookup field list. Now every lookup performed in that panel will clearly show which records are marked for deletion.

Totaling and the Undelete Scheme

Further, say you want this undelete scheme set up in a panel that totals to another panel, and you want totals updated when the user marks a record in that panel for deletion. That is, you want the totals to immediately update without having to wait for the Supervisor to run the Delete Report at the end of the day. Here's a way to handle this.

Say you have a Transaction Panel that includes the following fields:

Transaction Panel			
<i>Field</i>	<i>Name</i>	<i>Format</i>	<i>Comment</i>
Delete field 1	P1F1	U1	Displays <i>D</i> or blank
Delete field 2	P1F2	A5::C	Displays <i>eleted</i> or blank
Hidden delete field	P1F3	A7::CH	For lookups
Amount field	P1F4	G-ZZZ9.99	Data entry

Further, that Amount field totals to a field in the Customer Panel. Well, with this undelete scheme, don't put the Keep A Total on P1F4. Instead, have a hidden G-ZZZ9.99 field, P1F5, total to the Customer Panel. Give P1F5 the following field formula, to update on any change:

```
if P1F1="" then P1F4 else 0 endif
```

When the user puts a character in P1F1 (thusly marking the record for deletion), hidden P1F5 goes to 0. Since hidden P1F5 is the field with the Keep A Total, the total updates. If the user then decides to undelete by removing the character from P1F1, hidden P1F5 reverts back to the value found in P1F4. Again, the total updates.

Undelete Scheme With Totaling			
<i>Field</i>	<i>Name</i>	<i>Format</i>	<i>Comment</i>
Delete field 1	P1F1	U1	Displays <i>D</i> or blank.
Delete field 2	P1F2	A5::C	Displays <i>eleted</i> or blank.
Hidden delete field	P1F3	A7::CH	For lookups.
Amount field	P1F4	G-ZZZ9.99	Data entry. No Keep A Total.
Hidden amount field	P1F5	G-ZZZ9.99	Updates to P1F4 on any change. Has the Keep A Total.

Physically Deleting the Record

The record, of course, isn't deleted simply by saving it with *D* in P1F1. If you need to offer the user a way to physically delete records marked for deletion, create a special Delete Report like Delete Report 3a above, but assign it an Exception List Index that excludes all records with P1F1 blank. That would be an index with a single field on its Exception List: P1F1. That index now holds only records that are marked for deletion. Again, you'd only want the Supervisor (or someone you trust with deleting records) to run this report, probably at the end of each day, so make sure you only put it on *their* menu.

```

-----FIRST PAGE HEADER-----
-----Turn Print Off-----
-----Turn File Off-----
Warning: This routine deletes records permanently from the database!
Please exercise caution when running it. It will present you with a
lookup of possible records to delete. For each one you want to delete,
highlight it and hit Enter.

-----Prompt for Value of Report Variable 1 -----
-----Store Value in Report Variable 1 -----
-----Stop [Sub]Report if 0 Is in Report Variable 1 -----
-----Turn Print On-----
-----Turn File On-----
-----Store Value in Report Variable 2 -----
Date:      :
Time:      :
User ID:    :

-----OTHER PAGE HEADER-----
--Empty--
TWO-LEVEL REPORT HEADER
--Empty--
REPORT BODY
-----User Chooses Next Record By Lookup-----
:
:
:
:
-----Delete Record-----
TWO-LEVEL FOOTER
--Empty--
PAGE FOOTER
--Empty--
FINAL FOOTER
--Empty--

```

```
Print records
deleted.
```

```
REPORT: Delete Report

Destination:                Printer LPT1    Append to Disk File
1 - Printer On/Off
2 - Disk File On/Off
   Filename: DELETE.RPT
3 - Index Number            6
4 - Search Conditions       No Search
5 - Sort Direction          Forward
6 - Disk File Mode WP/DOS   DOS Text
7 - Print Margins           Top        Bottom    Left        Text Lines
                           6           0         5          54
8 - Edit Report Form
9 - Edit Report Name
```

Left Margin for
hole punches

398 Securing Data Entry

record. Any record the user fails to select this way remains in the database, marked for deletion. Unless the user then unmarks that record's Delete field (essentially undeleting it), it will again show up in the report lookup when this Delete Report is run.

Reports That Control Record Creation

Here's an example of a report that holds the user's hand during record creation, not letting them see the record that's actually created:

```
[Destination: Screen Only]
-----FIRST PAGE HEADER-----
This routine will create a new account. Please make sure you've
cleared creation of this new account with your supervisor.

You'll need the client's Social Security Number for this routine.

-----Prompt for Value of Report Variable 1 -----
-----Store Value in Report Variable 1 -----
-----Stop [Sub]Report if 0 Is in Report Variable 1 -----

-----Prompt for Value of Report Variable 2 -----
-----Prompt for Value of Report Variable 3 -----
-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
=====CREATE RECORD LINK/PANEL: 0 1 =====
-----REPORT BODY-----
-----Store Report Variable 2 in Field 5 -----
-----Store Report Variable 3 in Field 6 -----

=====SAVE RECORD=====
-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----

-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----

--Empty--
```

CreateRecord Report 1

Warning text.

Check to see if we should continue. See details below.

Create exactly one new record in Panel 1. See below for more details.

Starting at the top of the above Report Definition, the Destination obviously needs to Screen Only in the Initial Report Definition Screen. Otherwise the warning text will go to the printer (you can, of course, surround the warning text with Turn Print/File On/Off codes instead of using the Screen Only Destination).

The *Prompt for Value of Report Variable 1* code is formatted U1 and has the following prompt:

Shall we continue? (Y/N) :

The following formula is then used in the *Store Value in Report Variable 1* code:

```
if rv1 <> "Y" then 0 else 1 endif
```

That allows the *Stop [Sub]Report if 0 Is in Report Variable 1* code to stop the report cold if the user answered anything other than *Y* to the prompt.

In the Report Body I used **Ctrl-F7, 6** to call the Subreport Menu:

```

-----Subreports & Record Creation-----
1 - Include Subreport
2 - Create Record Through Link
3 - Create Record From Panel List
4 - Subreport Using Virtual Link
0 - Return to Edit
-----Selection: 0-----

```

I chose option 3 above. That gave me the *CREATE RECORD* portion of the Report Body. The 0 on the CREATE RECORD line signifies this record creation isn't using a link to get the panel in which the record will be created. That is, it's using the Panel List to get to that panel. The 1 on that line signifies the target panel for this record creation is Panel 1.

So what's all this, just after the CREATE RECORD portion of the Report Body, still within the Report Body?:

```

-----SAVE RECORD-----
-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----
-----TWO-LEVEL FOOTER-----

```

Here, Report Variable 200 is being set to 0, and the report is being stopped immediately after. If I don't do something like this, the report will attempt the CREATE RECORD again and again, once per record found in the panel on which the report is based. The two codes you see above, at the very end of the Report Body, cause the report to stop after one record is processed in the panel on which the report is based.

Also note that it's irrelevant which panel we use to base this report on, as long as it has at least one record. That's because the CREATE RECORD portion of its Report Body is using the Panel List to find the panel in which to create the new record. So the report can start anywhere in the database to do this.

You might want to consider even more protection here, by altering the above the report so it sends records the user creates to a disk file, providing the Supervisor a running record of record creations that's a lot easier to read than a Transaction Log.

Take a look at this. It's the same report, but with a different Destination in the Initial Report Definition Screen, the User ID being stored in Report Variable 4, and fields being sent to the disk file. Also note that I have to surround the warning text in the First Page Header by Turn Print/File On/Off codes, otherwise that text will be sent to the printer and/or file (I always turn *both* printer *and* file on and off, just in case I later decide to switch from one to the other, or use both).

```

[Destination: Append to Disk File]
-----FIRST PAGE HEADER-----
-----Turn Print Off-----
-----Turn File Off-----
This routine will create a new account. Please make sure you've
cleared creation of this new account with your supervisor.

You'll need the client's Social Security Number for this routine.

-----Prompt for Value of Report Variable 1 -----
-----Store Value in Report Variable 1 -----
-----Stop [Sub]Report if 0 Is in Report Variable 1 -----

-----Prompt for Value of Report Variable 2 -----
-----Prompt for Value of Report Variable 3 -----
-----Turn Print On-----
-----Turn File On-----
-----Store Value in Report Variable 4 -----

-----OTHER PAGE HEADER-----
--Empty--
-----TWO-LEVEL REPORT HEADER-----
--Empty--
-----REPORT BODY-----
=====CREATE RECORD LINK/PANEL: 0 1 =====
-----REPORT BODY-----
-----Store Report Variable 2 in Field 5 -----
-----Store Report Variable 3 in Field 6 -----

=====SAVE RECORD=====
##### Date ##### Time #####
##### Record fields #####
##### Blank line #####

-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----

-----TWO-LEVEL FOOTER-----
--Empty--
-----PAGE FOOTER-----
--Empty--
-----FINAL FOOTER-----
--Empty--

```

CreateRecord Report 2

Turn Print/File off.

Turn Print/File back on.
Put USER.FIELD[1] in RV4.

Date, Time, & RV4.
Record fields.
Blank line.

The above report works like the previous one, but it appends to a disk file a copy of each record *created*, including who created it, and when.

Reports That Control Editing

Perhaps you need even more security in a particular panel. You not only want to hold the user's hand during record creation, but also editing of existing records. Again, the reason may be the same in both cases: you don't want the user *in the panel* when they edit. That is, you don't want them to see the record itself. Or you feel the editing process needs certain guiding text to keep the user from making a silly mistake. Here's an example:

[Destination: Screen Only]	EditRecord Report 1
-----FIRST PAGE HEADER-----	
This routine will allow you to change the last name of a client. Please make sure you've cleared this with your supervisor.	Just like CreateRecord Report 1
-----Prompt for Value of Report Variable 1 -----	
-----Store Value in Report Variable 1 -----	
-----Stop [Sub]Report if 0 Is in Report Variable 1 -----	
-----User Chooses Next Record By LookUp-----	
This client's last name is [REDACTED]. What should it be now? You may hit F1 or ESC if you don't want to change it.	Offers a lookup. Then shows user's choice.
-----Prompt for Value of Report Variable 2 -----	Prompts "Your choice?:"
-----OTHER PAGE HEADER-----	
--Empty--	
-----TWO-LEVEL REPORT HEADER-----	
--Empty--	
-----REPORT BODY-----	
-----Store Report Variable 2 in Field 3 -----	
That client's last name is now [REDACTED].	Put new LastName in LastName field. Show new LastName.
-----Store Value in Report Variable 200 -----	Stop after one record processed
-----Stop [Sub]Report if 0 Is in Report Variable 200-----	by setting RV200 to 0.
-----TWO-LEVEL FOOTER-----	
--Empty--	
-----PAGE FOOTER-----	
--Empty--	
-----FINAL FOOTER-----	
--Empty--	

The above report offers the user a lookup of clients sorted by Last Name (assuming you tie it to an index beginning with Last Name, of course). When they choose a client, they're shown the Last Name of that client and asked what it should be now. After they type in the new Last Name and hit **Enter**, the report updates it and stops. Only one record is allowed to be processed this way with each run of the report. That is, they only see the lookup once.

And, as we've done before, we can change this report into one that appends each edit to a disk file:

```

[Destination: Append to Disk File]
-----FIRST PAGE HEADER-----
-----Turn Print Off-----
-----Turn File Off-----
This routine will allow you to change the last name of a client.
Please make sure you've cleared this with your supervisor.

-----Prompt for Value of Report Variable 1 -----
-----Store Value in Report Variable 1 -----
-----Stop [Sub]Report if 0 Is in Report Variable 1 -----
-----User Chooses Next Record By LookUp-----

This client's last name is [REDACTED].
What should it be now?
You may hit F1 or ESC if you don't want to change it.

-----Prompt for Value of Report Variable 2 -----
-----OTHER PAGE HEADER-----
-----Empty-----
-----TWO-LEVEL REPORT HEADER-----
-----Empty-----
-----REPORT BODY-----
-----Store Report Variable 2 in Field 3 -----

That client's last name is now [REDACTED].

-----Turn Print On-----
-----Turn File On-----
-----Store Value in Report Variable 4 -----
[REDACTED]
[REDACTED]
[REDACTED]

-----Store Value in Report Variable 200 -----
-----Stop [Sub]Report if 0 Is in Report Variable 200 -----

-----TWO-LEVEL FOOTER-----
-----Empty-----
-----PAGE FOOTER-----
-----Empty-----
-----FINAL FOOTER-----
-----Empty-----

```

EditRecord Report 2

Just like
CreateRecord
Report 2.

Turn Print/File
back on.
RV4=USER.FIELD[1].
Date, Time, RV4.
Fields in record.
Blank line.

Hiding Data Entry

[Load UD.STR for this section.
Find the *Hiding Data Entry* panel.]

Suppose you have a need to allow the user to enter data in a particular field but you want the result of the data entry to be hidden after saved. Further, you don't want other viewing users to see the data on the screen while it's being entered. That is, you don't want their keystrokes to echo on the screen for all to see. This is typically done in password fields in computer applications in general, where all that echoes to the screen is a series of asterisks.

Let's make this simple and say you want to do this with an A3 field. First off, that field is going to be hidden, which means we'll have to provide the user with a different field in which to enter his data.

Actually, what we need is a series of three A1::E fields, where each moves the cursor to the next after the user enters data in it (the ::E display modifier, remember, moves the cursor this way). These three A1::E fields must be right next to each other, giving the appearance of a single A3 field. Next we'll need yet another series of three A1 fields, though these will be hidden.

So far, we have this:

- Three A1::E fields
- Three A1::H fields
- One A3::H field

The three A1::H fields will grab the data entered by the user in the three A1::E fields. The A3::H field will concatenate the three A1::H fields. We need to make sure that, as the user enters data in each A1::E field, its corresponding A1::H field grabs that data before the A1::E field changes it to an asterisk.

The trick here is the formulas we assign to the A1::E and A1::H fields, and the Edit Order in which we place the fields. Let's give all these fields names, so we can reference them easily here with formulas:

Format	Name
A1::E	P1F1
A1::E	P1F2
A1::E	P1F3
A1::H	P1F11
A1::H	P1F12
A1::H	P1F13
A3::H	P1F21

Here's a combination of formulas and Edit Order that works:

Order	Format	Name	Formula	Trigger
1	A1::H	P1F11	if P1F1<>"*" then P1F1 else "" endif	any change
2	A1::H	P1F12	if P1F2<>"*" then P1F2 else "" endif	any change
3	A1::H	P1F13	if P1F3<>"*" then P1F3 else "" endif	any change
4	A1::E	P1F1	if P1F11<>"*" then "*" else P1F1 endif	any change
5	A1::E	P1F2	if P1F12<>"*" then "*" else P1F2 endif	any change
6	A1::E	P1F3	if P1F13<>"*" then "*" else P1F3 endif	any change
7	A3::H	P1F21	cat.c[P1F11;P1F12;P1F13]	creation

The Edit Order above is only a *relative* Edit Order. That is, P1F11 doesn't need to be number one in that Edit Order. It just needs to come before P1F12, and so on. What's important here is that the A1::H fields precede the A1::E fields in the Edit Order.

One important note about this scheme. It only works in Create mode, not Edit mode. After the above record is saved, the A3::H field will never change. So you'll need to use this scheme under the control of a Go To Panel menu item that has Panel Access Rights set to Create OK - No Edit/Delete.

To get around this Create Only problem, we can use my Moment function (see *An Alternative Solution: Using the Concepts of MOMENT and MODULO* in my **Fields: Issues** chapter for more on the moment function). In UD.STR's Hiding Data Entry Panel, take a look at how differently Password Field 1 and Password Field 2 behave during Edit mode. Password Field 2's field formula turns on the value found in the Moment field; whereas Password Field 1's field formula doesn't:

Password Field 1

```
If (86400*today)+now=P21F12 then P21F10  
else cat.c[P21F5;P21F6;P21F7;P21F13;P21F14;P21F15] endif
```

Automatically computed at any change and when record is saved.

Password Field 2

```
cat.c[P21F5;P21F6;P21F7;P21F13;P21F14;P21F15]
```

Automatically computed when record is created.

Password Field 1 looks at the Moment field to see if the current moment is the same as the moment found in the Moment field. As soon as one second has passed after hitting **F6** (Edit) or **F9** (Create), these two values will be different, so the ELSE clause rules during data entry. This causes the CAT.C statement to be inserted into Password Field 1. That statement grabs values found in the Intermediary fields. At that moment, the Intermediary fields hold the true Password constituents.

On Save, the Moment field updates one more time, to match the current moment, and the Intermediary fields blank out. When that happens, the IF clause rules, which returns Password Field 1 back to the value it had before the Moment field updated a second time, effectively ignoring the newly blanked out Intermediary fields.

This is complicated. Play with that UD.STR's Hiding Data Entry Panel a while to convince yourself that Password Field 1 works fine in Edit, whereas Password Field 2 doesn't. Then read over this discussion again.

Application Maintenance Utilities

This chapter is for both beginners and the experienced.

DPEXport and DPImport

DPEXport and DPImport (DPEXP.COM and DPIMP.COM) usually are used together. DPEXport, when applied to an .STR, will produce a text file of the .STR's contents in readable English. This text file will have the same filename as the .STR, except its extension will be .STE. Just how to read an .STE file is laid out pretty well in the Reference Manual, so I won't get into that here.

After creating an .STE with DPEXport, DPImport can be used to create a new .STR from the .STE file. So why do any of this?

One reason for creating an .STE from your .STR is to hunt down possible corruption that appears in your application. Say you see some odd looking low ASCII characters in Help screens where you know you never put any. You might create an .STE with DPEXport and then load the .STE in a text editor and search for these characters and delete them. After running DPImport on the new .STE, they shouldn't appear in the application anymore.

Another reason for running DPEXport on your .STR, and DPImport on the subsequent .STE, is to let that process clean up any corruption that may have crept into the .STE during an extensive development session. Running DPEXport and DPImport like this is a good practice after every such session.

A third reason for running DPEXport and DPImport like this would be to make some changes to the application that would be easier done in the .STE file than when the .STR is loaded in DataPerfect. For instance, you might want to globally change some text string that appears in many Help screens. You can just load the .STE file in a text editor and do this, though I'd make sure you confirm each Replace.

.STE Editing Caveats

Please heed a couple caveats when editing an .STE file. When editing the .STE file with a *text editor*, turn off word wrap first. In WordPerfect Corporation's Editor 3.x, you do this with **Shift-F1, E, W, T, N, Enter, F7, F7, F7**. When editing an .STE file with a *word processor*, save it in generic word processor format, not DOS text or standard ASCII text. Saving a word processor document in generic word processor format won't replace soft returns with hard returns. So a paragraph, for instance, is saved as one long line. Saving a word processor document in DOS text or ASCII text format puts a hard return at the end of each line in a paragraph. The latter will ruin the .STE file.

Editing an .STE file isn't for the faint of heart. You can blow away much of your application this way. Simply changing a field's format to one that's incompatible

with its panel can trash an entire panel. Don't forget that your text editor or word processor doesn't know you're editing an application's heart and soul, so it's not going to give you error messages when, say, changing a field format to one that's too big for its panel. Do this sort of thing in DataPerfect if you're not sure what you're doing, and leave DPEXport and DPImp for simply cleaning up an .STR file that may be corrupt, or as a followup to a lot of development. I don't remember the last time I actually edited an .STE manually with a text editor or word processor.

DPImport Caveats

Be aware that when you run DPImport, it deletes your application's .IND file before it starts the actual .STE conversion to an .STR. This is for your own good, just in case any changes you made to the .STE file are now incompatible with the existing .IND file. By deleting the exiting .IND file, DPImport is forcing you to regenerate indexes after loading the new .STR. If you don't want to regenerate indexes (you have a lot of data and this will take many hours, or even days, for instance), there are ways around this. But you must know when this isn't safe. Read my sections on *When It's Okay to Overwrite an .STR* and *Cleaning the .STR Without Re-indexing* before attempting this. They're in this chapter.

Importing Reports

One more thing many developers don't know you can do with DPEXport and DPImport. You can take selected reports from the REPORTS: section of an .STE file and import just those reports into an existing .STR without having to import the entire .STE, and *without having to regenerate indexes*. To do this, create a text file that begins with the string

```
REPORTS:
```

all by itself on the first line (flush left).

Now put in the reports you took from the REPORTS: section of the .STE file. These reports follow the above line with a blank line between them and *REPORTS:*. Make sure there's a blank line at the end of the last report. Give the new text file a name like REPORTS.STE when you save it. Take all the precautions I already discussed when editing an .STE (no word wrap, etc.).

Now run DPImport. Tell it to import the new REPORTS.STE. It will ask you which .STR to import these reports into. When done, you'll find the new reports at the end of the Report List.

DPDiagnostics

This utility (DPDIAG.EXE) is indispensable for any serious DataPerfect application developer. It should be used after any development session that involves any of the following:

- Altering a field format.
- Altering an index
- Creating or deleting a field
- Creating or deleting an index
- Altering a link definition
- Deleting a link
- Editing your .STE and then running DPIMP on it

What DPDiagnostics does is analyze your .STR to make sure that there's no conflict between related entities. For instance, you may have initially created a link that used a particular index and later deleted that index thinking it was unnecessary. Likewise, you may have initially created a field formula that accessed another field in the same panel and later deleted that other field, or deleted or altered the link that formula used to grab that other field if it was in a different panel. DataPerfect doesn't tell you when doing such things will cause a problem like this. The best way to detect this sort of error is to run DPDiagnostics on that .STR from time to time.

If you run DPDIAG from the command prompt, you get this help screen:

```
USAGE: dpdiag [database] [options]

/f-outputname  Specify output filename (no spaces)
/I-#           Specify the maximum field count for index optimization
              Validation
/l            List only the panel number and panel name
/n            Do not create an output file
/o            Suppress all optimization messages
/q-#          DPPrint printer port, 1=LPT1, 2=LPT2, 3=COM1, 4=COM2
/w            Suppress all warning messages
```

I suggest you initially run it with none of the startup options that suppress its output, just to see what it does. So if your .STR is MYAPP.STR, just do this at a command prompt:

```
dpdiag myapp
```

Its output will be to MYAPP.FIX unless you use the /F option to specify a different output filename. You'll find the information in the output file easy to understand. The output is divided up into Optimization Messages, Warning Messages, and Error Messages.

Optimization Messages

I ignore these, so I always use the /O switch above. DPDiagnostics is usually trying here to show you where an index either contains so many fields that it may slow down your application, or it's not being used by a lookup, link, menu, or report. Regarding the former, this is a judgment call. Regarding the latter, you must realize that DPDiagnostics doesn't take into account the Smart Lookups algorithm version 2.3 introduced. See *Smart Lookups* in my **Lookups** chapter if you don't know what they are.

Warning Messages

DPDiagnostics says Warning messages "indicate less serious problems that may keep your database from functioning smoothly and efficiently." You usually can suppress these. Here are the ones I see most often in my applications:

Mismatching field formats between a list field and its counterpart index field.

When this message is given as a Warning and not an Error, it usually involves a mismatch between G and H fields that are of the same size. This is of no consequence. More serious mismatches are reported as Errors.

Link's Index in Destination Panel contains an Exception List.

I use Exception List Indexes with panel links often. There's nothing wrong with this. DPDiagnostics just wants you to be aware that your panel link here is giving the user limited access to its target panel.

Exception List on Index contains more than one field.

Again, I do this frequently. There's nothing wrong with it.

Error Messages

DPDiagnostics says Error messages "indicate problems that may damage the database." Take them seriously. Here's a list of Error messages I extracted from DPDIAG.EXE. When you get one of these messages, DPDiagnostics will reference the panel, field, etc., involved:

- Report index does not exist in panel
- Destination Panel does not exist. Error within Subreport Return Code
- Report Variable to be printed does not exist
- Error within Skip if RV is False Code
- Error within Skip [Sub]Report if RV is False Code
- Error within Skip To Record At RV Code
- Error within Store Report Variable in Field Code

- Error within Store Report Variable in Field Code
- Field does not exist in panel
- Error with the field to be printed
- Error in Store Value in Report Variable Code
- Illegal Formula
- Mismatching field formats between a list field and its counterpart index field
- Field(s) shown below for Index do NOT exist
- Field(s) shown below for Index Exception List do NOT exist
- Destination Panel for Menu Option does NOT exist
- Destination Index for Menu Option does NOT exist

DPOrder

DPOrder (DPO.EXE) allows the developer to change the order of Reports on the Report List, or Panels on the Panel List. Though it provides a wonderful service to DataPerfect application developers, it doesn't always work as planned. Most problems I've seen as a sysop of the DataPerfect forum on CompuServe are these:

DPOrder fails to complete because of out-of-memory problems.

This should have no dangerous consequences as long as you backed up before running it. You might try to increase your conventional memory using the usual methods (unload TSRs, optimize with a memory manager, etc.) and try again. Alternatively, if all you're trying to do is reorder your reports on the Report List, you can do that with STE Manager© (see *STE Manager©* in this chapter) or follow the approach of fellow DataPerfect application developer, Mark Nepon:

- With a *copy* of the .STR, delete all but the first panel from the Panel List.
- Now load it with DPOrder and reorder your reports. This probably won't give you the out-of-memory error.
- DPEXP this new .STR.
- Load the newly created .STE in a text editor.
- Save its reports section to a file named REPORTS.STE. That section starts with *REPORTS:* and goes to the end of the file.
- DPEXP the original .STR and delete the *REPORTS:* section of its .STE and retrieve the REPORTS.STE you previously created in its place. Save this .STE.
- In the original .STR, delete all its reports from the Report List.
- Exit DataPerfect and DPIMP the new .STE.

After an apparently successful DPOrder operation,

the application now has problems with some of its links.

Though I haven't seen this myself, I've heard of it happening. In all these cases, DPOrder was used to reorder panels. Apparently DPOrder sometimes fails to keep certain relational properties of a complex application intact when you use it to reorder panels. I've never heard of this happening when using it only to reorder reports. Unless you have a very simple application, I suggest you not use DPOrder on panels.

STE Manager©

This is one of the more interesting utilities available for .STR files. It's currently free copyrighted shareware owned and developed by fellow DataPerfect application developer, Bob Butler. I included a copy of it and its documentation on your diskette.

STE Manager©, in its basic operation, allows you to do this. First, running STE-MGR.COM on an .STR file produces a Transaction Log with the same name as the .STR file, but with the {L} extension. So it creates a new file MYAPP.{L} from MYAPP.STR. To see just what this .{L} file looks like, go ahead and copy one of your .STR files to your STE-MGR directory and run STE-MGR.COM there. Then choose that .STR copy from the .STR list STR-MGR presents you. Don't choose the MANAGER.STR from that list—that's the special .STR that ships with STE Manager©.

After hitting **Enter** on your .STR file, STE-MGR.COM processes it, showing you its progress on the screen, like this:

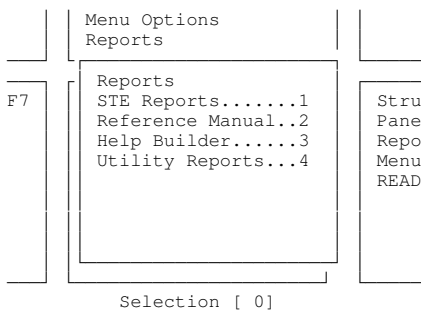
```
LOG EXPORT OF .STE DATA
PANEL:
PANEL:  ] Panels named here as they appear on the Panel List
PANEL:
MAIN MENU
SUB MENU
REPORT:
REPORT:  ] Reports named as they appear on this submenu
REPORT:
SUB MENU
REPORT:
REPORT:  ] Reports named as they appear on this submenu
REPORT:
SUB MENU
REPORT:
REPORT:  ] Reports named as they appear on this submenu
REPORT:
REPORT:
REPORT:  ] Reports named as they appear on this submenu
REPORT:
SUB MENU
REPORT:
REPORT:  ] Reports named as they appear on this submenu
REPORT:
REPORT:
REPORT:  ] Reports named as they appear on this submenu
REPORT:
```

Reports named as they appear on the Report List

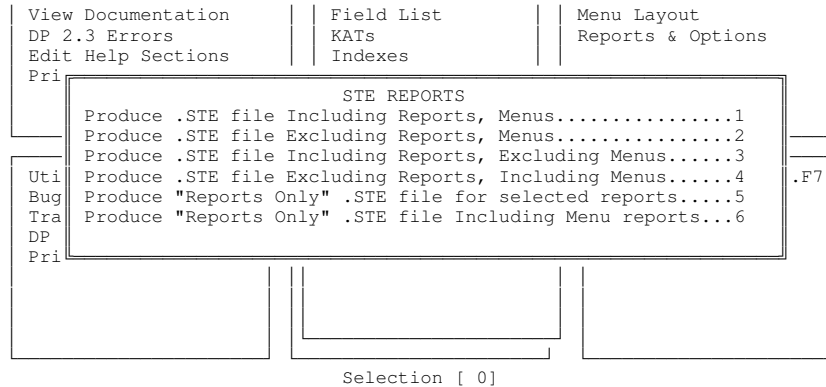
After importing this odd Transaction Log, the elements of your .STR file are now in the MANAGER.STR database. With it loaded in MANAGER.STR, you can manipulate most aspects of your .STR file. For instance, you can change the position a report occupies on the Report List or copy it from a menu back to the Report List. Once you're done changing the structure of the imported database, STE Manager© lets you then create an .STE that reflects those changes.

Here's the STE Manager© Main Menu (MANAGER.STR):

```
Selection [ 0]
```



Choosing **1** from that submenu calls the STE Reports menu:



The above six STE REPORTS selections offer you a way to produce different types of .STE files. The first four will each produce an .STE file that, when imported with DPImport, will produce a new .STR. The final two Reports Only choices produce .STE files that can be imported with DPImport into an existing .STR, even if its data files have data. Doing that will put the reports in the Reports Only .STE at the end of the existing Report List.

As I mentioned before, among other interesting things, STE Manager© lets you copy or move reports between their menu assignments and the Report List. Let's say, when developing a DataPerfect application, you like to delete all the reports on your Report List after assigning them to menu items (don't forget that assigning a Report List report to a menu item just creates a copy of the original Report List report). Some developers like to do this just to keep the Report List empty for future reports, or because they like to give the user access to the Report List (I advise against this), but don't want them to have access to certain Report Definitions.

Well, suppose, later, you want to copy one of those menu item reports back to the Report List. With that application's .STR/Log file loaded in STE Manager© as a database, you'll find a record for each report in the STE Manager© Report Panel, with each record showing that report's menu assignment (if any) and its numerical position on the Report List (if any). If it isn't in the Report List, it won't have a number in that field. Just put an appropriate number in that field and the report will end up on the Report List when you have STE Manager© create a new .STE file for that application.

There's a lot more you can do with STE Manager©. Be careful, however. Unlike DataPerfect, it won't let you know when you put in a illegal value for this or that application entity. You need to really understand DataPerfect to use this utility.

Application Maintenance Issues

This chapter is for the experienced DataPerfect application developer only. Some of the techniques talked about here require a strong stomach.

Upgrading a Client to a New Version of DataPerfect

If your client is running a DataPerfect application under DataPerfect 2.2 or 2.3, upgrading their application to 2.3's current version is simply a matter of running the same application under the new DataPerfect program files (DP.EXE and DP.SYS). No export/import of the .STR or data is necessary. Make sure you always use the DPEXP.COM and DPIMP.COM that go with the new version of DP.EXE.

This isn't true, however, if the application is running under DataPerfect 2.0 or 2.1. Databases created with 2.0 or 2.1 are not compatible with 2.3, though all their data can be safely imported into the same application run under the new version of DataPerfect. In such cases, follow these steps, backing up the database first (this process will take quite a while on a large database):

- Load the application with the new version of DataPerfect.
- Do not create, edit or delete any record in the database.
- Export all data in the database to a Transaction Log (**Shift-F9, A, 1**).
- Delete all data in the database (**Alt-F5, 4**).
- Exit the application.
- Exit DataPerfect.
- Run DPEXP on the .STR file.
- Run DPIMP on the .STE file created above.
- Run the new version of DataPerfect.
- Load the .STR file created above and choose **1** (Create a New Index File with New Indexes for All Data Files) from the menu.
- Import the Transaction Log you created above (**Shift-F9, 8**).

The database is now upgraded to the current version of DataPerfect 2.3.

Compatibility Between Different Versions of DataPerfect

As I said above, you should convert a database that's been running under DataPerfect 2.0 or 2.1 with the steps outlined above. But other issues arise that involve compatibility between different versions of DataPerfect. For instance, suppose you get the latest version of DataPerfect 2.3c. Until now, your client's application has been running under DataPerfect 2.3b.

As I mentioned above, all you need to do now is overwrite the old DataPerfect program and utility files with the new ones, and load the application. No export/import of the data or the .STR is necessary in this case.

But there are caveats here you must be aware of here. The most important one is to not run different versions of DataPerfect on the same network, even if they share the same version number. If they have different date stamps, they can be different enough to cause problems on a network when accessing the same database simultaneously.

That said, it still may come in handy to use older version of DataPerfect when developing an application. Say you're working with an .STR that needs some field formulas changed. The .STR is so complex that any attempt to edit a formula in the Specify Formula screen causes an Error 104, signifying DataPerfect is too low in free workspace for that formula edit operation to complete.

In such a case, it's perfectly fine to load the .STR with an earlier version of DataPerfect, starting with the September 1993 version. That's the version of DataPerfect 2.3 that introduced the User ID Panel facility. If your application has menus but no User ID Panel active, you can go back as far as the initial release of DataPerfect 2.3 (February 1993). But if the User ID Panel facility was used in your application, the February 1993 version won't load the .STR. In either case, you'll be working with a version of DataPerfect that has more workspace available. Though the menu facility and the User ID Panel facility both decreased DataPerfect 2.3's available workspace, the August 1994 version (the initial release of DataPerfect 2.3b) decreased it even more.

By loading the application with an earlier version of DataPerfect 2.3, you may often find that a field formula that couldn't be edited without causing an Error 104 will now let you edit it. When done editing the formula, load the application under the current version of DataPerfect 2.3 and continue your development.

When It's Okay to Overwrite an .STR

Suppose you did some work on your client's .STR tonight, outside the presence of its current data. You're pretty sure it now does exactly what your client wants. Well, here's a really simple question that inevitably arises: Can you just overwrite your client's old .STR tomorrow with the new one without regenerating indexes? The official answer has always been the same:

Never. You must always regenerate indexes after overwriting an old .STR with a new one.

The official answer has always been false, but it's certainly the safest way to handle this tomorrow in the client's office. The more accurate answer is this:

Sometimes you may overwrite an old .STR with a new one. It just depends on the circumstances.

Let's outline those circumstances for you.

If you're working on a copy of the .STR your client is currently using, you may overwrite the old .STR with the new one if, since the last time indexes were regenerated with the old .STR, no index was created or deleted with the old .STR. Further, all real fields in the new .STR are exactly the same as those in the old .STR. That is, in working on the new .STR, you didn't add, delete, or alter a real field (that includes changing its field type or length). The general rule is that you must not change the .STR in such a way that its relationship to the data files is altered. If, for instance, all you did was add or change a few reports and create a new menu, or even add a ::C field to display something interesting, you haven't done anything to the .STR that would cause a change in either the .IND file or the application's various data files.

What's the rationale behind having to regenerate indexes if, with the old .STR, you created or deleted an index since the last index regeneration? This has to do with the way DataPerfect stores index blocks in the .STR. To refresh your memory, here's that graphical representation I gave you in *The .STR File and Index Regeneration* of my **Indexes** chapter. It represents three incarnations of a changing .STR:

Old Index Block Pool	1	2	3	4	5	6	7
New Index Block Pool							

Indexes Just Regenerated

Old Index Block Pool	1	2	3	4		6	7
New Index Block Pool	8	9					

2 Indexes Created, 1 Deleted

Old Index Block Pool	1	2	3	4	6	7	8	9
New Index Block Pool								

Indexes Regenerated Again

Again, think of the above index blocks as physical places in the .STR where the .STR hooks into the .IND file. My analogy in the **Indexes** chapter was that the .STR file is a hand and the .IND file a glove, and these index blocks are hand fingers that fit and move glove fingers. The gloved hand then manipulates data files.

Our developer cleans up his .STR by running DPEXP and DPIMP on it. He loads it to make sure it's working as planned, regenerating indexes in the process.

Now he installs it at his client site. What he installs, then, is the first .STR above (that represented by the first graph).

After a few weeks of successful operation, the client calls with an enhancement request. They need a new report that sorts differently than the ones available. The developer goes to work on this, using a copy of the same .STR he initially installed above. To enhance the application the way the client wants, the developer won't need to alter the format of a field, or delete or create a field, but he must create two new indexes. He does. In this process he notices an index he never needed, so he deletes it. Now his .STR is in the state represented by the second graph. After finishing the enhancements and testing them, he runs DPEXP and DPIMP on the new .STR and loads it, regenerating indexes in the process. All looks fine, so he installs it at the client site. What he installs is the third .STR above.

Does the developer need to regenerate indexes after overwriting the old .STR with the new one? Yes. The .IND file the first .STR worked with, which is still at the client site, expects an .STR with index blocks physically arranged the way you see in the first graph. But the .STR that's attempting to hook into that .IND has its index blocks physically arranged the way you see in the third graph. The .STR hand won't fit the .IND glove. You overwrite the old .STR with the new one, and then regenerate indexes. All is well now. And, again, their .STR now has index blocks exactly where your copy of that .STR has them. Both look like you see in the third graph.

Note that in the second .STR, two indexes were created and one deleted. The index regeneration produced the third .STR. You might wonder, however, if it was the creation of the new indexes that cause the problem that requires index regeneration at the client site. Not true. If all that was done was that one index was deleted, you would still end up with an .STR that's incompatible with your client's .IND file:

Old Index Block Pool	1	2	3	4	5	6	7
New Index Block Pool							

Client's .STR

Old Index Block Pool	1	2	3	4	6	7
New Index Block Pool						

Developer's .STR

1 Index Deleted
Indexes Regenerated

In the above example, the old .IND file expects all seven index blocks the way they were in the first graph. So the .IND glove has an unfilled finger.

If all you did was create two indexes and not delete any, you get this:

Old Index Block Pool	1	2	3	4	5	6	7
New Index Block Pool							

Client's .STR

Old Index Block Pool	1	2	3	4	5	6	7	8	9
New Index Block Pool									

Developer's .STR

2 Indexes Created
Indexes Regenerated

Again, a mismatch exists between the .IND that worked with the first .STR and the new .STR. There are too many fingers on the new .STR for that .IND glove.

That covers when you can't overwrite an .STR with a new one without regenerating indexes, relative to when indexes were last regenerated. But issues other than when indexes were last regenerated push you into index regeneration after overwriting an .STR with a new one. Even if no indexes were created or removed since the last index regeneration at your client's site, you'll still need to regenerate indexes after overwriting his .STR with a new one if you created, deleted or modified a real field. In fact, if you did create, delete or modify a real field, you'll need to do more than just regenerate your client's indexes. You must do at least this:

- With your client's old .STR still in place, load his application and, in each panel you added, deleted or modified a real field, export that panel's data to a Transaction Log (**Shift-F9, A, 2**). One Transaction Log per affected panel.
- Remove all data in each such panel (**Alt-F5, 1**), followed by exiting application and deleting each such panel's data file from DOS.
- Copy over his .STR with your new one.
- Load the application and import each Transaction Log.
- Regenerate indexes for all panels.

Alternatively, export and import *all* data in the database this way:

- With your client's old .STR still in place, load his application and export all data (the entire database) to a single Transaction Log (**Shift-F9, A, 1**).
- Exit the application and delete all the application's data files, including its .IND and .TXX files.
- Overwrite the old .STR with the new one.
- Load the application and have DataPerfect create new .TXX and .IND files.
- Import the Transaction Log.

Though it can be time consuming with large databases, the latter method is better than the former one. It regenerates indexes without fear of being stopped due to duplicates. Read on for a discussion of the duplicates problem later in this chapter (*Removing Duplicates in a Panel*).

One final point about overwriting a client's .STR with a new one. When you made your changes to this .STR, you probably exported and imported it at least once, using DPEXP and DPIMP. This is good practice. But if you did, you must be sure to do the following, which can all be done with the new .STR before or after overwriting the old one:

- Reset all auto-incrementing fields.
- Re-select the User ID Panel, if active.
- Re-create all application passwords, if present.

Removing Duplicates in a Panel

As odd as it may seem, a DataPerfect database will occasionally have duplicates (duplicate entries for the same record in at least one index). I've never seen this have any consequence in real time running of the application, even when duplicates are found in panels that carry totals to other panels. Just about the only way you'll ever find out you *have* duplicates in your database is when you regenerate indexes. You'll never see them in lookups or reports, or anything else that uses an index.

If there *are* duplicates in an index, then, when you regenerate indexes in that panel, DataPerfect will stop when it comes to a duplicate record, warning you of its existence. DataPerfect will suggest that you need to redefine the indicated index so this doesn't happen. If you know there's nothing wrong with that index's definition, don't bother redefining it. That won't help.

Anyway, each time you see this error message while reindexing, reindexing stops and waits for you to hit **Enter**. This can be very time consuming if you have a lot of indexes in a particular panel, and that panel has a lot of records. DataPerfect is going to stop for every duplicate it finds in every index in which it occurs in that panel, which usually means every index in that panel.

If you elect to just hit **F1** at this point (instead of hitting **Enter**), you just trashed your .IND file and will now have to regenerate all over again. So do yourself a favor and backup before attempting index regeneration on any panel at all. Better

yet, instead of regenerating indexes the usual way (**Shift-F9, 1**), use one of the methods I describe below for removing duplicates in a database, even if you have no reason to suspect you have any duplicates. These methods automatically regenerate indexes during processing, while giving you the added bonus of making sure you have no duplicates when done. *Simple index regeneration doesn't remove duplicates.*

How Duplicates Are Created

Before I show you how to clean your database of duplicates, let's explain how this can happen in the first place. When you tell DataPerfect to delete a record, a few things must take place for the deletion to complete:

- 1. DataPerfect deletes the physical record from the data file and updates that panel's indexes.
- 2. DataPerfect then packs the data file by copying an existing record into the space created by Step 1 and updating pointers in that panel's indexes.
- 3. DataPerfect then deletes the original copy of the record that it just copied in Step 2.

Graphically, here's how this looks:

1	2	3		5	6	7		
---	---	---	--	---	---	---	--	--

Step 1
Record 4 physically deleted

1	2	3	7	5	6	7		
---	---	---	---	---	---	---	--	--

Step 2
Record 7 physically copied to fill hole

1	2	3	7	5	6			
---	---	---	---	---	---	--	--	--

Step 3
Original Record 7 is physically deleted

Most database programs don't automatically pack a data file. Traditionally, they have you run an external program (usually called PACK.EXE) to fill the holes created in that data file by Step 1. DataPerfect likes to keep data files compact, so it packs them on the fly.

Well, the problem is the split second that exists between Steps 2 and 3. During that moment there are two copies of the record that will fill the newly created space. If you turn off your computer at that moment, crash, or simply fall prey to a transient computer or DOS glitch, you have accurate indexes in that panel, but also have a duplicate copy of a record. You'll never know it exists until you attempt to

regenerate that panel's indexes. When DataPerfect comes to that record, you'll get an error message that tells you to fix this or that index. In fact, there's nothing wrong with your indexes. They've been working fine for years in that panel. You just have some duplicate records that need to be purged somehow.

Removing Duplicates in a Single Panel

Okay, here's how to remove duplicates in a single panel, which is also the way I recommend you perform index regeneration:

1. Backup up the database.
2. Load the application and export that panel's data to a Transaction Log (**Shift-F9, A, 2**).
3. Delete the data from that panel with **Alt-F5, 1** and then exit the application and delete that panel's data file from DOS.
4. Delete or rename DP{LOG}.PRB if it exists.
5. Load the application.
6. Load the problem panel. When told *The index shows that records are present in the data file, but the file is not found*, choose option 2: *Delete the Index(es) if you have deleted the file*.
7. Import the Transaction Log.
8. All duplicates, if any, will be thrown into a file called DP{LOG}.PRB for your perusal. None will land in the database.

Let me mention something important about Step 3. If you don't **Alt-F5, 1** first, any variable-length data in that panel will still reside in the .TXX file. **Alt-F5, 1** cleans that .TXX data up. Deleting the panel's data file from DOS, of course, doesn't. On the other hand, if you don't follow **Alt-F5, 1** with a DOS delete of the panel's data file, the duplicates will still be in that panel's supposedly empty data file. They're still there even though you don't see them when you load the application, and then load that panel. The .STR doesn't see them because the indexes for that panel don't see them. Likewise, **Alt-F5, 1** fails to delete these duplicates because it uses that panel's indexes to see what's there. Because the indexes don't see the duplicates, **Alt-F5, 1** doesn't delete them.

Removing Duplicates in the Entire Database

1. Backup up the database.
2. Load the application and export all data to a single Transaction Log (**Shift-F9, A, 1**).
3. Delete all data in the database (**Alt-F5, 4**).
4. From DOS, delete or rename DP{LOG}.PRB if it exists.
- 4a. Export and import your .STR with DPEXP and DPIMP.
5. Load the application and tell DataPerfect to create a new .IND.
6. Load a panel.
7. Import the Transaction Log (**Shift-F9, 8**).

8. All duplicates, if any, will be thrown into a file called DP{LOG}.PRB for your perusal. None will land in the database.

Note Step 3. Unlike **Alt-F5, 1**, which deletes a single panel's data without removing duplicates, **Alt-F5, 4** deletes all data in the database and *leaves no duplicates in data files*. It doesn't leave duplicates because, unlike its **Alt-F5, 1** counterpart, **Alt-F5, 4** doesn't use indexes to delete the records. It actually physically deletes (from DOS) the data files, the .IND file, and .TXX file, and then recreates the .IND and .TXX files.

And lastly, Step 4a isn't really necessary. But you might as well take advantage of the fact that you don't have any data in the database and clean your .STR. As you should know by now, however, if you perform this step you must be sure to do the following at some point before your client loads the application:

- Reset all auto-incrementing fields.
- Re-select the User ID Panel, if active.
- Re-create all application passwords, if present.

Cleaning the .STR Without Re-indexing

You may occasionally feel the need to clean your client's .STR. Perhaps you suspect corruption has crept into it. Whatever your reason, you'd like to be able to export and import it with DPEXP and DPIMP without having to regenerate all indexes. If you run DPIMP on the .STR in the same directory as the .IND file, DPIMP will delete the .IND file before importing the .STE.

If you haven't created or deleted an index definition since the last time indexes were regenerated with this .STR, you may clean an .STR without index regeneration this way:

- Copy the .STR to a dedicated directory.
- Run DPEXP and DPIMP on the .STR in that new directory.
- Still in the new directory, load the application and have DataPerfect create a new .TXX and .IND
- Still in the new directory, load a panel and have DataPerfect create new indexes for all panels (**Shift-F9, 1, 2**)
- Still in the new directory, reselect the User ID Panel and redefine application passwords.
- Copy the .STR back to the original directory, overwriting the old one.

Again, you can do this without problem only if you have *not* created or deleted an index definition since the last time indexes were regenerated with this .STR. Otherwise don't even *think* about doing the above.

The Big Clean: Cleaning the Entire Application

Consider doing what I call the Big Clean once a year on major applications:

1. Backup the entire database to two different media.
2. Export all data in the database to a single Transaction Log (**Shift-F9, A, 1**).
3. Exit the application and delete all data files, the .TXX file, and the .IND file. Leave the Transaction Log and the .STR alone.
4. Export and import the .STR with DPEXP and DPIMP.
5. Load the application and have DataPerfect create the .TXX and .IND files.
6. Import the Transaction Log (**Shift-F9, 8**).
7. Reselect the User ID Panel and redefine application passwords.

The above procedure does all the following:

- Cleans the .STR.
- Tightens the .TXX file.
- Removes any duplicate records in the database.
- Regenerates indexes without worrying about duplicates interfering.
- Resets auto-incrementing fields to the next highest number, no matter what they're set at just before the import.

Fixing a Corrupt .TXX File

Every version of DataPerfect prior to 2.3b (date stamped 08/19/94) contains a bug that eventually bites databases the make heavy use of the variable-length text fields. This usually shows up as low ASCII characters showing up in these fields, like ASCII 1 (☺) or 2 (☹). When it gets really bad, the corruption will show up as data from one record appearing in another record, even in fields that aren't variable-length. When the database gets to this point of corruption, all can be lost.

The developer or user can get a false sense of security by simply editing away such corruption by either deleting the corrupt record or putting it in Edit mode and using the **Del** or **Backspace** key to delete the corrupt characters. Though this appears to work, it actually just spreads the corruption to other records. This sort of corruption can't be deleted or edited away with any version of DataPerfect earlier than 2.3b.

Here's what to do:

1. Backup the database.
2. Export all data in the database to a single Transaction Log (**Shift-F9, A, 1**).

3. Exit the application and delete all data files, the .TXX file, and the .IND file. Leave the Transaction Log and the .STR alone.
4. Export and import the .STR with DPEXP and DPIMP.
5. Load the application with the current version of DataPerfect, and never use a version earlier than 2.3b again. Have DataPerfect create the .TXX and .IND files.
6. Reselect the User ID Panel and redefine application passwords.
7. You now can take one of a few different paths here:

Alternative 7a

Import the newly created Transaction Log into the database and edit your variable-length text fields as much as needed from within the database itself. Editing those corrupt fields with DataPerfect 2.3b or later won't cause more corruption, or spread existing corruption. Editing those fields with an earlier version will.

Alternative 7b

- Exit the application and load the Transaction Log into a text editor. The Transaction Log is a simple text file containing no embedded nulls or other binary information, so any standard text editor will do. Each line ends with the DOS text file conventional <CR><LF>. (This isn't true of WPMerge files. See *The Nature of the Export File, Including Some Caveats* in my **Export/Import** chapter for a discussion of that.)
- Once the Transaction Log is loaded in your text editor, you have a couple of options. The important characters to delete from the Transaction Log are the control characters (ASCII 0-31). When DataPerfect produces a Transaction Log, it converts these characters from their single-character control versions to three-character alpha strings that begin with the double-quote and backslash characters like this:

```
^A is converted to "\A
^B is converted to "\B
```

A simple way to deal with this is to simply do a Search and Replace on

"\

Replacing it with nothing. Of course that will leave the terminating A, B, etc., in the Transaction Log. Alternatively, you can do a more complete job with a Search and Replace on each of the thirty-two control conversion combinations. The thirty-two strings to Search and Replace are as follows:

<i>ASCII Decimal Value</i>	<i>Transaction Log Text Conversion</i>
0	" \@
1-26	" \A through " \Z
27	" \[
28	" \ \
29	" \]
30	" \ ^
31	" \ _

- Load the application (again, with a version of DataPerfect no earlier 2.3b).
- Import the clean Transaction Log (**Shift-F9, 8**).
- With the database loaded, manually edit, in Edit mode, data in variable-length text fields that need editing.

Crippling Applications

This section deals with how to handle sending out applications for evaluation purposes. Put another way, it involves how to cripple a DataPerfect application in a strategic way. In this situation, you also want to be able to ship the client an uncrippled .STR and nothing else, and have them overwrite their crippled one with the new one, without have to regenerate indexes or reset auto-incrementing fields.

Date Crippling

One way to do this is with a key report you code to stop working after a particular date. When the client pays you, you ship them an .STR with an uncrippled report, telling them to overwrite the old .STR with the new one.

Another way to do this with a key date field that. For instance, a date field in a Transaction Panel. If, say, you want field P1F1 to not accept dates beyond January 15, 1997, you could use a field formula like this:

```
if P1F1 <= date[15;1;1997] then P1F1
else 0 endif
```

When the client pays you, ship them an .STR with no formula on that field.

Record-Number Crippling

Here you put a range on a field. When the client pays you, ship them an .STR *without* the range on that field. If you want to put the range on an auto-incrementing field, do it before you format it as auto-incrementing (it won't work the other way around). I still suggest staying away from auto-incrementing fields, however. In this situation, it interferes with a smooth installation of the uncrippled .STR because you'll have to reset all the auto-incrementing fields in the new .STR. See *The Recursive Link* in my **Links** chapter for reason to avoid auto-incrementing fields.

Password-Protected Zip File

Here you send a crippled .STR along with a zipped good .STR (an .STR file compressed with PKZIP©). The zipped good .STR is password-protected. You give them the password when they pay, and have them overwrite the old .STR, provided you didn't include any auto-incrementing fields in either .STR, and you never made changes to either .STR that would require index regeneration (read the previous section on *When It's Okay to Overwrite an .STR* if unsure of this).

As of this writing, to create a password-protected zip file of all files in the c:\myapp directory, using *private* as the password, you do this:

```
pkzip myapp c:\myapp\*. * -sprivate
```

The zip file *myapp.zip* will now reside in the current directory and can only be unzipped this way:

```
pkunzip myapp -sprivate
```

Try it. It's a nice feature of PKZip©.

As of this writing, DataPerfect is a DOS character-based database manager in a Windows graphics world, with no manufacturer supporting it. So where does this leave the DataPerfect application developer?

Support Avenues

Though ©Novell, Inc. owns DataPerfect, they don't support it. This is one of the conditions of their gracious release of DataPerfect to the public as free copyrighted shareware. Under that agreement, Lew Bastian, is allowed to continue to fix and enhance DataPerfect as he sees fit. Though it has no telephone support at this time, a community of very helpful DataPerfect application developers volunteer support for DataPerfect through two online channels: CompuServe© and the Internet.

CompuServe Support

Peer support of DataPerfect on CompuServe has been around for many years and, as of this writing, is very active. Currently, DataPerfect's home on CompuServe is Section 11 of WordPerfect Users Forum (GO WPUSERS). I've been a sysop in that forum since about 1991, mainly responsible for managing its DataPerfect section. The DataPerfect message section (Section 11) of WordPerfect Users Forum is complemented by a library section as well (Library 11), where users can find many files related to DataPerfect application development. This library always includes the latest version of DataPerfect, including its various utilities and runtime executables. This is also a good place to find many user-contributed files.

Internet Support

There are two ways to access Internet support for DataPerfect: the DataPerfect Users Discussion Group Home Page and the DATAPERF LISTSERV.

DataPerfect Users Discussion Group Home Page

Fellow DataPerfect application developer, John Cabrera, currently manages the DataPerfect Users Discussion Group Home Page:

`http://members.aol.com/compusof1/`

John's page offers a wealth of information about DataPerfect, along with links to allow easy downloading of various DataPerfect-related files, including its latest version. If you have any problems accessing this page, let Web Master John Cabrera know. He can be reached directly via his Internet address:

compusof1@aol.com

DATAPERF LISTSERV

Web Master John Cabrera manages the DATAPERF LISTSERV. A LISTSERV is a mailing list dedicated to a particular group of users—in this case, DataPerfect users. Subscribers to that group all help each other out. It's free to subscribe. You can subscribe to it via John's Web page, or just send an Internet eMail message to

listproc@listproc.wsu.edu

Place the following in its message body:

subscribe dataperf YourFirstName YourLastName

That should be the first and only line of the message. You'll then be subscribed to the group and will receive, in your eMail box, all messages posted to that group since the last time you checked eMail box. To reply or compose a message to the group, address messages to this address:

dataperf@listproc.wsu.edu

Note that that's not the same address you send your subscription request to.

The DataPerfect Users Cooperative

There's a movement afoot to provide DataPerfect application developers the tools to port their existing DOS applications to various Windows environments, and to provide these developers tools that will allow them to develop new applications in these environments the way they're used to doing in DataPerfect. As of this writing, this movement is called the DataPerfect Users Cooperative, rallying most of its support from the CompuServe forum and LISTSERV mentioned above. It has already elected a five-person board of directors. This organization is in its infancy, changing rapidly. If you're interested in joining, or at least getting more information on it, contact James Trybalski via eMail at

103000.3437@compuserve.com

Please keep abreast of this development by subscribing to the DATAPERF LISTSERV and, if a CompuServe member, grabbing messages in Section 11 of WPUSERS forum.

- .IND file
 - introduction 28
- .STE editing caveats 407
- .STR file
 - and index regeneration 98
 - cleaning the .STR without re-indexing 425
 - index block and 98, 99, 419-421
 - introduction 27
 - when it's okay to overwrite 418
- .TMP files 29
- .TXX file
 - fixing a corrupt 426
 - introduction 28
- ::C field 15, 18, 22, 24, 31, 36, 38, 39, 41-44, 46, 47, 49, 52, 67, 88, 100, 103-105, 112, 113, 141, 142, 144, 155, 156, 241, 274, 280, 289, 366, 372, 373, 376, 386, 390-392
- ::E field 39
- ::H field 39
- ::I and ::J fields 21, 25, 39, 57, 133-137, 143, 422, 425, 426, 428, 429
- ::M field 40
 - controlling record creation with 388
- ::N field 38, 42, 43, 47, 58, 85
- ::1-9 (Truncate Leading and Trailing Blanks and Leave n Space 199
- ::B (Truncate Both Leading and Trailing Blanks) 199
- ::C (Center Characters) 200, 201
- ::D (Delete All Blanks) 199
- ::E (Delete Zero Subfields from the End) 199
- ::L (Left Adjust Characters) 200, 201
- ::N (New Occurrence of Field) 200, 205
 - examples 205
- ::P (Postal Bar Code) 200
- ::Q (Enclose Alphanumeric Fields in Double Quotes) 200
- ::R (Right Adjust Characters) 200
- ::S (Suppress Leading Blanks) 198
- ::T (Truncate Trailing Blanks) 198
- /Z command line switch 375
- A field 32
- Access Report List, menu facility 353
- Acknowledgments xi
- application files
 - Big Clean: Cleaning the Entire Application 426
 - cleaning the .STR without re-indexing 425
 - crippling applications 428
 - removing duplicates in a panel 422
 - when it's okay to overwrite an .STR 418
- application maintenance
 - .STE editing caveats 407
 - Big Clean: Cleaning the Entire Application 426
 - cleaning the .STR without re-indexing 425
 - compatibility between different versions of DataPerfect 417
 - crippling applications 428
 - DPDiagnostics 409
 - DPEXport 407
 - DPImport 407
 - DPImport caveats 408
 - DPOrder 411
 - error messages, DPDiagnostics 410
 - exporting and importing transaction logs 335
 - importing reports 408

- optimization messages,
 - DPDiagnostics 410
 - removing duplicates in a panel
 - 422
 - STE Manager 413
 - upgrading a client to a new
 - version of DataPerfect 417
 - warning messages,
 - DPDiagnostics 410
 - when it's okay to overwrite an
 - .STR 418
- application password
 - introduction 349
 - rights granted 349
 - User ID Panel vis a vis
 - application passwords 363
 - when DataPerfect prompts for
 - application passwords 350
- APPLY.FORMAT function
 - explanation 291
- Auto-Display Record 18
- Auto-Edit, Auto-Create Menu 19
- auto-enter field 39
- auto-incrementing field
 - conditional incrementation 133, 137
 - reasons for avoiding 135
- Auto-Save 18
- backward-referring computed fields 44
- backward-referring non-updatable fields
 - 47
- Basic Default Panel 376
- Big Clean: Cleaning the Entire
 - Application 426
- Browse mode 15
- Browse mode lookups 16
- carriage returns and spaces in formulas
 - 300
- cascade update
 - vs. Keep A Total 155
- case variable 289
- CASES statements vs. IF-THEN
 - statements 289
- caveats
 - .STE editing 407
 - clipboard 346, 348
 - compatibility between different
 - versions of DataPerfect 417
 - data link 125, 129
 - DPIImport 408
 - dummy reports 244
 - exception list indexes 101
 - formula error messages 282
 - Keep A Total 151
 - merge file 331
 - Open Filename in Report
 - Variable 270
 - regarding two-digit years 55
 - Run Report option 355
 - time fields and uniqueness 57
 - two-digit years 55
 - User ID Panel 359
- center output 200
- chapters, outline 2
- Choose Index, menu facility 354
- choosing between ::C and ::N fields 41
- choosing between G fields and N fields
 - 49
- cleaning the .STR without re-indexing
 - 425
- clipboard
 - a caveat regarding field formulas
 - and help screens 348
 - in a Specify Formula screen 344
 - in Define Panel mode 343
 - in Report Definition mode 345
 - introduction 343
 - vs. screen capture 343
- closing off a data link 374
- closing off access to the panel list 374
 - /Z command line switch 375
 - menu facility 375
- color, panel, change 18
- Comma Delimited format 341
- comma delimited output
 - ::Q (Enclose Alphanumeric
 - Fields in Double Quotes) 200

- compatibility between different versions of DataPerfect 417
- Complex Default Panel 377
- CompuServe support 431
- computed field
 - and DataPerfect's work space 49
 - backward-referring 44
- Conditional Page Eject, report option 216
- CONTAINS function
 - explanation 299
- controlling data entry with reports
 - editing 401
 - providing an undelete 395
 - record creation 399
 - record deletions 393
- controlling record creation with the ::M field 388
- controlling record deletions with DPMouse 391
- CONVERT function
 - explanation 291
- Cooperative, DataPerfect Users 432
- corrupt .TXX file, fixing 426
- counter
 - report variable 225
- create
 - controlling record creation with indexes 387
 - controlling record creation with the ::M field 388
 - Create Record From Panel List, report option 215, 218
 - Create Secondary Merge Report, report option 215
 - data link 118
 - exception list 21
 - field 11
 - index 12
 - Keep A Total 148
 - menu text 351
 - panel link 113
 - panel text 10
 - reports that control record creation 399
- Create mode 15
- Create Record From Panel List, report option 215, 218
- Create Secondary Merge Report, report option 215
- Create/Edit Menu Text, menu facility 351
- crippling applications 428
- data link
 - caveat 129
 - choosing between the panel link and 128
 - closing off 374
 - create 118
 - data link options caveats 125
 - define 118
 - introduction 117
 - options 122
- data link options
 - Auto-Create 123
 - caveats 125
 - Check During Data Entry Off 123
 - Edit Target Field/Target Index/Field List 122
 - No Create, No Access 123
 - Prompt-Create 123
 - Remove Data Link 122
- Data Link Subgroup Lookup and the USER.FIELD function 367
 - explanation 72
- DATAPERF LISTSERV 431, 432
- DataPerfect Users Cooperative 432
- DataPerfect Users Discussion Group Home Page 431
- date field
 - and the year 2000 problem 55
 - as a special numerical field 54
 - international dates 56
 - report option 211
 - two-digit vs. four-digit years 55
 - when not to use for dates 51
- define
 - data link 118
 - field 11
 - index 12, 20

- Keep A Total 148
- link 20
- panel link 113
- DEFINE FIELD Options 21, 25
 - Define Initial Formula or Define Field Formula 22
 - Define Search Field List 25
 - Initial Value 25
 - Initialize at Create/Save/Change 23
 - Keep A Total 26
 - Lookup Field List 22
 - Range Check 25
 - Remove Last Total 26
 - Validation Time 25
- Define Menu screen, menu facility 350
- Define Panel Option
 - Auto-Edit, Auto-Create Menu 19
 - Auto-Save 18
 - Change Color 18
 - Change Edit Order 19
 - Edit Filename 17
 - Edit Panel Name 17
 - Recompute Field Offsets 19
- delete
 - controlling record deletions with DPMouse 391
 - Delete an Existing Entry, menu facility 358
 - Delete Record, report option 218
 - field, notes on 66
 - index 21
 - reports that control record deletions 393
- Delete an Existing Entry, menu facility 358
- disk file mode 160
 - Open Filename in Report Variable 267
 - Turn File On or Off, report option 212
- display
 - sneaking print mode indicators into panel fields 202
- display modifier 14
- Do Report in Subgroups, report option 215
- DOS delimited text 337
 - Comma Delimited format 341
- DPDiagnostics (DPDIAG.EXE) 409
 - error messages 410
 - optimization messages 410
 - warning messages 410
- DPEXport (DPEXP.COM) 407
- DPImport (DPIMP.COM) 407
 - caveats 408
- DPMouse
 - and field protection 390
 - controlling record deletions with 391
 - using to conditionally close a panel link 390
- DPOrder (DPO.EXE) 411
- dummy report example
 - a report that branches to other reports 245
 - gathering preliminary information from various panels 253
 - prompting the user with the number of hits 256
 - report that double-sorts records 259
- duplicates, removing from panel 422
- edit
 - exception list 21
 - field format 11
 - index field list 21
 - panel text 10
- Edit an Existing Entry, menu facility 357
- Edit Key Word, menu facility 354
- Edit mode 15
- Edit Order, change 19
- Edit Password, menu facility 353
- Edit Report Form 164
- Edit Report Form Screen
 - knowing your place 174
- Edit Report Form Screen sections
 - delineated 166
- Final Footer 170
- First Page Header 166

- Other Page Header 167
- Page Footer 170
- Report Body 168
- The Report Algorithm 170
- Two-Level Report Footer 170
- Two-Level Report Header 167
- elapsed time
 - calculating across the 24-hour barrier 58
 - the simple case 58
 - using MOMENT and MODULO 61
- Eliminate Line if Blank, report option 213
- Epilogue 431
- error messages, DPDiagnosics 410
- exception list
 - aiding computed fields in parent panel 103
 - aiding lookups with 101
 - dividing data file record access with 105
 - exception list index bug in version 2.2 107
 - lowest numbered index: a caveat 101
 - speeding reports with 102
 - what it is 99
- export, import
 - Comma Delimited format 341
 - DOS delimited Text 337
 - reasons for exporting or importing data 329
 - transaction log 335
 - WordPerfect Merge files 330
- expression 281
 - APPLY.FORMAT function 3, 52, 53, 75, 202, 203, 291-293, 300, 301
 - CASES vs. IF-THEN 289
 - identity operator 47, 285, 286, 288, 302
 - perfect matches and the identity operator 286
 - USER.FIELD function 49, 142
- F field 35
- field
 - ::T field 172, 173, 189, 198-202, 222
 - auto-incrementing 3, 22, 40, 94, 97, 133, 336, 429
 - choosing between ::C and ::N fields 41
 - choosing between G fields and N fields 49
 - code 32
 - create 11
 - deleting, notes on 66
 - display modifier 14, 15, 292, 388, 403
 - DPMouse and field protection 390
 - edit format 11
 - fundamentals 31
 - index field list 13, 20, 21, 41, 46, 90-93, 99, 104, 114, 194, 333, 334
 - name 32
 - sneaking print mode indicators into panel fields 202
 - truncate 172, 198, 199, 203, 261
 - type 32
 - variable-length text 3, 28, 31, 32, 203-205, 341, 376
- Field Offsets, Recompute 19
- Fields: Introduction 31
- Fields: Issues 41
- file
 - .IND 28-30, 89, 90, 98, 99, 408, 419, 420, 422, 425-427
 - .STR 19, 27-30, 89, 90, 98, 99, 135, 137, 281, 348, 359, 375, 408, 413, 414, 417, 419, 429
 - .TMP 29
 - .TXX 4, 8, 9, 27-31, 33, 270, 376, 422, 424-427
 - application files 27
 - Big Clean: Cleaning the Entire Application 426
 - cleaning the .STR without re-indexing 425

- crippling applications 428
- fixing a corrupt .TXX file 426
- program 27
- removing duplicates in a panel 422
- specifications 30
- when it's okay to overwrite an .STR 418
- File, Turn On or Off, report option 212
- Files and Specifications 27
 - application files 27
 - program files 27
 - specifications 30
- flat-file DBMS vs. relational DBMS 109
- footer
 - final 166, 170, 171, 178, 190, 215, 217, 218, 227, 232-234, 242-244, 318, 346, 347
 - Include After Last Record, report option 218
 - Number of Records in Report 218
 - Number of Records on Page, report option 218
- format
 - alphanumeric field 12, 22, 32, 290
 - choosing between ::C and ::N fields 41
 - choosing between G fields and N fields 49
 - D field 35, 51, 53, 54, 97
 - date field 2, 15, 35-38, 42, 43, 51-57, 59, 61, 64, 78, 87, 94, 97, 98, 103-105, 144, 152-154, 227, 228, 241, 242, 256, 257, 259, 260, 294, 312, 326, 376, 377, 428
 - field 14
 - field type 32
 - G field 34, 35, 49, 52, 97, 114, 198
 - H field 14, 34, 35, 48, 50, 51, 65, 94, 97, 104-106, 144, 153-155, 197, 273, 404, 410
 - N field 15, 24, 33, 34, 39, 41, 49, 50, 52, 114, 143, 151, 205, 273, 339, 358, 360, 364, 390
 - numeric field 22, 33, 34, 199
 - time field 37, 38, 56-61, 64, 169
 - variable-length text field 8, 28, 30, 32, 33, 200, 203, 269, 295, 299, 424, 426-428
- formula
 - a caveat regarding field formulas and help screens 348
 - CASES statements vs. IF-THEN statements 289
 - elapsed time 57-60, 62-64
 - error messages: a warning 282
 - expression 281
 - IF-THEN statements vs. CASES statements 289
 - Initial Formula, controlling data with 375
 - introduction 281
 - legal value 282
 - note about DataPerfect's notion of truth 321
 - operand 281
 - operator 281
 - perfect matches and the identity operator 286
 - screen capture, using in Specify Formula screen 345
 - spaces and carriage returns in 300
 - Specify Formula Screen 22, 23, 32, 102, 134, 141, 145, 211, 219, 221, 223, 225, 267, 281, 282, 284, 304, 344-346, 348, 418
 - troubleshooting 301
 - well-formed formula 282
- function
 - CONTAINS 2, 26, 93, 104, 109, 117, 138, 140, 156, 197, 199, 240, 271, 277-279,

- 299, 300, 302, 336, 384, 392, 410, 426
- CONVERT 52, 53, 55, 63, 166, 203, 212, 291-298, 341, 417
- SUBFIELD 3, 70, 294, 295, 298
- Go to Panel List, menu facility 356
- Go to Panel, menu facility 352
- Go to Report List, menu facility 357
- H field
 - in reports 50
- header
 - Include Before First Record, report option 216
 - Skip if Start of Two Level, report option 216
 - Two Level Report 220
- help screens, a caveat 348
- hidden field 39
- Hidden Panel 385
- hiding data entry 403
- identity operator
 - perfect matches 286
- IF-THEN statements vs. CASES statements 289
- import, export
 - Comma Delimited format 341
 - DOS delimited Text 337
 - reasons for exporting or importing data 329
 - transaction log 335
 - WordPerfect Merge files 330
- importing reports 408
- Include After Last Record, report option 218
- Include Before First Record, report option 216
- Include Subreport, report option 229
- incrementation
 - absolute 24, 134, 136-140
 - absolute incrementation using a recursive panel link 134
 - absolute incrementation using recursive links on a Network 137
 - conditional incrementation using a recursive panel link 133
 - reasons for avoiding auto-incrementing fields 135
- index
 - Choose Index, menu facility 354
 - cleaning the .STR without re-indexing 425
 - controlling record creation with indexes 387
 - define 12, 20
 - delete 21
 - exception list 20, 21, 48-50, 64, 78-80, 88, 89, 94, 99-107, 116, 163, 167, 170-174, 245, 252, 265, 266, 318, 319, 326, 327, 330, 386, 397, 398, 410, 411
 - how indexes sort 90
 - introduction 89
 - number 159, 162, 163, 189, 330, 338, 339, 341, 354, 355, 398
 - picking fields for the index field list 92
 - primary sorting field 78, 86, 87, 193, 194, 196, 259, 307-309, 312, 327
 - primary sorting field and the Skip To iteration code 307
 - reports that double-sort records 259
 - reverse index 94
 - reverse sorting by number 94
 - sorting backwards 94
 - uniqueness 92
- index number, report 163
- index regeneration
 - .STR File and 98
 - cleaning the .STR without re-indexing 425
- indicator
 - display mode indicator 38, 173, 185
 - print mode 51, 189, 198-200, 202-205, 292

- print mode indicator 3, 51, 172, 173, 198-204, 292
- Initial Report Definition Screen
 - Edit Report Form 164
 - Index Number 163
 - Search Conditions 163
 - Sort Direction 163
- Initial Value 25
- Initialize at Create/Save/Change 23
- international dates 56
- Internet support 431
- Introduction 1
- iteration control
 - combining Skip To with Stop If 309
 - how report lookups display 318
 - introduction 303
 - note about DataPerfect's notion of truth 321
 - Repeat If 307, 320, 324, 325
 - Repeat If, explanation 320
 - Single Record Report From Lookup off a Menu 317
 - Skip If 3, 179, 209, 216, 259, 286, 304-308, 322, 324, 410
 - Skip If, explanation 303
 - Skip To 182, 209, 214, 222, 223, 225, 255-261, 263, 264, 280, 303-315, 321, 326-328, 410
 - Skip To, explanation 306
 - skipping records, introduction 223
 - Stop If 244, 246, 250, 260, 261, 269, 270, 280, 306, 307, 309-312, 315, 316, 324, 327, 328, 361
 - Stop If, explanation 306
 - troubleshooting 328
 - User Chooses Next Record By Lookup 84, 87, 254, 256, 273, 275, 313, 314, 316, 317, 327, 328, 355, 393-395, 398, 402, 403
 - User Chooses Next Record By Lookup, explanation 313
- iteration control example
 - date-range report 326
 - getting report to continue after last record in lookup selected 327
 - limiting a report to a particular number of records 325
 - limiting a report to one record 324
 - monthly statements for accounts with positive balance 326
 - report that prints particular number of iterations per record 325
- Iteration Control, report option 215
- Keep A Total
 - caveat 42, 151
 - defining 148
 - introduction 147
 - using it to update records in foreign panels 151
 - vs. cascade update 155
 - without a parent record to receive the total 150
- keeping a saved field from being edited 372
 - with DPMouse© 390
- Labels, report option 216
- Launch Shell Macro, menu facility 357
- left-align output 200
- legal value 282
- link
 - action, panel link 116
 - cascade update/delete 46, 121, 122, 125, 130, 131, 143, 144, 146
 - caveat regarding data links 129
 - choosing between the panel link and the data link 128
 - controlled panel link access to subrecords 366
 - controlling the data link's Create/Edit mode lookup display 118

- data link 20, 40, 41, 72-74, 111, 115-132, 142, 143, 145, 149, 190, 229, 274-276, 345, 367, 368, 373-375, 382, 384, 388
- data link options 122
- data link, create 118
- define 20
- field list, panel link 114
- flat-file DBMS vs. relational DBMS 109
- index, panel link 114, 116
- link index 77-80, 111, 113, 114, 143, 247
- many-to-many 110, 111
- many-to-one 110
- one-to-many 110
- one-to-one 110
- options menus 120
- panel link 20, 24, 40, 41, 43-46, 52, 70-72, 75-79, 81-86, 88, 94, 95, 97, 98, 103-107, 111-122, 125, 128-130, 132-134, 136-145, 149, 156, 175, 188-190, 193, 229, 230, 233, 236, 237, 240, 241, 272, 274-277, 280, 293, 345, 352, 366, 367, 374, 376, 377, 379-381, 384-386, 390-392, 410
- panel link options 120
- panel link, create 113
- panel link, making safer 141
- pyramidal design as a data integrity strategy 383
- recursive 3, 24, 40, 57, 85, 133, 134, 136-140, 145, 380, 381, 429
- the two types of DataPerfect links 111
- troubleshooting 143
- using DPMouse to conditionally close a panel link 390
- virtual 137, 140, 141, 175, 188, 189, 217, 229, 239-243, 246-248, 252, 254, 255, 257, 278, 360, 400
- virtual link vs. Subreport Using Virtual Link 140
- link options menus 120
- linkage
 - many-to-many 110, 111
 - many-to-one 110
 - one-to-many 110
 - one-to-one 110
- lookup
 - Browse mode 16
 - Data Link Subgroup Lookup 72, 73, 125-127, 145, 367, 368
 - field list 21-23, 25, 42, 69-72, 74-76, 79, 81-87, 96, 318, 319, 328, 375, 396
 - fundamentals 69
 - hidden field, on a 83
 - making lookups look better (Browse mode) 74
 - non-updatable field, on a 85
 - saving a lookup definition, a note about 85
 - Smart Lookup Algorithm 79, 80
 - Smart Lookups 2, 72, 75, 77, 81, 85-87, 95-97, 145, 319, 410
 - strategy in defining a Browse mode lookup 81
 - subfield lookups 70
 - troubleshooting 86
- maintenance
 - Big Clean: Cleaning the Entire Application 426
 - cleaning the .STR without re-indexing 425
 - compatibility between different versions of DataPerfect 417
 - crippling applications 428
 - removing duplicates in a panel 422

- upgrading a client to a new version of DataPerfect 417
 - when it's okay to overwrite an .STR 418
- maintenance
 - .STE editing caveats 407
 - application 2, 4, 99, 136, 330, 336, 365, 407, 417
 - DPDiagnostics 409
 - DPEXport 407
 - DPImport 407
 - DPImport caveats 408
 - DPOrder 411
 - error messages, DPDiagnostics 410
 - exporting and importing transaction logs 335
 - importing reports 408
 - optimization messages, DPDiagnostics 410
 - STE Manager 413
 - warning messages, DPDiagnostics 410
- many-to-many 110, 111
- many-to-one 110
- menu facility
 - Access Report List 353
 - caveats regarding Run Report option 355
 - Choose Index 354
 - controlling access to data accessed from a menu 365
 - create/edit menu text 351
 - Define Menu screen 350
 - Delete an Existing Entry 358
 - Edit an Existing Entry 357
 - Edit Key Word 354
 - Edit Password 353
 - Go to Panel 352
 - Go to Panel List 356
 - Go to Report List 357
 - introduction 350
 - Launch Shell Macro 357
 - move menu prompt 352
 - Normal Report Mode 355
 - Panel Access Rights 353
 - Password option for reports 355
 - Restrict Modification to First Level 353
 - Run Report 354
 - Submenu 357
 - Subset 354
 - troubleshooting 368
 - User Set-Up option for reports 355
 - using to prevent creation via link 391
- merge
 - Create Secondary Merge Report, report option 215
 - WordPerfect Merge files 330
- merge file vs. transaction log, strategies 336
- moment function
 - special use in reports 64
- Move the Menu Prompt, menu facility 352
- Must Be Updated fields (::M) 40
 - controlling record creation with 388
- Number of Records in Report, report option 218
- Number of Records in Section, report option 218
- Number of Records on Page, report option 218
- numeric field
 - choosing between G fields and N fields 49
- one-to-many 110
- one-to-one 110
- Open Filename in Report Variable, report option 212, 267
- operand 281
- operator 281
 - MODULO 61, 63, 154, 262, 272, 320, 325, 405
 - perfect matches and the identity operator 286

- optimization messages, DPDiagnosics 410
- Other Page Header
 - Include Before First Record, report option 216
 - Skip if Start of Two Level, report option 216
- overwriting an .STR 418
- Page Eject vs. Skip to Bottom of Page 214
- Page Eject, report option 214
- Page Number, Set, report option 211
- panel
 - Basic Default Panel 376
 - clipboard, using in Define Panel mode 343
 - closing off access to the panel list 374
 - Complex Default Panel 377
 - creating panel text 10
 - Go to Panel List, menu facility 356
 - Go to Panel, menu facility 352
 - Hidden 385
 - hiding data entry 403
 - moving 11
 - Panel Access Rights, menu facility 353
 - Printer Control Panel 271
 - removing duplicates in 422
 - Report Records Panel 260
 - screen capture, using in Panel Define mode 344
 - sizing 11
 - sneaking print mode indicators into panel fields 202
 - text, creating 10
- Panel Access Rights, menu facility 353
- panel filename, edit 17
- panel link
 - a note about a panel link's index 116
 - absolute incrementation using a recursive panel link 134
 - absolute incrementation using recursive panel links on a network 137
 - action 116
 - choosing between the data link and 128
 - conditional incrementation using a recursive panel link 133
 - controlled panel link access to subrecords 366
 - create 113
 - define 113
 - field list 114
 - index 114
 - making safer 141
 - options 120
 - recursive 133
 - using DPMouse to conditionally close 390
 - virtual link vs. Subreport Using Virtual Link 140
- panel link options
 - Cascade Off, Cascade Update, Cascade Update/Delete 121
 - Create/Edit Window Field List 121
 - Define Lookup List 121
 - Define Related Records Window 121
 - Delete Window 121
 - Display/Hide Link 121
 - Edit Target Field/Target Index/Field List 121
- panel name, edit 17
- Panel Option
 - Auto-Display Record 18
- panel text, creating 10
- password
 - application 349, 350, 356, 358, 359, 363, 364, 368, 422, 425-427
 - application, introduction 349
 - assigning a password to a report on a menu 355
 - definer 364

- Edit Password, menu facility 353
- hiding data entry while entering 403
- password-protected Zip file 429
- rights granted by application passwords 349
- User ID Panel vis a vis application passwords 363
- perfect matches and the identity operator 286
- pick list field 373
 - closing off a data link to protect a pick list panel 374
- preventing inadvertent editing 371
 - with DPMouse© 372
- primary sorting field
 - with Skip To iteration control code 307
- print margins 160
- print mode indicator
 - sneaking print mode indicators into panel fields 202
- print mode indicators that alter field output spacing 198
 - ::1-9 (Truncate Leading and Trailing Blanks and Leave n Space 199
 - ::B (Truncate Both Leading and Trailing Blanks) 199
 - ::E (Delete Zero Subfields from the End) 199
 - ::S (Suppress Leading Blanks) 198
 - ::T (Truncate Trailing Blanks) 198
- print mode indicators that don't alter field output spacing 199
 - ::C (Center Characters) 200
 - ::D (Delete All Blanks) 199
 - ::L (Left Adjust Characters) 200
 - ::N (New Occurrence of Field) 200
 - ::P (Postal Bar Code) 200
 - ::Q (Enclose Alphanumeric Fields in Double Quotes) 200
 - ::R (Right Adjust Characters) 200
- Print Report Variable, report option 211
- Print, Turn On or Off, report option 212
- printer control
 - Open Filename in Report Variable, report option 267
 - Open Filename in Report Variable, report variable 212
 - Printer Control Panel 271
- Printer Control Panel 271
- Printer Control, report option 212
- program files 27
 - runtime 27, 391, 431
- Prompt for Report Variable, report option 214
- protection
 - /Z command line switch 375
 - Basic Default Panel 376
 - closing off a data link 374
 - closing off access to the panel list 374
 - Complex Default Panel 377
 - controlled panel link access to subrecords 366
 - controlling access to data accessed from a menu 365
 - controlling data entry of Zip Codes with ZipKey© 382
 - controlling record creation with indexes 387
 - controlling record creation with the ::M field 388
 - controlling record deletions with DPMouse 391
 - controlling user access 361
 - crippling applications 428
 - DPMouse xiii, 4, 107, 142, 143, 350, 366, 367, 372-375, 390-392

- DPMouse and field protection 390
- Hidden Panel 385
- initial formula 21-23, 25, 276, 375, 376
- Initial Formula or Value 375
- keeping a saved field from being edited 372
- keeping subpanel data current 43
- making panel links safer 141
- password-protected Zip file 429
- pick list field 373
- preventing inadvertent editing 371
- pyramidal design as a data integrity strategy 383
- tracking user activity with the USER.FIELD function 360
- User ID Panel vis a vis application passwords 363
- user-stamping records 364
- using DPMouse to conditionally close a panel link 390
- using menu facility to prevent creation via link 391
- ZipKey©, controlling data entry with 382
- pyramidal design as a data integrity strategy 383
- Ralph Alvy 1
- range check 25
- real field 41
- Record Number, report option 218
- recursive link
 - absolute incrementation using a recursive panel link 134
 - absolute incrementation using recursive links on a network 137
 - conditional incrementation using a recursive panel link 133
- recycled report variable 227
- regeneration, index 2, 98, 419-425, 429
 - .STR File and 98
 - cleaning the .STR without re-indexing 425
- relational DBMS vs. flat-file DBMS 109
- remove last total 26
- removing duplicates in a panel 422
- Repeat If
 - explanation 320
- report
 - ;;1-9 (Truncate Leading and Trailing Blanks and Leave n Spaces) 199
 - ;;B (Truncate Both Leading and Trailing Blanks) 199
 - ;;C (Center Characters) 200
 - ;;D (Delete All Blanks) 199
 - ;;E (Delete Zero Subfields from the End) 199
 - ;;L (Left Adjust Characters) 200
 - ;;N (New Occurrence of Field) 200
 - ;;P (Postal Bar Code) 200
 - ;;Q (Enclose Alphanumeric Fields in Double Quotes) 200
 - ;;R (Right Adjust Characters) 200
 - ;;S (Suppress Leading Blanks) 198
 - ;;T (Truncate Trailing Blanks) 198
- Access Report List, menu facility 353
- algorithm 170
- branching to other reports 245
- caveats regarding Run Report option on a menu 355
- clipboard, using in Report Definition mode 345
- date-range report 326
- destination 159
- disk file mode 160
- dummy report 175, 243-246, 248, 252-254, 256, 257, 261
- Edit Report Form 164
- Edit Report Form Screen sections delineated 166

- finding what panel a report is based on 162
- gathering preliminary information from various panels 253
- general theory in creating 171
- getting report to continue after last record in lookup selected 327
- Go to Report List, menu facility 357
- importing reports 408
- index number 163
- Initial Report Definition Screen 158
- introduction to reports 157
- iteration control 3, 84, 163, 167, 172-174, 182, 209, 215, 223, 224, 243, 253, 257, 259, 303, 305-307, 315, 317, 321, 322, 324, 327, 355, 361
- knowing your place in the Edit Report Form 174
- labels, printing 9-12, 16, 162, 166, 209, 216, 217
- limiting a report to a particular number of records 325
- limiting a report to one record 324
- monthly statements for accounts with a positive balance 326
- name 159
- note about DataPerfect's notion of truth 321
- Open Filename in Report Variable, report option 267
- primary sorting field 193
- print margins 160
- print mode indicators that alter field output spacing 198
- print mode indicators that don't alter field output spacing 199
- Printer Control Panel 271
- printing totals at the top of the invoice 235
- prompting the user with the number of hits 256
- Repeat If, explanation 320
- report variables and truth values 323
- reports that control editing 401
- reports that control record creation 399
- reports that control record deletions 393
- Run Report, menu facility 354
- search conditions 163
- Skip If, explanation 303
- Skip to Bottom of Page 182, 209, 214, 222, 225, 304, 305
- Skip To, explanation 306
- sort direction 163
- Stop If, explanation 306
- subgroup report 51, 72, 73, 125-127, 145, 182-184, 187, 193-196, 199, 209, 215, 295, 296, 367, 368
- subreport 48, 87, 103, 112, 140, 141, 158, 174, 175, 187-196, 217, 218, 229-236, 238-257, 259, 261-265, 275, 278, 280, 305-308, 310-312, 314, 315, 318, 326, 327, 346, 399, 400, 410
- subreport that depends on existing link 229
- subreport, introduction 229
- subreports as subroutines 232
- subreports: going from version 2.2 to 2.3 230
- that branches to other reports 245
- that double-sorts records 259
- that prints a particular number of iterations per record 325
- Two Level 29, 220, 315
- Two-Level Reports in Subreports 192

- variable-length text fields in 203
- virtual link vs. Subreport Using Virtual Link 140
- report field
 - ::1-9 (Truncate Leading and Trailing Blanks and Leave n Spaces 199
 - ::B (Truncate Both Leading and Trailing Blanks) 199
 - ::C (Center Characters) 200
 - ::D (Delete All Blanks) 199
 - ::E (Delete Zero Subfields from the End) 199
 - ::L (Left Adjust Characters) 200
 - ::N (New Occurrence of Field) 200
 - ::P (Postal Bar Code) 200
 - ::Q (Enclose Alphanumeric Fields in Double Quotes) 200
 - ::R (Right Adjust Characters) 200
 - ::S (Suppress Leading Blanks) 198
 - ::T (Truncate Trailing Blanks) 198
 - bar coded Zip Code output 200
 - F field 197
 - print mode indicator 198
 - print mode indicators that alter field output spacing 198
 - print mode indicators that don't alter field output spacing 199
 - variable-length text fields in reports 203
 - Zip Code output, bar code 200
- report option
 - Conditional Page Eject 216
 - Create Record From Panel List 215, 218
 - Create Secondary Merge Report 215
 - Date, Select 211
 - Delete Record 218
 - Do Report in Subgroups 215
 - Eliminate Line if Blank 213
 - Include After Last Record 218
 - Include Before First Record 216
 - Iteration Control 215
 - Labels 216
 - Number of Records in Report 218
 - Number of Records in Section 218
 - Number of Records on Page 218
 - Open Filename in Report Variable 212
 - Page Eject 214
 - Page Number, Select 211
 - Page Number, Set 211
 - Print Report Variable 211
 - Printer Control 212
 - Prompt for Report Variable 214
 - Record Number 218
 - Select Report Field 211
 - Skip if Start of Two Level 216
 - Skip to Bottom of Page 214
 - Store Value in Report Variable 211
 - Subreports 217
 - Time, Select 211
 - Turn File On or Off 212
 - Turn Print On or Off 212
 - Two-Level Report 215
- report options menus 209
- Report Records Panel 260
- report structure
 - Subgroup Reports in Subreports 194
 - Subreport 184
- report variable
 - counter 225
 - introduction 219
 - Open Filename in Report Variable, report option 212, 267
 - Print Report Variable, report option 211
 - printing data not already in fields 220

- Prompt for Report Variable,
 - report option 214
- recycled 227
- self-referencing 225
- Store Value in Report Variable,
 - report option 211
- truth value 323
- Report Variable, Open Filename in,
 - report option 212
- Report Variable, Print, report option 211
- Report Variable, Prompt for, report
 - option 214
- report, general structure
 - Basic Report 177
 - primary sorting field 193
 - Subgroup Report 183
 - Subreport 184
 - Two-Level Reports 181
 - Two-Level Reports in Subreports 192
- requirements
 - application files 27
- Restrict Modification to First Level,
 - menu facility 353
- right-align output 200, 201, 203, 293
- Run Report, menu facility 354
 - caveats 355
- screen capture
 - in Panel Define mode 344
 - in Specify Formula screen 345
 - vs. clipboard 343
- search conditions, report 163
- search field list 25
- Select Report Field, report option 211
- self-referencing report variable 225
- Single Record Report From Lookup off a
 - Menu 317
- Skip If
 - explanation 303
- Skip if Start of Two Level, report option
 - 216
- Skip To
 - combining Skip To with Stop If 309
 - explanation 306
 - internal logic of 308
 - strategic placement of 308
- Skip to Bottom of Page vs. Page Eject
 - 214
- Skip to Bottom of Page, report option
 - 214
- skipping records, introduction 223
- Smart Lookup Algorithm 79, 80
- sort direction, report 163
- sorting backwards 94
 - by date 97
 - by number 94
- spaces and carriage returns in formulas
 - 300
- statement
 - CASES vs. IF-THEN 289
 - moment function 64, 65, 262, 272, 405
 - string identity operator 105, 285-287, 298, 299, 302
 - SUBSTRING function 294, 298
- STE Manager 413
- Stop If
 - combining Skip To with Stop If 309
 - explanation 306
- Store Value in Report Variable, report
 - option 211
- strategy in defining a Browse mode
 - lookup 81
- SUBFIELD function
 - explanation 294
- subfield lookups 70
- Submenu, menu facility 357
- subpanel, keeping data current 43
 - backward-referring computed fields 44
 - backward-referring non-updatable fields 47
 - cascade update 46
- subreport
 - as subroutines 232
 - going from version 2.2 to 2.3 230
 - introduction 229
 - parallel 235, 236, 238, 239, 243-245, 248, 253-255, 257-259, 265

- Subreport Using Virtual Link,
 - introduction 239
- that depends on existing link 229
- Using Virtual Link 140, 141,
 - 175, 188, 189, 217, 229,
 - 239-243, 246-248, 252,
 - 254, 255, 257, 278, 400
- virtual link vs. Subreport Using Virtual Link 140
- subreport that depends on existing link 229
- Subreport Using Virtual Link
 - vs. virtual link 140
- Subreport Using Virtual Link,
 - introduction 239
- Subreports, report option 217
- subroutines
 - subreports as 232
- Subset, menu facility 354
- SUBSTRING function
 - explanation 294
- support, DataPerfect 431
- T field 37
- time field
 - calculating elapsed time across the 24-hour barrier 58
 - computing elapsed time: the simple case 58
 - report option 211
 - using MOMENT and MODULO 61
 - using to guarantee uniqueness 57
- totaling
 - caveat 42, 151
 - implementing a Keep A Total 148
 - Keep a Total 2, 3, 21, 25, 26, 41-43, 92, 112, 128, 144-156, 218, 335-337, 385, 386, 397, 414
 - Keep A Total, introduction 147
 - printing totals at the top of the invoice 235
 - when there's no parent record to receive the total 150
- tracking user activity with the USER.FIELD function 360
 - user-stamping records 364
- transaction log
 - exporting and importing with 335
- transaction log vs. merge file: strategies 336
- troubleshooting formulas 301
- troubleshooting iteration control 328
- troubleshooting links 143
- troubleshooting lookups 86
- troubleshooting menus 368
- truncate
 - ;;1-9 (Truncate Leading and Trailing Blanks and Leave n Spaces) 199
 - ;;B (Truncate Both Leading and Trailing Blanks) 199
 - ;;E (Delete Zero Subfields from the End) 199
 - ;;S (Suppress Leading Blanks) 198
 - ;;T (Truncate Trailing Blanks) 198
- truth
 - note about DataPerfect's notion of truth 321
 - report variables and truth values 323
- TSR
 - ZipKey© 382
- Two Level Report
 - Skip if Start of Two Level, report option 216
- two-digit vs. four-digit years 55
- U field 32
- update
 - cascade update 3, 46, 92, 121, 122, 125, 130, 131, 143-146, 156
- upgrading a Client to a new version of DataPerfect 417
- Use-Mention distinction 283
- User Chooses Next Record By LookUp
 - explanation 313

- getting report to continue after
 - last record in lookup selected 327
 - how report lookups display 318
 - placing in the First Page Header 315
 - placing in the Report Body 317
- User ID Panel
 - caveats 359
 - controlled panel link access to subrecords 366
 - controlling access to data
 - accessed from a menu 365
 - controlling user access 361
 - Data Link Subgroup Lookup and the USER.FIELD function 367
 - deselecting 368
 - introduction 358
 - tracking user activity with the USER.FIELD function 360
 - USER.FIELD function 360
 - user-stamping records 364
 - vis a vis application passwords 363
- USER.FIELD function
 - and the Data Link Subgroup Lookup 367
 - controlled panel link access to subrecords 366
 - controlling access to data
 - accessed from a menu 365
 - tracking user activity with 360
 - user-stamping records with 364
- user-stamping records 364
- utility
 - DPDiagnostics 409
 - DPEXport 407
 - DPIImport 407
 - DPOrder 411
 - STE Manager 413
 - ZipKey© 382
- validation time 25
- variable-length text field
 - ;;N (New Occurrence of Field) 200, 205
 - in reports 203
- warning messages, DPDiagnostics 410
- warnings
 - formula error messages 282
- well-formed formula 282
- WordPerfect Merge files 330
 - the nature of the export file, including some caveats 331
 - what happens during a WordPerfect Merge file import 334
- WordPerfect Users Forum 431
- work space
 - computed fields 49
- year 2000 problem 55
- Zip Code
 - controlling data entry with ZipKey© 382
- Zip Code output, bar codes 200
- ZipKey© 382