APPX LIBRARY

Application Design Manual 4.2

APPX 4.2, 2006

> Written for APPX 4.2 in 2006. The current Application Design Manual, kept up to date for APPX 6, is at docs.appx.com.

APPX Software, Inc.

Made on October 8, 2026 from the pages of the APPX Library, https://wiki.appx.com/manuals/library/design-manual-42/. The pages are the current version.

Contents

Chapter 1-1 **Overview of Application Design** 26
Introduction to Application Design 26
Terminology 27
A Specification Approach 28
Prototyping an Application 28
Integrated Documentation 29
Components of the Application Design Facility 29
The Data Dictionary (Data Elements and Structures) 31
Objects or Processes 31
The Image Editor 33
Integrated Language Facility (ILF) 33
The ILF Editor 34
The Debugging Facility 34
Design Tools 34
Language Facility 34
Utilities 35
Application Chooser 39
Application Properties 40
Named Resources 42

Chapter 1-2 **Getting Started** 46
Before You Begin 46
System Administration Tasks 46
Managing Design Files 46
Planning Your Application 58
Design Methodologies 58
Approaches to System Design 58
Analyzing Your Requirements 59
Entering Specifications in APPX 60
Defining Keystroke Macros 61

Chapter 1-3 **Data Dictionary** 62
The Data Dictionary: Cornerstone of an Application 62
Characteristics of the APPX Data Dictionary 62
Accessing the Data Dictionary 63

The Components of the Data Dictionary ... 63
Domains and Domain Types ... 64
The Functions of a Domain ... 65
The Advantages of Using Domains ... 65
An Example of a Domain ... 66
Fields and Field Types ... 66
Work Fields ... 68
Audit Trail ... 68
Storage Options and Files ... 68
Permanent Files ... 69
Disk Files (Temporary) ... 69
Memory Files ... 69
Working-Storage Files ... 70
File Organization ... 70
One-Record File Organization ... 70
Sequential File Organization ... 71
Indexed File Organization ... 71
File Keys ... 71
Default Value Records ... 72
Data Dictionary Processing ... 72
When Processing is Required ... 73
Restructuring Files ... 73
Security in the Data Dictionary ... 74
Record Protection Fields ... 74

Chapter 1-4 **Understanding Process Design** ... **76**
APPX Processes ... 76
Processes Compared With Programs ... 76
Process Concepts ... 77
The Menu Process ... 77
The Job Process ... 78
The Input Process ... 78
The Output Process ... 79
The Update Process ... 81
The Query Process ... 81
The Inquiry Process ... 82
The Status Process ... 82
The Subroutine Process ... 82

Relationships Between Processes ... 82
Process Genealogy ... 82
Process Components ... 83
Preparing for Runtime ... 86
The Sequence of Operation ... 87
An Example of the Process Cycle ... 88

Chapter 1-5 **Interprocess Communication** ... 89
Interprocess Communication: overview ... 89

Chapter 1-6 **Customizing Your Application** ... 92
Runtime Application ... 92
Predefined Fields (PDFs) ... 92
Using Predefined Fields ... 92
Categories of Predefined Fields ... 93
Predefined Processes ... 93
Accessing Non-APPX Programs ... 94
The CALL Statement ... 94
The RUN Statement ... 95
Command Process Invocation ... 95

Chapter 1-7 **The Documentation Facility** ... 96
An Integrated Facility ... 96
The Standard Documentation Screen ... 96
Technical and User Documentation ... 97
The Help Option ... 98
Application Design Documentation Menu ... 98

Chapter 1-8 **Application Design Tools** ... 101
Application Design Tools Menu ... 101
Application Parameters ... 102
Change All References ... 103
Design Transfer ... 105
Cross Reference ... 108
Cross Reference Options ... 109
Other Design Tools ... 111
Print Technical Documentation ... 112

Chapter 2-1 **Data Dictionary Overview** ... 113
APPX Data Dictionary ... 113

Accessing the Data Dictionary ... 113
Characteristics of the Data Dictionary ... 114
Components of the Data Dictionary ... 115
Domain Concepts ... 115
Domain Types ... 116
Functions of a Domain ... 116
Some Examples of Domains ... 117
Mask Definition ... 118
Display Mask Relationship (Data Dictionary vs. Image) ... 118
External, Internal, and Universal Data Formats ... 119
Fields ... 120
Field Types ... 120
Physical versus Logical Categories ... 121
Physical Fields ... 122
Work Fields ... 122
Logical Fields ... 123
Substring Fields ... 123
Synonym Fields ... 124
Group Fields ... 124
Designating Keys ... 126
Key Order versus Sort Order ... 128
Files ... 129
File Storage Options ... 129
File Organization ... 131
Temporary versus Permanent Files ... 133
Record Protection ... 134

Chapter 2-2 **Data Dictionary Concepts** ... **135**
Processing the Data Dictionary ... 135
Tracking Processing Requirements ... 136
Summary of a Data Dictionary Process ... 136
Relationships of the Processed Dictionary to Other APPX Components ... 137
Application Integration Issues ... 138
Integrity ... 138
Database Security ... 139
Documentation ... 142
Accessing the Data Dictionary Documentation Screens ... 142
Integration with Database Management and System Administration ... 144

Database Management and the Data Dictionary ... 145
System Administration and the Data Dictionary ... 145

Chapter 2-3 **Domains** ... **146**
Introduction to Domains ... 146
Domains Screen ... 146
Domains Toolbox Option ... 149
Domains Data Lookup ... 150
Domains Additional Attributes ... 150
Common Domain Specifications ... 151
Alpha Domains ... 151
Alpha Domain Validation Table ... 153
Alpha Domain Additional Attributes ... 155
Format Domains ... 158
Format Domain Validation Table ... 160
Format Domain Additional Attributes ... 162
System Domains ... 164
System Domain Additional Attributes ... 166
Text Domains ... 166
Text Domain Additional Attributes ... 168
Token Domains ... 169
Token Domain Token Table ... 171
Token Domain Additional Attributes ... 173
Numeric Domains ... 174
Numeric Domain Additional Attributes ... 176
Numeric Storage Types ... 179
Six Types of Characters Used in Numeric Masks ... 180
Mask Characters Influenced by Values in System Administration ... 182
Text in Numeric Masks ... 183
Formats Excluded by Numeric Masking Facility ... 183
Date Domains ... 184
Logic Domains ... 186
Same As Domains ... 187

Chapter 2-4 **Files and Fields** ... **189**
Introduction to Files and Fields ... 189
Files ... 189
Files Toolbox Option ... 192

Files Rules Option ... 195
Files Additional Attributes Option ... 197
Files Record Protection Option ... 200
Files Default Process Option ... 201
Fields ... 203
Fields Toolbox Option ... 205
Fields Toolbox Copy Fields Option ... 207
Fields Data Lookup Option ... 209
Fields Additional Attributes ... 211
Fields Key Control Option ... 211
Common Fields Specifications ... 212
Occurrence ... 213
Domain Fields ... 214
Domain Fields Additional Attributes Option ... 216
Domain Fields (System Domains) ... 220
Domain Fields (Numeric Domains) ... 221
Alpha Fields ... 224
Alpha Fields Validation Table ... 227
Alpha Fields Additional Attributes ... 228
Format Fields ... 230
Format Fields Validation Table ... 232
Format Fields Additional Attributes ... 234
System Fields ... 236
Text Fields ... 238
Text Fields Additional Attributes ... 239
Token Fields ... 241
Token Fields Token Table ... 243
Token Fields Additional Attributes ... 245
Numeric Fields ... 247
Date Fields ... 251
Logic Fields ... 254
Logic Fields Additional Attributes ... 256
Synonym Fields ... 257
Special Uses for Synonym Fields ... 259
Substring Fields ... 261
Group Fields ... 263
Group Header Fields ... 264

Group Trailer Fields ... 266
Special Uses for Group Fields ... 266

Chapter 2-5 **Work Fields** ... **268**
Introduction to Work Fields ... 268
Work Fields Toolbox Option ... 269
Work Fields Data Lookup Option ... 271
Work Fields Additional Attributes Option ... 272
Common Work Fields Specifications ... 272
Domain Work Fields ... 273
Domain Work Fields Additional Attributes ... 274
Domain Work Fields (Numeric Domain) ... 278
Domain Work Fields (Numeric Domain) Additional Attributes ... 280
Alpha Work Fields ... 281
Alpha Work Fields Validation Table ... 283
Alpha Work Fields Additional Attributes ... 285
Format Work Fields ... 288
Format Work Fields Validation Table ... 290
Format Work Fields Additional Attributes ... 291
Text Work Fields ... 293
Text Work Fields Additional Attributes ... 295
Token Work Fields ... 297
Token Work Fields Token Table ... 298
Token Work Fields Additional Attributes ... 300
Numeric Work Fields ... 302
Date Work Fields ... 306
Logic Work Fields ... 310

Chapter 3-1 **Overview of APPX Processes** ... **313**
Overview of Chapters 3-1 thru 3-16 ... 313
Navigation Aids ... 314
The APPX Process Facility ... 314
Accessing the Process Facility ... 315
Process Types ... 316
Predefined Processes ... 316
Frames ... 318
Images ... 319
Process Control Files ... 321

Automatic and Optional Child Invocations ... 322
Event Points ... 324
Process Lineage ... 326
The Process Cycle ... 328

Chapter 3-2 **Getting Started** ... **329**
Introduction to Process Concepts ... 329
The Image Editor ... 329
Automatic Data Lookup and Display ... 330
Frame/Image Positioning ... 331
Screen Positioning ... 331
The Effect of Child Invocation on Screen Display ... 332
Page Positioning ... 332
The Effect of Scrolling Frames on Screen and Page Positioning ... 333
Matrix Scrolling Mode ... 333
Columnar Scrolling Mode ... 335
Input Process Concepts ... 336
Input Entry Levels ... 336
Input Keymaps ... 337
Input Modes ... 337
Input Interactive Phases ... 338
Automatically Saved Records ... 338
Output Process Concepts ... 339
Output Frame Classes ... 339
Output Frame Class Execution ... 340
Print Beside ... 341
Update Process Concepts ... 344
Query Process Concepts ... 344
Query Setup versus Execution ... 344
Query Files ... 345
Job Process Concepts ... 346
Inquiry Process Concepts ... 346
Status Process Concepts ... 346
Subroutine Process Concepts ... 347
Performance Tuning ... 347
Saving Executable Modules ... 347
EM Caching ... 348
Documentation ... 349

Accessing Process Documentation Screens ... 350
Security ... 351

Chapter 3-3 **Process Definition** ... **352**
Introduction to Process Definition ... 352
Common Process Specifications ... 353
Process Additional Attributes Option ... 355
Process Rules Option ... 357
Process Documentation Option ... 359
Process Security Option ... 360
Process Toolbox Pulldown Menu ... 361
Audit Information Overlay ... 363
Invoke a Process Overlay ... 364
Process Jump to Pulldown Menu ... 366

Chapter 3-4 **Menu Processes** ... **368**
Introduction to Menu Processes ... 368
Menu Process Specifications ... 368
Menu Process Rules Option ... 371
Menu Process Optional Children Option ... 372
Menu Image in the Image Editor ... 373

Chapter 3-5 **Job Processes** ... **375**
Introduction to Job Processes ... 375
Job Process Specifications ... 375
Job Process Toolbox Option ... 377
Job Process Rules Option ... 379
Job Process Additional Attributes Option ... 380
Job Process Automatic Child Processes ... 382

Chapter 3-6 **Input Processes** ... **384**
Introduction to Input Processes ... 384
Input Process Specifications ... 385
Input Process Additional Attributes Option ... 386
Input Process Rules Option ... 389
Input Frame Specifications ... 392
Input Frame Additional Attributes Option ... 395
Input Frame Rules Option ... 397
Input Frame Automatic Children Option ... 400
Input Frame Toolbox Pulldown Menu ... 401

Input Image Specifications ... 403
Scrolling Input Image Specifications ... 404
Scrolling Input Image Additional Attributes Option ... 405
Input Image Rules Option ... 407
Input Image Optional Children Option ... 409
Scrolling Input Image in the Image Editor ... 410
Non-Scrolling Input Image Specifications ... 412
Non-Scrolling Input Image Additional Attributes Option ... 414
Non-Scrolling Input Image in the Image Editor ... 416

Chapter 3-7 **Output Processes** ... 419
Introduction to Output Processes ... 419
Output Process Specifications ... 420
Standard Column Heading Image Generation ... 422
Output Process Additional Attributes Option ... 423
Output Process Rules Option ... 425
Output Frame Specifications ... 427
Output Frame Additional Attributes Option ... 430
Output Frame Rules Option ... 433
Output Frame Automatic Children Option ... 436
Output Frame Toolbox Pulldown Menu ... 437
Output Image Specifications ... 438
Scrolling Output Image Specifications ... 439
Scrolling Output Image Additional Attributes Option ... 441
Non-Scrolling Output Image Specifications ... 442
Non-Scrolling Output Image Additional Attributes Option ... 444
Output Image Rules Option ... 446
Output Image in the Image Editor ... 449
Designing GUI Output ... 450
Printing PDF Output ... 453
PDF Print on Screen Dialog ... 454

Chapter 3-8 **Update Processes** ... 457
Introduction to Update Processes ... 457
Update Process Specifications ... 457
Update Process Toolbox Option ... 459
Update Process Rules Option ... 460
Update Process Additional Attributes Option ... 462

Update Frame Specifications ... 465
Update Frame Toolbox Option ... 467
Update Frame Rules Option ... 468
Update Frame Automatic Processes Option ... 469

Chapter 3-9 **Query Processes** ... **470**
Introduction to Query Processes ... 470
Query Process Specifications ... 471
Query Process Toolbox Option ... 473
Query Process Rules Option ... 475
Query Process Additional Attributes Option ... 477
Enduser Selections Option ... 479
Enduser Selections Toolbox Option ... 481
Enduser Selections Expression Option ... 482
Enduser Selections Add'l Attributes Option ... 484
Designer Selections Option ... 485
Sort Order Option ... 488
Grand Totals Option ... 491
Populating Query Files ... 493

Chapter 3-10 **Inquiry Processes** ... **494**
Introduction to Inquiry Processes ... 494
Inquiry Process Specifications ... 495
Inquiry Process Toolbox Option ... 497
Inquiry Process Rules Option ... 498
Inquiry Process Additional Attributes Option ... 500
Inquiry Frame Specifications ... 504
Inquiry Frame Toolbox Option ... 506
Inquiry Frame Rules Option ... 507
Inquiry Frame Additional Attributes Option ... 508
Inquiry Frame Automatic Processes Option ... 514
Inquiry Image Specifications ... 514
Inquiry Image Toolbox Option ... 517
Inquiry Image Rules Option ... 518
Inquiry Image Additional Attributes Option ... 520
Inquiry Image Optional Processes Option ... 524

Chapter 3-11 **Status Processes** ... **526**
Introduction to Status Processes ... 526

Status Process Specifications ... 526
Status Process Toolbox Option ... 528
Status Process Rules Option ... 529
Status Process Additional Attributes Option ... 530

Chapter 3-12 **Subroutine Processes** ... 533
Introduction to Subroutine Processes ... 533
Subroutine Process Specifications ... 533
Subroutine Process Toolbox Option ... 535
Subroutine Process Rules Option ... 536
Subroutine Process Additional Attributes Option ... 537

Chapter 3-13 **Automatic and Optional Children** ... 540
Introduction to Automatic and Optional Children ... 540
Child Specifications ... 540
Child Toolbox Option ... 544
Child Rules Option ... 545
Child Additional Attributes Option ... 547
Child Constraints Option ... 549
Child Use Query Option ... 551
Child GUI Attributes Option ... 553

Chapter 3-14 **Using the Image Editor** ... 554
Image Editor ... 554
Image Editor Options ... 556
Control Cursor Position Option ... 560
Image Editor Toolbox Option ... 561
Image Editor Toolbox Rules Option ... 563
Editing or Defining Items on Images ... 564
Image Item Specifications ... 564
Image Item Toolbox Option ... 569
Image Item Toolbox Rules Option ... 571
Item Repeat Option ... 571
Data Lookup Option ... 573
Image Item Additional Attributes Option ... 577
Alphanumeric Item Additional Attributes ... 577
Date/Time Item Additional Attributes ... 583
Display Mask (Date/Time) ... 586
Logic Item Additional Attributes ... 588

Numeric Item Additional Attributes ... 588
Defining Image Characteristics ... 591

Chapter 3-15 **Using GUI Features of the Image Editor** ... **595**
GUI Features of the Image Editor ... 595
Image Editor Toolbar ... 596
Making Selections ... 598
Using Guide Bars ... 598
Cursor Types ... 598
Moving and Resizing ... 599
Image Editor Right-Click Menu ... 599
APPX Data Palette ... 601
Editing or Defining Data Objects for Images ... 602
Data Object Specifications ... 604
Painting Buttons, Labels, Pictures, and Boxes or Lines ... 605
Painting Buttons ... 607
Painting Labels ... 611
Painting Pictures ... 615
Painting Boxes or Lines ... 619
Changing and Deleting Graphical Items ... 623
Use of Graphical Attributes ... 623
Defining Graphical Attributes for Images ... 624
Title Bar Options ... 630
Window Options ... 631
Shared Options and Name ... 632
Defining Graphical Attributes for Optional Children ... 632
Defining Graphical Attributes for Image Fields ... 639
Image Field Control Options ... 645
Image Field Font Options ... 646
Image Field Border Options ... 647
Image Field Other Options ... 647
Image Field Tab Order Options ... 648
Making Color Selections ... 648

Chapter 3-16 **Using Event Points** ... **652**
Event Points ... 652
Process-Level Event Points ... 652
Start of Process ... 653

Set Run-Time Defaults ... 653
Post PCF Read ... 654
Pre-PCF Update ... 654
Post PCF Update ... 655
Query Process Event Points ... 655
Start of Query Execution ... 656
Establish PCF Range ... 656
Pre-User Selection ... 657
Post User Selection ... 657
End of Query Execution ... 658
End of Process ... 658
Frame-Level Event Points ... 659
Start of Frame ... 659
Select Image ... 659
Global Pre-Display ... 659
Global Option Intercept ... 660
End of Frame ... 660
Effect of Optional Frames on Event Point Chronology ... 660
Effect of Repeating Frames on Event Point Chronology ... 660
Effect of Scrolling Frames on Event Point Chronology ... 661
Image-Level Event Points ... 661
Pre-Display ... 661
Pre-Display/Verify ... 662
Option Intercept ... 662
Verify ... 663
Child-Level Event Points ... 663
Pre-Invocation ... 663
Post Invocation ... 663
Using the Predefined Field NEXT CHILD ID ... 664
File Level Event Points ... 664
File Open ... 665
Post Read ... 665
Pre-Update ... 665
Post Update ... 666
File Close ... 666

Chapter 4-1 **ILF Integration** ... **667**
Using ILF Routines ... 667

An Example of a Statement ... 667
The Components of a Statement ... 668
The Statements Included in the ILF ... 668
When to Invoke ILF Routines: Event Points ... 670
Event Point Definition and Description ... 670
The Steps in Using the ILF ... 670
The ILF Editor ... 671
The Compilation Step ... 672
Resolving Compilation Errors ... 672
Resolving Runtime Errors ... 674
Using the ILF Debugger ... 674

Chapter 4-2 **True/False Status Indicators** ... 675
How APPX Uses T/F Status Indicators ... 675
Setting T/F Status Indicators ... 676
T/F Status Indicator Values ... 677
Statements That Affect T/F Status Indicators ... 677
Some Examples of T/F Usage ... 679
T/F Execution Condition Values ... 680
Valid/Invalid Execution Conditions ... 681
Using Hyphens as Placeholders ... 681
Simple T/F Examples for Comparison ... 682
Using AND and OR Statements ... 683
Using AND After an IF Statement ... 684
Using OR After an IF Statement ... 684
Adding Execution Conditions ... 685
Potential Misuse of AND and OR ... 685
AND and OR with Statements Other Than IF ... 685

Chapter 4-3 **Specifying Statements** ... 687
Specifying Statements: overview ... 687
Patterns of Statement Keyword Use ... 687
Rules for Statement Components ... 688
The T/F Execution Conditions ... 688
The Statement Keyword ... 689
The Application ID ... 689
The File Name ... 689
The Field Component ... 690

Partial-Record I/O ... 690
The Label ... 692
The Occurrence Number ... 692
Characteristic-Type Statements ... 692
The Appearance Number ... 693
The Arithmetic Operator ... 694
The Relational Component ... 694
The Message Text ... 695
The FAIL (or FT) Parameter ... 695
A Non-APPX Program Name ... 696
Miscellaneous Operands ... 696
Using Predefined Fields (PDFs) in Statements ... 697
PDFs and Field-Naming ... 697
The A through Z and AI through ZI PDFs ... 698
Specifying Constant Values ... 698
Universal Data Formats ... 699

Chapter 4-4 **The ILF Editor** ... **700**
The Editor and Automatic Compilation ... 700
The Features of the Editor ... 700
Invoking the ILF Editor ... 701
The Layout of the Editor Screen ... 702
The Help Option ... 704
The Edit Modes and Options ... 705
Save and Exit ... 706
Add Mode ... 706
Delete Mode ... 709
Change Mode ... 711
Toolbox ... 712
Move ... 713
Copy ... 715
Search ... 715
Replace ... 717
XCopy (external copy) ... 718
Delete All Statements (Option 10) ... 720
Restore All Deleted Statements (Option 11) ... 721
Using Comments ... 721

Chapter 4-5 **The Debugger** **722**
Analyzing Logic Errors 722
The Features of the Debugger 722
Invoking the Debugger 722
The Layout of the Debugger Screen 724
Key Information 724
Display and Debugging Area 725
Data Values Area 726
Options 728
The TRAP Statement 728
Conditional Execution of the TRAP Statement 729
“Hard-Coded” versus Dynamic Trapping 729
The Options on the Debugger Screen 730
Go 730
Return 731
Step/Into 731
Step/Ret 731
Step/End 732
[F3] Select (Clear or Set Trap) 732
Add Mode (Debugger Screen) 732
Watch Points 733
Inquire Mode (Debugger Screen) 735
Change Mode (Debugger Screen) 735
Other Options 735
Suggested Uses for the Debugger 736
Leaving the Debugger 736
Clearing Out Break Points 737

Chapter 4-6 **ILF Keyword Reference** **738**
* (Comment) 738
** (Comment) 738
AND 739
APPEND 741
AT FIELD 743
BEG AT 744
BEG LOOP 746
BEG READ 748
BLANK 752

BLINK ... 753
BOLD ... 754
BRIGHT ... 755
CALC ... 756
CALC - Basic Functions ... 759
CALC - Exponential Functions ... 760
CALC - Trigonometric Functions ... 760
CALL ... 761
Using the CALL Statement to Access User-Provided Functions ... 762
CANCEL ... 763
CLOSE ... 764
CNV BIN ... 765
CNV PORT ... 767
CNV TEXT ... 767
COLOR ... 769
COMMIT ... 770
COMPUTE ... 772
COPY ... 773
CREATE ... 774
DATE ADD ... 776
DATE BTW ... 777
DELETE ... 779
DIM ... 780
DISABLE ... 781
DISPLAY ... 782
ENABLE ... 783
END ... 784
END AT ... 785
END LOOP ... 787
END PROC ... 789
END READ ... 789
ERROR ... 791
EXIT ... 792
FONT ... 792
GOSUB ... 793
GOTO ... 795
IF ... 796

IF DIFF ... 798
IF EXIST ... 799
INPUT ... 801
INQUIRY ... 802
INVERSE ... 803
JOB ... 804
LABEL ... 806
LOCAL ... 807
LOGOUT ... 807
MARK TRX ... 808
MENU ... 808
MESSAGE ... 809
NO BOLD ... 810
NO INPUT ... 811
NO ULINE ... 812
NORMAL ... 813
OK INPUT ... 814
OPEN ... 815
OR ... 817
OUTPUT ... 819
OVERFLOW ... 821
PASS ... 822
PASS Lists ... 823
PICTURE ... 824
POSITION ... 826
QUERY ... 827
READ ... 828
READNEXT ... 830
RECEIVE ... 832
REFRESH ... 834
RELEASE ... 834
RESTORE ... 834
RETURN ... 836
REWRITE ... 837
ROLLBACK ... 838
RUN ... 839
SAVEPNT ... 841

SCAN ... 842
SCRATCH ... 843
SET ... 845
When SET Destination is Alpha ... 846
When SET Destination is a Number ... 847
When SET Destination is a Date ... 848
When SET Destination is a Logic Field ... 849
SET DATE ... 850
SET DAY ... 851
SET MNTH ... 852
SET TEMP ... 853
SET TOT ... 855
STATIC ... 856
STATUS ... 856
STORE ... 857
SUBR ... 859
TIMEOUT ... 860
TOTAL ... 862
TRAP ... 863
ULINE ... 864
UPDATE ... 865
USE CC ... 866
VERIFY ... 867
WARNING ... 868
WRITE ... 869

Chapter 4-7 **Predefined Fields and Processes** ... 871
Runtime Application ... 871
Predefined Fields (PDFs) ... 871
Using Predefined Fields (PDFs) ... 871
Categories of PDFs ... 872
Current Status PDFs ... 872
Process Control PDFs ... 873
Terminal Management PDFs ... 874
Report Management PDFs ... 874
General Purpose PDFs ... 876
Information Headings for PDF's ... 876
Key to Table 4-7-1: Predefined Fields Attributes ... 877

Table 4-7-1: Predefined Fields Attributes ... 877
A, B, C...Z ... 881
AI, BI, CI...ZI ... 882
ACCESS PATH ... 882
ALTERNATE IMAGE NUMBER ... 883
APPLICATION ... 883
AUTO READNEXT ... 883
BETWEEN ... 884
BILLING CODE ... 884
CANCEL OK ... 884
CHILD ID ... 884
COLOR BACKGROUND ... 885
COLOR FOREGROUND ... 885
COMMIT MODE ... 885
COMPANY ADDRESS 1 ... 887
COMPANY ADDRESS 2 ... 887
COMPANY ADDRESS 3 ... 887
COMPANY NAME ... 887
CURSOR AT APPEARANCE ... 888
CURSOR AT OCCURRENCE ... 888
CURSOR COLUMN ... 888
CURSOR ROW ... 889
DATABASE ... 890
DEFAULT MODE ... 890
ENTRY LEVEL ... 890
ERRORS ... 891
EXACT KEY REQUIRED ... 891
FONT ... 892
FORM LENGTH ... 892
FORM WIDTH ... 892
INCLUDE CHILDREN ... 892
INCLUDE RECORD ... 893
INTERACTIVE PHASE ... 893
KEYMAP NAME ... 894
LANGUAGE ID ... 894
LINES LEFT ON PAGE ... 895
LOG BANNER ... 895

LOG COPIES ... 895
LOG FILE DISPOSITION ... 895
LOG FILE PATHNAME ... 896
LOG FORM ID ... 896
LOG FORMAT ... 896
LOG HARD COPY ... 897
LOG MODE ... 897
LOG NOTIFY USER ... 897
LOG ON SCREEN ... 898
LOG PRINTER ID ... 898
LOG PRIORITY ... 898
LOG REPORT OPTIONS ... 899
LOG SPOOL DATE ... 899
MODE ... 899
NEW PAGE AFTER ... 899
NEW PAGE BEFORE ... 900
NEXT APPLICATION ... 900
NEXT CHILD ID ... 900
NEXT DATABASE ... 901
NEXT LANGUAGE ID ... 901
NEXT PROCESS NAME ... 901
NEXT VERSION ... 902
OPERATIONAL PHASE ... 902
OPTION ... 903
PAGE NUMBER ... 903
PAN COLUMN ... 903
PAN ROW ... 904
PRINT BANNER ... 904
PRINT COPIES ... 904
PRINT FILE BROWSABLE ... 905
PRINT FILE DISPOSITION ... 905
PRINT FILE PATHNAME ... 905
PRINT FORM ID ... 906
PRINT FORMAT ... 906
PRINT HARD COPY ... 906
PRINT MODE ... 906
PRINT NOTIFY USER ... 907

PRINT ON SCREEN 907
PRINT PAGE ESTIMATE 907
PRINT PRINTER ID 908
PRINT PRIORITY 908
PRINT RECORD LIMIT 908
PRINT REPORT OPTIONS 909
PRINT SPOOL DATE 909
PRINT SUMMARY ONLY 909
PROCESS CODE 909
PROCESS NAME 910
PROCESS TYPE 910
RECORD NUMBER 911
RETURN CODE 911
SCROLL TO END 912
SELECTED APPLICATION 912
SELECTED DATABASE 912
SELECTED VERSION 913
SKIP AFTER 913
SKIP BEFORE 913
SOURCE APPLICATION 913
SOURCE VERSION 914
STATUS CODE 914
SUBHEADING 915
SUBMIT DATE 915
SUBMIT MODE 915
SUBMIT NOTIFY USER 916
SUBMIT OPTIONS 916
SUBMIT PRIORITY 916
SUBMIT PROCESS 916
SUBMIT QUEUE 917
TEMP 1 917
TEMP 2 917
TEMP 4 918
TEMP 30 918
TEMP 30 CTR 918
TEMP 30 LJ 918
TEMP 30 RJ 919

TEMP 32 ... 919
TEMP 79 ... 919
TEMP 80 ... 919
TEMP 132 ... 920
TEMP 256 ... 920
TEMP 512 ... 920
TEMP 2K ... 920
TEMP 8K ... 921
TEMP 32K ... 921
TEXT AT POSITION ... 921
USER ID ... 921
VERSION ... 922
WARNINGS ... 922
Predefined Processes ... 922
Designer-Invoked Processes ... 923
APPX MENU ... 924
BEGIN LIVE OPERATIONS ... 925
CHECK GUI CLIENT ... 926
DISPOSITION ... 926
DISPOSITION (BACKGROUND) ... 927
ENTER BILLING CODE ... 928
GET CMD LINE PARM ... 929
GET ENV VAR ... 930
GET FILE FROM CLIENT ... 930
PROCESS QUERY RECORD ... 931
SELECT DATABASE ... 932
SEND FILE TO CLIENT ... 932

CHAPTER 1-1

Overview of Application Design

Introduction to Application Design

The application design component of APPX enables a designer to develop and modify an application system quickly and effectively. This chapter briefly describes the APPX facilities that make this possible.

Application Design is accessed by selecting the DESIGN AN APPLICATION option from the Main APPX Menu shown in *Figure 1-1-1* below.

Main APPX menu

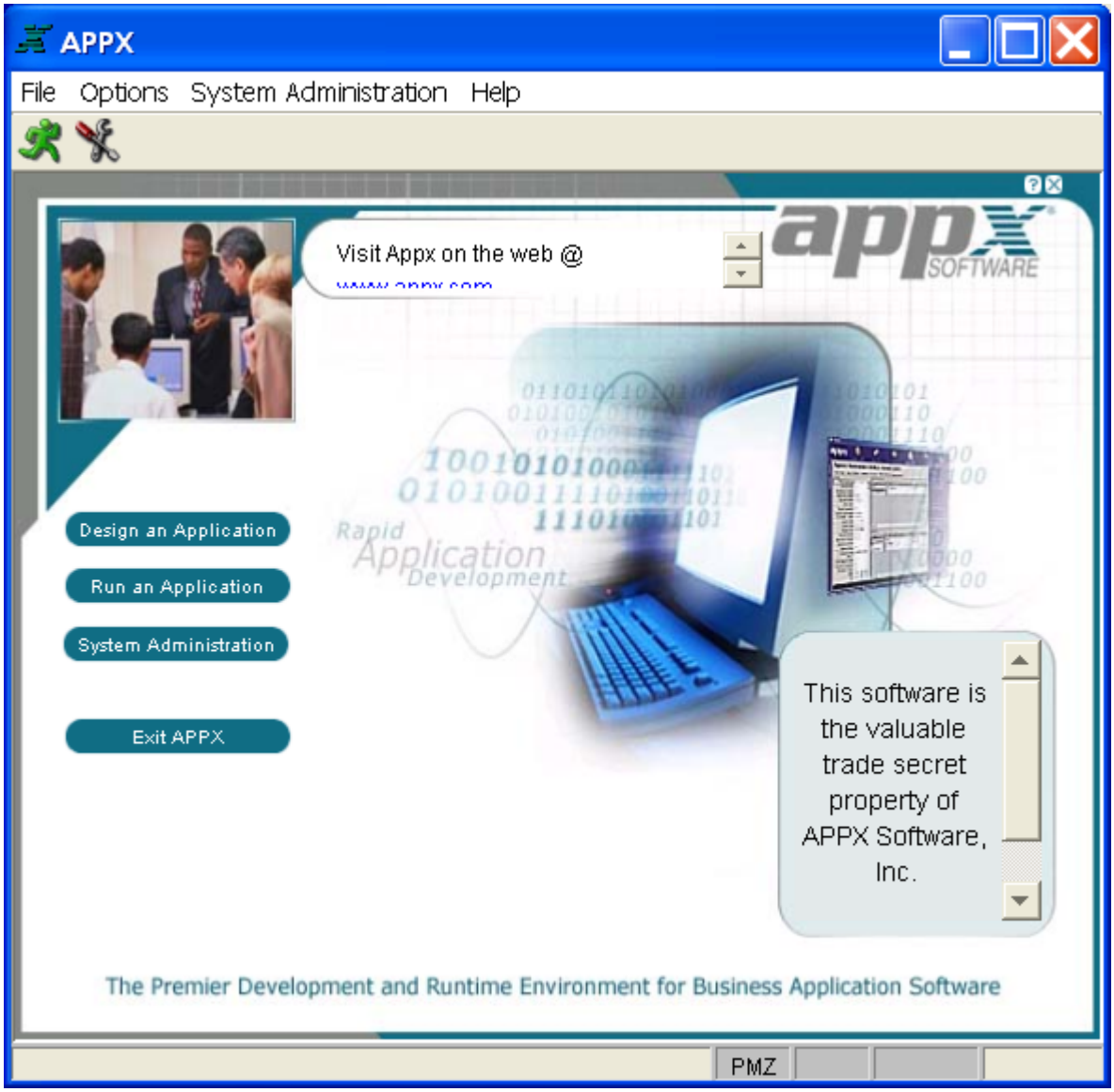


Figure 1-1-1

Terminology

There are some terms often applied to the people and systems associated with the APPX environment. The following definitions will help clarify how these terms are used as you read this manual.

- An *application system*, or application, is a set of interrelated computer processes, data structures, and procedures designed to perform one or more business-related tasks (for instance, to produce a payroll or maintain accounts payable information).
- An *application designer*, or designer, is a person who uses APPX to develop or modify applications that are run by end users.

- An ***end user***, or user, is a person who runs or executes an application written in APPX to perform a business-related task, such as preparing payroll.

The target audience for this manual is the application designer; in general, when directions or instructions are provided, they are addressed to a designer. The terms *designer* or *end user* are used only when such a distinction is essential to convey precise meaning.

A Specification Approach

APPX allows you to develop applications using a specification approach. Using this method, you supply responses to a number of pre-established questions or choices, and APPX produces the desired application products (menus, input screens, reports, etc.). The specification method to system design differs from the more traditional technique of using a procedural language (such as COBOL) to develop applications. APPX actually incorporates both techniques because it has its own Integrated Language Facility (ILF) that supplements the specification tools. You can develop complete applications using only the specification method. Or, in complex situations, ILF routines can supplement the specifications to perform more sophisticated operations or to provide explicit instructions to APPX. These ILF routines become an integrated part of application design.

Prototyping an Application

One method of system design involves the development of a preliminary specification-driven application to use as a prototype during the definition stage of an application project. Prototyping is an iterative design method by which you develop sample screens and reports, review them with the intended end users, modify the samples, and repeat the review process until the samples are satisfactory. APPX is an excellent prototyping tool because you can specify and modify screen images and reports very quickly. After you and the end users reach a mutual understanding about the basic structural and operational requirements of an application, you can supplement the basic specifications with ILF routines to address the more complex processing requirements.

There are several advantages of using a prototyping approach:

- With APPX, you can continue to enhance the prototype application you develop until it is complete enough to use as the real application in production. You do not have to discard the prototype and start over after all the requirements are defined.
- Prototyping allows you to minimize the coding and testing cycle because you actually begin testing as soon as you review the prototype application with the end users.
- There is less need to freeze the design requirements before you can begin productive work and demonstrate some of the capabilities of the proposed application. Users always find it easier to critique examples rather than concepts (i.e., written specifications).

- APPX's facilities provide ease of modification and enhancement; you do not have to discourage users from proposing changes and additional features.
- Prototyping encourages earlier and more productive involvement of end users, reducing the chances of finding design flaws in the latter stages of development.

Integrated Documentation

APPX allows you to generate a comprehensive set of documentation for an application without having to leave the normal development environment. In many application design environments, documentation is written after the application is complete using facilities separate from the primary design tool. With APPX, documentation is written concurrently; therefore, when the application is complete, so is the documentation.

There is another way to use the documentation facility if your environment calls for producing a specification of formalized requirements. You can use the documentation screens as a tool to specify all the requirements that the application must meet, without actually developing the entire application during the first pass. At the end of this definitional phase, you are able to generate a comprehensive requirements document suitable for review and approval by designers or end users.

Components of the Application Design Facility

The application design facility is an APPX application and, as such, operates like any other APPX application. This approach may prove instructive for designers because they can choose to pattern their own development after application design.

To enter application design, you select the DESIGN AN APPLICATION option from the Main APPX Menu. Once you have selected an application from the Application Chooser, the primary Application Design screen is displayed, as shown in *Figure 1-1-2* below, which contains the Application Design Specifications Menu.

Application Design Specifications menu

Figure 1-1-2

Application Design is divided into five general categories, as indicated by the folder tabs: **Specifications**, **Tools**, **Language**, **Documentation**, and **Utilities**.

As you can see, the Specifications tab is selected by default since this screen contains the heart of application design, the Application Design Specifications Menu. The menu options are grouped by function. **Application Specifications** include basic application properties and named resources, which link graphics files for use throughout the application. **Data Elements and Structures** encompass domains, file structures, and work fields, which together constitute the Data Dictionary. **Form Based Objects** include menus, inputs, inquiries, and outputs. These objects or processes all involve designing images. **Processing Objects** include queries, updates, subroutines, job streams, and status checks. These objects or processes do not involve designing images.

Each of these options, along with those found on the other folder tabs, will be described in detail later in this manual. The remainder of this chapter provides an overview of the primary application design tools that you will use most frequently.

The Data Dictionary (Data Elements and Structures)

The *Data Dictionary* component of APPX allows you to specify information about the data and files that are used in an application. Data dictionary options, shown in *Figure 1-1-3* below, are available from the Application Design Specifications Menu under the heading "Data Elements and Structures".

Figure 1-1-3. Data Dictionary Options

The dictionary specifications provide a central location for all data and file attributes that the processes (menus, inputs, outputs, etc.) reference as they are developed. The APPX dictionary is active, or integrated, meaning that all the information a designer specifies in the dictionary is used by, and places restrictions on, all other parts of an application. For example, if you specify in the dictionary that the value of a field called AMOUNT cannot exceed 100, an end user could not enter a value of 105 on any input screen where this field appears. In addition, if you expand the length of a field from seven to eight characters in the dictionary, after the dictionary is processed the field appears as eight characters wherever it is referenced in the application. As a designer, you do not need to make any other modifications to accommodate this change in field length, unless the change would cause fields to overlap on a screen or in a report.

A preliminary discussion of the dictionary can be found in Chapter 1-3: Data Dictionary while Chapters 2-1, 2-2, 2-3, 2-4, and 2-5 define the Data Dictionary in more detail.

Objects or Processes

The process definition component allows you to specify all types of objects or processes that, when integrated with the Data Dictionary and other application specifications, form the basic building blocks of an application. Options for objects or processes, shown in *Figure 1-1-4* below, are available from the Application Design Specifications Menu under the headings "Form Based Objects" and "Processing Objects".

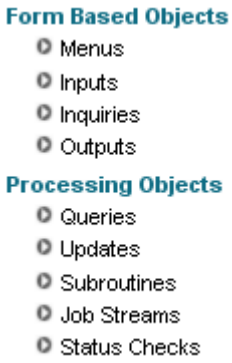


Figure 1-1-4. Objects or Processes Options

Form Based Objects utilize the Image Editor to define or edit images and include the following process types:

- ***Menus***, which provide a list of options available for selection by the user. Because an option on a menu may invoke another menu, an application can include many related menus.
- ***Inputs***, which allow a user to enter, modify, view, and delete data.
- ***Inquiries***, which allow a user quick access to viewing and optionally printing a specific record in a pre-defined format.
- ***Outputs***, which allow a user to produce reports or lists in a specified format that can be viewed on a screen or printed on hard copy.

Processing Objects do not involve designing images and include the following process types:

- ***Queries***, which permit a user to select a subset of records from a file and sort the records.
- ***Updates***, which allow data to be modified programmatically.
- ***Subroutines***, by which the designer can develop "stand-alone" procedural routines that can be invoked during the execution of any other type of process.
- ***Job Streams***, which controls the execution of a series of related processes.
- ***Status Checks***, which provides the capability to perform various kinds of error checking.

Together they comprise the nine different types of objects or processes within APPX. You provide the structure for an application by connecting the various processes that you define.

A preliminary discussion of APPX processes can be found in Chapter 1-4: Understanding Process Design. while Chapters 3-1 thru 3-16 of this manual provide more comprehensive information. These chapters also discuss the relationships between processes and describe how you make the appropriate connections to produce an integrated application.

The Image Editor

The *Image Editor* is a "screen painting" facility within processes that support images. It allows you to specify exactly how an image should appear on a screen display or in a printed report. A key step in the definition of a menu, input, output, or inquiry process is painting the image(s) that specify the format the user will see on the screen or report. There is no separate menu option to select the Image Editor directly. Instead, you automatically enter the Image Editor directly from within process definition when defining an image for a menu, input, output or inquiry process.

The standard features of the Image Editor can be employed in character mode using only a keyboard. However, you need a mouse to take advantage of the GUI features within the Image Editor. GUI tools provide designers with the ability to create screen images complete with pictures, buttons, icons, wallpapers, and other graphical features. Another GUI tool is a data palette offering drag and drop selection and placement of data fields.

Basic functionality of the Image Editor is fully explored in Chapter 3-14: Using the Image Editor and a complete discussion of its GUI features is presented in Chapter 3-15: Using GUI Features of the Image Editor.

Integrated Language Facility (ILF)

As a part of APPX's application design component, the *Integrated Language Facility (ILF)* allows you to develop procedural routines within an application. Each ILF routine consists of one or more *statements*. APPX's language includes many statement *keywords* that allow you to perform a wide variety of functions. ILF routines provide you with three primary capabilities:

- ***Explicit control over certain operations***, such as file opening and closing. Based on the information you have specified in the data dictionary, APPX provides automatic control of many operations. In some instances, however, you may choose to override a standard operation by defining and executing an ILF routine.
- ***Advanced file processing*** that provides you with precise control over record access, selection, and update. For example, you can perform complex cross-file updating.
- ***More complex processing functions*** than the specification method supports. An application may require a complex logic operation in which a number of conditions must be checked before a certain action occurs. For example, you may want to specify that for out-of-state customers only, if their outstanding balance exceeds their credit limit by more than $1,000, a warning message is to be issued. The ILF supports such sophisticated logic and truth table operations.

The Integrated Language Facility is explored in depth in Chapter 4-1: ILF Integration.

The ILF Editor

The ILF includes an *ILF editor* that allows you to develop and modify statements in routines. The editor is fully integrated with the data dictionary and processes, and provides automatic edit checking to help eliminate errors. The editor includes a number of facilities (such as move, copy, and search) that allow you to enter and edit statements efficiently. You enter the editor automatically by selecting the desired *event point* option while designing a process.

For more comprehensive descriptions of the editor, refer to The ILF Editor section of Chapter 4-1: ILF Integration as well as Chapter 4-4: The ILF Editor.

The Debugging Facility

APPX provides a debugging and testing tool, called the *statement debugger*, that allows you to step through the execution of any ILF routine and analyze how specific field values are affected by each statement. This tool is very helpful in resolving logic errors within an application. You control when and where the debugger operates by setting and clearing traps to examine sections of an ILF routine that you suspect may contain logic errors. The debugger also allows you to modify field values and monitor the effect of such changes on subsequent processing.

A complete description of the features of the debugger can be found in Chapter 4-5: The Debugger.

Design Tools

APPX also provides a number of tools that are designed to increase your efficiency in developing applications. One tool, called *design transfer,* allows you to copy existing processes as a starting point for developing other new ones. For example, you may want to “borrow” a report layout to serve as a pattern for a similar report. Another tool allows you to quickly change all existing references in an application to a new name.

A complete description of these tools can be found in Chapter 1-8: Application Design Tools.

Language Facility

When you select the Language tab from the primary Application Design screen, the Application Design Language Menu is displayed, as shown in *Figure 1-1-5* below.

Application Design Language menu

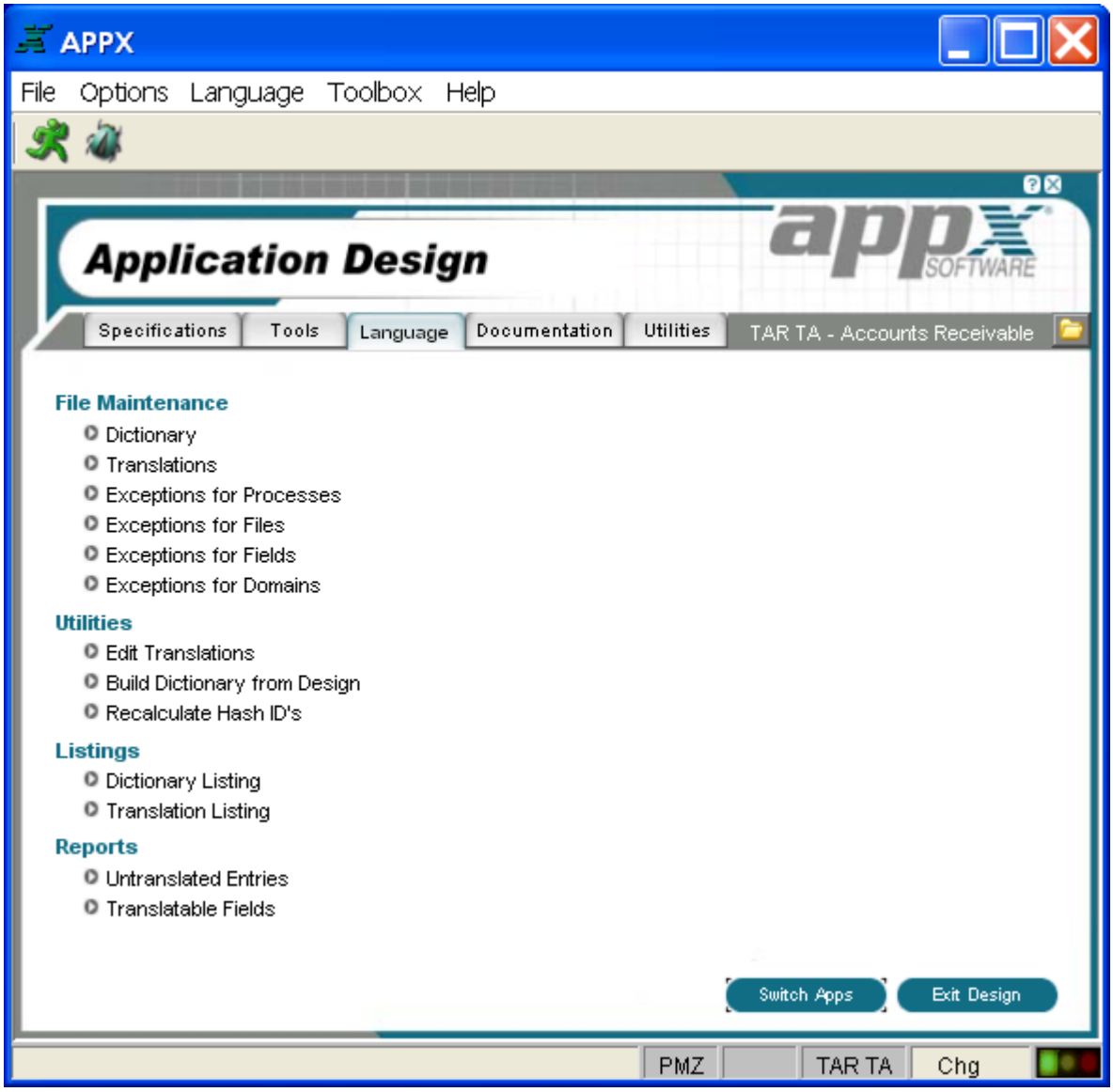


Figure 1-1-5

The Application Design Language Menu provide options to maintain language translations for the current application, produce listings and reports, and other functions. For more information regarding this facility, refer to the *Language Translation Manual*.

Utilities

When you select the Utilities tab from the primary Application Design screen, the Application Design Utilities Menu is displayed, as shown in *Figure 1-1-6* below.

Application Design Utilities menu

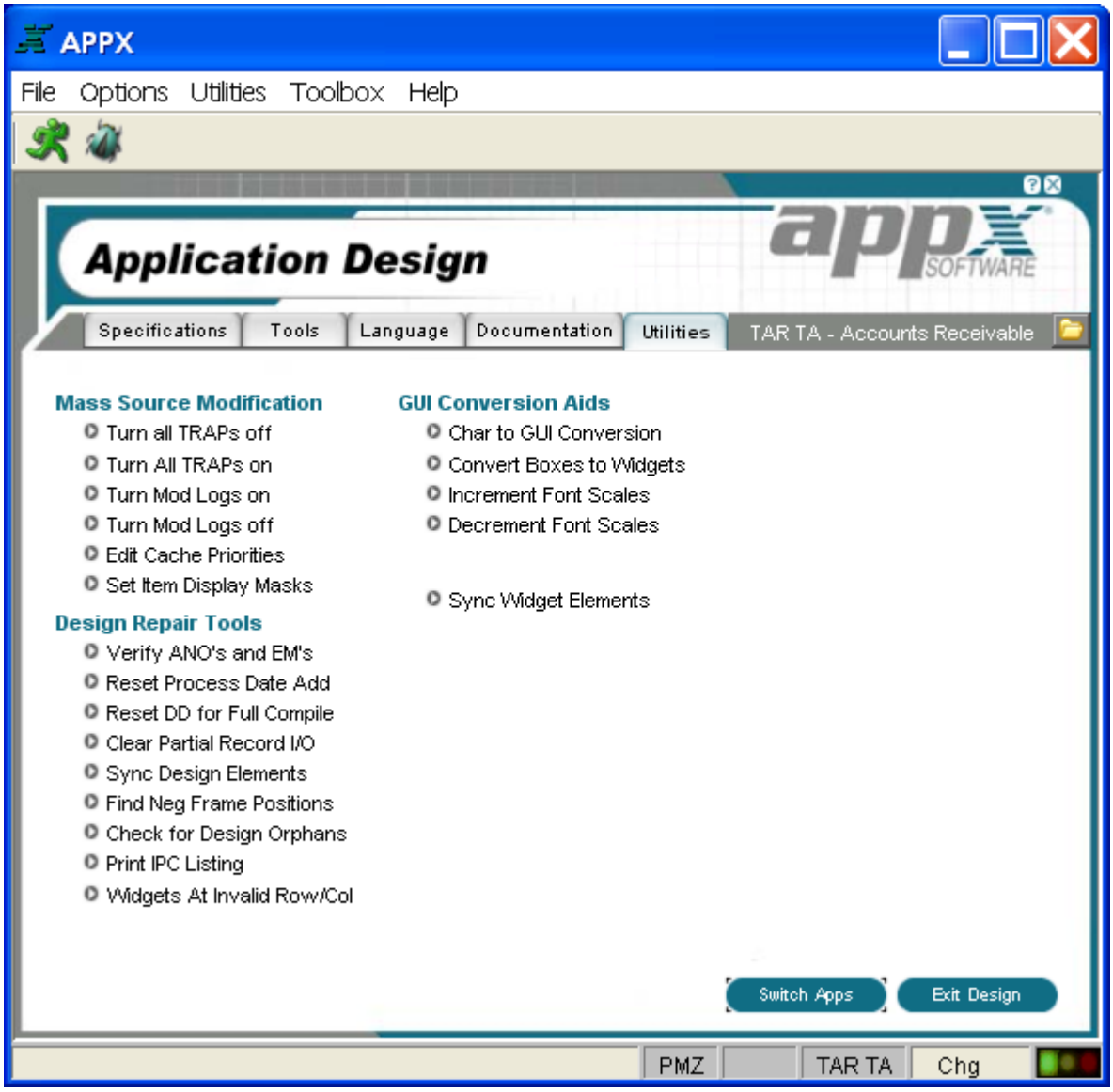


Figure 1-1-6

The Application Design Utilities Menu offers options for a variety of special purpose utilities including the following:

Source Modification

- Turn All TRAPs OFF turns all debugger traps off within the application so the debugger is not invoked during process testing.
- Turn All TRAPs ON turns all the debugger traps on within the application so the debugger is active during process testing.

- Generate Key Entry Frames regenerates the Key Entry frames for all inquiry processes in the application. This can be performed individually by process by using the regenerating option from the Additional Attributes option while editing the inquiry process.
- Renumber is used to renumber record sequence numbers within the application.
- Check Images for Item Overlap scans all images in the application and finds those that have items overlapping or immediately adjacent to one another. The images can be edited directly from this process. This option is most common used after expanding one or more fields in the data dictionary.
- Edit Process Cache Priority allows you to edit the cache priority of each process in the application. This function can be performed for each process by selecting the ADDITIONAL ATTRIBUTES option at the process level.

Design Repair Tools

- Orphan Detection to generate a series of reports that assess design elements in the application. An error is generated if any higher level design elements in the ancestry are missing. These errors refer to “orphans”. For example, an Image for Process “XXX” and RECORD Frame “100” would be an orphan if either the Process or the Frame were not currently on file.

  Note that these reports may not list all orphans. For instance, if there was a second Image for the same Process and Frame in the prior example, they would both be orphans. However, only the first Image would be listed because it sufficiently identifies any missing elements in the ancestry to remedy the orphan status of both Images.

  If you add a missing design element, the orphans become visible again. You then decide if you want to delete any design elements from the application.
- Verify All References runs a special version of “Cross Reference All” for the current application. You can select the REFERENCED BUT NOT DEFINED option to generate a list of all design elements and processes that are referenced by other elements or processes yet do not exist. Or, you can select the DEFINED BUT NOT REFERENCED option to list all design elements and processes that exist but are not referenced anywhere in the application.
- Frames With Negative Row POs lists all frames within the application where the Row Position” field is negative or where the frame contains more rows that the screen can display at one time.
- Check for Duplicate ANO and EMs generates a report of any process that have a duplicate date added. These result in duplicate Executable Modules and may cause problems. The report also shows duplicate “area numbers” or ANO’s. These are internal pointers used to locate variables in memory.

  If a duplicate EM is reported:
  1. Delete all EMs for the application
  2. Run the RESET PROCESS DATE ADD FIELDS option on the Toolbox’s Conversion menu.

If a duplicate ANO is reported:
1. Delete all EMs for the application
2. Initialize the BITMAP, ELEMENT, FILE, and INTIAL design files for the application
3. Process the data dictionary for the application
4. Restructure the application's end user data files

- IPC Listings generates a list of IPC files for either the applications processes or files, focusing on any exceptions from standard values. Any exceptions found can be corrected by this utility.

  IPC (Interprocess Communication) fields are used to define and control the sharing characteristics of APPX design elements. APPX designers don't have direct access to these fields as they are automatically maintained by APPX based on design entry in fields such as File Type and Share Class.
- Convert CHR Boxes to Widgets. Boxes were originally implemented as CHR records in APPX Version 4.0. With Version 4.1, they have been re-implemented as records in the WIDGET file. CHR boxes are now obsolete.

  This jobs allows process selection by a query. For each process selected, any box CHR records found are converted to WIDGET records. A log of the changed processes is produced. The conversion of a box to a widget may fail if a widget is already defined in the same location on the image.

GUI Conversion Aids

- Turn All TRAPs OFF turns all debugger traps off within the application so the debugger is not invoked during process testing.
- Turn All TRAPs ON turns all the debugger traps on within the application so the debugger is active during process testing.
- Generate Key Entry Frames regenerates the Key Entry frames for all inquiry processes in the application. This can be performed individually by process by using the regenerating option from the Additional Attributes option while editing the inquiry process.
- Renumber is used to renumber record sequence numbers within the application.
- Check Images for Item Overlap scans all images in the application and finds those that have items overlapping or immediately adjacent to one another. The images can be edited directly from this process. This option is most common used after expanding one or more fields in the data dictionary.
- Edit Process Cache Priority allows you to edit the cache priority of each process in the application. This function can be performed for each process by selecting the ADDITIONAL ATTRIBUTES option at the process level.
- Edit Processes Descriptive allows you to view and edit the descriptive names for every process in the application.

Application Chooser

When you select the DESIGN AN APPLICATION option from the Main APPX Menu, the Application Chooser screen is displayed, as shown in *Figure 1-1-7* below.

Application Chooser screen

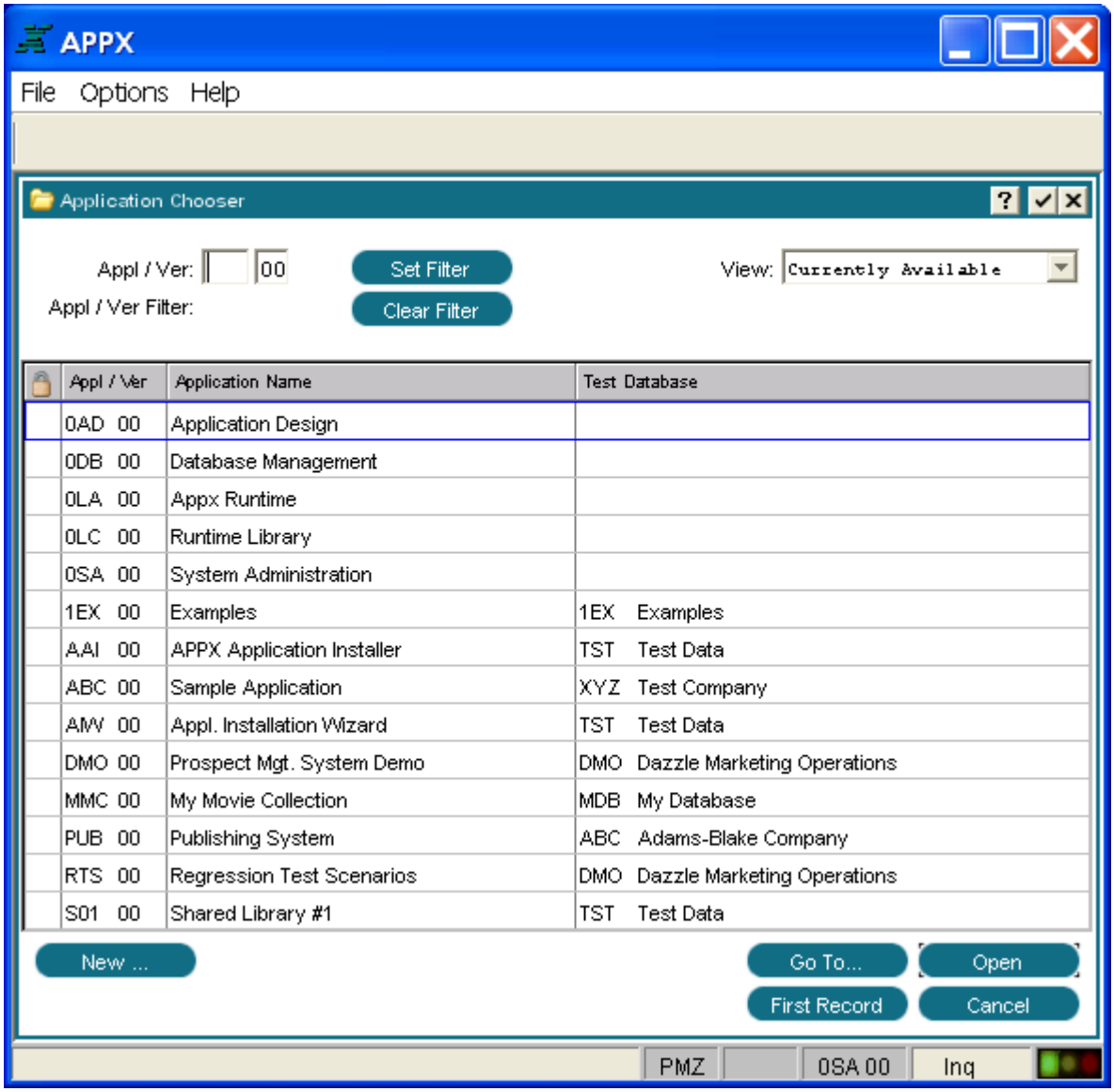


Figure 1-1-7

The Application Chooser screen displays a list of the applications in your system from which you can make your choice.

The SET FILTER option allows you to constrain the list to a particular application or version. The CLEAR FILTER option clears the filter and redisplays the unfiltered list. The "Currently Available" view lists only those applications for which you have design rights. The "All Applications" view lists all the applications in your system but you must have design rights in order to enter Application Design for that application.

The NEW option allows you to specify a new application and version, add the entry to the Applications file in System Administration, and perform Design File Management functions for the new application.

Application Properties

When you select the APPLICATION PROPERTIES option from the Application Design Specifications Menu, the Application Properties screen is displayed, as shown in *Figure 1-1-8* below.

Application Properties screen

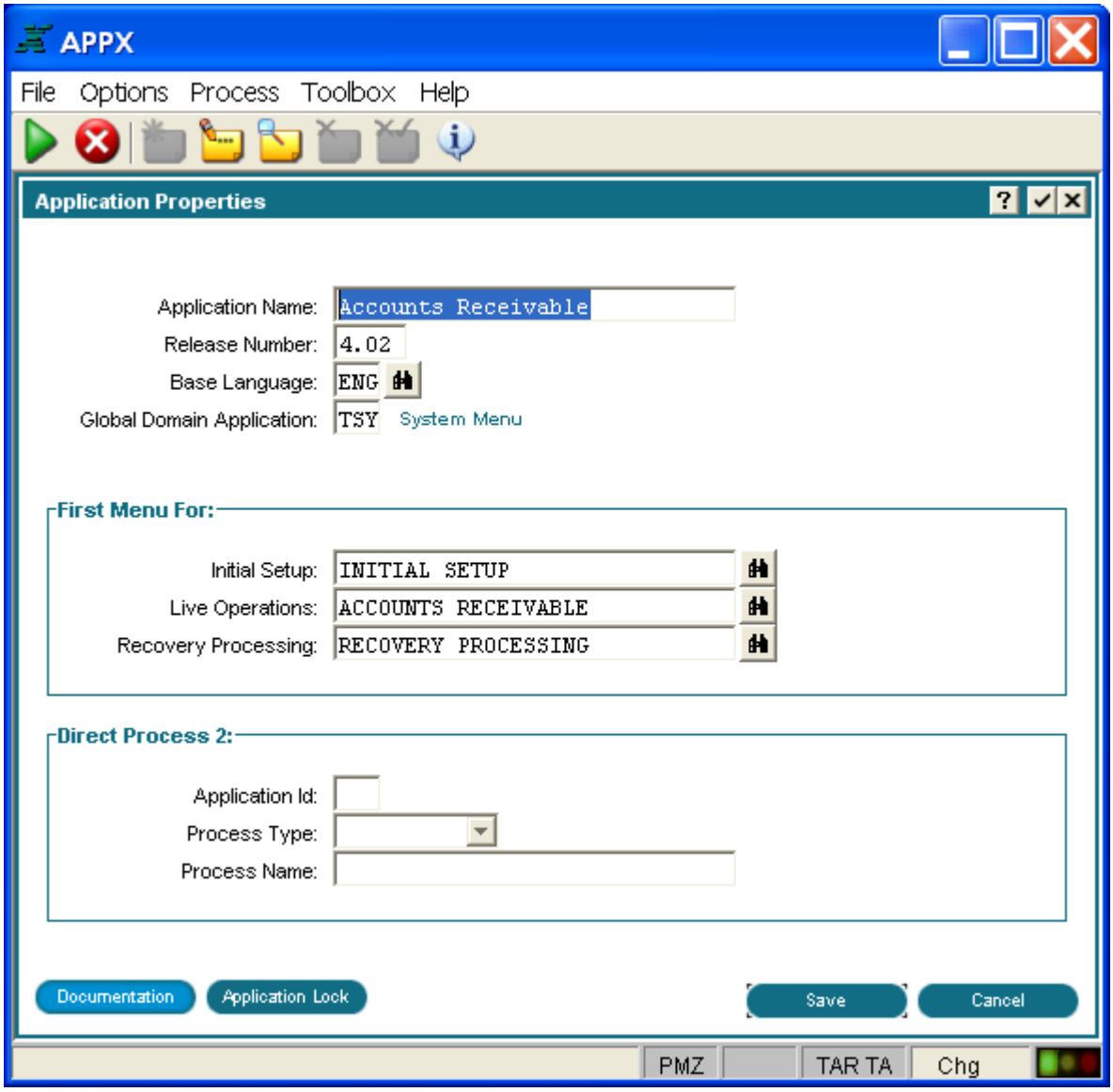


Figure 1-1-8

The Application Properties screen contains the following fields:

- Application Name. Enter a descriptive name for this application.
- Release Number. Supply the release number for this application.
- Base Language. The language the application is presented to the user in. This encompasses all parts of the application made visible to the user such as menus, input processes, warnings, errors, etc. The Language ID file can be scanned to make a selection but this field is needed only if the Language Translation feature is in use at your site.
- Global Domains App. Enter the three-character ID of the application that contains your global — or universal — domains.

You use the *Global Domains* application to help default domain names when entering the data dictionary. If you add a domain-type field, and the field name contains the name of a domain within the application, that domain name is supplied as the default for that field's domain. If no match is found in the current application, and if the field contains a domain name that exists in the *Global Domains* application you identify here, APPX will use that domain name as the default.

- First Menu For —
- ***Initial Setup.*** Enter the name of the startup menu that should be presented to the user when the application's operational phase is *Initial Setup.*
- ***Live Operations.*** Enter the name of the startup menu that should be presented to the user when the application's operational phase is *Live Operations.*
- ***Recovery Processing.*** Enter the name of the startup menu that should be presented to the user when the application's operational phase is *Recovery Processing.*
- Direct Process 2 —
- ***Application ID.*** Enter the application ID that should be executed when the end user chooses *Direct Process 2* while in this application.
- ***Process Type.*** Enter the type of process that should be executed when the end user chooses *Direct Process 2* while in this application.
- · ***Process Name.*** Enter the name of the process that should be executed when the end user chooses *Direct Process 2* while in this application. If this is a menu, all of the optional processes defined on the menu will also be listed under the *Application* selection from the user's menu bar while in that application. If *Direct Process 2* is defined as a single non-menu process, then only that process will appear as a selection under *Application*.

The Applications Properties screen contains these options:

- Documentation contains user and technical documentation regarding the application.
- Application Locks are the method by which APPX applications can restrict run-time and design access in the licensing of their applications. Applications locks may or may not be included for this application. The Author ID field and each of the Lock fields in this overlay refer to individual Product Registrations which must be present in your APPX System Registration before you will have access to the corresponding feature within this application.

Named Resources

The NAMED RESOURCES option is used to define a resource name for referencing a group of related graphic files within the application. Once a named resource has been defined, any references to the named resource on a GUI Attribute screen will result in the bitmap for the appropriate state being displayed on the image or object. This feature is used when designing GUI images that include buttons, icons and pictures. The type of image file being used varies based on where the graphical attributes are being defined. Typically, a resource is used in this manner:

Table 1-1-1. Named Resources

Defined At	Graphic File Normally Contains
Frame	Title bar icon, wallpaper bitmap
Image	Title bar icon, wallpaper bitmap
Child	Menu icon (pull-down, toolbar, pop-up)
Button	Button icon for one or more states (enabled, disabled, pressed, rollover, etc.)
Picture	Picture or image file

When you select the NAMED RESOURCES option from the Application Design Specifications Menu, the Named Resources screen is displayed, as shown in *Figure 1-1-9* below.

Named Resources screen

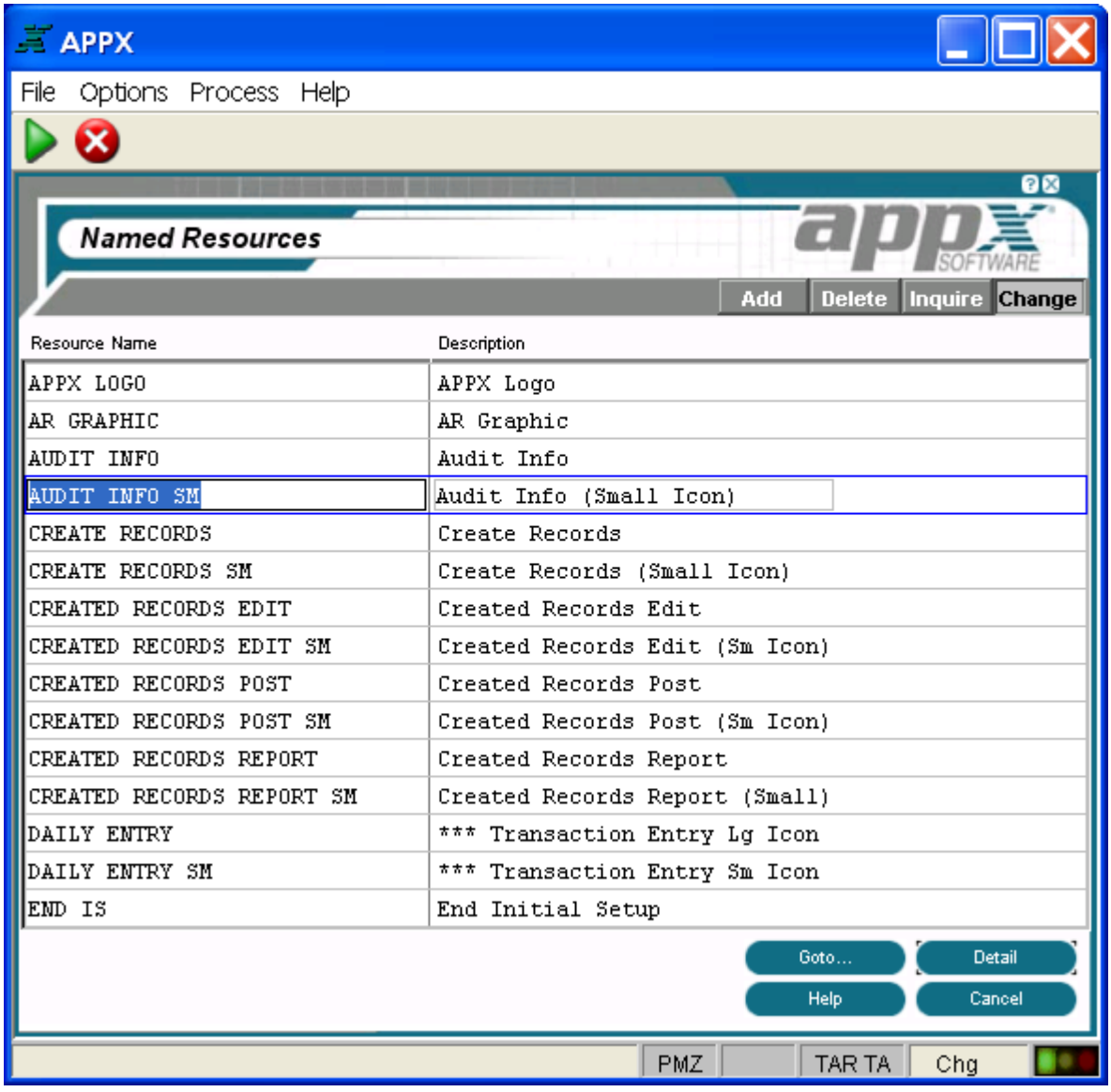


Figure 1-1-9

The Named Resources screen is a scrolling display which contains the following fields:

- Resource Name for assigning a name to a related group of bitmaps
- Description to further define the Resource Name

Select DETAIL to further define the selected resource. The detail display shown in *Figure 1-1-10* allows you to define the "states" for this resource.

Named Resources Detail screen

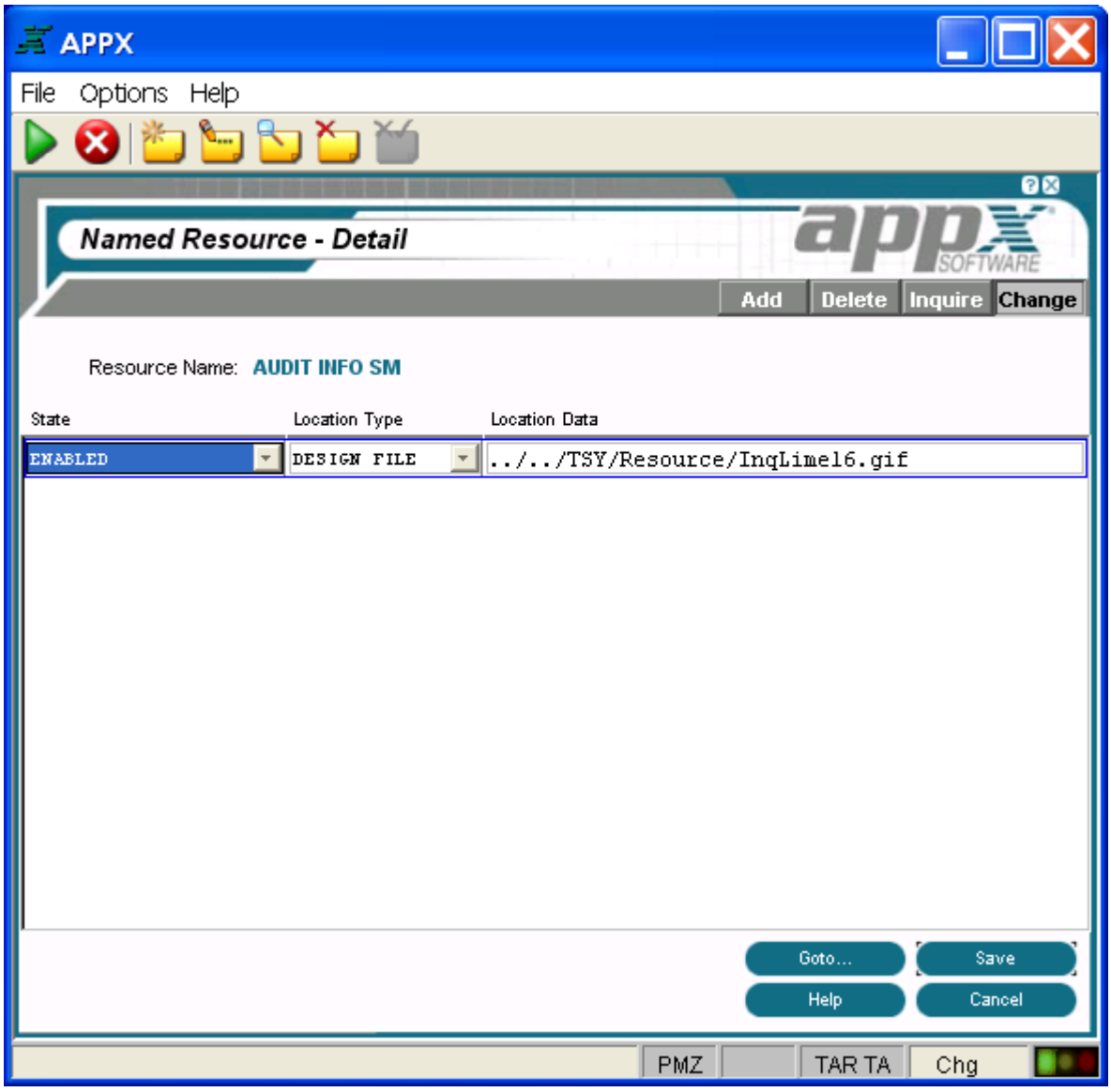


Figure 1-1-10

The Named Resources Detail screen contains the following fields:

- State identifies a separate bitmap for each of the various states which an object may assume.

ENABLED

To indicate the object is active and available for use to the user

DISABLED

To indicate the object is not available for use to the user

SELECTED

To indicate the object is currently selected by the user

ROLLOVER

To indicate the mouse pointer is positioned over the object

ROLLOVER SELECTED

To indicate the mouse pointer is positioned over a selected object

PRESSED

To indicate the object has been pressed (clicked on) by the mouse pointer

WALLPAPER

To indicate that the resource can be used as a wallpaper for a menu or input image.

- Location Type for defining the location of the bitmap

CLIENT URL

For a URL to be interpreted by the client

CLIENT FILE

For a specific file located on the client

DIRECT

Indicates that the bitmap is stored in a 2K field within the resource record

SERVER URL

For a URL to be interpreted by the server

SERVER FILE

For a specific file located on the server

DESIGN FILE

For a bitmap file stored in the application's Resource directory

- Location Data for defining the specific location information that corresponds to the Location Type used.

CHAPTER 1-2

Getting Started

Before You Begin

Before you begin to design an application, you should fully understand the concepts presented in the *APPX User Manual* and the *APPX System Management Manual*. In addition, there are some system administration tasks discussed in the next section that you or your APPX system administrator must perform. These are all one-time actions that need to be taken when APPX is first installed on your computer, or when a new application is being created.

System Administration Tasks

The following steps are among those performed by a system administrator (a user defined with system administrator privileges in the Users file):

- The addition of your name to the Users file.
- The addition of the name of the new application to the Applications file.
- The addition of your name to the Application Designers file for the new application.
- The establishment of one or more test databases for the application. Each database must appear in the Databases file, the database/application combination must be defined, and your name must be associated with the database/application in the Database Managers file.

These steps are all part of APPX's overall security measures that prevent the unauthorized use of the APPX design facilities.

Managing Design Files

All of the information that you specify about an application, including specifications for the data dictionary, processes, statements, and painted screens, is stored in a number of APPX *design files*. The information stored in these design files comprises an application. When a user runs an application process, APPX produces the appropriate actions and/or screens by interpreting the information that you specified.

When you make modifications to an existing application, APPX applies the changes to the appropriate design files. Before you enter specifications for a new application, you first provide an estimate of the number of records that each design file will contain. Next, you create the design files for the new application. APPX uses the estimated file sizes to generate the initial file extents; if this initial space is exceeded, additional space is made available. The design or operation of an application is not restricted by these initial estimates.

To create design files, select the DESIGN FILE MANAGEMENT option from the System Administration menu, which accesses the Design File Management overlay. The Design File Management overlay then accesses the Design File Management menu shown in *Figure 1-2-1* .

Design File Management menu

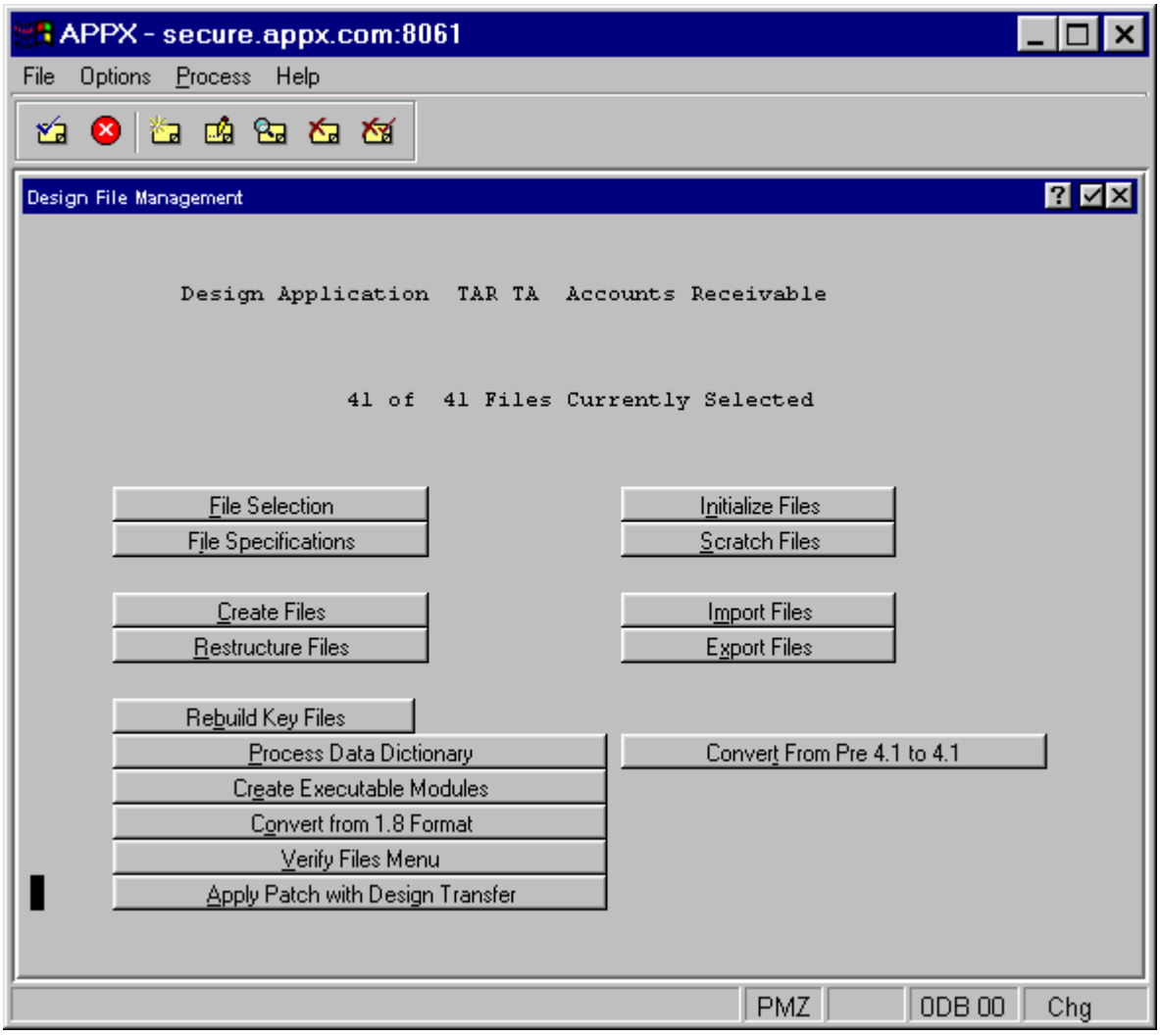


Figure 1-2-1

The Design File Management menu contains the following options:

File Selection

Normally, you perform an operation (creating files, for example) on all the design files of an application at the same time. There are times, however, when you may want to operate selectively on one or some of the design files. This option allows you to designate specific files. When you select FILE SELECTION, the screen shown in *Figure 1-2-2* appears.

Design File Management File Selection screen

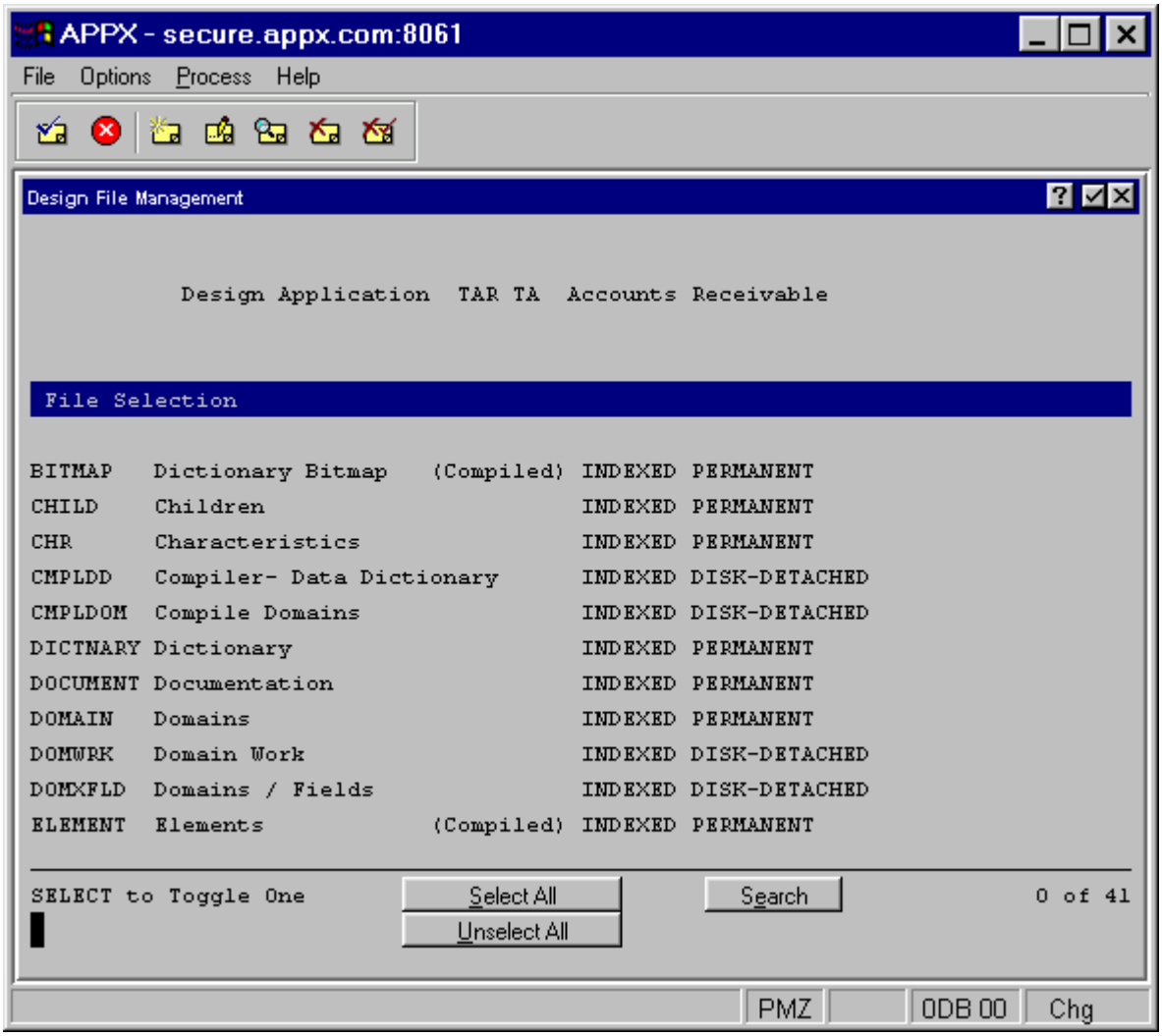


Figure 1-2-2

Click SEARCH to find a specific file or files that begin with a specified letter or letters. Use the SELECT key (F3) to select or unselect a given file. Select SELECT ALL to select all files. Use UNSELECT ALL to unselect all files. When you have selected the files you want, select END to continue.

File Specifications

Use this screen to estimate the approximate number of records for each of the design files of a new application. Remember, the numbers you enter represent the total number of records of each type for the entire application. When you select FILE SPECIFICATIONS, the screen shown in *Figure 1-2-3* appears. You may also use this option to verify you have selected the files you intended to before proceeding with other file management tools that may destroy data.

Design File Management File Specifications screen

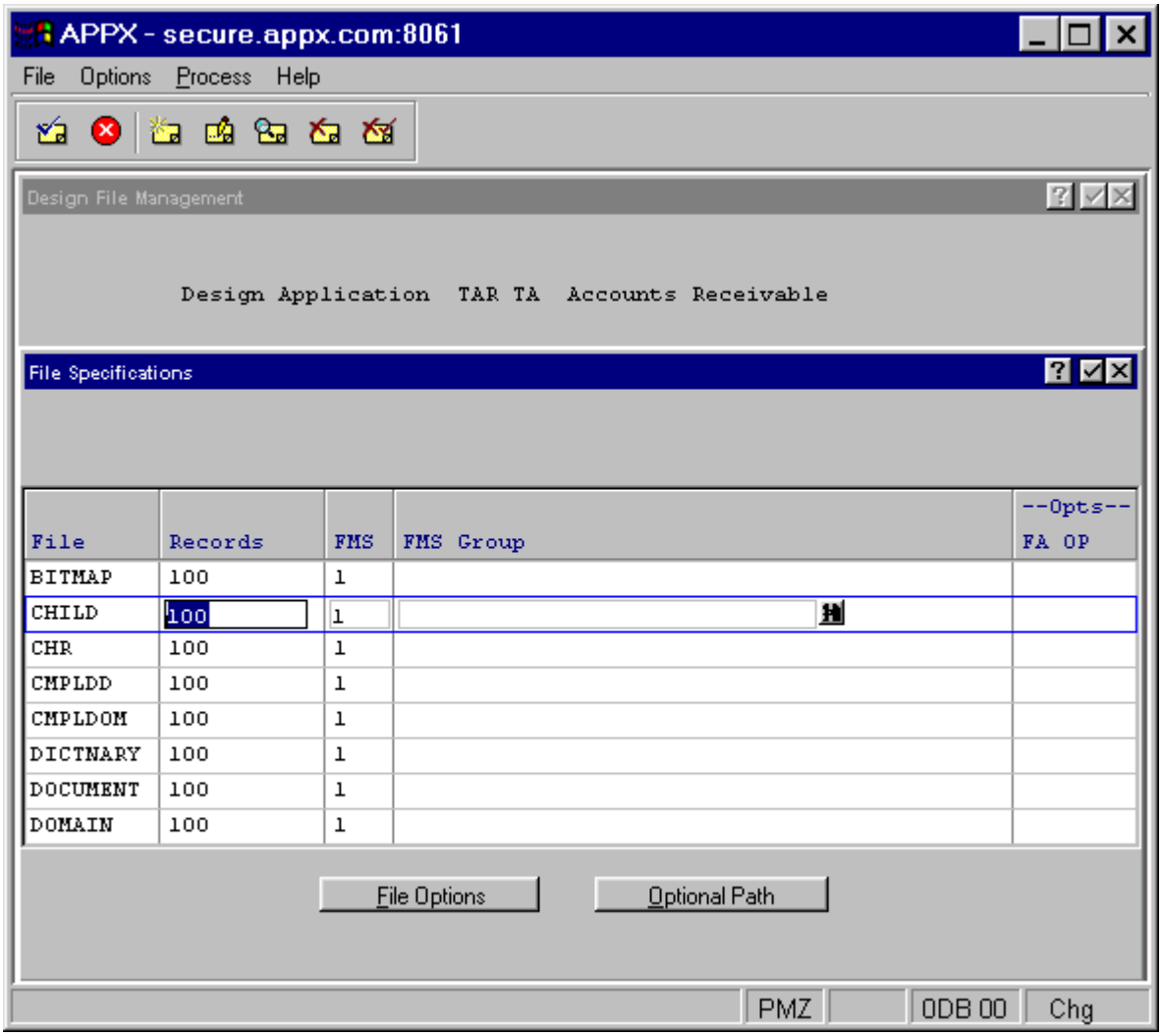


Figure 1-2-3

Following is a description of each type of design file for your information.

- ***BITMAP***. The bitmap file contains Data Dictionary processing control information.
- ***CHILD***. This file contains one record for each optional or automatic child defined in an application.
- ***CHR***. The characteristic file contains one record for each set of characteristics specified in an application. A *characteristic* is an attribute of background text on an image, such as bold, blink, or underline.
- ***CMPLDD***. This work file contains information used in the compilation of the data dictionary.
- ***CMPLDOM***. This work file contains compile information for domains.
- ***DICTNARY***. This file contains dictionary entries for this application that are used by the language translation feature.
- ***DOCUMENT***. The documentation file contains one record for each page of documentation in an application.

- ***DOMAIN***. The domain file contains one record for each domain specified in an application. You can estimate this number by counting one domain record for each field to be specified in an application, because many fields have a single domain associated with them.
- ***DOMWRK***. This work file is used during the processing of domain fields.
- ***DOMXFLD***. This work file is used when processing fields with domains.
- ***ELEMENT***. The element file contains one record for each dictionary field after processing.
- ***EXCEPT***. Used by the language translation feature, this file contains exceptions, or overrides, to previously defined translations.
- ***FIELD***. The field file contains one record for each field specified in an application.
- ***FILE***. This file contains one record for each file specified in an application after processing.
- ***FRAME***. The frame file contains one record for each frame specified in an application.
- ***IMAGE***. The image file contains one record for each image defined in an application.
- ***INITIAL***. The initial file contains two records for each record in the file.
- ***ITEM***. The item file contains one record for each item specified in an application. An *item* is a field that is entered or displayed on an input screen or displayed on a report. To compute an estimate, multiply the number of images by the average number of items for each image.
- ***PARAM***. The parameter file contains one record per application.
- ***PROCESS***. The process file contains one record for each process defined in an application, including menu, input, output, update, query, and subroutine processes.
- ***RECORD***. The record file contains one record for each file specified in an application.
- ***RESNAMES***. The file contains names of resources such as icon files, pictures, etc.
- ***RESOURCE***. The file contains path names of graphic resources.
- ***ROW***. The row file contains one record for each row specified on all the menu, input, and output images in an application. To compute an estimate for the rows file, multiply the number of images by the average number of rows for each image.
- ***SAVTRACE***. This file contains saved trace information used in the debugging of APPX applications.
- ***SELECT***. The select file contains one record for each child constraint and each query selection constraint that is specified in an application.
- ***SORT***. This file contains one record for each query-related sort specification defined in an application.
- ***SPELLIN***. This work file is used during spell checks and contains the original text entries.
- ***SPELLOT***. This spell check work file contains corrected text entries.
- ***STMT***. This file contains one record for each statement included in an application's ILF routines.
- ***SXAP2, SXDOM2, SXFLD2, SXPROC2, SXREC2***. These work files are used to store selections of various APPX elements during the application design process.
- ***TOKEN***. The token file contains one record for each tokenized value specified in an application.
- ***TRACE***. This work file contains trace information used in the debugging of APPX applications.

- ***WIDGET***. This file contains graphic file names for buttons.
- ***XLAT***. This file contains translation maps used to translate a dictionary entry of one language to the dictionary entry of another language.
- ***XREF***. This work file is used during the generation of an element (files, fields and processes) Cross Reference report.
- ***XREFNAV***. This work file is used during the generation of a Cross Reference report and contains Hyper Links, or direct paths for the maintenance of APPX elements shown on the report.

The File Specifications screen contains the following fields.

- ***File***. The files are itemized in a scrolling input window.
- ***Records***. Enter the approximate number of records you will need. If you do not know how many records you will need, leave the default value of *100*; the files can be optimized later.
- ***FMS*** (File Management System) is a system-dependent value that designates a specific file management system on your computer. Refer to the instructions in your system installation guide for specific values that apply to your computer system. (two numeric characters).
- ***FMS Group*** (optional) allows you to specify the file management group that this file belongs to. The *FMS Group* and the *FMS* (above) reference the system defaults file in system administration, which provides default values for certain file specifications.

Additionally, there are two options available on the File Specifications screen:

- ***File Options*** can be entered to provide necessary information for accessing files for certain FMS types. For instance, the database server and database name for RDBMS's such as Oracle or Sybase. Beginning with APPX 4.1, specification-based FMS controls are available for some FMS types. The older FMS Control String is still supported but the specification-based controls are encouraged. If specification-based controls are available for this FMS type, you will receive a message. Clearing your old FMs Control String and selecting ENTER will give you access to the specifications for this FMS type. See *Figure 1-2-4.*
- ***Optional Path***. APPX allows you to enter a unique path for each file, if necessary. Usually, all files are located in the same APPX-defined directory. However, for unusually large disk files or unusually small disks, it may be necessary to designate a different directory. See *Figure 1-2-5*.

Select ENTER to save the information for this file.

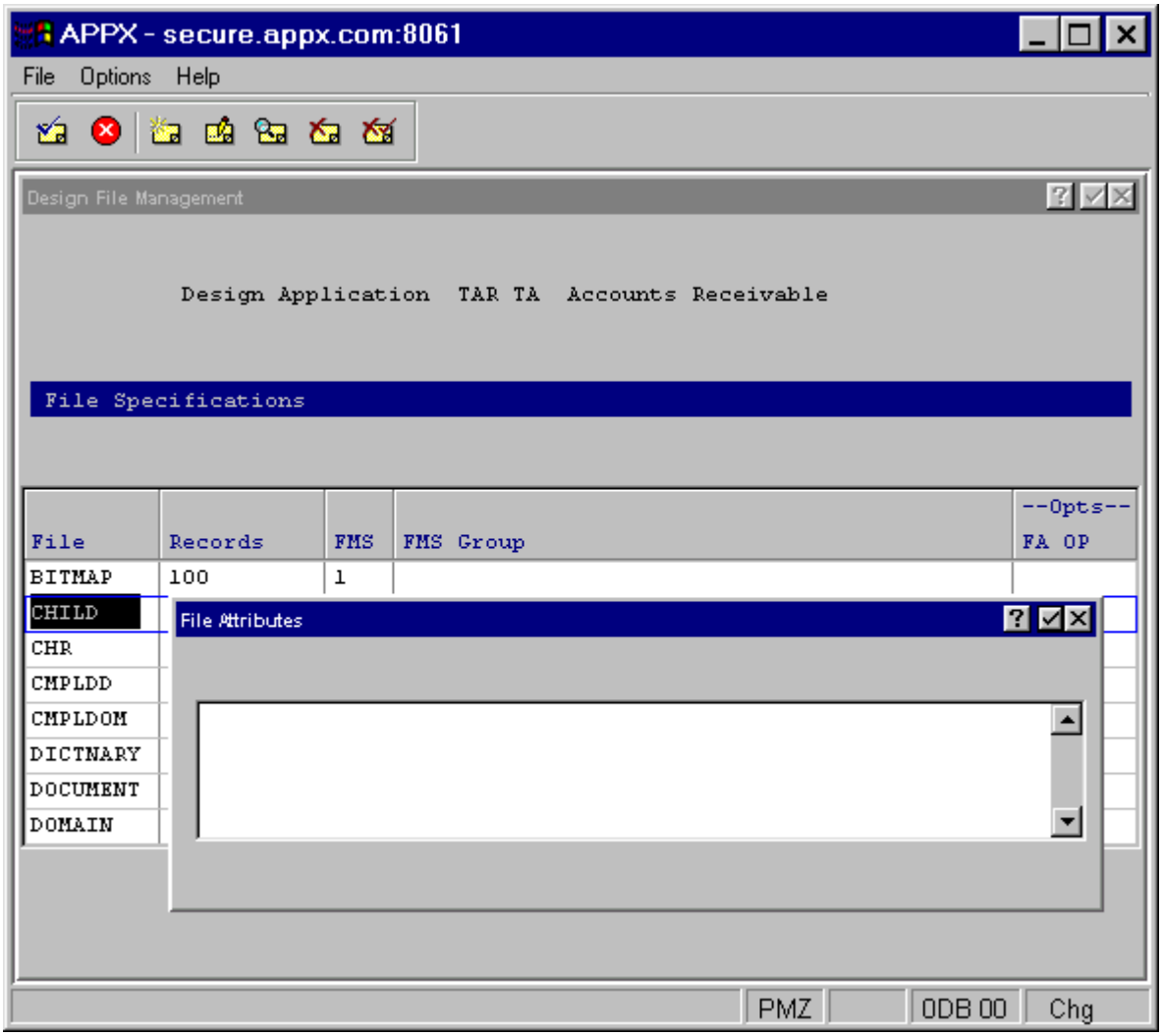


Figure 1-2-4. File Options Overlay

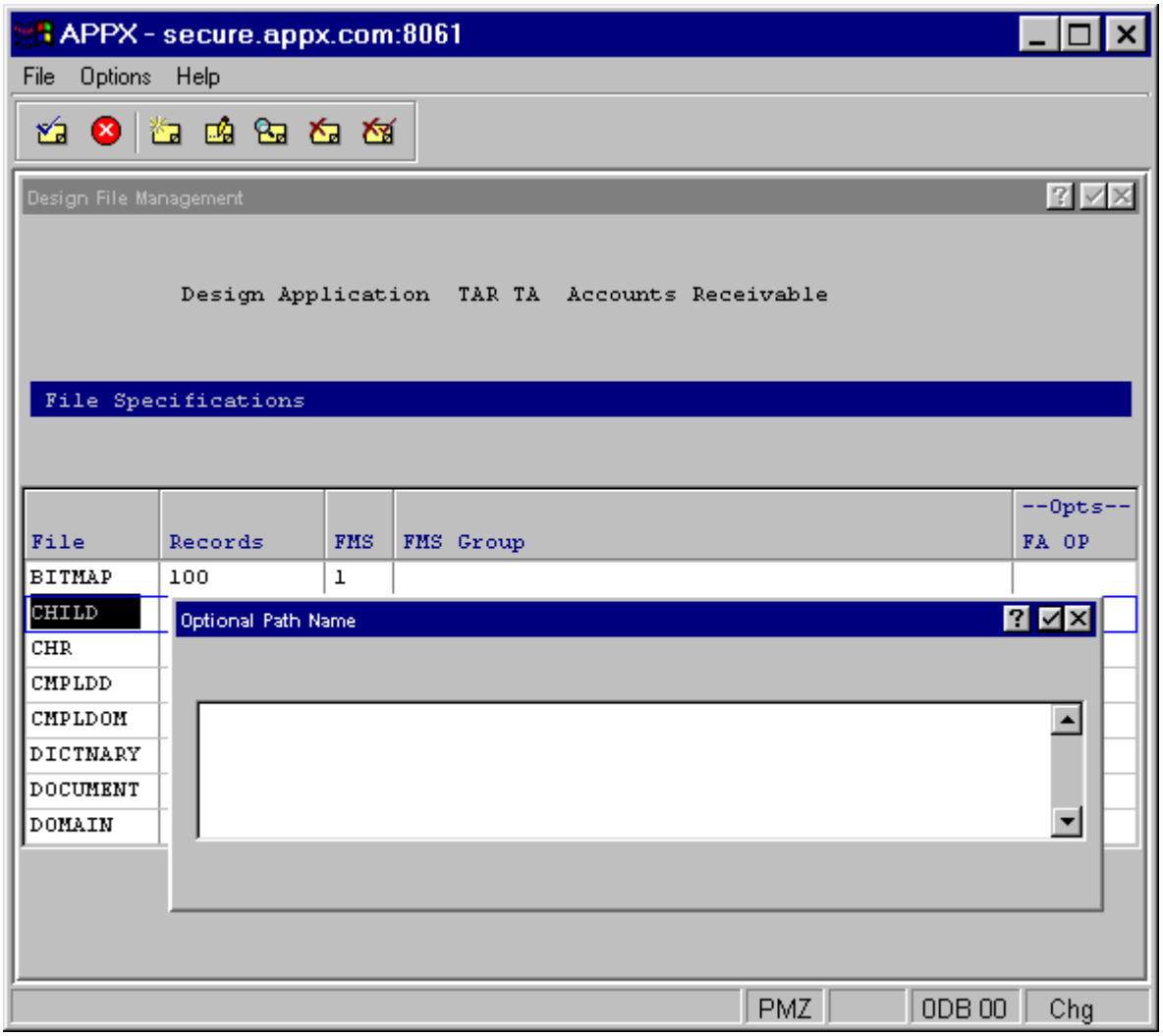


Figure 1-2-5. Optional Path Name Overlay

Create Files

Select this option to create the design files for which you have just provided estimated sizes. When this step completes, you can enter the Application Design menu and begin to enter specifications for an application.

> **NOTE:** When running *Create Files* for a database in database management, you may get some strange error messages, one of which might be, "Data Dictionary Defini." These messages occur because the ELEMENT and BITMAP files are "out of sync" with your version of APPX. To correct this situation, initialize ELEMENT, BITMAP, FILE, and INITIAL; run PROCESS DATA DICTIONARY from the *Toolbox* menu; and then run RESTRUCTURE FILES on data structure files.

Restructure Files

When a structural change is made to one or more files in an application, this option is needed. A structural change is one that affects the record layout of a file. Examples of this include adding a field, deleting a field, or changing the length of an existing field. APPX does not allow users to run a process that references a file if it needs to be restructured.

Initialize Files

Occasionally the need arises to erase the contents of one or more files. This option allows you to delete all the records in a file at once. You will receive a warning message when you select this option, advising you that you will destroy data if you continue. You can then choose to return to the menu instead of initializing the file. If you continue, you will see a "status" screen that shows you each file as it is being initialized. Select PROCEED to continue.

Scratch Files

Occasionally the need arises to eliminate one or more files completely. This option allows you to remove the selected files from the disk. You will receive a warning message when you select this option, advising you that you will destroy data if you continue. You can then choose to return to the menu instead of scratching the file. If you continue by selecting PROCEED, you will see a "status" screen that shows you each file as it is being scratched.

Import Files

If you are converting an existing application from an outside design, you can use this option to import the existing data. It must be in the specific format required by the import process.

When you select this option, the overlay shown in *Figure 1-2-6* appears.

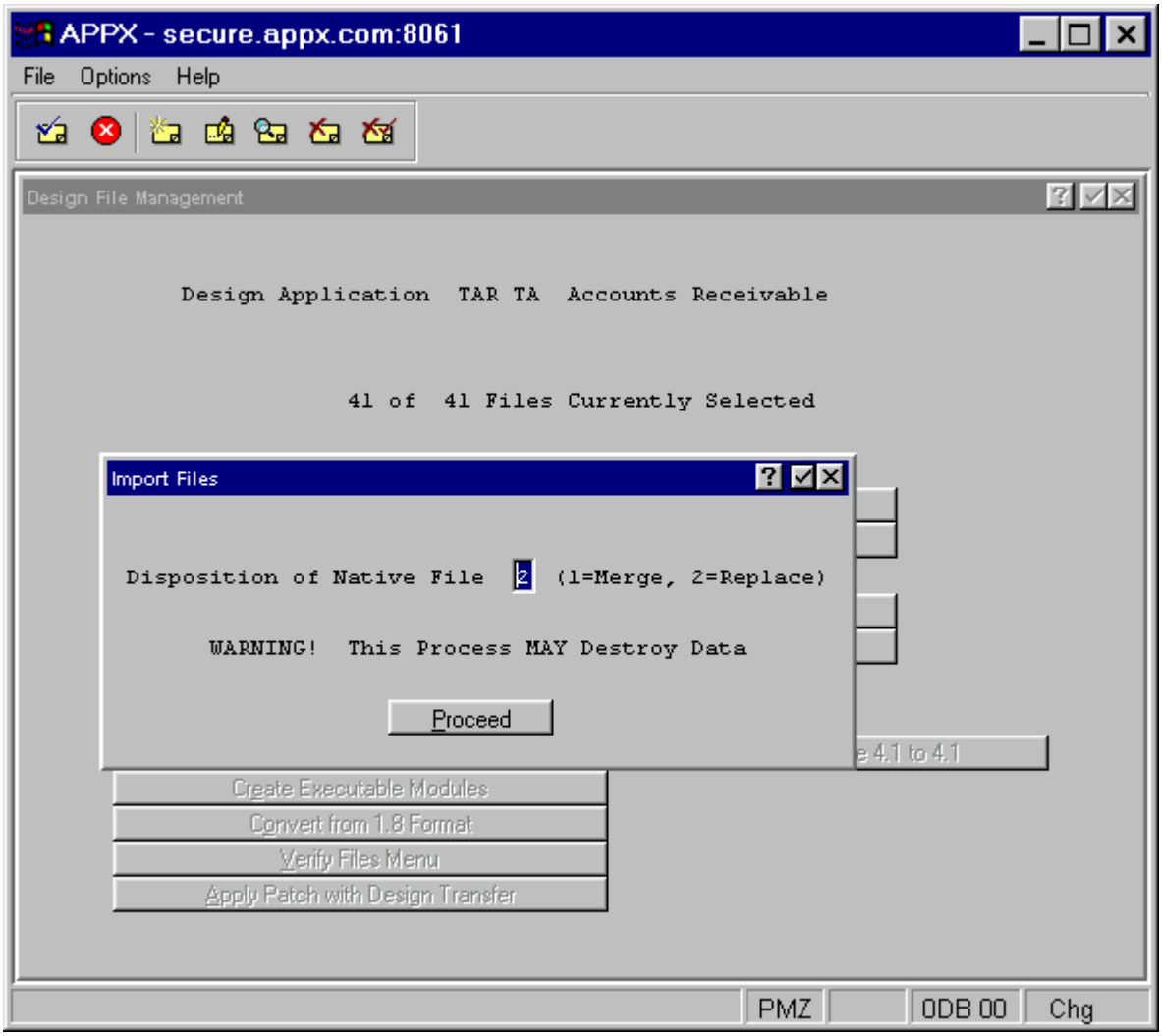


Figure 1-2-6. Import Files Overlay

The following fields are available on the Import Files overlay.

- ***Disposition of Native File.*** Enter *1* if you wish to merge the external file with any data already in the existing design file. Enter *2* if you want the external file to replace the existing design file. If there are no records in the existing design file, enter *2*.

Select PROCEED to begin the import. A "status" screen will show you each file as it is being imported.

Export Files

This option allows you to create external files for transporting to other systems. Select PROCEED to begin the export. A "status" screen shows you each file as it is being exported.

Rebuild Key Files

This option allows you to rebuild the key files (.key extension) for the selected files in the event these key, or index, files become corrupted and need to be recreated. When executed, the key file is deleted and initialized, then the data file (.dat extension) is read. The appropriate keys for each data record read are written to the key file. A "status" screen shows you each file as its key file is being rebuilt.

Process Data Dictionary

If your site has Application Design Rights, this option invokes a menu that allows you to select from these options:

- Enter and modify information about the application's domains (field prototypes).
- Specify and maintain information about the application's files and fields.
- Define and maintain the application's work fields.
- Compile the data dictionary when newly defined, or after any changes have been made to existing information.

Otherwise, this option automatically compiles the data dictionary for all domains, files and work fields in the specified application.

Create Executable Modules

This option compiles all processes in the application and creates Executable Modules (object code) that are used during runtime. This is recommended immediately following APPX version upgrades or other mass changes to eliminate processes from having to be compiled at runtime. An error log is created and contains all processes with compile errors.

Convert From 1.8

Converts APPX applications from a version prior to 2.0 to the 2.0 format. Applications in versions prior to 2.0 are not compatible with APPX versions 2.0 and later.

Verify Files Menu

The options contained on this menu, shown below in *Figure 1-2-7*, are used to verify the integrity and structure of APPX files. These utilities are used in the event an APPX file appears corrupted or otherwise dysfunctional.

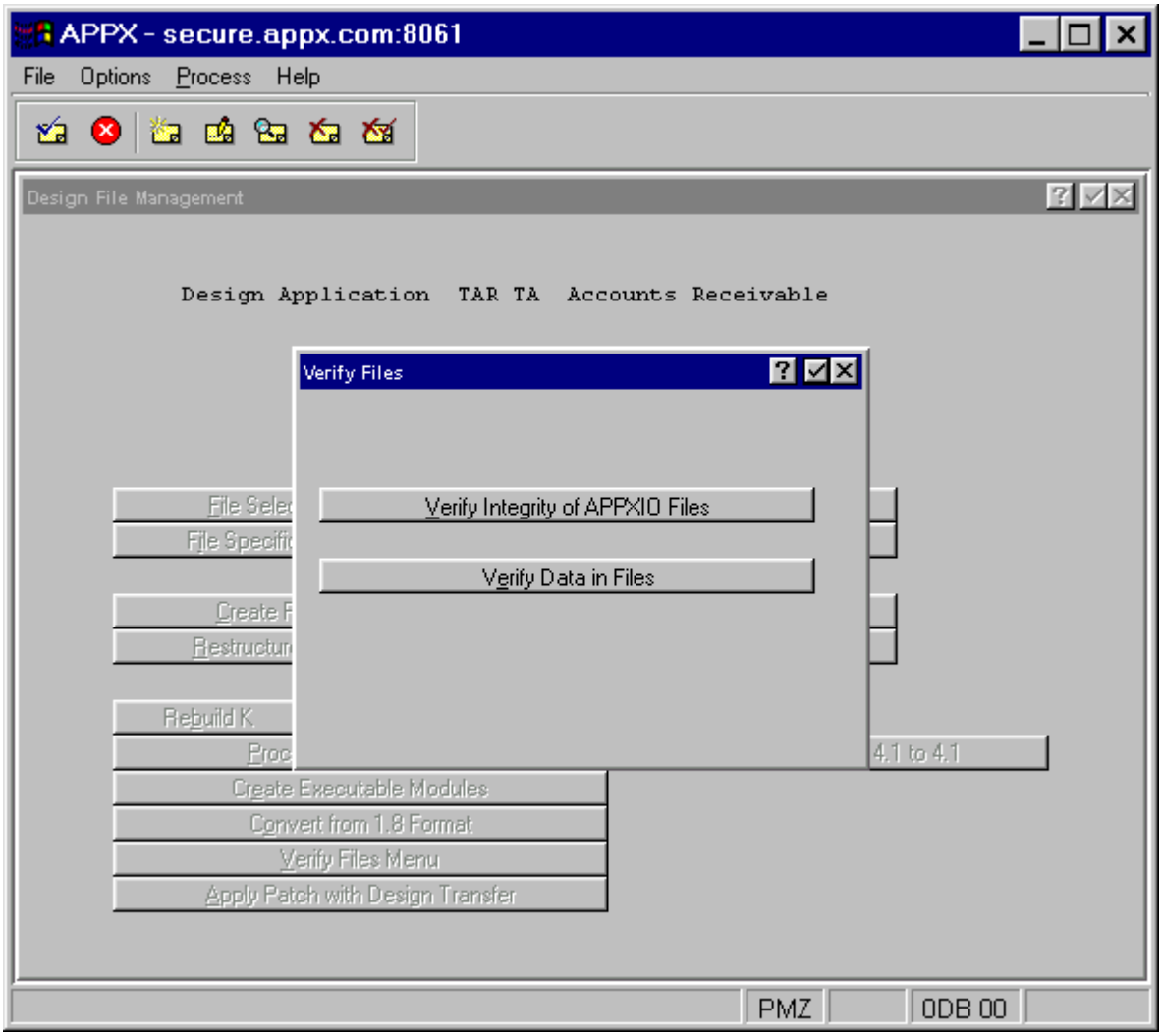


Figure 1-2-7. Verify Files Menu

The two options on the Verify Files Menu are:

Verify Integrity of APPXIO Files

Is a utility that checks database integrity of all selected files. A log is created in your default print directory and is displayed on your screen. Any error messages contained in this log should be reported immediately to APPX Technical Support.

Verify Data in Files

Verifies the field contents in each record of a selected file or files. This is dependent upon the *field type* as defined in the data dictionary. Referential integrity can also be verified and is dependent on "*data lookups*" defined in the data dictionary for each file. Any errors encountered are placed in an error log that is created in your default print directory.

Apply Patch With Design Transfer

This option accesses the design transfer process to transfer all design components from an indicated application into the selected application. This feature can be utilized by APPX developers to "patch" their applications at customer sites. The developer can create a "patch application" which contains new or modified design components. The patch application can be distributed to customers and transferred into the appropriate application using this feature. The Release Number field in Application Parameters will be set to the value in the patch application.

Once the design files are created, proceed with developing an application's processes. The following section provides some approaches you may want to consider as you perform your analysis and design.

Planning Your Application

This section describes a typical APPX approach for developing and implementing application systems. Because the system design process is complex, and because designers approach system design from different viewpoints, there is no single best way to use APPX. It is a flexible tool that can accommodate a variety of design methods. This section presents an approach that new designers may want to use as a pattern to develop their initial applications. As you become more familiar with APPX, you can make adjustments that suit your own design style.

Design Methodologies

In complex application environments, it is likely that a designer has followed some general methodology as a guideline for defining the components (including files and data fields, inputs, outputs, and menus) of a proposed application. In less complex applications, the methodology may be much less formal, using only outlines, lists, notes, and sketches.

APPX places no restrictions on the design methodology you choose, and enables you to translate the system requirements/conceptual design into a functional APPX application.

Approaches to System Design

There are three fundamentally different approaches to defining a data processing system: data-driven, output-driven, or process-driven. Each of these is discussed below.

Using a *data-driven* approach, you first define all the data fields for an application. Next, you specify the relationships between the data fields and then develop supporting processes. Proponents of this approach believe that if they can define all data-related information in advance, processes to support the entry, update, review, and retrieval of data will flow naturally as a by-product of data definition.

In an *output-driven* design, you first define all the required outputs (both screen displays and printed reports) and, then, you develop the data structures, and input and update processes that you need to derive them. If the desired outputs can be defined, they can work "backwards" through the system design to define all the supporting structures that allow the system to produce the specified outputs.

Using a *process-driven* approach, you define each step in the work process (such as shipping finished goods, preparing invoices, etc.) and develop parallel steps in the application system to support the normal work flow. Proponents of this approach feel that a computerized application should be developed to reflect the day-to-day business operation of a company or organization. If the processes are well-defined, the details of data entry and retrieval can be logically defined as a secondary task.

APPX supports any of the three approaches that are described above. After you have developed the preliminary definition of a proposed application, however, there is a preferred sequence of steps to make the building of an application in APPX more straightforward.

Analyzing Your Requirements

Before you actually enter the APPX Application Design menu, you may want to think through the application by following a series of steps. You may choose to document the results of your analysis in some written fashion, although it is not required. On the other hand, there is no penalty for "thinking through" your design by using APPX directly as a prototyping tool, and making additions and revisions as you go along. The specification screens are organized so that you can develop your application in a logical way.

Although the sequence of design steps is not fixed, the following list represents one approach to performing a preliminary analysis before entering information into APPX:

1. Identify the tasks or functions you expect the application to perform. Organize the tasks into menus. Remember that, in APPX, a menu is a type of process, and you define menus just as you do inputs and outputs.
2. Next, consider what processes are required to accomplish each task defined above. Some tasks may require more than one process to achieve the desired result. Identify processes by type (menus, inputs, outputs, etc.) and assign names to each one. You may want to assign corresponding names to related processes, such as CUSTOMER QUERY and CUSTOMER LIST.
3. Define and analyze your application's data requirements. Identify the files and fields to be used in the application. For each file, specify the information that you need to provide when you define the file. Define the keys and other fields contained in each file to determine what attributes each field will have. Analyze all of the files in your application to identify fields with similar attributes that can be created as domains.
4. After you gather this information you are ready to create the design files for the new application.

Remember that APPX is very flexible, and allows for revisions and additions at any time. Do the best analysis that you can in advance, enter the specifications in APPX, and make modifications as you discover things you omitted.

Entering Specifications in APPX

At this point, you are ready to enter information into APPX and actually begin the application specification process. To initiate this action, select the Application Design menu, and proceed as follows:

1. From the Application Design menu, select the *Database Management* option. At this point, you may want to select the *Domains* option, and proceed to specify all the domains in your application. Domains are generic definitions for fields in an application that share a common set of attributes. If you later find that you forgot to enter any domains, you can enter them as required. Then select the *Files* option and specify the information for the first file. From here, you move automatically to the Field screen, where you specify each field contained in the first file. Continue to enter the specifications in the same manner for each file.
2. After you complete the data dictionary specifications, the data dictionary will process so that it is in a format APPX can use as you define the application's processes.
3. Return to the Application Design menu and select the *Processes* option. Typically, your first step is defining menus. Enter the required information, and paint the first menu screen. There are two ways to do this:

- You can first paint each option on the menu and immediately identify and specify all the details of the process to be invoked when this option is selected. Then, paint the second option and specify it in its entirety and continue to modify the original menu as each option is added.
- OR, you can paint the entire menu screen at one time, identifying all options in a single pass, and then go back and specify the details of each process in any sequence you choose.
- Either method results in a complete specification of a menu, its options, and the processes to be invoked when each option is chosen. This pattern (of two or more fundamentally different ways of proceeding) is common throughout APPX, and demonstrates its flexibility.

1. If you can separate your application into discrete functional components, you may wish to define and specify one component at a time in APPX. You can then review what you have entered, and begin to test the partial application to see if it works properly to that point. This approach may help you isolate problems and inconsistencies within one area before going on to the next. APPX allows you to define test databases to test your application immediately after making any changes, without leaving the application design environment.
2. After you make a first pass at specifying the entire application, or one or more distinct application components, you are ready to begin testing. You first need to define and create the appropriate application data test files. This is a function of database management and, depending on how

security privileges are assigned in your environment, you may require assistance from your database administrator to create the test files.

3. Test all the processes and capabilities in the new application. Verify your results and use the statement debugger as needed to help trace logic problems. When you are satisfied that your application operates properly, the database administrator can create production data files and the application is ready to use.
4. Once an application is in production, APPX supports all of the steps involved in the normal maintenance cycle during which changes and enhancements are applied. When you make changes to the data dictionary, APPX will process the files that are affected; if structural changes occur (such as adding a new field to a file), you need to perform a *restructure*. You do not have to take any extra steps if you make changes to processes. APPX automatically keeps track of changes and compiles new versions if required.

Defining Keystroke Macros

As you become a more experienced APPX designer, you may find that you often perform the same tasks that require the same sequence of keystrokes every time you need to perform them. This applies to the ILF editor (writing procedural code), the Image Editor (painting screen and report images) and any other task that is performed by executing a specific sequence of keystrokes. With APPX's *Keystroke Macro* feature (**available only in character mode, not in the APPX Client**), you can "program" these keystrokes into an option number you select and then execute the series of keystrokes anytime just by selecting the option you defined. APPX "remembers" this option and the macro associated with it so it is available for use every time you log in.

To define a macro, begin by pressing the `CTRL` and `R` keys simultaneously (the R stands for *remember).* When you see a message at the bottom of your screen that tells you to *Define Option*, select the option number you want to assign to the series of keystrokes that you will program into the macro. To assign option 77 to the macro, for example, you'll press `OPTION`, `OPTION`, `7`, `7`). The `OPTION` key is defined by your System Administrator.

After assigning an option number to the macro, a new message will tell you to *Define macro*. Now you simply execute your task and APPX will record all of the keystrokes you make. When you are finished, press the `CTRL` and `R` keys again to let APPX know your key sequence is complete. Then any time you want to execute those same keystrokes, just select `OPTION` `77` and the keystrokes will be played back.

CHAPTER 1-3

Data Dictionary

The Data Dictionary: Cornerstone of an Application

A *data dictionary* is a repository of information describing an application's data. In APPX, this information includes both data specifications and supporting documentation.

Data specifications include information such as a file's *Descriptive Name* or a field's *Length*. The data dictionary is the cornerstone of application development. All processes operate on the basis of specifications in the data dictionary. In APPX, you use the data dictionary to define all the attributes of data.

Characteristics of the APPX Data Dictionary

The data dictionary is *active* in that its specifications are used by, and place real-time restrictions on, both designers creating applications and users operating them. For example, assume you defined a numeric, five-digit CUSTOMER ID field. APPX uses the length specification to provide the proper *mask* when a designer paints the field onto an input image. APPX also uses data dictionary specifications as a basis for validation when an image is executed. The system displays an error message if a user attempts to enter a non-numeric character into the CUSTOMER ID field. An active data dictionary maximizes consistency, productivity, and integration. It minimizes the chance for error, both when designing and using applications.

To enhance performance, you process APPX's dictionary before using it. Specifications entered into the data dictionary are translated into a format more directly usable by APPX during process execution. The process also searches for logic errors (such as invalid key definitions or illegal record lengths), constructs a default value record for each file, computes field lengths, establishes the starting position for each field within a record, and determines whether or not the changes may affect the structure of existing data files.

The dictionary provides for integrated documentation, including entry screens for online help text and printed technical documentation.

To ensure integrity, the dictionary is automatically monitored for structural changes. Examples of structural changes include adding a field to a file or expanding the length of an existing field. By contrast, changing the standard column heading for a field is not a structural change. Whether you make a structural or nonstructural change to the dictionary, all processes that reference changed elements are automatically updated. APPX flags existing files that require restructuring, and the restructure facility (available as an option on the Database Management menu) enables your database

administrator to convert them to the new structure. This facility reduces the need to "freeze" the file structures in applications. Data dictionary changes requiring restructure should be made when the file is not in active use by the end user.

Accessing the Data Dictionary

You access the APPX Data Dictionary from the Application Design menu by selecting the DATABASE MANAGEMENT option. The Database Management overlay includes options to edit and process the Data Dictionary. These options are introduced in the following sections of this chapter and described in detail in Chapters 2-1 thru 2-5 of this manual. The final option, Physical File Management, is covered in *Chapter 3, Database Management,* of the *System Management Manual.*

The next section provides a schematic diagram of the data dictionary design options.

The Components of the Data Dictionary

The principal components of APPX's data dictionary are the application's *domains*, *files*, *fields*, and *work fields*.

Domains

Are generic definitions, or patterns, for fields that share a set of common attributes. For example, your application may include a number of different fields that store a vendor number—a purchase order vendor number field, an accounts payable vendor number field, an invoice entry vendor number field, and so on. Each of these fields is the same length and has the same alpha and numeric components. You define these common attributes in a "vendor number" domain, and have each of these fields reference the same domain.

Files

Are data storage structures that consist of a collection of related ***records***. Each record contains one or more ***fields***. The record layout for a file is determined by the fields defined for that file within the data dictionary.

Fields

Are the most elemental data storage structures of applications. They store the individual items of information available to users of APPX applications.

Work fields

Are temporary fields that are not stored in a file, but which you can use in APPX processes. You cannot save the contents of work fields after a user exits APPX, although you can move work field values to other fields that are stored in permanent files.

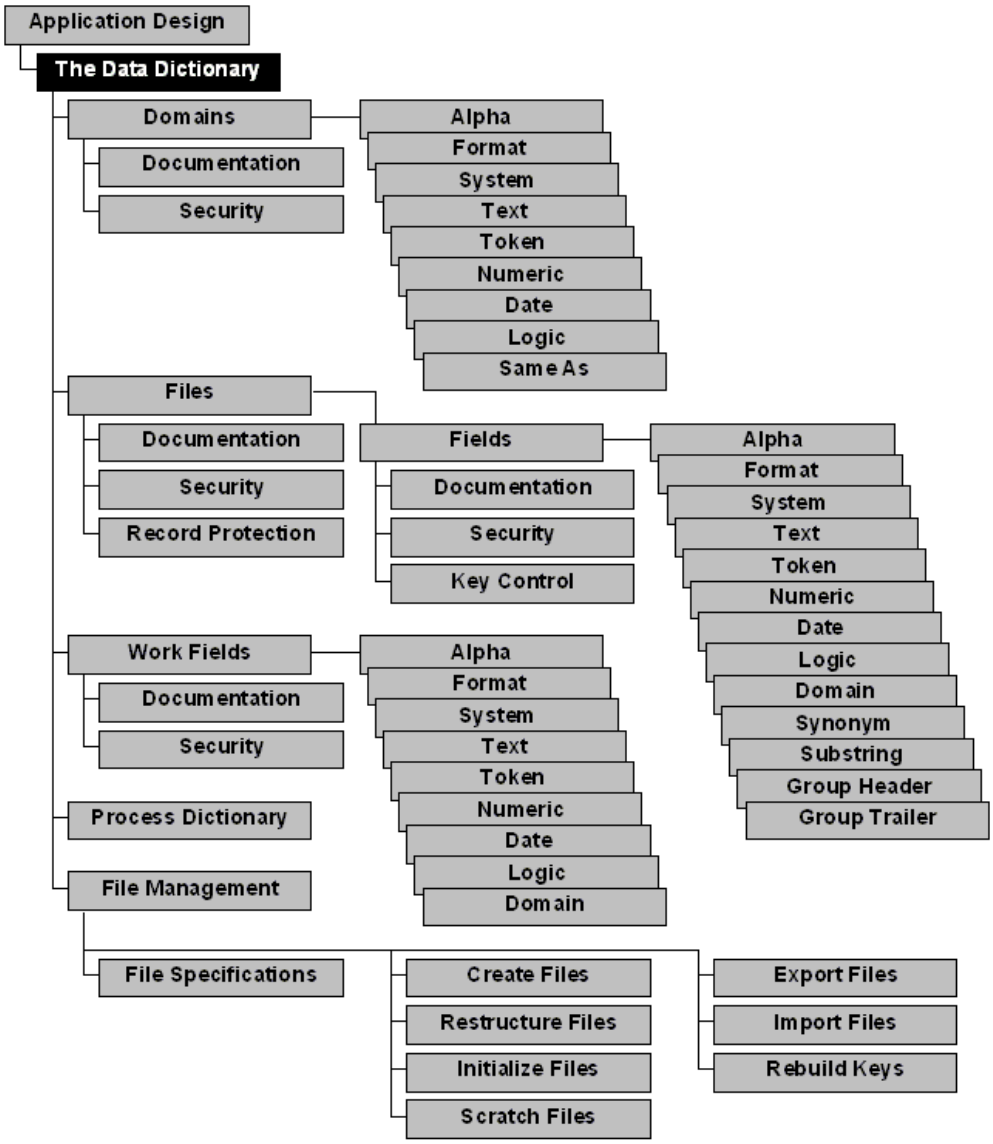


Figure 1-3-1. The Data Dictionary

Domains and Domain Types

Domains are generic definitions, or patterns, for fields and work fields that share a set of common attributes. These attributes include such things as the physical space a field requires, the set of values it can contain, and the way a field is presented to a user.

There are nine types of domains in the APPX data dictionary*:*

- *Alpha* domains are patterns for fields comprised of letters, numbers, and special characters (-, #, or &, for example), where the primary characteristic is field length.
- *Format* domains are like alpha domains except that an edit mask is the primary characteristic.

- *System* domains are patterns for fields that contain data provided by the system, often referred to as audit fields.
- *Text* domains are patterns for fields comprised of blocks of text.
- *Token* domains are like alpha domains except they have specific values that are stored as corresponding numbers.
- *Numeric* domains are patterns for fields restricted to number, decimal (.), and sign (-) entries.
- *Date* domains are patterns for fields that contain dates and/or times, from a hundredth of a second through a century.
- *Logic* domains are patterns for fields with possible values of yes or no , or to represent an unknown value.
- *Same As* domains are patterns for fields that share the same characteristics as a domain in another application in your database.

The Functions of a Domain

Each domain specification provides one or more of the following:

Storage definitions

From which APPX determines the internal storage length and format of any field that references the domain.

Display characteristics

For items painted on an image, such as display masks or justification options.

Constraints and restrictions

On a field's valid values, including range checks, validation tables, edit masks, and data lookup.

Other field attributes

Including column headings, descriptive names, default values, and security codes.

The Advantages of Using Domains

Domains provide a number of significant benefits, including:

Minimization of development effort

By eliminating duplicate entry of common field attributes and assumed values.

Minimization of maintenance effort

By providing a single location from which changes flow through to all related fields in an application.

Assurance of consistent database definition

Throughout the system. The length or edit attributes of the customer number, for example, will never vary from CUSTOMER file, to INVOICE file, to CASH RECEIPTS file, and so on.

An Example of a Domain

One example of a domain might be employee number which should have the same number of characters and the same editing characteristics wherever it is used in a system. You can define such common attributes—including a descriptive name if desired—once, in an EMPLOYEE NUMBER domain. Domain attributes apply to all fields referencing that domain and cannot be overridden at the field level. Thus, any attributes that are not appropriate for each field based on the domain should be left blank in the domain definition. For instance, the employee master file may include employee number fields for both the employee and the employee's supervisor. While "Employee Number" is a good descriptive name for the first field, "Supervisor Number" is more appropriate for the second field. To accommodate this situation, just leave *Descriptive* blank in the domain definition and enter individual names instead. Meanwhile, both employee number fields remain subject to the same format and editing criteria defined for the domain.

Fields and Field Types

Fields represent the smallest units of data storage in an APPX application. Each field contains an item of information in a database. While a domain establishes a set of common attributes (such as required physical storage space) for one or more fields, it is the field that is used to store application data. The data information stored in a field is referred to as a *field value*. Fields are grouped together as records stored within files.

There are thirteen types of fields in the APPX data dictionary:

Domain

Fields are fields that directly express the attributes defined in a domain while they define and occupy new physical space in a file. There are nine distinct domain-type fields: Alpha, Format, System, Text, Token, Date, Logic, Numeric, and Same As. Refer to the Domains and Domain Types section for additional information. You can choose to set up physical fields to exclusively reference domains, or you can set up individual field types directly.

Alpha

Fields are comprised of numbers, letters, and special characters (-, #, or &, for example).

Format

Fields are like alpha fields except that an edit mask is the primary characteristic.

System

Fields, often referred to as audit fields, display data that is provided by the system.

Text

Fields allow the user to enter blocks of text on the data entry screen.

Token

Fields are like alpha fields, except that they have specific values that are stored as corresponding numbers.

Numeric

Fields are restricted to number, decimal (.), and sign (-) entries.

Date

Fields are patterns for fields that contain dates and/or times, from centuries through hundredths of a second.

Logic

Fields are fields with possible values of yes or no , or to represent an unknown value.

Synonym

Fields provide alternative names for referencing existing APPX fields. They occupy no physical space in a file aside from the fields they reference.

Substring

Fields represent discrete portions of longer alphanumeric fields. They occupy no physical space in a file aside from the fields they are a part of.

Group header

Fields are used to group other fields by identifying the beginning and ending fields in a group of fields. The group can then be referenced by the name of the group header field. Group headers occupy no physical space in a file aside from the fields that comprise the group.

Group trailer

Fields are optional placeholders to allow for terminating a group without referencing a physical field. They provide a streamlined method to allow for additions to the end of groups. They also provide visual clues as to the ends of groups when scanning the Data Dictionary. Group trailers occupy no physical space in files.

Work Fields

You can define temporary domain-type fields called *work fields* to calculate, temporarily store, or display values that are not stored in a file. A work field is analogous to a record that has no file storage characteristics and contains a single field. A work field has no sequence number and no relationship to any files, records, or other work fields. You can manipulate work fields in much the same fashion as any other field.

You cannot save a value in a work field after a user exits APPX. To save a work field value, you must copy it into a field that is part of a permanent file's record. You cannot define a group of work fields because they are not arranged in a predictable order. Work fields cannot be keys because they are not part of a record structure within a file.

Audit Trail

APPX provides for an automatic audit trail within the data dictionary by the use of audit fields (i.e., system fields or system domain fields). These fields track when and by whom records are updated for a file within an APPX input process, as described below:

Date-added

Fields contain the date or date/time that the record was added to a file.

Date-changed

Fields initially contain the date or date/time that the record was added to a file, and are updated each time any user modifies the record.

User-added

Fields contain the user ID of the person who originally added the record to a file.

User-changed

Fields initially contain the user ID of the person who added the record to a file, and are updated each time any user modifies the record.

Storage Options and Files

You determine whether the records you define in the data dictionary are to be stored in files and, if so, whether the files are maintained on disk or in memory. There are several factors to consider when selecting which type of storage to use:

- *Permanence*—Do the records need to be retained permanently or only for a specified period during current processing?

- ***Capacity***—Is capacity a limitation that needs to be considered? How much space do you expect the collection of records to consume?
- ***Shareability***—Are there many users that may need to access records in the collection simultaneously?
- ***Performance***—How important is it that records can be accessed very rapidly during processing?
- ***Database dependency***—Are there security considerations that must be addressed when defining file attributes and storage options?

There are three types of storage options within APPX: *disk* (permanent and temporary), *memory,* and *working-storage*. Each type implies a number of characteristics and offers certain advantages, as described in the following sections.

Permanent Files

Permanent files are disk files and are shareable among users and stored on a disk surface defined in database management. The system will always take longer to retrieve data from files stored on disk than to retrieve data from files stored in memory. Disk files have an advantage over memory files because they are less limited in capacity.

Disk Files (Temporary)

As described in the preceding paragraph, permanent files are stored on disk. You can, however, define temporary files that are stored on disk as well. Temporary files are not shareable between users and not retained after you exit APPX. Note that the system will always take longer to retrieve data from files stored on disk than to retrieve data from files stored in memory. Disk files have an advantage over memory files because they are less limited in capacity.

There are three different levels of interprocess communication between disk files: *subprocess*, *related*, and *detached*. See Chapter 1-5 for information about interprocess communication.

Memory Files

Memory files are user-specific files (i.e., they can never be shared between users) loaded into the system's main memory, and are not retained after a user exits APPX. Because the files are maintained in memory, the quantity of data these files contain is generally relatively small. Excessive or improper use of memory files can result in a memory overflow condition.

The principal reason to use a memory file is to improve performance. The system will always take less time to retrieve data from files maintained in memory than to retrieve data from files stored on disk.

There are three different levels of interprocess communication between memory files: *subprocess*, *related*, and *detached*. See Chapter 1-5 for information about interprocess communication.

Working-Storage Files

A file with a *working-storage* specification can be used to enter data, perform edit checks and computations, and execute statements without actually storing data. This type of file provides you with the equivalent of work fields that can be grouped in a record format. It may be beneficial to use this storage-type when dealing with a group field, such as a general ledger account number with distinct but related subparts.

These files are often used in record comparisons. You create a *work record* that contains the same fields as the disk-based record and, using a group field that includes the entire record, copy the contents of the disk record into the work record with a single statement. You can then compare this stored record with another record from the disk file.

Calculator functions are another example of the use for this type of file. You could use a *working-storage* file-type to receive principal and interest rate data, and compute and display a monthly payment amount. Users could re-enter different values, and the computation and display can be performed as often as necessary without saving the entered or calculated information.

Working-storage fields have a SUBPROCESS scope. They can be shared between a process and its subprocesses.

File Organization

APPX supports three types of file organization: *one-record*, *sequential*, and *indexed*. Your selection of a file organization-type will affect performance and the methods used to access data in a file.

One-Record File Organization

A *one-record* file is stored sequentially, and is therefore subject to all the restrictions that apply to sequential files (see the next section). Unlike other sequential files, however, a single-record file communicates to an APPX input process that it can expect at most one record. Based upon this information, the user interface is streamlined. For example, APPX automatically checks for the existence of a record in a file. If there is no record, APPX places the user in add mode. If there is a record in the file, APPX places the user in change mode.

A parameter file is an example of a single-record file. Using a single-record file to hold specifications that are referenced by more than one process in an application avoids unnecessary duplication, and simplifies design and maintenance.

Sequential File Organization

Sequential files organize records according to the sequence in which they are added to a file, by adding each new record at the end. The system retrieves records based upon their relative position in the sequence by using unique record numbers. A sequential file does not contain keys so its records cannot be retrieved by a key path. To select a record based upon field value, APPX must process all records, ignoring those that do not satisfy selection requirements.

In most computer systems, APPX retrieves records from uncompressed, fixed-length record sequential files faster than from indexed files.

A sequential file can contain an unlimited number of records, subject only to available storage space or operating system limitations. You cannot, however, simply delete specific records from this type of file. To eliminate specific records, you must first transfer all records that are to be retained to another file and then scratch the original file.

Indexed File Organization

For an *indexed* file, the system retrieves records based on the content of specified key fields. APPX requires one primary key field for each file and optionally permits up to 16 alternate keys.

Key fields, when used by the system to retrieve records, are referred to as *key paths*. This method of record retrieval is inherently slower than the method employed for sequential files but is more flexible and allows you to perform more sophisticated processing operations.

An indexed file can contain an unlimited number of records, subject only to restrictions imposed by available storage space and operating system limitations. Unlike records in a sequential or a one-record file, you can directly change or delete records from indexed files, even if they are of variable length.

File Keys

For each indexed file, you specify a *primary key* and up to 16 *alternate keys*. A primary key is required for each file, while alternate keys are optional.

A primary key provides the default key path by which to access and retrieve specific records in an indexed file. Unless otherwise specified by a user or designer, the primary key path is always used. The field you designate as the primary key should generally be short and must be unique. A primary key is often used as a link to data in other files. Examples of primary keys include a customer number in a customer file, a product code in a product file, and a date/time stamp in a purchases history file.

Alternate keys provide optional key paths by which to access and retrieve specific records in an indexed file. A user or designer may optionally choose to change from the primary key path to any alternate key path for record access and retrieval. Examples of alternate keys to a customer file might include customer name or phone number.

APPX allows you to have duplicate alternate key values in a file, although a unique primary key is required. For each key path, you specify whether duplicate key values are allowed.

When you define a field to be the key to your file, there are two fields in the Key Control pop-up box in addition to *Key Type*:

Key/Field Order

Can be defined as ASCENDING or DESCENDING. The default is ASCENDING.

User Selectable?

Determines whether or not a user can switch to a key path with the Select Access Path option from within an input process. The default is "yes" .

A field does not have to be designated as a key to be used as a sort field in query processes. During sorting, APPX can use any fields to generate the required sort order.

Default Value Records

A *default value record*, created during compilation of the dictionary, includes the default values optionally established by the designer for each field in a file. APPX uses the default value record as a modifiable starting point on an input screen, for example, or to initialize a record before setting selected fields and writing the record to a file.

Default value records are maintained by APPX; they are not stored in the data file.

Data Dictionary Processing

Processing the data dictionary analyzes the data dictionary specifications and translates them into a format more directly usable by APPX during process execution. It does not generate machine or object code.

Dictionary processing performs the following actions:

- Creates an element record from domain and field specifications.
- Creates a file record from record specifications.
- Searches for logic errors (such as invalid key definitions or illegal record lengths).
- Constructs the default value record for each file.
- Computes field lengths.

- Establishes the starting position for each field within a record.
- Determines whether or not changes may affect the structure of existing files.

You must process the data dictionary for each new application before its domains, files, and fields can be referenced by processes. In addition, APPX will process the appropriate files or work fields any time you change the specifications. If the data dictionary requires processing, APPX prevents processes that reference it from executing.

When Processing is Required

The data dictionary's input processes track modifications that, when they occur, require the dictionary (or selected portions of it) to be processed. By tracking situations that require processing, APPX can eliminate errors by removing the possibility that other components within APPX will access obsolete data specifications.

The following paragraphs describe conditions for which processing is required:

Domains

Require processing after they are added to the dictionary and whenever their specifications have been modified. Processing of domains is automatic in APPX. If a domain requires processing, all fields that reference the domain also require it. If a domain is deleted from the dictionary, any remaining fields that reference the domain must be modified and processed.

Files

Require processing after they are added to the dictionary and whenever their specifications are modified. Processing of files is automatic in APPX. In addition, a record/file must be processed whenever fields are added or deleted, or a field specification is modified. If a record/file is deleted from the dictionary, its associated fields are automatically deleted. The dictionary does not have to be processed.

Work fields

Must be processed whenever a work field is added or deleted, and whenever a specification for any work field is modified. Processing of work fields is automatic in APPX. If any work field requires processing, all work fields are processed.

Restructuring Files

As noted previously, you or your database administrator must run the restructure facility when a structural change is made to one or more files in an application. A structural change is one that affects the layout of a record. Examples of this include adding a field to a record, deleting a field from a record, or changing the length of an existing field. APPX does not allow users to run a process that references a file if it needs to be restructured.

Security in the Data Dictionary

Within the APPX data dictionary, you use *field* and *record* security specifications to control which users can view and modify data presented on images in applications.

To secure fields on input and output images, you assign a *read rights security code*. Read rights govern whether or not a user can see the contents of a field as defined on an image—either input (transaction entry or maintenance screens, for example) or output (journals, lists, or reports). If a user does not have read rights to a secured field, the contents of the field are invisible and non-editable. Refer to the discussion on Database Security in *Chapter 2-2* for more information.

Field security specifications, unlike record security specifications, do not restrict access to fields when developing or modifying an application within application design. For example, a designer can define statements to modify values in secured fields, or to load the contents of secured fields into unsecured fields. The contents of a secured record, however, cannot be accessed or modified with statements.

You define read and modify security specifications for each of the following components of the dictionary:

Domain
Security specifications cannot be overridden for any domain-type fields and work fields that reference them.

File
Security specifications control read and write access to files.

Field
Security specifications control read and write access to fields. If security specifications are blank, access to the field is unrestricted.

Work field
Security specifications control read and write access to work fields. If security specifications are blank, access to the work field is unrestricted.

In addition to the file/field-level security described above, you can establish record-level security as described in the next section.

Record Protection Fields

Record protection fields instruct APPX to verify whether or not a user is allowed to delete a record or change a primary key value in a record. Use of these fields allows you to maintain *referential integrity* within an application.

Delete Protection

Fields govern whether or not a user can delete a record while executing an input process. These fields can be used to prevent the creation of “orphan” records, by prohibiting the deletion of any record that is referenced by other records in the application.

Key Protection

Fields govern whether or not a user can modify the *primary key* to a record while executing an input process. These fields can be used to prohibit modification of primary keys that are referenced by other records, thus also preventing “orphan” records.

Record Access

Fields contain designer-designated security codes that automatically restrict the ability to retrieve a record (and by implication any field on the record) to users with security profiles that include this code. If a user does not have access to the record, APPX responds as if the record is not on file. The designer must maintain the values in this type of field.

CHAPTER 1-4

Understanding Process Design

APPX Processes

The data dictionary, described in the previous chapter, provides the central location for file and data information for an APPX application. The other key components of an application are the processes you develop. A *process* consists of a set of related specifications and ILF routines to perform one or more desired operations. Processes include references to data dictionary specifications. There are nine types of APPX processes: ***Menu***, ***Job***, ***Input***, ***Output***, ***Update***, ***Query***, ***Inquiry***, ***Status***, and ***Subroutine***.

Processes Compared With Programs

A process is very similar to a program in traditional data processing systems, in that:

- You execute a process to perform one or more related functions or activities.
- A process is the smallest unit of an application that can be executed separately.
- A process consists of a series of detailed instructions performed in a specified sequence. These instructions consist of process specifications, their associated painted images, data dictionary references, and ILF routines.
- APPX compiles the process source specifications into instructions more suitable for machine execution.

There are, however, some significant differences between a process and a traditional program, including the following:

- A program consists entirely of line-by-line procedural coding, while a process is comprised of an integrated set of specifications.
- You must compile a program as a separate step before executing it but a process is compiled automatically at runtime, if required.
- A program must include a detailed definition of the files and fields that are referenced, whereas all data referenced by a process (including local temporary data) is defined in the data dictionary.
- In a program, you must explicitly control when each instruction is executed, while an APPX process automatically executes based on the specifications.
- You use a standard editor to develop and modify a program, while in APPX you use the specification screens to guide development.
- Each program represents a separate file, but all the processes for an entire application are stored in a set of APPX design files.

You access the process definition facility from the Application Design menu. The first screen you see allows you to specify the type and name of the process you are defining or editing. After you identify the process, you see displayed the remainder of the specifications that are applicable to the type of process (menu, input, etc) you selected. From this point on, APPX moves you automatically through the family of screens that comprise the process definition facility. On the facing page is a schematic diagram of the process definition design components.

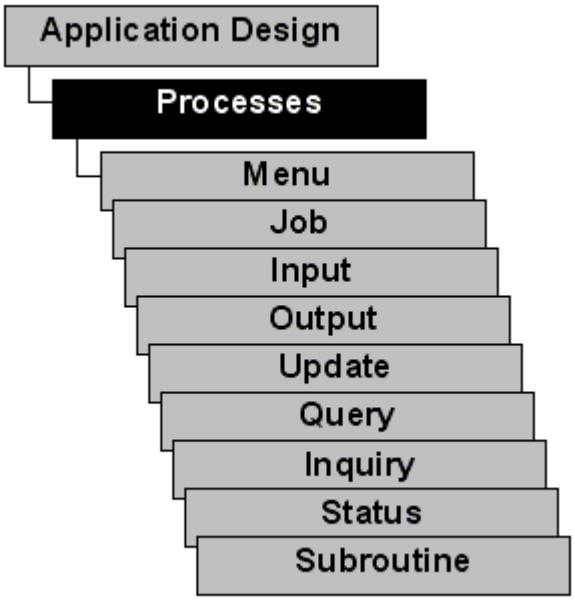


Figure 1-4-1. The Process Family

Process Concepts

An application consists of one or more related processes that together support all the required business-related functions. As noted earlier, there are several different types of processes. The next sections of this chapter discuss some of the features of each type of process. Chapters 3-1 thru 3-16 give a more extensive overview of the process facility and provide definitive descriptions of the specifications needed for individual process types.

The Menu Process

You use a menu process to permit users to select and run other processes. In a typical application, you begin process definition by designing and specifying the main menu that is presented when a user first enters the application. You then proceed to identify what happens when each of the options on the menu is selected. APPX allows you to define up to 249 options (choices) on each menu. An option may invoke another menu, and the number of “nested” menu levels that you can specify is unlimited.

When you define a menu (as with each type of process), you first provide some basic information about it, including a unique name. Process names must only be unique within a process-type. For example, you can have both a menu and input process called ORDER ENTRY.

Next, you paint a menu *image* using the Image Editor and then identify the process associated with each option on the menu. Because APPX supports panning (the ability to move about on a surface that is larger than the terminal's physical screen size), a menu can be any dimension up to 449 rows and 250 columns.

Refer to Chapter 3-4: Menu Processes for detailed specifications on defining menu processes.

The Job Process

You use a job process to serve as a "manager" for a group of related processes that are to be performed in a specified sequence. For example, you may want to allow users to select an option from a menu that produces three reports, each with an associated query. To accomplish this automatic sequence of steps, you can specify a job process that, in turn, controls the execution of all query and output processes.

Refer to Chapter 3-5: Job Processes for detailed specifications on defining these processes.

The Input Process

An input process allows users to enter, modify, delete, and inquire about data in an application. Included in the basic information you provide for an input process is the *Process Control File* (called the PCF) associated with the process. For example, the PCF for a customer file maintenance input process would be the CUSTOMER file. Each input process can have at most one PCF.

Each input process consists of one or more *frames* (refer to the description of process components later in this chapter). For example, if a file has more input items than will fit on a single screen, you can define two frames that operate one after the other so the users can enter all of the required information.

Each frame may have one or more *alternate images* (also described in more detail later), which are the screen representations that appear to the users. If more than one image has been defined, one of them is selected to display based on previous user input.

A complete input process provides many different *event points* (described later) at which you can incorporate an ILF routine (set of statements) to supplement the basic process specifications. For example, you may want to calculate an extended price based upon the unit price and the quantity entered by a user.

As with a menu process, you paint the images that are to appear in an input process. Within the context of the *Image Editor*, you specify where each input item (user-entered field) is to appear. APPX automatically references the data dictionary to validate each item you paint, and displays the field as a

series of “pseudo blanks” based on its length defined in the dictionary. The image editor also allows you to define any of several characteristic attributes, such as bright, inverse, and underline, for both items and background text.

Although the data dictionary already contains information about how a field should be formatted on an image (its default display mask), you can override this on each image. For example, you may want to display a date field, normally displayed as “12/31” (MM/DD), as “December 31”.

There are many other optional or automatic features provided in an input process including:

Data lookup
To retrieve and optionally display data from another related file (such as, in an order entry process, using a CUSTOMER ID to look up and display a name from the CUSTOMER file).

Automatic retrieval
Of the next record on the file each time a user completes an input action.

Partial key
Option, by which you designate whether a user can enter a partial key or needs to enter a full key to identify a record on file.

Automatic update and/or deletion
Of records in related files. If so specified, APPX effectively allows a user to change key values in one or several associated files.

Automatic field validation
Against the data dictionary when data is entered by a user.

- *Scrolling of records.* Each frame and a selected alternate image may appear once or, you can specify that it scrolls. APPX supports both vertical and horizontal scrolling.

Refer to Chapter 3-6: Input Processes for detailed specifications on defining these processes.

The Output Process

You use an output process to define a report that the application is to produce. Included in the initial information you provide for an output process is the report name, the PCF to be used, an “In Progress” message, and whether you want to use standard page and column headings.

You then proceed to specify each frame for the output process. Output processes may consist of the following special frame types:

Report class
Frames (*start* and *end*) that operate only at the beginning or end of a report.

Page class

Frames (*start* and *end*) that operate only at the top or bottom of each page.

Range class

Frames (*start* and *end*) that operate when a sort control break occurs (if specified in the related query).

Record class

Frames that operate on each individual record included in the PCF (and as constrained by a query, if so specified).

For each frame, you define the related images. For example, in a start-of-report frame, you paint the image of text and/or items you want to appear at the beginning of a report. You use the same image editor facility as described earlier under input and menu processes. For the record class frame, you specify all the data fields that you want to appear on the "detail" lines of the report. For an end-of-range frame, you can specify subtotal or grand total information and text. You specify as many frames and images as required to produce the report.

As with other process types, there are a number of event points available to supplement the basic specifications with ILF statements.

APPX lets you specify reports very quickly by using the standard defaults provided. You can also define very complex report formats by using the optional features available. Some of the report specification features include:

- The ability to use standard page and/or column headings; or, you can design customized headings for any output process.
- Vertical and horizontal scrolling capabilities.
- Automatic totaling.
- Sort and control breaks up to a maximum of 255 levels.
- Comprehensive page-change, line, and spacing control.
- Support for printing forms, such as invoices and checks.
- Sophisticated, relative image positioning; that is, the ability to print the components of a report below or beside each other.
- The ability to browse a report on the screen.
- The ability to use default display masks for items as defined in the data dictionary, or to override them for a specific report.
- The ability for an end user to specify several print disposition values, such as the printer ID, type of form, and number of copies.
- The ability to automatically build Range-Start and Range-End frames based on runtime query specifications.

Refer to Chapter 3-7: Output Processes for detailed specifications on defining these processes.

The Update Process

You use an update process to perform file update functions. For example, you may define an update process to update a master file with transactions that have accumulated since the last update occurred. An update process still has only a single PCF (in this example, the transaction file), although it can also update records in the master file using statements in an ILF routine. An update process can produce a log file of all informational and error messages generated during the course of processing.

Refer to Chapter 3-8: Update Processes for detailed specifications on defining these processes.

The Query Process

A query process allows you to:

Select
A subset of data from a file by providing (or having a user provide) selection criteria.

Sort
The selected data by specifying (or having the user specify) one or more sort fields.

You frequently use a query process in conjunction with an output process (described later), to produce a predefined report format for the data as selected and sorted in the query. You can also use a query before an update process to select subsets of a file to be processed in a prescribed sequence.

Included in the general information you specify about a query is the name of the PCF (process control file) and the key you want to use to access the file (if other than the primary key).

In specifying the selection criteria for a query, you can identify as many field constraints as you wish for both key and non-key fields. Each constraint allows you to specify a relation (such as *equal to* or *greater than*) between fields, or between a field and a constant. There is also an *includes* relation that you can use to constrain records by the presence of words or word strings in text. Note that these selections are case-sensitive.

The Record Selection screen also provides an area that allows you to combine several individual constraints into logical expressions, including parentheses, ANDs, and ORs. As a designer, you can predefine some or all of the selection constraints, or you can allow a user to specify the constraints dynamically at runtime.

As with the selection criteria, you can choose to sort a file by any number of key or non-key fields. You can predefine any or all of the sort criteria, and allow end users to modify sort requirements dynamically. Each sort field is potentially a “control break/subtotal” level in an output, and you or a user can specify print-related information about each sort field, such as whether to print subheadings or go to the top of a new page.

Refer to Chapter 3-9: Query Processes for detailed specifications on defining these processes.

The Inquiry Process

You use an inquiry process for quick access to a specific record or set of records. Inquiry has all the options that output does, in addition to the ability to ask for the record key.

Refer to Chapter 3-10: Inquiry Processes for detailed specifications on defining these processes.

The Status Process

You use a status process to do special error-checking procedures. For example, you may want to ensure that certain reports have been printed before closing an accounting month.

Refer to Chapter 3-11: Status Processes for detailed specifications on defining these processes.

The Subroutine Process

You may want to include several standard functions in an application that you can accommodate by defining and invoking a single ILF routine. If you define each of these standard functions using a subroutine process, you can invoke the same process at appropriate points throughout an application. By invoking the same set of statements during an event point within another process, you can save development and maintenance time, and ensure that the common function operates consistently each time it is used.

Refer to Chapter 3-12: Subroutine Processes for detailed specifications on defining these processes.

Relationships Between Processes

You have now learned about the nine different types of processes in APPX and the functions you can perform with each type. An essential part of application design is to specify the relationship between the processes you define. For example, when a user selects an option from a menu, an input screen may appear. Or another option when selected, may cause the generation of a report preceded by a query. In both of these examples, the functions operate together properly as a result of the specifications you provide. With the exception of a start-up menu that displays automatically when a user first enters APPX, no APPX process is invoked until you specify its relationship with at least one other process.

Process Genealogy

A simple genealogical metaphor describes the relationships between and among processes. For example, a *parent* is a process that invokes one or more *child* processes. In APPX, a child is associated with only a single parent (that is, the process that invoked it) at each invocation occurrence. But the

same process can be invoked by more than one process in the same application and, thus, may have several parents. Two processes with the same parent are *sibling* processes. *Ancestor* processes include the parent process and any *grandparent* processes. *Descendent* processes include any child and *grandchild* processes (and so forth). All these relationships can be clearly depicted with diagrams that resemble a family tree. The term *process family* is applied to a parent and all its descendents.

Any or all of these family-related terms can be applied to any APPX process (menu, input, output, etc.), depending on its relationship in the context of an application. For example, a menu process can be the parent of several children (all the processes that correspond to the options on the menu), one of which is a job process. This job process can be the parent of a query and output process. The same menu described previously as a parent can also be the child of a previous process such as another menu.

Process Components

Each process operates as one or a series of related parts, called *frames*. Each frame, in turn, can include one or more *automatic children,* as well as one or more *images* that comprise all or part of a terminal screen or report. Each image can have *optional children*. Both automatic and optional children can have selection criteria associated with them. An image is comprised of *background text* and *items*, which are data fields on a screen or report that are displayed and/or filled in by a user. Processes also have *event points* at which ILF routines are defined and invoked. This section discusses the relationship between the various process components.

Processes

You have learned about each of the nine different process types. Although each type is distinct and has its own unique features, there are many similarities among all process types. The process itself is the first component you define; you specify several pieces of information, including a name, description, security group, and process control file.

Frames

A *frame* is the next component level of a process. Every type of APPX process (except subroutine, menu and status processes) includes one or more frames that are invoked in the sequence you specify. A frame can serve two primary purposes:

Controlling

The sequence of activities that occur within a process.

Grouping

A set of alternate images.

To demonstrate the sequencing function, consider a customer input process in which there are so many fields for a user to enter that they cannot all fit on a single screen. You therefore design a two-screen input process that will display to the user one screen at a time in a designated sequence. To accomplish this, you define two frames, numbered so that they execute one after the other (frames operate in ascending sequence), for each customer record.

The other primary function of a frame is to control the presentation of alternate images. In this sense, think of an analogy to a picture frame that you could use to mount any of several photos. Another version of a customer input function demonstrates this type of usage. In this version, there are two major types of customers, domestic and international. The information you need to capture for each type of customer has many similarities but some fields apply uniquely to only a domestic or international customer. To satisfy this requirement, you define two frames. On the image for the first frame, the user enters the information that applies to both customer types and also indicates the customer type. In the second frame, you define two alternate images as part of a single frame. Then, depending on the customer type entered in the first frame, the second frame selects which of the two alternate images is to appear on the screen.

In most cases, a frame executes exactly once before a subsequent frame is invoked. You may, however, specify that a frame is optional or repeating and, through statements, control whether a frame executes at all, or more than one time. As an example of an optional frame, you may have defined an input process for entering customer information. In the first frame, a user enters the customer name and several informational items, including whether the customer is domestic or international. For international customers only, a second frame allows the user to supply additional information. The second frame does not operate for domestic customers, and no image appears in this case.

When you are defining a process, APPX automatically displays the frame specification screen after you enter the process-level information.

Automatic Children

An *automatic child* is a process that is invoked directly by another process without a user's intervention or choice. For example, a job process may automatically invoke, in sequence, a query and an output process. One of the steps during application design is to specify if a process is to invoke any automatic children. Only input, output, inquiry, job, or update processes can have automatic children associated with them.

Selection Criteria

Another factor in process relationships is whether there are constraints on the data selection for the process control file of a child. For example, if an input process for orders invokes a line item input process as an automatic child, you may constrain the line item file to select only records with a matching order number. You can specify *selection criteria* for either optional or automatic children, and you can define constraints for both key fields and non-key fields. Constraints must be specified as equalities, such as the ORDER NUMBER of the records in the child process must be equal to the ORDER NUMBER of the record selected in the parent process.

Images

An *image* is the visual representation of *painted* text and items that comprise a display on a terminal screen or report. An individual image may occupy an entire screen surface or page size, or only a portion of it. A complete report page, for example, may consist of several images, including page headings, column headings, detail lines, subtotals, and a grand total.

You control the layout of each screen or report by specifying the size and placement of each image. For example, you may specify that a menu starts in the upper left-hand corner of a screen, and consumes all the rows and columns of a typical terminal screen. Or, you can create a "pull-down" menu by positioning a menu in the top center portion of the screen, making it stand out by drawing a box around it (the box is controlled by a YES/NO question for each image).

An image is uniquely associated with only one frame within a process. As noted previously, a single frame may have more than one image associated with it, called alternate images. You use an ILF routine at the frame level to control which image is to appear during process execution.

You paint screens and report pages using APPX's image editor facility. This facility allows you to paint *background text* (constant, non-editable areas on an image) and *items* (data from fields) according to your layout requirements. In addition, you can specify *characteristics* for text and items, by which you can highlight parts of the image with such attributes as underline, bright, and inverse video. You can also define graphical components and attributes on your images to give them a decidedly Windows look and functionality.

Background Text

Background text consists of descriptive text and identifying labels on an image. For example, column headings on reports and field labels on input screens are usually part of background text. You often design the background text as the framework within which to define individual items, described in the following section.

Items

An *item* is the appearance of an individual field on an image. For example, an input process image may include several items so a user can enter, review, and modify data in a file. You can specify whether an item in an input process is modifiable or not. You also specify items in output process images, but they cannot be modifiable.

When you define an item on an image, it automatically assumes the attributes of its data dictionary specifications (such as field length and display mask) unless you override them for this image. You also can specify the item's display characteristics, as described in the following paragraphs.

Characteristics

A *characteristic* is an attribute that affects how items and text appear on the terminal screen or in a report. Examples include underline, inverse/normal, blink, and bright/dim. Some characteristics are applicable only to a terminal image and cannot be applied to a printed report.

When you define an item on an image, you can specify one or more characteristic attributes to apply to the item. For example, you could display a single item as modifiable, bright, and inverse. A similar set of characteristics may be applied to any portion of background text on an image.

Graphical Characteristics

Using the GUI features of APPX's Image Editor, you can add buttons, icons, pictures, wallpaper, title bars, and other graphical components to your menu, input, output, and inquiry process images.

Optional Children

An *optional child* is a process that a user invokes by selecting an option on a menu (or from an input process, discussed later). On a menu, for example, Option 1 may invoke an input process, Option 2, an output process, and, Option 3, a job process. One of the steps during application design is to specify all the optional children for each menu. These processes are called *optional* children because they are never invoked unless a user selects the corresponding option on a menu.

Although a menu is the primary place from which to invoke optional children in APPX, you can also invoke optional children from an input process. For example, you may allow the user to select from among three options, descriptions of which are on the input image.

As with automatic children, you can specify selection criteria that constrain the records to be processed in the optional child's process control file.

Event Points

An *event point* is one of a number of predefined chronological points during the execution of a process at which an ILF routine can be invoked. Certain process components, including processes, frames, images, and children have event points associated with them. For example, there is a *Pre-Display* event point for an image. If you develop an ILF routine for this event point, it will be invoked just before the image is displayed on the screen (or written to a report). Event points have names that describe when the associated ILF routine executes.

Preparing for Runtime

When you choose to run a process after initially defining or modifying it, APPX needs to prepare it for operation. During this preparation step, called *compilation*, the system translates the source specifications into detailed instructions that optimize runtime performance. This translated format of a process is called an *executable module*, abbreviated as *EM*. Because each process has its own EM,

usually only a small portion of an application needs recompilation at runtime. APPX is designed so that you can immediately invoke a new or modified process without having to execute a separate step to compile it. Whenever you choose to run a process, APPX automatically determines whether compilation is required by checking all referenced components in the application, including the data dictionary, to see if any changes have been made since the last compilation. If changes are detected, the system recompiles the process and creates a new executable module; if not, the system uses the current executable module.

The system also processes the data dictionary specifications into an executable format. The dictionary operates differently from processes, and requires a separate processing step. If you make any changes to the data dictionary specifications, APPX will process them before executing any process that references modified dictionary components. A separate step is provided to allow you to process the dictionary and to coordinate other related design activities, such as file restructuring.

The Sequence of Operation

The components of a process operate in a defined sequence so that there is a consistent pattern of operation, no matter what type of process is defined. For example, if a process has several frames that include optional and automatic children, and many of the frames also contain images, you need to understand APPX's processing sequence to achieve your intended design objective.

The primary control mechanism within a process is a frame. If a process has more than one frame, they operate in sequence based on a number you assign. APPX executes one complete *process cycle* (all frames one after the other) for each selected record in a process control file. For input processes, users select the records. For update and output processes, records may be selected by a previous query process; otherwise, the entire file is used. Thus, for an update or output process, if there are 100 selected records from the PCF, the entire frame cycle will execute 100 times.

A frame may include an image (or several alternate images), and invoke one or more automatic children, as well. APPX executes all the components of one frame before proceeding to the next.

Child processing is controlled in two fundamentally different ways. An optional child is always invoked by a user selecting an option key, and is always invoked from an image. An automatic child is invoked at the appropriate time without user intervention. For example, as soon as a user finishes entering data for an order in an order entry input process, a line item input process may automatically be invoked when the user selects ENTER .

Because of the basic difference between optional and automatic children as described above, there is not a conflict with the order of invocation. Because a child is also a process, it can include a number of its own components. The same rules regarding order of invocation as described previously also apply for a process that is itself a child. You can therefore design a process that has various levels for which the sequence of operation is always predictable.

An Example of the Process Cycle

The following example demonstrates the sequence of operation for a sample *process cycle.* In this example, a process has one frame that has ILF routines at the *Start of Frame* and *End of Frame* event points. The frame has one image, which has ILF routines defined for the *Pre-Display* and *Verify* event points. The frame also includes one automatic child, which has ILF routines defined at the *Pre-Invocation* and *Post Invocation* event points. For each selected record in the process control file, the sequence of operation is as follows:

- Read the process control file record.
- Invoke the *Start of Frame* event point.
- Perform image processing:

Invoke the *Pre-Display* event point.

Display the image on the screen.

User completes data entry.

Invoke the *Verify* event point.

Invoke the *End of Frame* event point.

- Perform automatic child processing:
- Invoke the *Pre-Invocation* event point.
- Perform processing as specified for the child process.
- Invoke the *Post Invocation* event point.

CHAPTER 1-5

Interprocess Communication

Interprocess Communication: overview

In APPX, you develop applications by defining and connecting processes. When a user runs an application, information may need to be shared among the processes that form the application. *Interprocess Communication*, or *IPC*, refers to the set of specifications and statements by which you control how data is communicated among processes.

Consider an application in which the user is asked to enter the CURRENT ACCOUNTING MONTH on the first screen that is presented. You want this information to be available to any process in the application. You can use the IPC facilities within APPX to communicate the CURRENT ACCOUNTING MONTH to all the processes within the application.

Now consider an order entry function that consists of an input process to enter sales order information, and a subsidiary input process to enter individual line items for each order. In this case, the order information entered in the first process must be communicated to the line item process. For example, line item pricing may depend on information about the customer entered for the order. Again, you can use the IPC facilities within APPX to communicate between these two processes.

Interprocess Communication is specified implicitly based on the following:

- Invocation type specification in the Automatic/Optional Child processes
- File Type of data dictionary record structures
- Share Class of Work Fields

Any process can be invoked as DETACHED or RELATED, or as a SUBPROCESS. When working with temporary files, it is important to know how the file type and the process invocation are related and interact. Child process invocations are defined at the following points in process specifications:

- Menu item specifications are optional child processes. The default invocation for menu items is *DETACHED*.
- Job/step item specifications are automatic child processes. The default invocation for job steps is *RELATED*.
- Input frames can invoke automatic child processes, and input images can invoke optional child processes. The default invocation for input child processes is *SUBPROCESS*.
- Output, update, and inquiry frames can invoke automatic child processes. The default invocation for these child processes is *SUBPROCESS*.
- ILF statements can invoke any process. The default invocation when entering ILF statements is *DETACHED*.

Any temporary file is available only to the user who creates it. When a user creates a DETACHED temporary file, it remains available until the user exits APPX.

When a RELATED temporary file is created, it remains available until a DETACHED process is invoked, or until the process stream returns to the parent that called it with a DETACHED invocation (for example, a menu). It is important to note that, although a DETACHED child process cannot access a RELATED temporary file, that file remains available to the parent that created it until the parent itself is ended. If the DETACHED child process attempts to access that file, a new, empty copy of the file will be created.

When a SUBPROCESS temporary file is created, it remains available until a DETACHED or RELATED process is invoked, or until the process stream returns to a parent that called it with a DETACHED or related invocation (for example, a job step or a menu). As with RELATED temporary files, it is important to note that, although DETACHED or related child processes cannot access a SUBPROCESS temporary file, that file remains available to the parent that created it until the parent itself is ended. If a DETACHED or related child process attempts to access that file, a new, empty copy of the file will be created.

Current record buffers, field contents, and file pointers are all available transparently through a SUBPROCESS invocation. They are not maintained for related or DETACHED process invocations.

Whether or not a job step is invoked as a “separate task” also affects the availability of a RELATED or DETACHED temporary file. If the designer allows a job step to be invoked as a separate task (i.e., submitted to background), APPX makes a copy of the RELATED or DETACHED temporary file and passes it to the child process being invoked as a separate task. The file is then available to the child process but, if the child process updates the file, those updates will never be available to the parent process.

APPX automatically scratches temporary files based on their interprocess communication characteristics. For example:

- DISK/SUBPROCESS and MEMORY/SUBPROCESS files are local temporary files. These temporary files are only accessible by a single process family, and exist only for the duration of that process family. Each time a process is run that references a SUBPROCESS temporary file, a separate copy is maintained and, when that process ends, APPX automatically scratches the temporary file.
- DISK/RELATED and MEMORY/RELATED files are normal temporary files. They are passed and shared to non-detached processes (to SUBPROCESS processes and related processes). APPX automatically scratches a RELATED temporary file when the non-detached process family ends.
- At the other extreme, DISK/DETACHED and MEMORY/DETACHED files are global temporary files. All processes that reference a global temporary file use the same temporary file. APPX automatically scratches a global temporary file only when the APPX session ends.
- The exception to these DETACHED/RELATED/SUBPROCESS (also called "Scoping", "Inheritance", or "Sharing") rules, occurs with QUERY processes. Please see the description under the predefined

field PROCESS CODE, in Chapter 4-7: Predefined Fields and Processes for a more detailed explanation of the special inheritance rules with respect to QUERY processes.

CHAPTER 1-6

Customizing Your Application

Runtime Application

APPX includes a component called the *runtime application*. This application provides a number of predefined fields and processes that are introduced in this chapter. You can make use of the features provided in the runtime application as you develop your own applications. (Refer to Chapter 4-7: Predefined Fields and Processes for a complete description of these features.)

Predefined Fields (PDFs)

In the runtime application, APPX includes a number of fields called *predefined fields* or *PDFs*. These fields contain various "points of reference," such as who the current user is and the company address. Although the names of these fields are pre-specified, you can use them as you would any other fields in your own applications. You can reference predefined fields in statements, for example, and set the values of some of them (such as ALTERNATE IMAGE NUMBER) to control processing flow. APPX keeps track of the values of predefined fields at all times, and maintains their values automatically. When you use a predefined field, you designate the application ID with three hyphens (---). APPX interprets the three hyphens as the runtime application.

Using Predefined Fields

You can make use of predefined fields in three ways:

- You can display and optionally allow a user to modify the value of a predefined field by painting it on an image. For example, you may want to display the value of COMPANY NAME on an input screen for the user's information.
- Using statements, you can check the value of any predefined field and, based on the result, control the flow of processing. For example, you may want to determine if there were any errors encountered during an output process and, if so, perform a special error-handling routine. You can examine the value of the ERRORS predefined field and, if it is greater than zero, pass control to the error-handling statements.
- You can use statements to set the value of certain predefined fields and control how a process operates. For example, you may want to control which alternate image is to appear on the screen by examining a user's response to a preliminary question. Depending on the response, you can set the value of ALTERNATE IMAGE NUMBER to the appropriate value and cause the proper image to be displayed.

Categories of Predefined Fields

There are five categories of predefined fields:

- ***Current status PDFs***, which provide information about current values, like the current user ID.
- ***Process control PDFs***, which control what will happen next in the application.
- ***Terminal management PDFs***, which provide information about such factors as current cursor position.
- ***Report management PDFs***, which provide information about the disposition of reports, such as the print destination.
- ***General purpose PDFs*** are used as index work variables or work fields in ILF statements.

The predefined fields are described in detail in Chapter 4-7: Predefined Fields and Processes and a table of PDFs is also provided.

Predefined Processes

APPX provides a number of standard processes that you can invoke, or are automatically invoked, at appropriate points in your applications. For example, APPX contains a standard DISPOSITION input process that allows users to specify certain print-related information that controls how, when, and where a report is to print. This DISPOSITION process resides as a predefined process in the runtime application. To invoke this process as a child in an application, you identify the application ID as "---", the process type as INPUT, and the process name as DISPOSITION.

You cannot modify any of the predefined processes, although you can, for example, develop your own version of a DISPOSITION process by defining an input process in your own application. Then, instead of executing the DISPOSITION input process from the "---" application, you invoke the DISPOSITION input process from your own application. It is recommended that you define all these types of processes in a separate application so that they can be referenced easily for system-wide use.

The following is a list and brief description of the predefined processes that are available in APPX:

- ***APPX MENU***, which allows a user to select a new database and provides access to system administration, application design, and database management.
- ***BEGIN LIVE OPERATIONS***, which transfers the run phase from initial setup or recovery processing to live operations.
- ***CHECK GUI CLIENT***, which determines if the current engine is connected to a GUI-enabled client.
- ***DISPOSITION,*** which allows a user to provide information that controls how and where a report is to print.
- ***ENTER BILLING CODE***, which allows a user to enter a value into the predefined field BILLING CODE.

- ***GET CMD LINE PARM***, which checks for the existence of a parameter on the command line used to invoke the current engine.
- ***GET ENV VAR***, which retrieves the contents of an environment variable.
- ***GET FILE FROM CLIENT***, which retrieves a file from the client system and stores it on the server.
- ***PROCESS QUERY RECORD***, which potentially adds a PCF record to the result set of a query.
- ***SELECT DATABASE***, which allows a user to select the appropriate database.
- ***SEND FILE TO CLIENT***, which retrieves a file from the server and stores it on the client system.

See Chapter 4-7: Predefined Fields and Processes for details on the predefined processes.

Accessing Non-APPX Programs

Although you can use APPX's own design facilities to develop all your new applications, you may have existing programs or applications written in some other language that you would like to run within the context of an APPX application. APPX allows you to invoke programs or subroutines written in any programming language. To invoke other programs, you use either the CALL or RUN statements within an ILF routine.

You may also want to invoke an APPX process from a non-APPX program or procedure. You can do this by invoking a specific APPX process using an operating system command.

The CALL Statement

The CALL statement allows you to invoke non-APPX subroutines when you want to pass and share information between APPX and a subroutine. For example, you could call a special calculator function written in assembly language that is available on your computer system. You pass the appropriate field values to the called routine, and receive back the result of the calculation (by passing the result field with *Share?* specified as 'Y' for "yes").

On some computer systems, unique identification of a subroutine name may require more than the 22 characters allowed in the CALL statement. To accommodate this situation, you can define an alphanumeric work field in the data dictionary and set its value to the complete program name. Then, when you specify the CALL statement, you enter the work field name as the program to be called. This approach also minimizes future maintenance by reducing the number of changes you need to make if a revision to the name is required.

There is an indicator that you set when you specify the CALL statement to indicate whether the called routine is to remain resident in memory after it completes execution, so that it can be re-executed by a subsequent CALL without having to read it from disk a second time.

The RUN Statement

The RUN statement allows you to invoke other programs with which APPX does not need to share information. For example, you could run a word processing program that is available on your computer system. Or, you could run a command editor available within your operating system. You can think of the RUN statement as an initiator of an entirely different program, to which APPX gives full operational control. The program that was run cannot stay resident in memory after it completes execution.

Command Process Invocation

A specific APPX process can be invoked from an operating system command. The database, application, process type, and process name are specified as part of the command. Refer to the *Installation Guide* for specific details on command syntax. Upon completion of the specified process, APPX ends.

This technique can also be used to execute processes in a background queue. If APPX is executed in the background, any terminal screen outputs will be processed as though the user pressed `END`.

CHAPTER 1-7

The Documentation Facility

An Integrated Facility

APPX provides an integrated documentation facility that simplifies incorporating documentation during development. As you specify an application's processes, files, and fields, each component has its own Documentation screen available. When you modify an existing component, you have an option to select the Documentation screen to review the previously entered documentation and modify it if required. Also, if you are set up as an application designer in system administration or if you have documentation design rights, user documentation can be updated from the Help screen by pressing change (*CHG*) mode.

The Standard Documentation Screen

APPX has a standard Documentation screen format that is used for all levels of components (processes, fields, etc.) throughout the system. The Documentation screen provides a text area for you to enter a description of this component of the application. A sample User Documentation screen is shown in *Figure 1-7-1*.

User Documentation screen

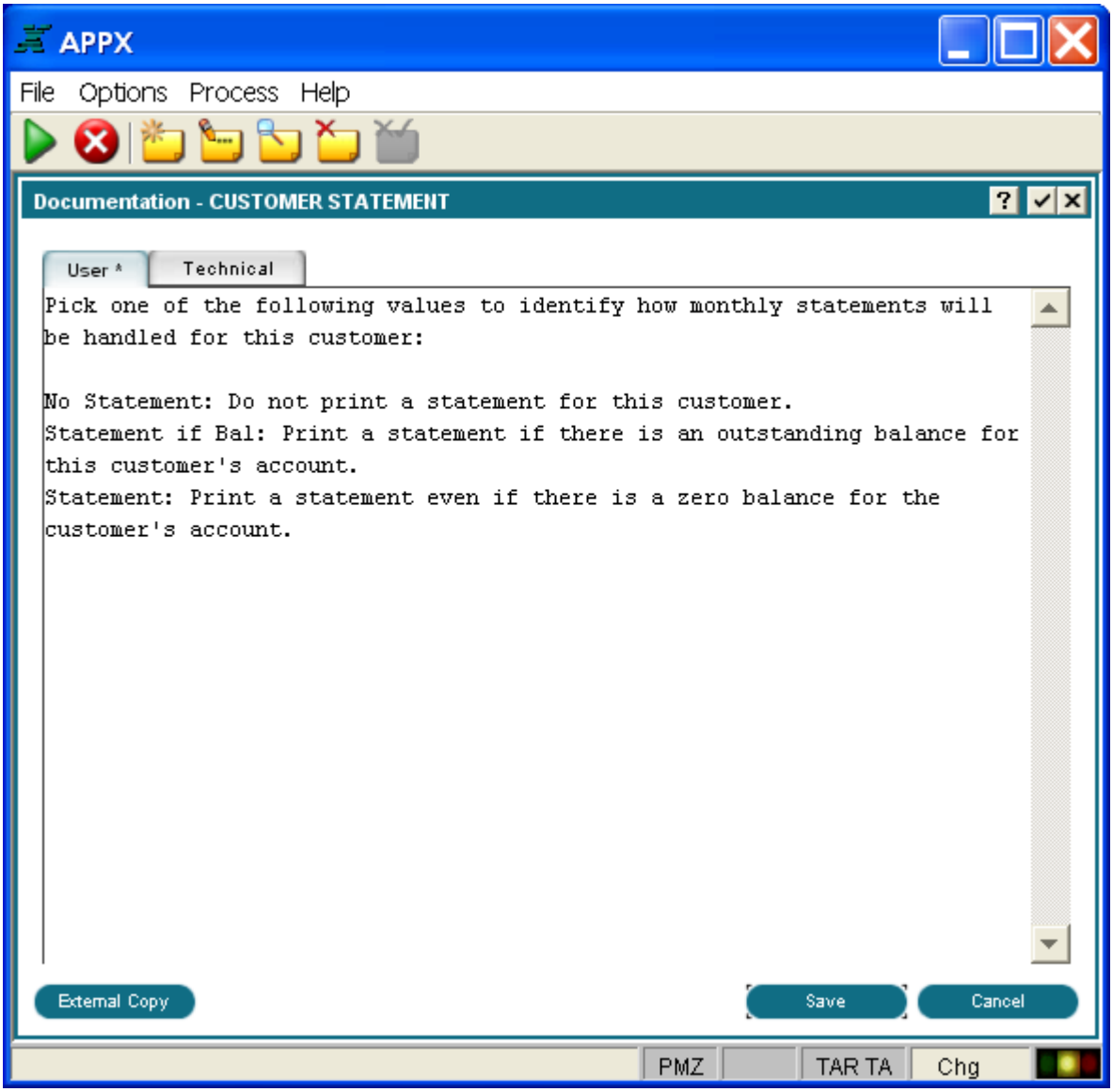


Figure 1-7-1

Technical and User Documentation

There are two major categories of documentation available within Application Design, known as technical and user documentation.

Technical documentation is of interest primarily to designers because it contains information about how an application is designed. It is the basis of the APPX printed technical documentation. See the Application Design Documentation Menu for details.

User documentation contains descriptive information about what an end user needs to know to operate an application, such as the allowable values or restrictions for an input field. It is the basis of the APPX online help facility. User documentation can be maintained both through Application Design and by using the DOCUMENTATION MANAGEMENT option from the System Administration Pulldown Menu on the APPX Main Menu (as described in *Chapter 4* of the *System Management Manual*.)

The Help Option

Users can at any time select the HELP button from the window or overlay title bar and display the documentation applicable to the item or option on which they then select via a mouse click. For example, if a user selected the HELP option then clicked on an item (field box), the documentation for the item, normally describing input rules and constraints, would appear on the screen. If the user selected the HELP option, they are requested to select which menu option they want to view help text for. By then selecting the desired option, the help text for the optional process that the option corresponds to is displayed.

Application Design Documentation Menu

When you select the Documentation tab from the primary Application Design screen, the Application Design Documentation Menu is displayed, as shown in *Figure 1-7-2* below.

Application Design Documentation menu

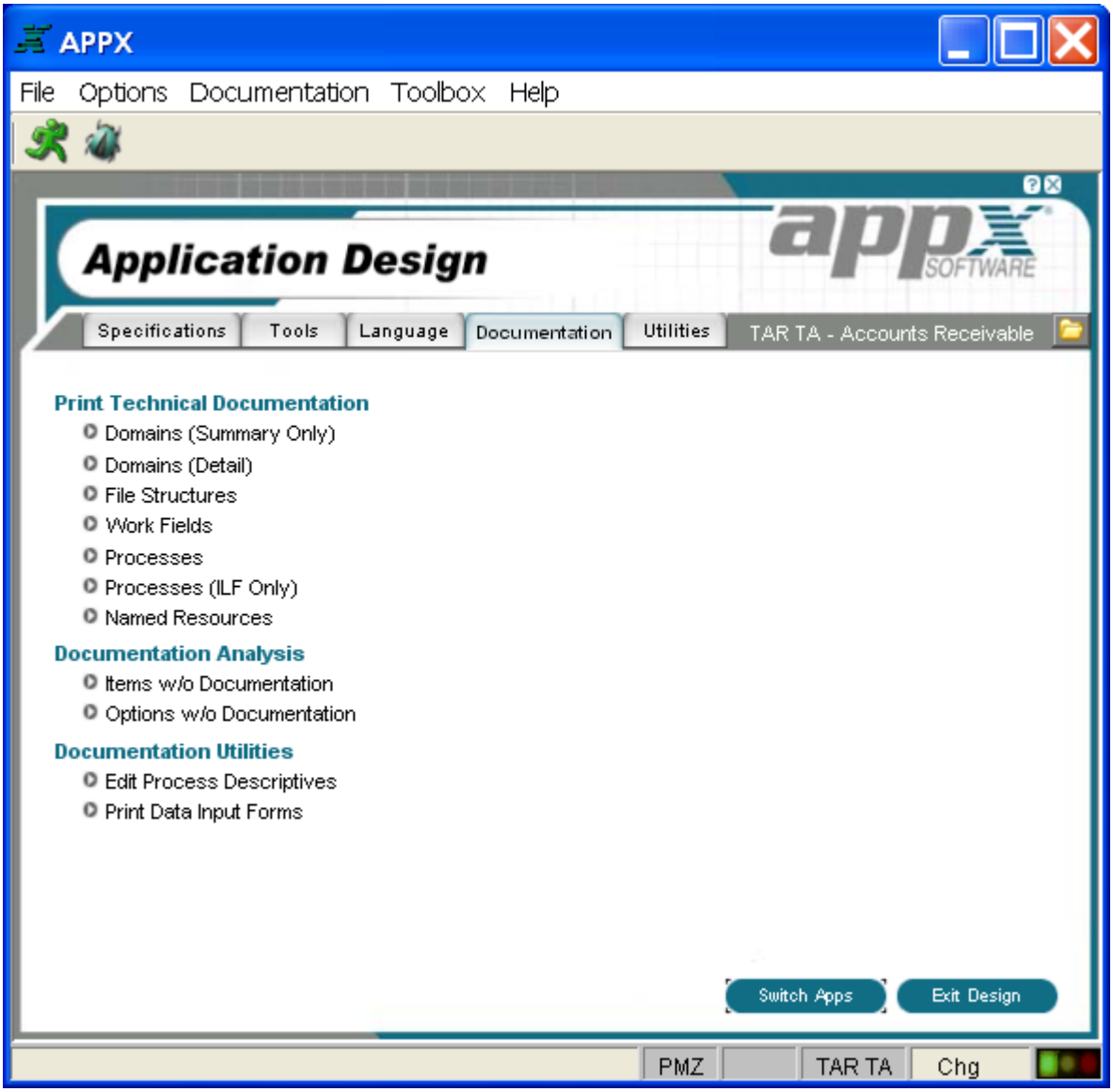


Figure 1-7-2

The Application Design Documentation menu options are grouped by function.

Print Technical Documentation allows printing of domains, records and fields, work fields, processes, and named resources. For each type, you will be shown a Scan screen that allows you to select those items that you wish to print. Items can also be selected based on the date they were last modified or, in the case or processes, by type (input, output, etc.). A standard disposition allows you to print either to the screen or on hard copy.

Documentation Analysis provides visibility of any items or options in your application without documentation.

Documentation Utilities includes a maintenance function to edit only process descriptions as well as a utility to print data input forms.

Options include:

- Items Without Documentation generates a list of design elements (domains, files, fields and work fields) without documentation.
- Options Without Documentation generates a list of options (optional processes invoked from menus) that have no documentation.
- Edit Processes Descriptive allows you to view and edit the descriptive names for every process in the application.
- Print Input Forms produces a report of interactive images found in input processes. The images are formatted as they would appear on a user's screen and can be "cut and pasted" into forms that can be used for data entry.

CHAPTER 1-8

Application Design Tools

Application Design Tools Menu

When you select the Documentation tab from the primary Application Design screen, the Application Design Tools Menu is displayed, as shown in *Figure 1-8-1* below.

Application Design Tools menu

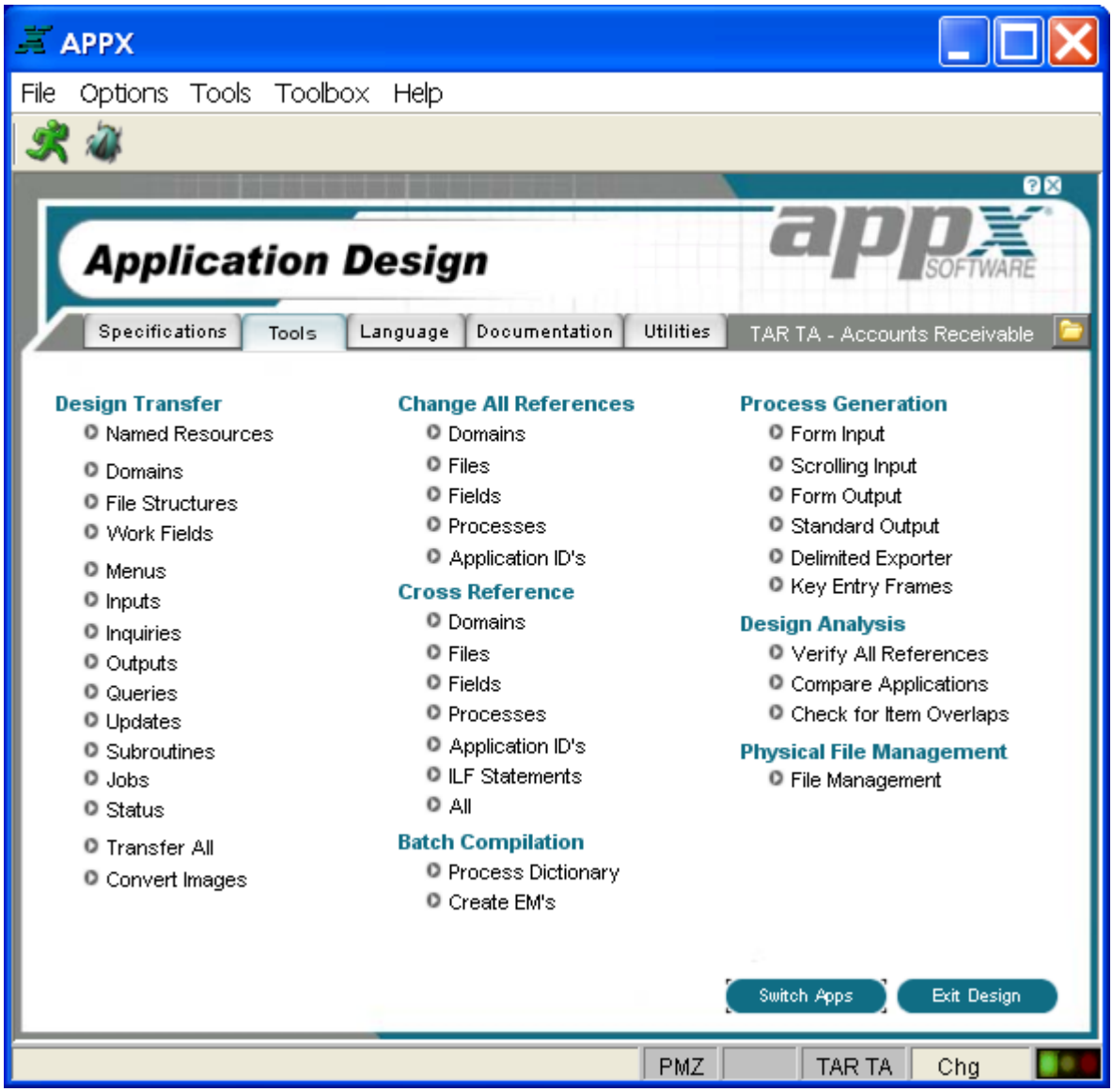


Figure 1-8-1

The Application Design Tools menu options are grouped by function.

Design Transfer options are described under the Design Transfer section.

Change All References options are described under the Change All References section.

Cross Reference options are described under the Cross Reference section.

Batch Compilation options are described under Other Design Tools.

Process Generation options are described under Other Design Tools.

Design Analysis options are described under Other Design Tools.

Physical File Management is described in depth in *Chapter 3, Database Management,* of the *System Management Manual.*

This set of tools provide a variety of special-purpose functions to facilitate the application development process.

Application Parameters

With the Application Parameters utility, the application designer can define or modify the parameters for an application. When you select APPLICATION PARAMETERS from the Utilities menu, the screen shown in *Figure 1-8-2* appears.

The Application Parameters screen contains the following fields:

- Application Name. Enter a descriptive name for this application.
- Release Number. Supply the release number for this application.
- Base Language. The language the application is presented to the user in. This encompasses all parts of the application made visible to the user such as menus, input processes, warnings, errors, etc. The Language ID file can be scanned to make a selection but this field is needed only if the Language Translation feature is in use at your site.
- Global Domains App. Enter the three-character ID of the application that contains your global — or universal — domains.

You use the *Global Domains* application to help default domain names when entering the data dictionary. If you add a domain-type field, and the field name contains the name of a domain within the application, that domain name is supplied as the default for that field's domain. If no match is found in the current application, and if the field contains a domain name that exists in the *Global Domains* application you identify here, APPX will use that domain name as the default.

- First Menu For —
- ***Initial Setup.*** Enter the name of the startup menu that should be presented to the user when the application's operational phase is *Initial Setup.*
- ***Live Operations.*** Enter the name of the startup menu that should be presented to the user when the application's operational phase is *Live Operations.*

- ***Recovery Processing.*** Enter the name of the startup menu that should be presented to the user when the application's operational phase is *Recovery Processing*.
- Direct Process 2 —
- ***Application ID.*** Enter the application ID that should be executed when the end user chooses *Direct Process 2* while in this application.
- ***Process Type.*** Enter the type of process that should be executed when the end user chooses *Direct Process 2* while in this application.
- · ***Process Name.*** Enter the name of the process that should be executed when the end user chooses *Direct Process 2* while in this application. If this is a menu, all of the optional processes defined on the menu will also be listed under the *Application* selection from the user's menu bar while in that application. If *Direct Process 2* is defined as a single non-menu process, then only that process will appear as a selection under *Application*.

The Applications Parameters screen contains these options:

- Documentation contains user and technical documentation regarding the application.
- Application Locks are the method by which APPX applications can restrict run-time and design access in the licensing of their applications. Applications locks may or may not be included for this application. The Author ID field and each of the Lock fields in this overlay refer to individual Product Registrations which must be present in your APPX System Registration before you will have access to the corresponding feature within this application.

Change All References

The Change All References utility performs a global search and replace operation throughout an application's design. For example, if you rename a field, you can change all references to the field from the old name to the new name. When you select CHANGE ALL REFERENCES from the Utilities menu, the menu shown in *Figure 1-8-3* appears.

Change All References menu

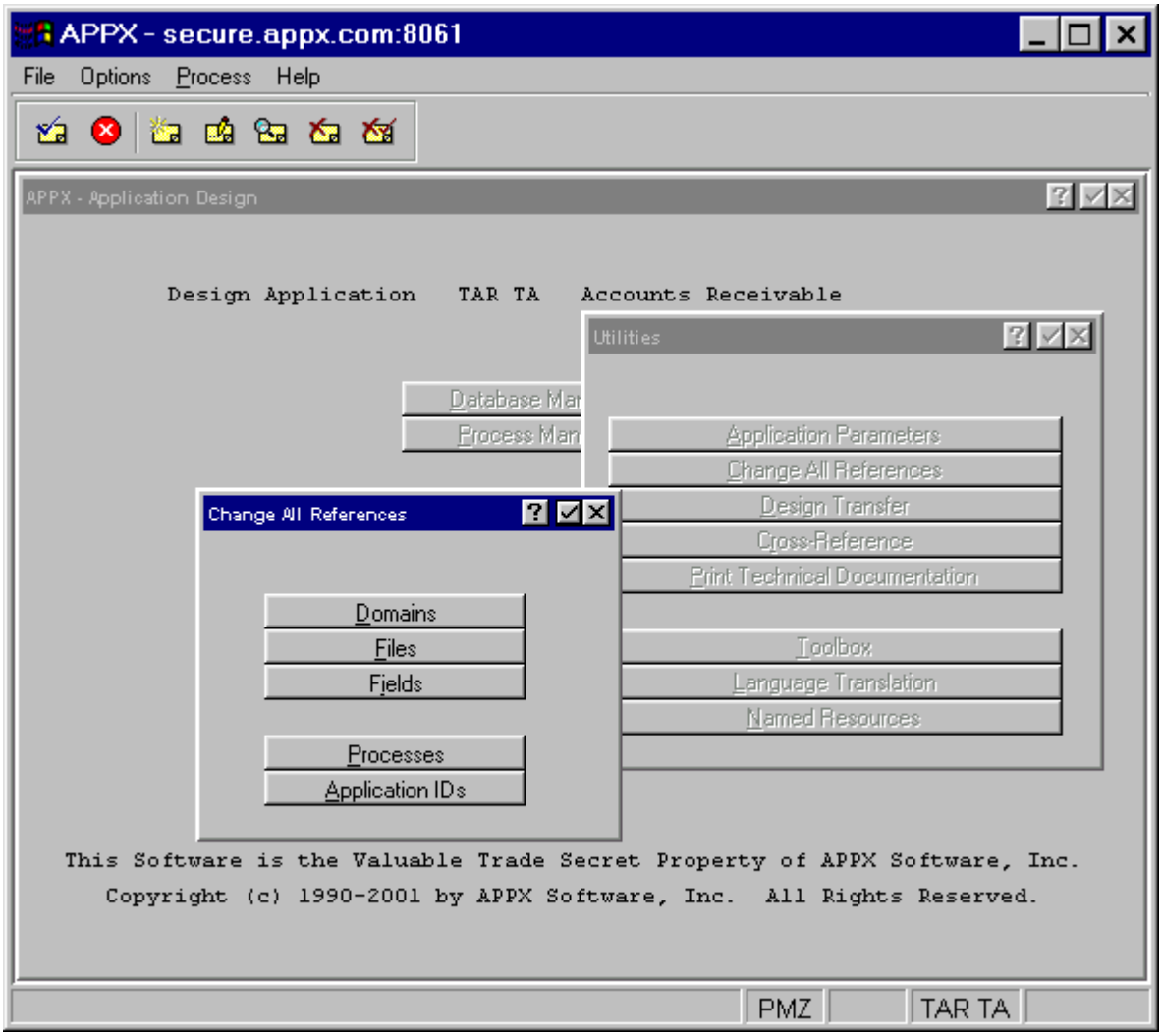


Figure 1-8-3

The Change All References menu allows you to choose what type of design component to change. When you select one of the menu items, a screen much like the one shown in *Figure 1-8-4* appears.

Change All References (Fields) screen

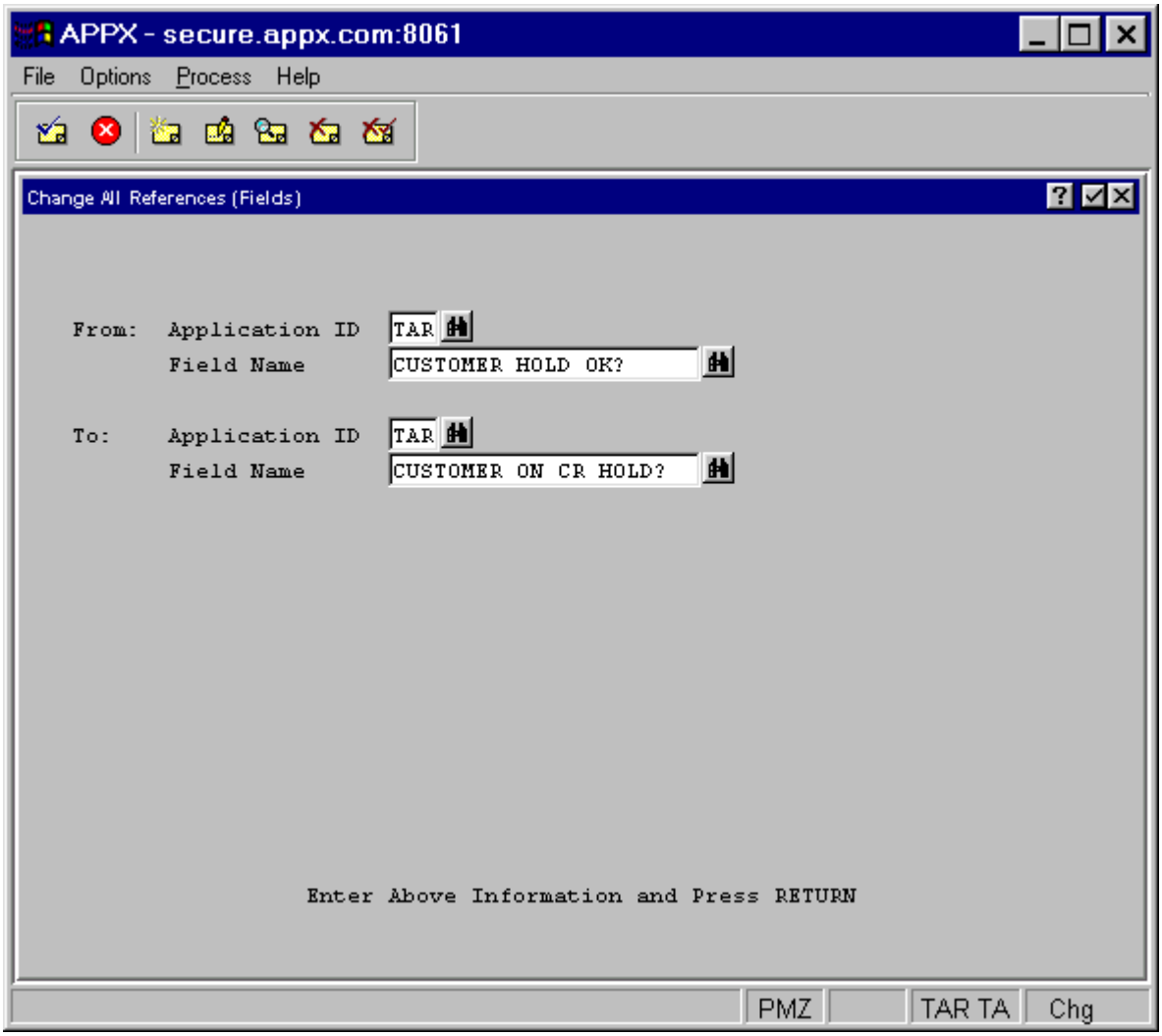


Figure 1-8-4

In this example, you would be allowed to change a field name. In other examples, you might change a process name, a domain name, or an application ID. In all cases, the application ID, along with any other appropriate information, will be requested. After entering the necessary values, press RETURN to make the changes.

Design Transfer

The Design Transfer utility permits all or a portion of an application's design to be transferred to or from another application or within the same application. For example, if you want to create a new report that is similar to an existing one, you can transfer the output process and assign the copied process definition a new name. After the transfer, you can then modify the copied process definition. When you select DESIGN TRANSFER from the Utilities menu, the menu shown in *Figure 1-8-5* appears.

Design Transfer menu

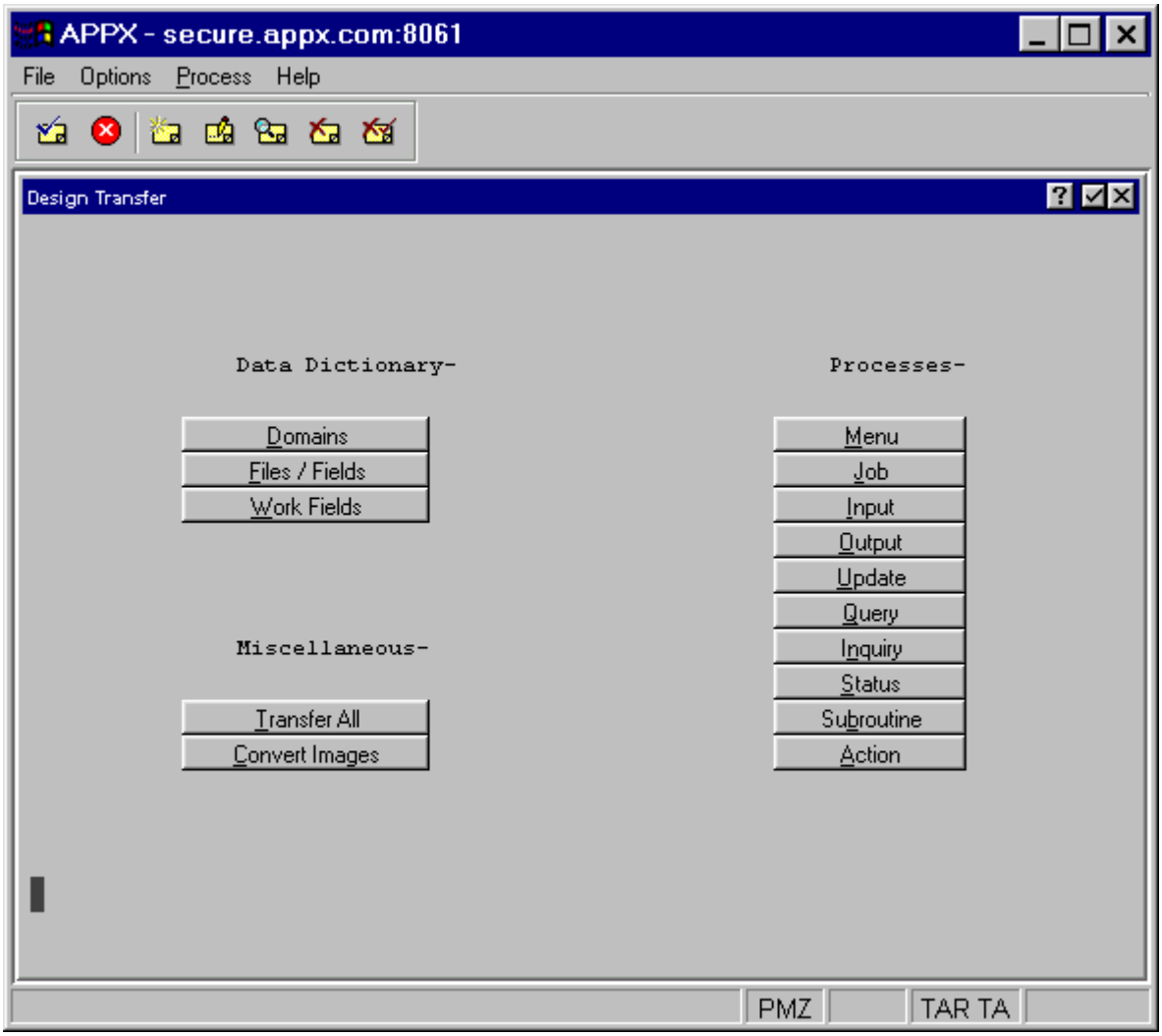


Figure 1-8-5

You can elect to transfer an entire application, or only parts of it, or you can transfer a process. When you select any option (other than TRANSFER ALL) from the Design Transfer menu, a screen like the one shown in *Figure 1-8-6* appears.

Design Transfer (Input Processes) screen

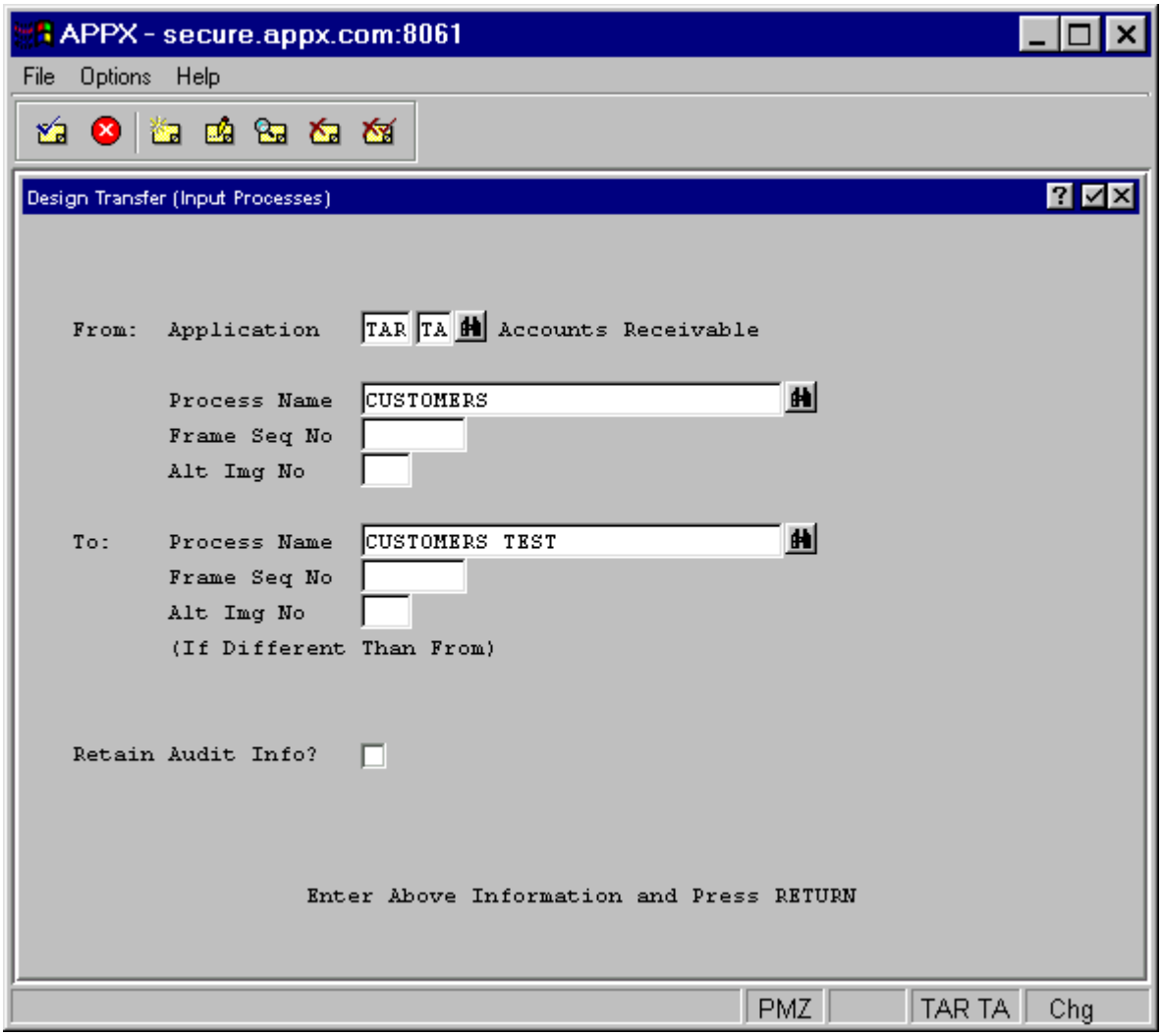


Figure 1-8-6

In this example, you would be transferring an input process. Notice that you have the option of transferring all or part of this process; if you enter only the process name, the entire process will be transferred. However, you can specify detail down to the image level. The same is true for design transfer for all types of processes. The amount of detail available for transfer depends on the type of transfer being made.

> **NOTE:** If you transfer a process from a version that is different from the one you are working on, and you do not transfer the process control file, APPX will issue an error when you attempt to edit the process. APPX does not "look" across versions for a file.

All transfer options produce a log of the activity. Any problems encountered during transfer are noted.

Cross Reference

The Cross Reference utility prints a “where-used” listing or interactive hyperlink display that locate references to application IDs, domains, records/files, fields and processes. This facilitates application maintenance by locating all references to selected components. When you select CROSS REFERENCE from the Utilities menu, the menu shown in *Figure 1-8-7* appears.

Cross Reference menu

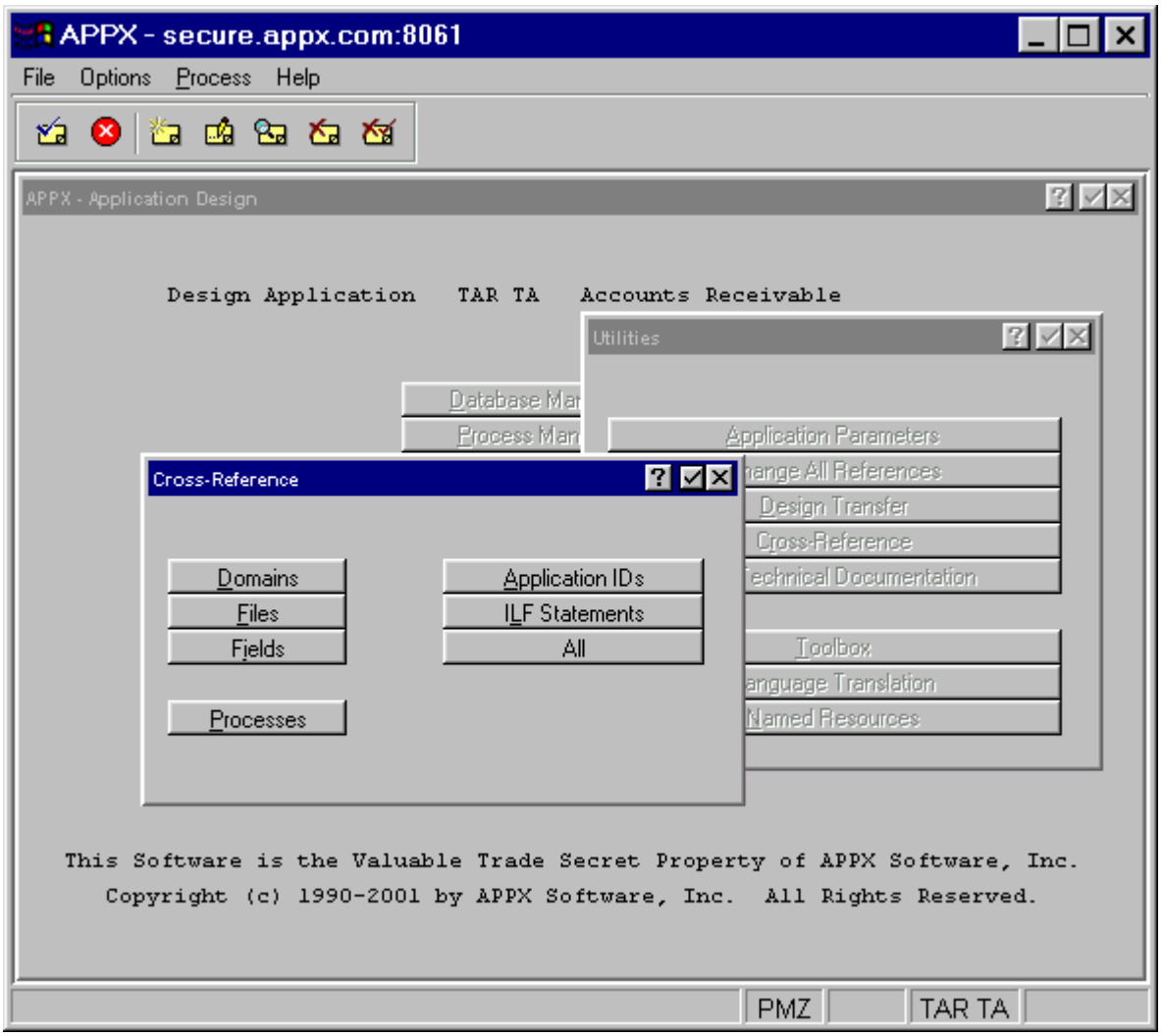


Figure 1-8-7

Notice that although you can choose a complete cross reference by selecting the ALL option, you can also choose to print a partial cross reference, based on the design component. When selecting a specific design component, you are presented with an input screen like the one shown in *Figure 1-8-8* on which you make your selections.

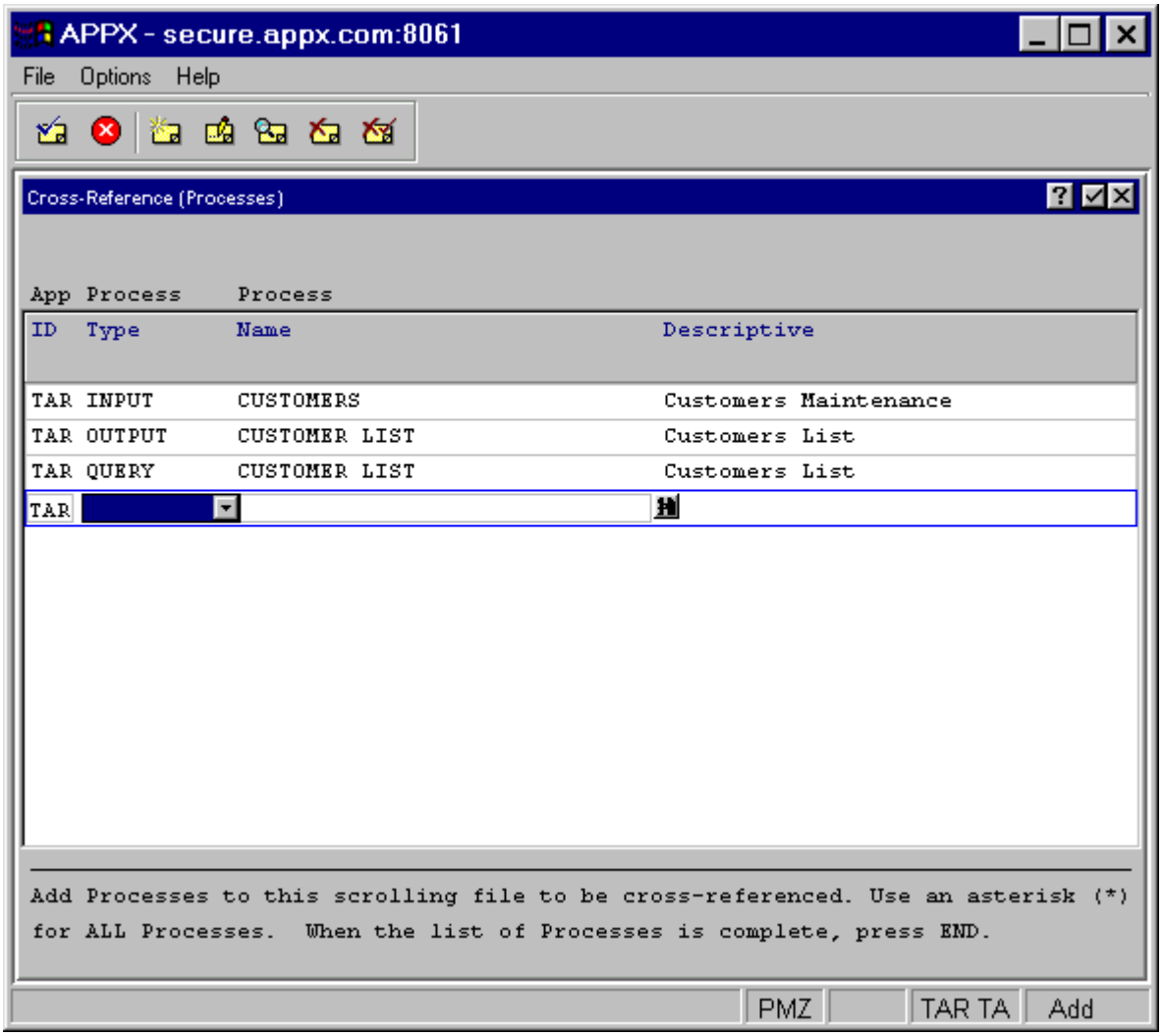


Figure 1-8-8. Cross Reference Item Selection Input Process

When you have finished selecting the elements you want to cross reference, select END from the title bar. The Cross Reference Options screen appears as described in the next section.

Cross Reference Options

When you have completed entry of elements you want cross referenced, the Cross Reference Options screen shown in *Figure 1-8-9* displays:

Cross Reference Options screen

Figure 1-8-9

Select whether you want to browse results interactively or generate a hard copy report. If you select the B*rowse Results Interactively* option, you are presented with an interactive list of elements that reference the selected elements. You can then go directly to any element you select via a hyper-text link and make changes by following the options shown on the bottom of your screen to "drill down" into the referenced elements and processes. Be advised that selecting this option prohibits the cross reference from being run in background. Click the checkbox to toggle between "yes" ☑ and "no" ☐.

If you select the *Produce Report?* option, a report listing all elements that reference the selected elements is generated. Note that a standard disposition screen will appear before the cross reference runs so that print and background options can be selected if you choose this option. Click checkbox to toggle between "yes" ☑ and "no" ☐.

Select PROCEED to run the cross reference or CANCEL to cancel it.

These additional options are available to you before running the cross reference:

- Define Applications To Search lets you select additional applications to perform the cross reference in. If no additional applications are selected, only the current application is searched. On this screen you also have the option of selected all applications within the current application version.
- Other Options currently only contains the Include Commented ILF? option which also searches all ILF code (rules) that has been designed as comments by the application designer. Click the checkbox to toggle between "yes" and "no" .

Other Design Tools

Other design tools on the Application Design Tools Menu include the following:

Batch Compilation

- Process Data Dictionary is used to process selected file names as well as all work fields. Domains are always processed whenever any portion of the data dictionary is processed.
- Create Executable Modules will compile executable modules (EMs) for all selected processes in the application. This eliminates the need to wait for a process to compile when it is invoked during testing or at runtime. The default is create EMs for all processes in the application.

Process Generation

- Form Input creates an input process for a selected file with one record per screen
- Scrolling Input creates an input process for a select file that scrolls, or contains multiple records per screen
- Form Output creates an output process (including Job, Query, and Output) for a selected file. There is one data record per page.
- Standard Output creates a complete output process containing multiple data records per page.
- Comma Delimited Update creates an update process used to create a comma delimited file for any file within the application.

Design Analysis

- Verify All References runs a special version of "Cross Reference All" for the current application. You can select the REFERENCED BUT NOT DEFINED option to generate a list of all design elements and processes that are referenced by other elements or processes yet do not exist. Or, you can select the DEFINED BUT NOT REFERENCED option to list all design elements and processes that exist but are not referenced anywhere in the application.

Print Technical Documentation

The Print Technical Documentation utility allows you to print complete technical documentation for any domain, file or field, work field, or process. When you select PRINT TECHNICAL DOCUMENTATION from the Utilities menu, the overlay shown in *Figure 1-8-10* appears. Each Print Technical Documentation option presents you with a selection screen, from which you can select any item to print. Items can also be selected based on the date they were last modified or, in the case or processes, by type (input, output, etc.).

Print Technical Documentation menu

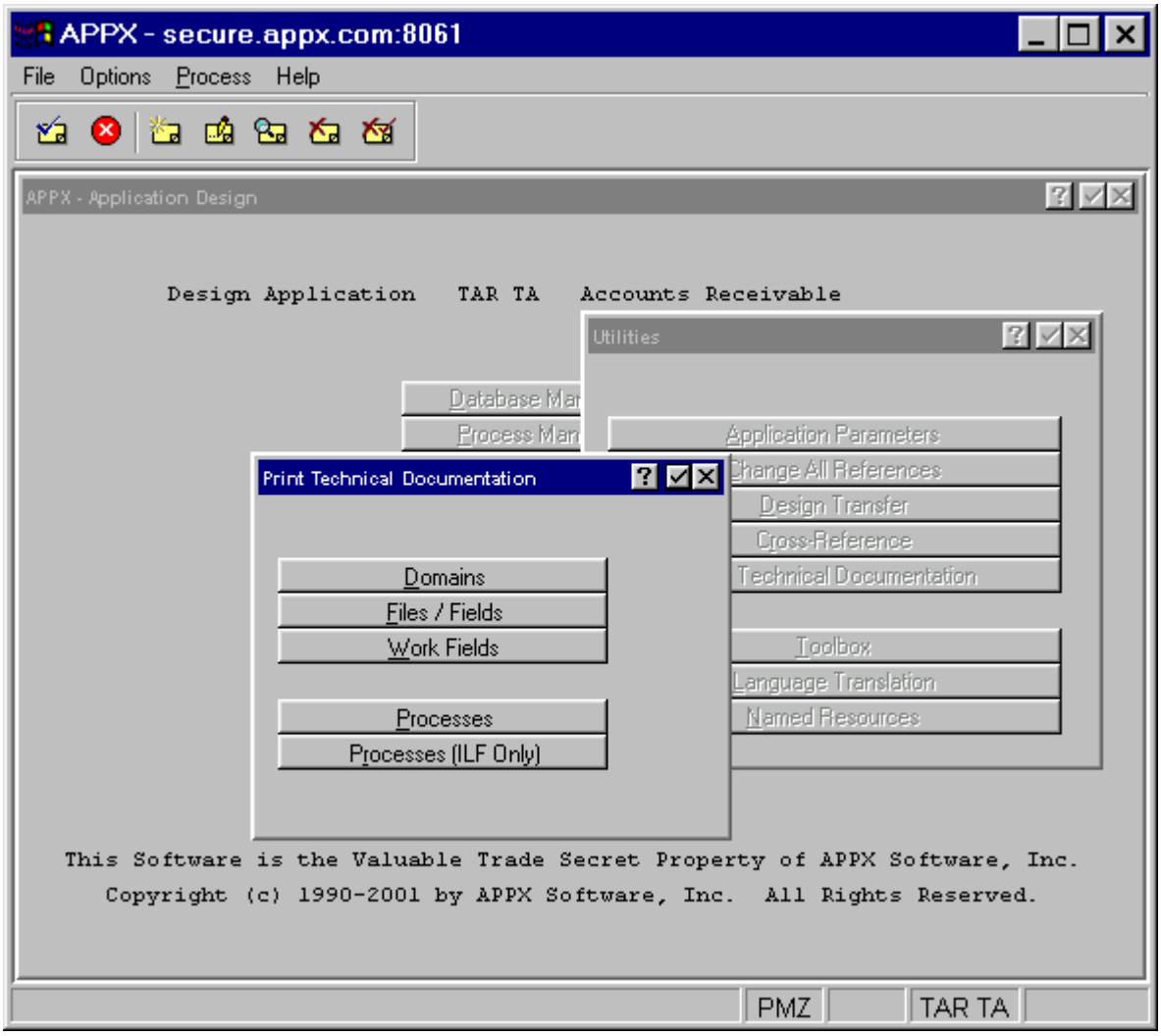


Figure 1-8-10

CHAPTER 2-1

Data Dictionary Overview

APPX Data Dictionary

A *data dictionary* is a repository for definitions of an application's data structures. In APPX, these definitions include both specifications (such as a file's *Descriptive Name* or a field's *Length*) and supporting documentation.

APPX's data dictionary is the cornerstone of application development. All processes operate on the basis of, and are subject to the specifications entered for, elements in the data dictionary. Dictionary specifications provide information and defaults for, or impose restrictions on, the database management facility.

Accessing the Data Dictionary

You access the APPX Data Dictionary by selecting DATABASE MANAGEMENT from the Application Design menu or from various Toolbox menus. The *Database Management* menu overlay (*Figure 2-1-1*) includes options to edit each dictionary component, domains, files/fields, and work fields, and to process the dictionary. Each of the Data Dictionary options is described in detail in the chapters that follow. Physical File Management is covered in *Chapter 3, Database Management,* of the *System Management Manual.*

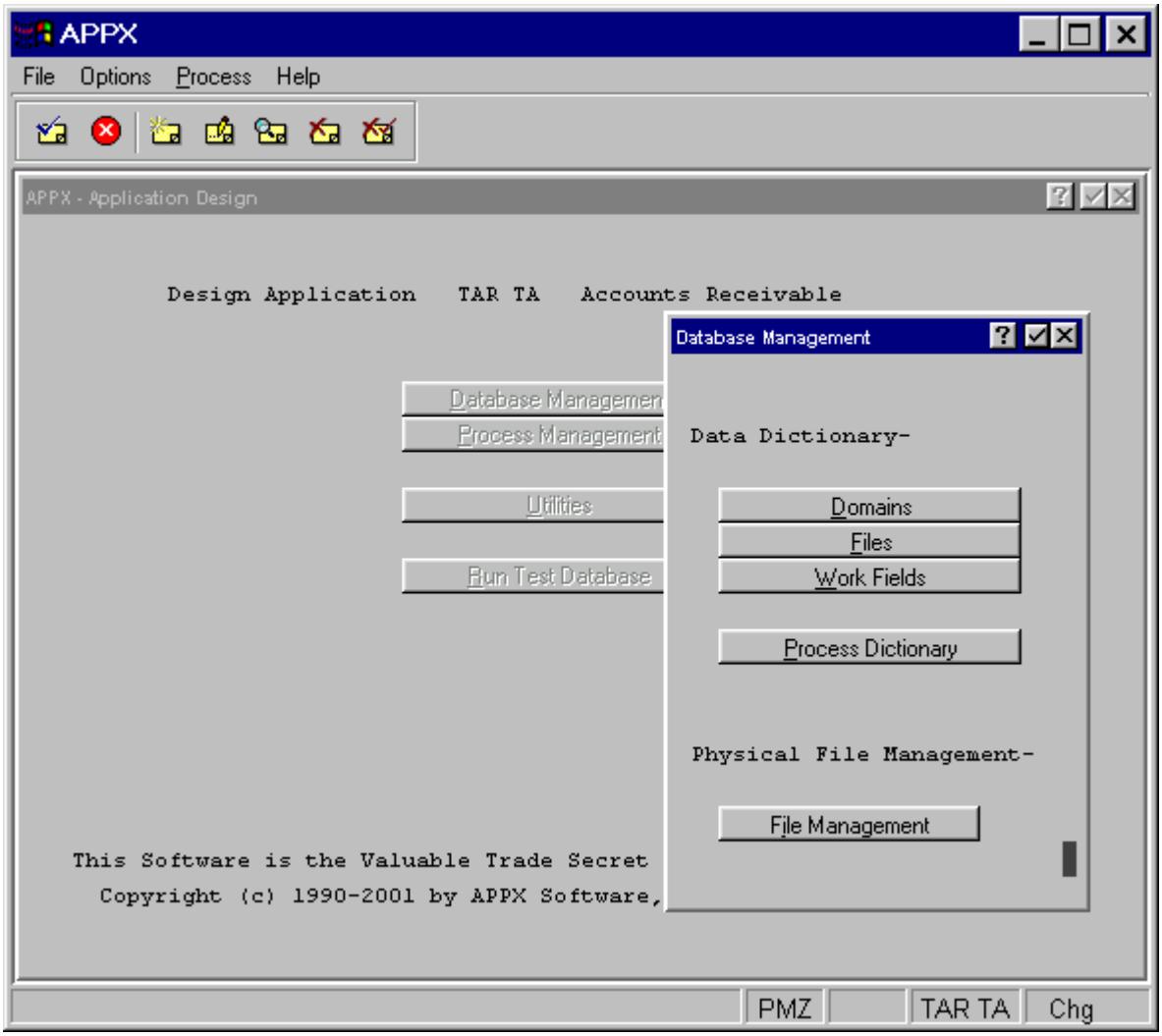


Figure 2-1-1. Database Management Menu Overlay

Characteristics of the Data Dictionary

There are four principal attributes that characterize the APPX data dictionary. The dictionary is:

- ***Active,*** in that its specifications are used by, and place real time restrictions on, both designers creating applications and users operating them. An active data dictionary maximizes consistency, productivity, and integration. It minimizes the chance for error, both when designing and using applications. For example, assume you have defined a numeric, five-character zip code field. APPX uses the length specification to provide a proper mask for the designer who is painting the field on an input image. APPX also uses data dictionary specifications as a basis for data validation when the image is executed. The system displays an error message if the application user attempts to enter a Canadian postal code (containing letters) into the numeric zip code field.
- ***Processed***, to enhance performance. The specifications you enter into the data dictionary are translated into a format more directly usable by APPX during process execution. Processing the

data dictionary also searches for logic errors (invalid key definitions or illegal record lengths, for example), constructs default value records for each file, computes field lengths, and establishes the starting position for each field within a record. Refer to Processing the Data Dictionary in *Chapter 2-2* for more information.

- ***Internally documented.*** The dictionary provides entry screens for the online help text and printed technical documentation. Documentation is an integral part of APPX development and user interface.
- ***Automatically monitored for structural changes,*** to ensure integrity. If you make such a change to the dictionary, all processes that reference changed elements are automatically updated, including images and event points. APPX flags existing data files that require restructuring, and permits you to automatically convert them to the new structure with the restructure utility in database management.

Components of the Data Dictionary

The principal components of the data dictionary are an application's:

- ***Domains,*** which are generic definitions, or patterns, for fields that share a set of common attributes. For example, your application may include a number of different General Ledger account fields, such as an Inventory account, a Sales account, a Cost-of-Goods-Sold account, and so on. Each of these fields is the same length and has the same alpha and numeric components. You can define these common attributes in an Account domain. Refer to Chapter 2-3: Domains for detailed information.
- ***Files,*** which are data storage structures that consist of a collection of related ***records***. Each record contains one or more ***fields***. The record layout for a file is determined by the fields defined for that file within the data dictionary. Refer to Chapter 2-4: Files and Fields for detailed information.
- ***Fields***, which are the most elemental data storage structures of applications. They store the individual items of alphanumeric, numeric, date/time, and logic information that are available to users of APPX applications. The dictionary includes both physical and logical fields. Refer to Chapter 2-4: Files and Fields for detailed information.
- ***Work fields,*** which are "temporary" fields that are not stored in a file but can be manipulated by processes in APPX (usually with statements). You cannot save the contents of work fields after a user has logged off, although you can pass work field values to other fields or processes. Refer to Chapter 2-5: Work Fields for detailed information.

Domain Concepts

Domains are generic definitions, or patterns, for fields and work fields that share a set of common attributes. These attributes include such things as the physical space a field occupies, the set of values it can contain, and the way a field is presented to a user.

It is important to distinguish between a domain and a field. A domain is simply a pattern or set of common attributes for one or more fields. It is a design component that stores field specifications, not application data. Fields store data in APPX applications according to the pattern established by the domain the field is based on, if any, in the data dictionary (or according to the individual field specifications if not based on a domain).

Domain Types

There are nine types of domains in the APPX data dictionary:

- *Alpha* domains are patterns for fields comprised of letters, numbers, and special characters (-, #, or &, for example), where the primary characteristic is field length.
- *Format* domains are like alpha domains except that an edit mask is the primary characteristic.
- *System* domains are patterns for fields that contain data provided by the system, often referred to as audit fields.
- *Text* domains are patterns for fields comprised of blocks of text.
- *Token* domains are like alpha domains except they have specific values that are stored as corresponding numbers.
- *Numeric* domains are patterns for fields restricted to number, decimal (.), and sign (-) entries.
- *Date* domains are patterns for fields that contain dates and/or times, from a hundredth of a second through a century.
- *Logic* domains are patterns for fields with possible values of yes ☑ or no ☐, or ? to represent an unknown value.
- *Same As* domains are patterns for fields that share the same characteristics as a domain in another application in your database.

Functions of a Domain

Each domain specification provides one or more of the following:

- ***Storage definitions***, from which APPX determines the internal storage length and format of any field that references the domain. A domain does not cause storage allocation. It simply defines the storage that APPX allocates when fields are created. Refer to the Fields section in this chapter for more information on physical storage allocation.
- ***Display characteristics*** for items painted on an image, such as display masks or justification options.
- ***Constraints and restrictions*** on a field's valid values, including range checks, validation tables, edit masks, and data lookup.
- ***Other field attributes*** including column headings, descriptive names, default values, and security codes.

Domains contribute a number of significant benefits to the APPX application design process. They:

1. Minimize the development effort, by eliminating duplicate entry of common field attributes.
2. Minimize maintenance efforts, by providing a single location where changes flow through to all related fields in an application.
3. Ensure consistent database definition throughout the system. The length or edit attributes of a customer number, for example, never vary from Customer file to Invoice file to Cash Receipt file, etc.

It is important to note that the attributes defined for a domain apply to any field referencing that domain and cannot be overridden at the field level. Therefore, only define characteristics that are appropriate for each field subject to the domain. For example, since an employee number should have the same format and edit characteristics wherever it is used in the system, define such common attributes once in an employee number domain. However, if the descriptive name for employee number can vary from one field to another, leave *Descriptive* blank in the domain definition. This would enable, for instance, the Employee master file to contain two fields based on the employee number domain, each with an appropriate descriptive name (i.e., "Employee Number" for the employee's own number and "Supervisor Number" for the number of the employee's supervisor).

Some Examples of Domains

Domains are designer-defined to satisfy the requirements of a specific application or system of applications. A domain can provide a pattern for a specific field in an application, or a category of fields.

A Product Number domain, for example, can describe the attributes of a Product Number field wherever it occurs in an application. You may even assign the same name to a domain and to a field. Zip codes, customer numbers, and vendor numbers are additional examples of domain definitions that relate to specific fields.

An Amount domain, on the other hand, might be linked to a virtually unlimited number of fields and work fields in the system, such as Invoice Amount, Inventory Amount, Order Amount, Requisition Amount, Line Item Amount, Amount to Pay, and so on. Such domains are generically named, and usually do not share a name with a specific field. Accounts and addresses are additional examples of domain definitions that relate to a variety of fields in an application or in the system.

Mask Definition

Masks are symbolic representations of fields. Each character position has a specific meaning with regard to the data it can store or display. Masks represent a visual approach to data definition, or data painting. When appropriately designed, masks enhance the ease of use and operation of an application, and the readability of displays and printed output, and they minimize the space that is required to store and display data.

You establish edit masks for format domains and fields. APPX uses the edit mask as a basis for both allocating a related field's physical storage and for online formatting and data validation. The number of characters in an edit mask define the storage requirements. Constant and placeholder mask characters define format and validation requirements:

- A ***constant mask character***, such as a "/" or a "-" in a phone number edit mask, automatically displays for any field painted on an image in an APPX process.
- ***Placeholder characters*** represent data-specific field positions. The individual placeholder character determines the allowable values for that position in an edit mask. For example, a "@" followed by a "#" in an edit mask indicates that the user must enter a letter followed by a number.

In addition, masks for some domains include reserved characters that affect the way APPX treats other mask characters, but do not relate directly to positions in the field value. For example, quotation marks in a date/time mask define mask text, but do not themselves display on an image. For more information, see the section on the date/time display mask in *Chapter 3-14 or 3-15*.

Display Mask Relationship (Data Dictionary vs. Image)

Both the data dictionary (for a domain or field) and the process facility (for an item on an image) provide display masks for numeric or date/time items. Data dictionary default display masks are assumed at the image item level unless specifically overridden in the item specifications.

You can override the dictionary display mask for either a numeric or date/time item in an input, output, or inquiry process. For a numeric item, you can enter a new display mask to alter the format of a dictionary display mask and, with an optional specification, elect to override the number of characters that display. For a date/time item, your display mask overrides the parts of a date/time field that display and/or the format in which they display. Refer to Numeric Item Specifications and Date/Time Item Specifications in Chapter 3-14: Using the Character Based Image Editor for a discussion of image masks.

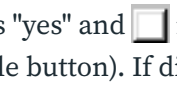
External, Internal, and Universal Data Formats

APPX communicates data in any of three formats (distinguishable by the intended recipient), and automatically translates between these formats as appropriate:

- ***External formats communicate data to an application user.*** They represent the displayable form and include all formatting characters (such as $, :, -, /, and #) called for by display masks, if any.
- ***Internal formats communicate data to the system.*** They represent the way APPX manages data (binary, packed decimal, or alpha, for example). The internal format is determined by *Storage Type*, *Field Type*, *Edit Mask*, and *Length* specifications.
- ***Universal formats communicate data to a designer.*** They represent a consistent (alpha) representation of data, which is independent of how information is stored or displayed. Default values, range checks, constants entered in a statement, record selection criteria, and query and child selection criteria are all maintained in Application Design in a universal format. Unlike external formats, universal formats do not include the formatting characters found in display masks. However, note that there is no difference in the external, internal, and universal formats for *format* fields (and domain-type fields based on a *format* domain). *Format* fields are displayed, stored, and coded using the same format.

Universal formats provide several significant advantages. They minimize maintenance overhead; you can change a field's storage or display characteristics without updating those portions of the APPX design facility that reference a field as a constant. They also permit you to reference and operate on a predictable format for the contents of a field, without having to remember exactly how data either displays or is stored in the system. A good example of this is a date field which is typically stored in packed decimal format. By means of appropriate display masks, the date April 15, 2002, may appear as 04/15/2002 to a user on an input or record selection screen and yet print as Apr 15, 2002, on a report. However, the designer would use the universal format of 20020415-------- to code this date wherever needed within Application Design.

The universal format for fields is described as follows:

- A date/time field consists of a mandatory 16 digits (or blank). When designing statements, range checks, or query and child selection criteria, paired hyphens for leading or trailing "parts" in a field are also permitted. These hyphens indicate that a corresponding part is not significant to the comparison or other operation.
- A logic field includes 1 or Y for "yes", 0 or N for "no", and blank for "unknown". Note that the external format of these values vary by how the logic field is displayed. If displayed as a checkbox, is "yes", is "no", and is "unknown". If displayed as a toggle button, is "yes" and  is "no" or "unknown" (since an unknown value is treated as a "no" when using a toggle button). If displayed as raw text, Y is "yes", N is "no", and blank is "unknown".
- A numeric field consists of digits, and an optional decimal point, optional leading or trailing sign (or blank).

- An alphanumeric field includes letters, digits, and special characters as well as justification, pad characters, paragraph symbols (for text fields), and constant characters (for format fields).

Fields

Fields represent the smallest unit of data storage in an APPX application. Each field contains an item of alphanumeric, numeric, date/time, or logic information in a database (field types are described in detail in the following section). While a domain establishes physical storage requirements and various attributes for a field, it is a field itself that actually stores application data. The data information stored in a field is referred to as a *field value*. For example, a telephone number domain may specify a length of ten alphanumeric characters to store. It is a template for all telephone number fields in the database that reference it. But, it is not until a customer telephone number field is actually created in the database management facility that APPX allocates storage space. The customer's telephone number would be stored in this field in, say, the customer record.

The relationships between various fields in a database are established in APPX files. The fields defined for a file in the data dictionary determine the record layout of the file.

With a few exceptions, APPX fields are *database-dependent* (unique to a specific database). Work fields and fields that are not stored in permanent, disk-based files are *database-independent*. Refer to Temporary versus Permanent Files in this chapter and Integration with Database Management and System Administration in *Chapter 2-2,* for more information on database dependency.

Field Types

There are thirteen field types in the APPX data dictionary:

Domain

Fields are fields that directly express the attributes defined in a domain while they define and occupy new physical space in a file. There are nine distinct domain-type fields: Alpha, Format, System, Text, Token, Date, Logic, Numeric, and Same As. Refer to the Domain Types section for additional information. You can choose to set up physical fields to exclusively reference domains, or you can set up individual field types directly.

Alpha

Fields are comprised of numbers, letters, and special characters (-, #, or &, for example).

Format

Fields are like alpha fields except that an edit mask is the primary characteristic.

System

Fields, often referred to as audit fields, display data that is provided by the system.

Text
Fields allow the user to enter blocks of text on the data entry screen.

Token
Fields are like alpha fields except that they have specific values that are stored as corresponding numbers.

Numeric
Fields are restricted to number, decimal (.), and sign (-) entries.

Date
Fields contain dates and/or times, from a hundredth of a second through a century.

Logic
Fields are fields with possible values of yes or no , or to represent an unknown value.

Synonym
Fields provide alternative names for referencing existing APPX fields. They occupy no physical space in a file aside from the fields they reference.

Substring
Fields represent discrete portions of longer alphanumeric fields. They occupy no physical space in a file aside from the fields they are a part of.

Group header
Fields are used to group other fields by identifying the beginning and ending fields in a group of fields. The group can then be referenced by the name of the group header field. Group headers occupy no physical space in a file aside from the fields that comprise the group.

Group trailer
Fields are optional placeholders to allow for terminating a group without referencing a physical field. They provide a streamlined method to allow for additions to the end of groups. Group trailers occupy no physical space in files.

Physical versus Logical Categories

Each of the APPX field types can be classified as *physical* or *logical*. The distinction between the two reflects a field's physical space requirements.

Table 2-1-1 outlines the properties of physical and logical fields. Each is discussed in more detail in the sections that follow. You can also refer to Chapter 2-4: Files and Fields, for more information.

Table 2-1-1. Properties of Physical and Logical Fields

Fields	Assumes Attributes From	Defines New Physical Space
Physical Fields	**Domain or Field**	**Yes**
Domain-Type Fields	**Field**	Yes
Alpha Fields	**Field**	Yes
Format Fields	**Field**	Yes
System Fields	**Field**	Yes
Text Fields	**Field**	Yes
Token Fields	**Field**	Yes
Numeric Fields	**Field**	Yes
Date Fields	**Field**	Yes
Logic Fields	**Domain or Work Field**	Yes
Work Fields		Yes
Logical Fields	Alphanumeric Field	**No**
Substring Fields	Any Field	Subset of existing Field
Synonym Fields	One or More contiguous Fields	Redefinition of existing Field
Group Fields		Redefinition of existing Field(s)

Physical Fields

Alphanumeric, date/time, logic, and numeric fields, and all work fields are physical fields. They define, or occupy, *new physical space*. They may, however, share this space with logical fields as explained later in this chapter in the section, *Logical Fields*.

Work Fields

Work fields are fields that are not stored in a record format but can be manipulated by processes in APPX (usually through the use of statements). A work field has no sequence number and no relationship to any files, records, or other work fields.

You use work fields to display values that are not stored, such as a calculated line item amount (quantity multiplied by price) or a constant value like π (pi). You can manipulate work fields in much the same fashion as any other field, with a few exceptions:

- You cannot save a value in a work field after a process completes. To save a value, it must be copied into a field in a record that is stored in a file.
- You cannot define a group of work fields because they are not arranged in any particular order or sequence.
- Work fields cannot be keys because they are not part of a record structure within a file.

Refer to Chapter 2-5: Work Fields for additional information.

Logical Fields

Substrings, synonyms, and group fields (as defined by group header and optional group trailer fields) are logical fields. They require no separate (new) allocation of physical space. They occupy space, or a portion of space, that is occupied by the field they reference. In other words, logical fields *redefine or share the physical space* that is occupied by a physical field (see *Figure 2-1-2*).

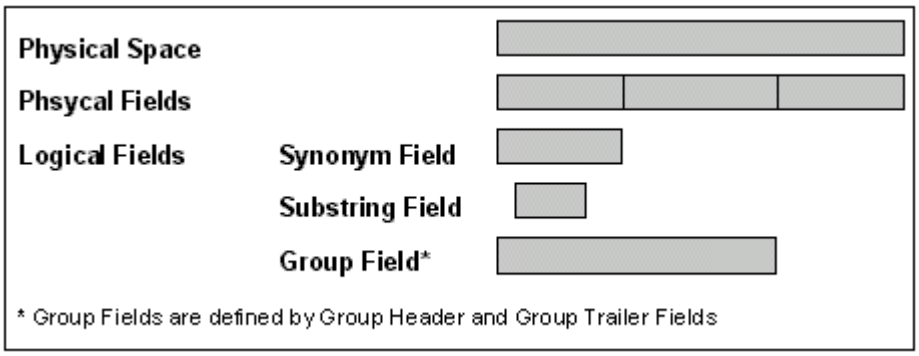


Figure 2-1-2. Physical Space Requirements

Logical fields draw a subset of basic attributes from one or more physical fields.

The full set of attributes for a logical field is actually a combination of those established by physical fields, APPX defaults, and you. A logical field draws certain attributes from a physical field when the dictionary is processed. At the same time, it assumes additional attributes established by APPX or by specifications you enter for a logical field itself. The specific source of each attribute depends upon the type of logical field.

Substring Fields

A *substring* field represents a discrete portion of an alphanumeric field (specifically, an alpha, format, or text field or a domain-type field based on an alpha, format or text domain). Substrings are defined for those instances where it is beneficial to be able to access both a field in its entirety and a discrete segment of a field, depending upon the specific requirements of an operation or display.

A substring field occupies a portion of the same physical space as the alphanumeric field it is a part of. For example, if you define a substring to access the three-character area code portion of an alphanumeric telephone number, the substring does not store a separate copy of the data in those referenced three characters. It references the same three characters that are physically stored in the telephone number field. As a result, a change to the data in either the substring or the alphanumeric fields affects both fields.

Because a substring occupies the same physical space as an alphanumeric field, its data is a subset of the data contained in that alphanumeric field. A substring's initial value is also a subset of the alphanumeric field's initial value.

Substrings are discrete segments of the internal formats of physical fields. As such, many of the formatting attributes of a physical field do not flow through to a substring. Those attributes a substring does assume are generally not format-related as, for example, *Descriptive*.

The set of attributes for a substring is essentially equivalent to the attributes of a basic alphanumeric field. (Remember that a basic alphanumeric field is an alpha field that is not specially formatted.) In other words, all alphanumeric domain and field specifications use their default values except for the *Length* specification, which you enter.

Refer to the Substring Fields section in *Chapter 2-4* for additional information.

Synonym Fields

A *synonym* field provides an alternative reference for an existing field and, by doing so, logically reorders fields in a file. With synonym fields, you can access the *same data* under more than one field name and with more than one field sequence number.

For example, you can use a synonym to create a group field that includes a physically noncontiguous field. (APPX requires that the fields in a group are contiguous, based on sequence number.) You simply define a synonym for the noncontiguous field using a contiguous sequence number.

A synonym field occupies the same physical space as the field it relates to. For example, assume you wish to access three fields together as a group. One of these three fields contains a noncontiguous sequence number, a condition that is not permitted within a group field. To overcome the problem, you define a synonym of the noncontiguous field, and assign the synonym a contiguous sequence number. The synonym produces a logically contiguous series of bytes that, for purposes of group manipulation, APPX treats the same as a physically contiguous series. Because of the synonym definition, the physical spaces are not required to lie in the same order as the logical fields in the group field.

A synonym does not store a separate copy of data in a referenced field. Both a synonym and the field it references access the same physical data. As a result, a change to the data in either a synonym or the field it references affects both fields.

Most of the attributes for a synonym field are identical to the attributes of the physical field that a synonym references. Refer to the Special Uses for Synonym Fields section in *Chapter 2-4* for additional information.

Group Fields

A *group field* (often referred to simply as a group) is a series of one or more fields with contiguous sequence numbers that can be referenced by a single field name. Groups are defined for those instances where it is of benefit to access fields both individually and as a unit, depending upon the specific requirements of an operation or display. For example, group fields are often used to define multipart keys.

A group field's primary purpose is to simplify data assignments and comparisons between two sets of fields with matching contents, formats, and storage types. You can only reference a group field in a statement or within selection criteria, or specify it as a key. You cannot, for example, paint a group as an item in an image.

APPX treats all components of a group field as physically contiguous, regardless of how they are stored. This is particularly significant when you use a group in an assignment or a comparison. For example, assume you compare two group fields, both 15-bytes-long. Group A is composed of two noncontiguous components: 10 bytes and five bytes. Group B is composed of three noncontiguous components: three bytes, 10 bytes, and two bytes. APPX does not compare the first component of group A with the first component of group B. Rather, it compares the first three bytes of group A with the first three bytes of group B, the next seven bytes of group A with the next seven bytes of group B, and so on. The same is true for an assignment. APPX treats groups as a contiguous string of bytes for purposes of both comparison and assignment.

You create a group by defining a *group header* and, optionally, a *group trailer* field.

A *group header* is a field that serves simply to mark the beginning and end of a series of contiguous (physically adjacent) fields in a concatenation or group. This group of fields can be referenced by the name assigned to the group header field.

A *group trailer* field is an optional placeholder to mark the position of the final field in a group. It simplifies group maintenance by providing a streamlined mechanism to allow for future additions to the end of a group. As with a group header field, a group trailer occupies no physical space in a file.

Group header and group trailer fields occupy no separate physical space in a file and serve only as placeholders to mark the beginning and end of a series of contiguous fields which you can access as a unit.

A group field itself, bounded by a group header and trailer, references one or more contiguous fields through a single field name. A group does not store a separate copy of data contained in component fields. Both a group field and the component fields it references access the same physical data. In fact, more than one group field can access the same component fields (nested group fields), as long as one group is completely contained within another group. Groups with partially overlapping physical storage are not permitted; in other words, any two groups cannot share a component field unless they share all the component fields in one of the groups. Refer to the Group Fields section in *Chapter 2-4* for additional rules and restrictions.

The attributes of a group field are *not* simply equal to the sum of the attributes of component fields. The content of a group is actually based on the *internal* format of all of its components. The only exception is the sort function of a query process, for which a group field reflects the attributes of its component fields. For example, for the purpose of the sort only, there is no difference between the numeric value in a group field and the numeric field itself.

APPX considers a group to be a *basic alpha field* (a single string of alphanumeric characters that are not specially formatted), regardless of how component fields are defined. If all components are not basic alpha fields, APPX performs no special translation when a group is used in a comparison or an assignment. Refer to the Field Types section for the definition of a basic alpha field.

The distinctions between the attributes of a group and its component fields have several important ramifications:

- By manipulating a group, rather than the component fields, you run the risk of placing invalid data into a component field. For example, if your group includes a packed decimal field and you set it to AAAA, you replace the data in the field with data that is not valid for a packed decimal number.
- APPX compares two strings of bytes with no translation or error checking. It does not warn you that a comparison is not based upon like component fields in a group. For example, assume you are using two groups (A and B) in the statement, IF A EQ B. A is composed of a customer number and a transaction number, and B is a transaction number and date. APPX compares these two group fields and returns a false, not an error, condition.
- Comparing two group fields does not always yield the same result as comparing the individual components of two groups. Assume the A and B group fields in the previous example are both comprised of customer number and transaction number, but the customer number in one is an alphanumeric and, in the other, a numeric field. If APPX compares a customer number of 100 in A to a customer number of 100 in B, it is capable of converting the values in these two fields to an intermediate format for comparison and returning a true condition. The same comparison, if based upon the groups rather than the component fields, returns a false condition.

These differences in attributes are particularly significant if you manipulate group data with statements. Refer to Chapter 4-1: ILF Integration for a discussion of APPX statements.

Designating Keys

In general terms, a *key field* is a field that you can use to identify and retrieve a record in an indexed file. Key fields in APPX include a single primary and up to 16 alternate keys.

Primary keys

Provide the assumed key path to access and retrieve specific records in an indexed file. They also establish the linkage between files. For example, you might include a Customer Number field (the primary key to a Customer file) in a Sales Order file to link an order with additional information about the customer placing it. Unless otherwise specified by a user or designer, APPX always uses the primary key path. The field that is designated as the primary key should generally be short and unique, and representative of how data is most frequently accessed in the file. Examples of primary keys include a customer number in a Customer file, a product number in a Product file, and a date/time stamp in a Purchases History file.

Alternate keys

Provide optional key paths to access and retrieve specific records in an indexed file. A user or designer may optionally choose to change from a primary key path to any alternate key path for record access and retrieval. Examples of alternate keys to a customer file might include customer name, phone number, contact name, and/or address.

By using a primary key rather than an alternate key, you gain two *referential integrity* advantages.

- ***Delete protection*** fields govern whether or not a user can delete a record while executing an input process. When used properly, these fields prevent the creation of orphan records because they prohibit deletion of records that are referenced by other records anywhere in the database.
- ***Key protection*** fields control whether or not a user can modify the primary key to a record while executing an input process. The proper use of these fields also helps to maintain the integrity of a database because they prohibit modification of keys that are referenced by other records in the database.

Three significant restrictions apply to the definition of primary and alternate key fields in APPX:

- Group trailers cannot be defined as key fields.
- Each indexed file must contain one (and only one) primary key.
- A maximum of 16 alternate keys can be defined for any file.

To maximize performance, you should minimize the number of key fields in each file. Unless you use a field for one of the following purposes, it should not be defined as a key:

- Online data retrieval. For example, you may define a customer number as a primary key so that your application users can retrieve specific customer records for display. Or perhaps, you might create a duplicate alternate key based on the customer number in a Sales Invoices file to display invoices online in customer number sequence.
- To relate data in one record to data in another, either within the same file or among files. Remember that primary keys are the only fields that can be used to automatically maintain referential integrity with the *Delete Protection* and *Key Protection* options.
- Data lookups for items in a process. If you include a customer number field on a sales order record, your users can automatically scan customer information when entering an order, if the customer number is a key to the customer file. In addition, APPX uses a key field to access related information when generating a standard subheading image in an output process.

A *unique key* prohibits an application user from entering more than one record with the same value in a key field. The term *duplicate keys* refers to a user's ability to define two or more records with the same value in a key field.

- Unique key fields provide key paths that you can use as record identifiers. To be able to identify a specific record in a file, you must define at least one key as unique. The primary key is generally designated as the unique key in a file.

- Unique keys are also used for online edit checking. If a user attempts to add a record with a value in a unique key field that corresponds to a record already in a file, APPX does not permit the user to save the entry. APPX considers this an error condition and prohibits the user from adding the record.
- Duplicate key fields are used for online record retrieval in a predetermined order. For example, to display sales orders on the screen in customer number sequence, Customer Number is defined as a duplicate alternate key to the Sales Orders file. The trade-off for this capability is that duplicate keys decrease performance and increase storage requirements.

Key Order versus Sort Order

The *key order* for a group of records refers to the sequence APPX uses to retrieve records along a key path in an indexed file. For example, if you scan through customer records within an APPX Customer input process, the customer records are presented in key order. Key order can be defined in ascending or descending sequence.

The *sort order* refers to the sequence an APPX query process arranges records in for use by a subsequent output, input, or update process. Sort order, also known as APPX's collating sequence, can be defined in ascending or descending sequence as well.

The principal differences between the key order and sort order sequences are as follows:

- Sort order is language-dependent.
- Sort order combines uppercase and lowercase alpha characters (i.e., AaBbCcDd...). Key order treats uppercase and lowercase separately (i.e., ABCD...abcd...).
- Sort order always presents numeric field values in sequence (negative to positive, if ascending, or positive to negative, if descending). Key order may not be sequential if there are both positive and negative values.
- Sort order uses the the alphanumeric value of token fields. Key order uses the numeric token ID.

For date and logic fields, APPX's sort order is identical to its key order (assuming both are defined as ascending or both descending). For other fields, however, the two sequences are not always identical. The distinction is important because an identical group of records can be sequenced differently by two APPX processes if one process uses a key path and the other uses the sort order established by a query process.

It is also important to understand that sort fields are not restricted to key fields. In other words, a field does not have to be a key to be used as a sort field in a query process. (Sort fields are a common example of the overuse of key fields.) An APPX sort creates the equivalent of a temporary key path (for example, it is only available during the production of a report), which is based on the sort field. If a sort field is defined as a key, APPX must constantly maintain the key path online. If a field is not a key, APPX only builds a key path when it is required by a sort.

While APPX's sort order combines upper and lowercase alpha characters, the key order retrieval for alphanumeric fields separates the series of uppercase alpha characters from the series of lowercase alpha characters.

Whereas numeric fields are always sorted in numeric sequence (say, -2, -1, 0, 1, 2, for ascending), the numeric key order can vary depending upon storage type. For instance, the key order for the same sample values in a packed decimal numeric field would be 0, 1, -1, 2, -2.

If a field is defined as a token, the key order retrieval is always in numeric sequence by token ID. Keys always operate on the actual data stored in a file which, in this case, is the token ID. The sort order established by a query process, however, is based on the corresponding alphanumeric value of the token field, not the token ID.

Files

The APPX dictionary establishes the *logical structure* of your database. The physical structure of a database and how it is stored on disk varies from system to system.

The data dictionary's file formats represent permanent data storage in an APPX application. *Files* consist of a collection of related records, each containing one or more *fields,* the most elemental data storage structures of an application.

APPX supports one-record, consecutive, and indexed organization. You can designate either fixed- or variable-length records and, optionally, whether you want a file stored on disk or in memory.

File Storage Options

For each file you define in the data dictionary, APPX requests storage information. You determine whether each record format is stored in a file and, if so, whether the file is maintained on disk or in memory. The attributes of each of these file storage options are summarized in *Table 2-1-3*, and discussed in more detail in the following sections.

Table 2-1-3. A Comparison of File Storage Types

Characteristic	Physical File	Disk File	Memory File
Permanent	Yes	Yes If So Specified	No
Capacity Limitation	Not Applicable	Amount of Disk	Paging Space
Shared Among Users	Yes	Yes (If Permanent)	No
Relative Performance	Not Applicable	Slower	Faster
Database Dependent	Yes	Yes (If Permanent)	No

An unstored record format is essentially a pseudo file that you can use to enter data, perform edit checks and computations, and execute statements without actually storing data. APPX does not read data into a record area before executing a process, and it does not store data after executing a process. In all other respects, record area operations are the same for unstored records as for records stored on disk or in memory.

There are two primary differences between unstored records and records stored in files (temporary or permanent). Unstored records:

- Can only communicate data from one process to another by implicit or explicit interprocess communication, passing and receiving definitions. Unstored records communicate only through areas in memory. Records stored in temporary and permanent files are less restrictive. One process can write a record to a temporary file, and a separate process can read the record from the temporary file. Unlike information in unstored records, this information can be saved and retrieved at will.
- Contain only the information that is in the current area in memory, which is the equivalent of a single record. Temporary and permanent files can contain more than a single record.

Unstored record formats are similar to work fields. You can think of a work field as an unstored record with a single field. Unlike work fields, however, unstored record formats:

- Establish relationships between one field and another. As a result, they eliminate a number of the limitations that are inherent in work fields. You cannot include a work field in an APPX group field. You can define a group field that consists of fields from an unstored file, simplifying field manipulation and comparison in many instances. Without an unstored option, you could only compare records by reading a record from disk, loading each field on the record into a work field and, then, comparing each work field to a field in a second record. With the unstored option you can create a work record, containing all of the fields in a disk-based record and, using a group of the entire record, set the contents of the disk record into a work record with a single statement.
- Can serve as process control files (PCFs) for APPX processes. If the record format includes a one-record organization specification, APPX completes one process cycle during process execution; in other words, APPX sequentially executes the frames for a process once and then ends. If the record format is other than one-record, the process continues to execute until the application user chooses to end it. For example, imagine a calculator utility that allows a user to enter principle and interest rate data and, then, computes and displays a monthly payment amount. If the unstored record format is one-record, the calculation is executed once and the program ends without saving the entered or calculated information. If the format is other than one-record, an application user can change the entered values as often as desired and APPX will continue to update the computation and the display until the user chooses to end.

Disk files are generally permanent files that are shareable among users and stored on a disk surface defined in *database management*. You can, however, define temporary files that are stored on disk.

The system always takes longer to retrieve data from files stored on disk than to retrieve data from files stored in memory. Disk files have an advantage over memory files because they have more capacity.

Permanent disk files are database-dependent, or stored as part of a specific database. As a result, APPX's database management utilities reference data dictionary specifications when operating on a disk file. Temporary disk files are database-independent.

Memory files are temporary files that are unique to a task. The contents of a memory file are lost when the task ends. Because files are maintained in memory and memory is relatively more expensive than disk space, the quantity of data in a memory file is generally limited. Excessive or improper use of memory files can result in memory overflow conditions.

The principle reason to use a memory file is to increase performance. The system always takes less time to retrieve data from files that are maintained in memory than to retrieve data from files that are stored on disk. By using memory files in strategic locations in a process, you enhance performance while maintaining your application as simply as possible. For example, assume you generate a report based on a 10,000-record customer file. For every customer record processed, APPX must also read a referenced record in a 10-record customer-type file. If the customer-type file is a disk-based file, APPX reads 20,000 records from disk, or a customer-type record for every one of the 10,000 customer records read. If, on the other hand, you read each of the 10 customer-type records into a memory file, your total number of physical I/Os is reduced from 20,000 to 10,001, which is a reduction of almost 50 percent.

File Organization

APPX supports three types of file organization: one-record, consecutive, and indexed. The type of file organization you select affects performance, space requirements, and the procedures you can perform with regard to a file.

If a file is used as a process control file (PCF), this specification affects the operation of a related process. If a file is one-record, APPX completes one process cycle during process execution; in other words, APPX sequentially executes the frames for a process once and then ends. If the file is other than one-record, the process continues to execute until the application user chooses to end it. Refer to Chapter 3-1: Overview of APPX Processes for additional information on PCFs and process cycles.

File organization is the only specification that affects record formats that are not stored in a file. If an unstored record is used as a PCF, the value you enter determines whether or not a process executes a single process cycle.

The following material is intended to help you choose the best type of organization for each file. The attributes of each type of organization are summarized in *Table 2-1-4*.

Table 2-1-4. A Comparison of File Organization

Attribute	One-Record	Consecutive	Indexed
Can Delete Records?	No	No	Yes

Attribute	One-Record	Consecutive	Indexed
Retrieval Method	Not Applicable	Position in File	Key Path
Relative Performance	Faster	Faster	Slower
Process Control File	One Process Cycle	Multiple Cycles	Multiple Cycles

- A one-record file is stored sequentially and therefore is subject to all the restrictions that apply to consecutive files. Unlike other consecutive files, however, a one-record file communicates a significant piece of information to an APPX input process; a process can expect at most one record. Based upon this information, the process simplifies development and maintenance requirements. APPX automatically checks for the existence of a record in the file. If there is no record, APPX places the user in add mode. If there is a record in the file, APPX places the user in change mode.
- A parameters file is an example of a one-record file. Use of a one-record file to hold specifications that are referenced by more than one process in an application avoids unnecessary duplication, and simplifies design and maintenance.
- Consecutive files organize records according to the sequence in which they are added to a file; each new record is added to the end of a file. The system retrieves records based upon their relative position in a sequence, or record number. A consecutive file does not contain keys, so its records cannot be retrieved by key path. To select a record based upon field value, APPX must process all records, ignoring those that do not satisfy selection requirements.
- Because file processing is generally based on the value contained in a specific field (such as the last name of a customer or the number assigned to a vendor), APPX must use a query process to *resequence* consecutive records by the order they should be processed in. A query maintains the generated sequence in a *pointer file*. For example, the pointer file for an Orders file that is sorted alphabetically by customer includes the following record sequence for the customers (Anderson's record is number 8, Jones' is 4, Smith's is 1, and Williams' record is 22). To print an alphabetically sorted list of outstanding orders, the system consults the pointer file, and the records are processed in the following order: 8 (Anderson), 4 (Jones), 1 (Smith), and 22 (Williams). Refer to the Query Files section of *Chapter 3-2,* Chapter 3-9: Query Processes, and the Child Use Query Option section of *Chapter 3-13* for more information about pointer files in a query process.
- On most systems, APPX retrieves records from consecutive files faster than from indexed files.
- A consecutive file can contain an unlimited number of records, subject only to operating system limitations and available storage space. You cannot, however, delete specific records in this type of file. To eliminate specific records, you must first transfer all records that you want retained to another file, and then scratch the original file.
- For an indexed file, the system retrieves records based on the content of specified fields, or primary and alternate keys. APPX requires one primary key field for each file, and permits up to 16 alternate keys. These key fields are identified in the field specifications screens in the data dictionary.
- Key fields, when used by the system to retrieve records, are referred to as *key paths*. This method of record retrieval is inherently slower than the method employed for consecutive files, but is more flexible and allows you to perform complicated operations more efficiently. Performance is

enhanced when records are retrieved by primary (as opposed to alternate) key paths, although indexed file retrieval is still slower than consecutive file retrieval.

- An indexed file can contain an unlimited number of records, subject only to the operating system's limitations, the restrictions that are imposed by the length of a key field, and the available storage space. Unlike records in a consecutive or one-record file, records in indexed files can be deleted.

Temporary versus Permanent Files

For every disk-based or memory-based file you define in the APPX dictionary, you must designate whether the file is temporary or permanent. This distinction affects how and/or when a file is created, opened, and scratched (see *Table 2-1-5*).

- Temporary files provide working space that is unique to a task and automatically managed by APPX during processing. At its maximum, the existence of a temporary file is limited to a single APPX session. If, based upon a file's specifications, it is not shared between processes, the existence of a file may be considerably less than the length of a session. Temporary files are database-independent; there is only one copy of a file for all databases that access it. Temporary files are never shared between users.
- Permanent files, on the other hand, are always shareable between processes, and are managed by APPX's database management. APPX never automatically scratches permanent files. If a file is disk-based, each database has its own copy of it; permanent, disk-based files are database-dependent. They are always shared between processes, regardless of file specifications.

Permanent files are created by APPX's database management. See the *System Management Manual* for more information.

Table 2-1-5. A Comparison between Temporary and Permanent Files

Attribute	Permanent File	Temporary File
Shared Between databases	No, except with statements	Yes
Shared between processes	Always	Dependent upon file specs
Shared between users	Yes	No
Outlasts a single session	Yes	No
Requires Database Management specifications	Yes	Yes, if disk-based. No, if memory-based

Temporary files are created when a process that references the file begins. By using a CREATE statement, you exercise explicit control over the time when a temporary file is created. For example, you can define a CREATE statement in a *Start of Process* event point to force APPX to create the file at the beginning of the process.

APPX never automatically opens temporary or permanent files at the beginning of a process. To enhance perceived performance (and quite often actual performance), APPX distributes file openings throughout a process. A file is physically opened when APPX first needs to access the file. By using an OPEN statement, you exercise explicit control over the time when a file is opened. For example, you can define an OPEN statement in a *Start of Process* event point to force APPX to open a file at the beginning of the process.

Any process can be invoked as DETACHED or related, or as a SUBPROCESS. When working with temporary files, it is important to know how the file type and the process invocation are related and interact. See *Chapter 1-5* for information about Interprocess Communication in APPX.

If APPX does not know for sure that a file exists, it attempts to open the file during the execution of a process. *File Does Not Exist* error messages can therefore occur at any time during process execution, unless all files are explicitly opened at the beginning of the process. The obvious trade-off is decreased performance.

Record Protection

APPX allows you to restrict the use of certain records by designating record protection fields at the file level. These fields are available when you select the RECORD PROTECTION option from the Files/Fields screenl.

- The ***Delete Protection*** field governs whether or not a record can be deleted by a user within an input process. This field can be used to prevent the creation of orphan records by prohibiting the deletion of records referenced by other records anywhere in a database. You must explicitly maintain the value in this field.
- The ***Key Protection*** field governs whether or not the *primary key* to the record can be modified by a user within an input process. This field can be used to maintain the integrity of a database by prohibiting the modification of keys referenced by other records anywhere in the database. You must explicitly maintain the value in this field.
- The ***Record Access*** field contains a designer-designated security code that automatically restricts retrieval of a record (and by implication any field on a record) to those users with security profiles that include this code. If a user does not have access to a record, APPX responds as if the record is not on file. The value in this field must be maintained by the designer.

CHAPTER 2-2

Data Dictionary Concepts

Processing the Data Dictionary

Processing the dictionary analyzes its specifications and translates them into a format that is more directly usable by APPX during process execution. It does not generate machine or object code.

As diagrammed in *Figure 2-2-2*, processing the data dictionary creates an *element* record from domain and/or field specifications, searches for logic errors (invalid key definitions or illegal record lengths, for example), constructs a default value record for each file, computes field lengths, establishes the starting position for each field within the record, and determines whether or not changes affect the structure of a file.

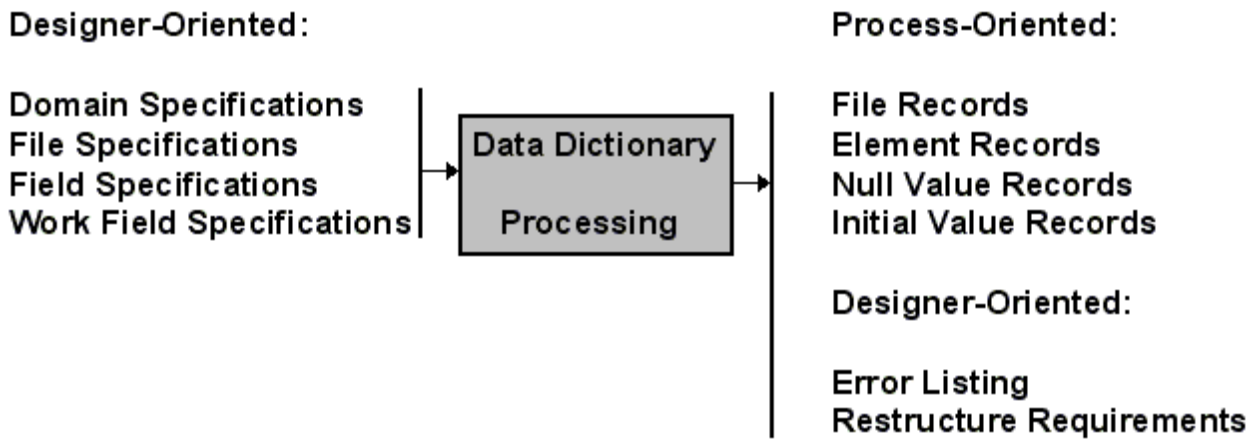


Figure 2-2-2. Components of Data Dictionary Processing

The data dictionary for each new application must be processed before its domains, files, fields, and work fields are referenced by processes. It is automatically processed any time you change the specifications for a domain, file, field, or work field. If the data dictionary requires processing, the processes that reference it cannot be executed. Normally, processing occurs automatically each time you exit a process that changed an element in the data dictionary. However, the *Process Data Dictionary* option on the Database Management menu allows you to initiate data dictionary processing yourself. This option is used most often to find and correct errors in your data dictionary.

Tracking Processing Requirements

The data dictionary input processes track specific types of modifications that, when they occur, require processing of the dictionary (or selected parts of it). They do so by maintaining various date/time stamps, and by comparing the date/time of unprocessed specifications with the corresponding processed formats.

By tracking those situations that require processing of a dictionary or dictionary component, APPX can:

Enhance performance
By limiting the magnitude of data dictionary processing.

Eliminate error
By removing the possibility that other components within APPX will access obsolete data specifications.

In general, the situations that give rise to a processing requirement are the following:

- Domains require processing after they are added to the dictionary and whenever their specifications are modified, including modification of an associated validation or token table. If a domain requires processing, all fields that reference it also require processing. If a domain is deleted from the dictionary, any remaining fields that reference the domain must be modified and processed.
- Files require processing after they are added to the dictionary, and whenever their specifications are modified. In addition, a file must be processed whenever fields are added or deleted, or a field specification is modified. If a file is deleted from the dictionary, its fields and element records are automatically deleted. The dictionary does not have to be processed.
- Fields are processed based upon a requirement maintained at the file level. If the specifications for any field change, the file and all its fields must be processed. If one or more fields are deleted from the dictionary, the file must be processed.
- Work fields must be processed whenever a work field is added or deleted, and whenever a specification for any work field is modified. For this purpose, work fields are considered to reside in a single work fields file, with the processing requirement maintained at the file *and* field level. If any work field requires processing, all work fields are processed. To improve performance, however, individual work fields that do not require processing on their own merit are simply verified against the current element record.

Summary of a Data Dictionary Process

Components of the data dictionary are only processed *when necessary*. Refer to the Tracking Processing Requirements section later in this chapter for additional information.

For those domains, files, fields, and work fields that *do* need processing, the following actions occur:

- Domains are processed before any fields in the dictionary.
- Fields are converted (in sequence number order) to a format more directly executable by processes within APPX. Attributes defined at the domain level are incorporated into the new format, and all overrides are resolved. Storage lengths for fields are calculated. The resulting processed fields are called *elements.*
- Files are processed to reflect changes to file specifications or to field specifications that affect a processed file, such as the various primary and alternate key specifications.
- Default value specifications are converted to an internal format and stored in an initial value record for each file. A null value record for each file is also created.
- Data is subjected to a limited validation. This validation is required because specifications may have been imported (either through the migration utility or a user-designed process), thus bypassing the error checks incorporated in APPX's data dictionary input processes.
- If modifications result in structural changes, a date/time field is updated, indicating the point where the structure changed.

A list of all the errors APPX encounters is generated at the end.

Relationships of the Processed Dictionary to Other APPX Components

Because the APPX data dictionary is active, various components within APPX depend upon an appropriately processed data dictionary. The components that are affected by processing are the following:

- The ***process compiler*** cannot generate an executable module if any of the data dictionary definitions that it references require processing.
- Before loading an executable module, the ***process loader*** performs a verification to ensure that the module is current. A new executable module must be created if any of the data dictionary definitions that the module references require processing.
- The ***database management*** utilities, including *Create Files* and *Restructure Files,* can only be used with data dictionary components that do not require processing. They will not, for example, create a file if the file or any of its fields require processing.
- Throughout the ***application design facility***, whenever a field or file is referenced, APPX verifies its processing status. Some examples are the image editor, ILF editor, and specifications entry screens.

Application Integration Issues

APPX applications operate as part of an integrated system of applications or stand alone, depending upon the specific requirements of each installation and application. You *design* integrated applications as if they will always be installed as a system.

When you first process a dictionary that includes one or more fields which reference domains in outside applications, your system *must* include the applications that contain the domains referenced. During processing, the field stores an internal copy of the specifications entered for the domain it references. Validation and token tables are also copied at this time.

After processing the data dictionary, the application can be installed as a stand-alone. When this stand-alone application is executed, the internal copy of the referenced domain's attributes are used, not the attributes in the referenced application. You can process a stand-alone application without reinstalling the referenced applications; APPX assumes the referencing fields do not require processing.

If the domain that is referenced by a field changes, and it is necessary to incorporate those changes in the field, you must reinstall the referenced domain's application. Processing the data dictionary then updates the internal copy of the specifications for that field.

Integrity

Integrity represents the accuracy, consistency, and completeness of a database. *Referential integrity* refers to the condition whereby all references between interconnected items of information in a database remain sound. It ensures that customer number 200 is not deleted when there are existing sales orders that reference customer 200, for example.

APPX includes features that, when properly implemented, ensure the integrity of a database. For example, the dictionary can be used to:

Prevent orphan records

By providing *Delete Protect* and *Key Protect* options.

- ***Prohibit duplication of primary keys***.

Prevent referencing nonexistent data

By providing the basis for online data validation throughout APPX and all APPX-based applications.

Other APPX components work with the dictionary to ensure database integrity. The process facility, for example, includes specifications that tell APPX to automatically update fields in a child process if equivalent fields are updated in its parent process. Other specifications require APPX to automatically delete records in a child process based upon deletion of related records in its parent process.

Database Security

Security refers to the function that protects programs and data from unauthorized access and/or modification. Within the APPX environment, security is optional and can be designed, implemented, or modified at any time during the life of an application.

Security in the Data Dictionary

Within the APPX data dictionary, you use field and file security specifications to control which users can view and modify data presented in images in applications. *Figure 2-2-3* illustrates the effect of each of these security definitions within a sample file containing six records of 13 fields each.

To secure fields in images in input and output processes, you assign two separate security codes in the dictionary:

Read rights

Govern whether or not a user can see the contents of a field as defined in an image, either input (transaction entry, maintenance, or inquiry screens, for example) or output (journals, lists, or reports). If a user does not have read rights to a secured field, the contents of the field are invisible and not modifiable.

Modify rights

Govern whether or not a user can change the contents of a field. A user with modify rights is automatically granted read rights, regardless of the value in the read security specification. If the user does not have modify rights but does have read rights, the contents of the field are visible but not modifiable.

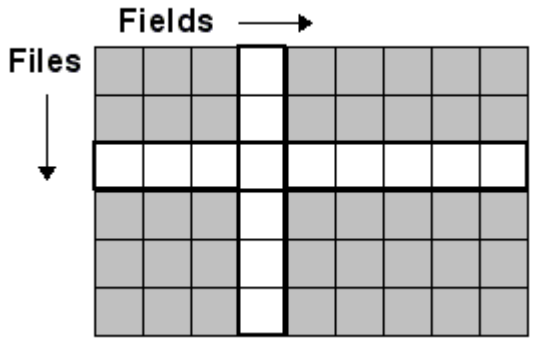


Figure 2-2-3. Field- and File-Level Security in the Data Dictionary

The APPX dictionary includes read and modify security specifications for domains, files, fields, and work fields, and records.

Domain

Specifications function as defaults for all fields and work fields that reference them.

File

Specifications control read and write access to the file.

Field security

Specifications can be defined if the domain specification is blank. If security specifications are blank at all levels, access to the field is unrestricted.

Work field security

Specifications can be defined if the domain specification is blank. If security specifications are blank at all levels, access to the field is unrestricted.

Record security

Is discussed in the Record Protection section of *Chapter 2-1*.

You define security in the data dictionary by selecting the TOOLBOX option from the Domains, Files/Fields, Fields, and Work Fields screens, which accesses the Toolbox overlay shown in *Figure 2-2-4*. From this overlay, you can select the SECURITY option. Upon completion, you are returned to the screen from which you selected the TOOLBOX option.

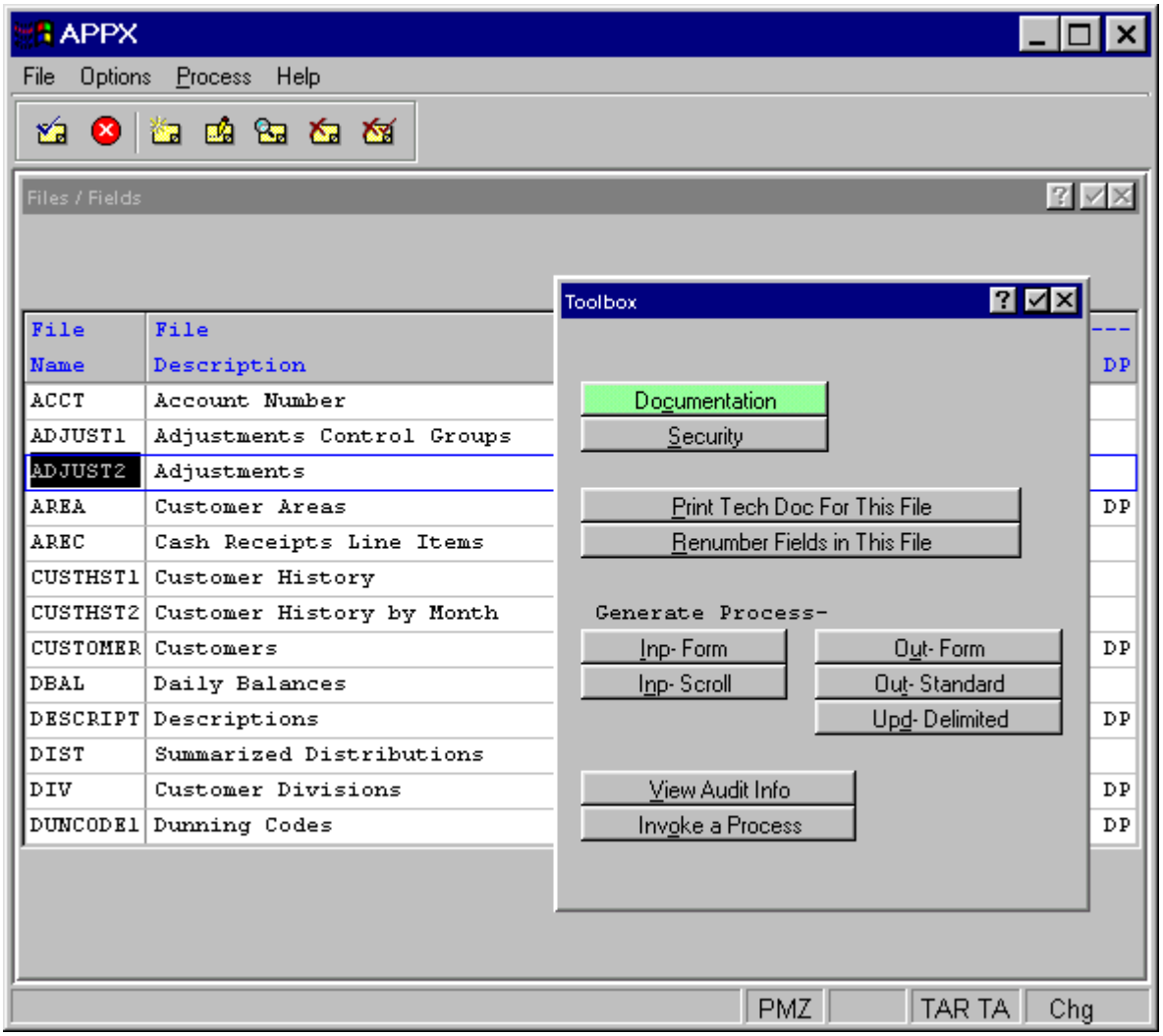


Figure 2-2-4. Toolbox Overlay

APPX does not permit you to secure a specific field or group of fields on a specific record. Field-level security applies equally to all records in a file, and file-level security applies to every field in a file.

Field security specifications, unlike the file security specifications, do not restrict access to fields from the application design facility. For example, a user with design rights could define statements to modify values in secured fields, or to load the contents of secured fields into unsecured work fields. The contents of a secured file, however, cannot be accessed with statements.

Assigning Security Codes

APPX's security facility was designed to provide maximum flexibility. As a result, there are no incorrect methods of assigning security codes. There are, however, methodologies that are more efficient or easier to establish and maintain than others.

Documentation

APPX provides screens to document each design component in the data dictionary as an integral part of its definition. When you create a data dictionary, you define:

User documentation
For domains, files, fields, and work fields. The text you enter on these screens is the basis for APPX's online help facility. (User documentation can also be directly maintained using the Documentation Management option from the APPX Main Menu, as described in *Chapter 4* of the *System Management Manual*.)

Technical documentation
For domains, files, fields, and work fields. The text you enter in these screens is the basis for APPX's hard-copy technical documentation.

Each user and technical documentation screen includes specifications that identify the domain, file, field, or work field you are documenting, and provides 18 lines of up to 78 characters to explain the component. The EXTERNAL COPY feature lets you access other documentation records and copy them into User and Technical Documentation screens.

When you enter text on any of the documentation screens, APPX provides a number of editing functions, including word-wrap, text insertion, text redisplay, and end paragraph control. Refer to the *User Manual* for a discussion of text editing capabilities.

Accessing the Data Dictionary Documentation Screens

Documentation is an integral part of the APPX data dictionary. You access:

User and technical documentation
For any domain, file, field, or work field in the dictionary by selecting Option 1 on a component's Toolbox screen. Option 1 accesses the Documentation overlay, from which you can access the User and Technical Documentation screens.

- In addition, if you are set up as an application designer or documentation designer in system administration, you can modify the user help text from the *Help* facility by selecting change (*CHG*) mode.

The external copy of a documentation record
By selecting an option on a component's User or Technical Documentation screens.

To access the common Documentation overlay, shown in *Figure 2-2-5*, you select TOOLBOX from the Domains, Files/Fields, Fields, and Work Fields screens and then DOCUMENTATION from the Toolbox overlay. Upon completion, you are returned to the screen from which you selected the option.

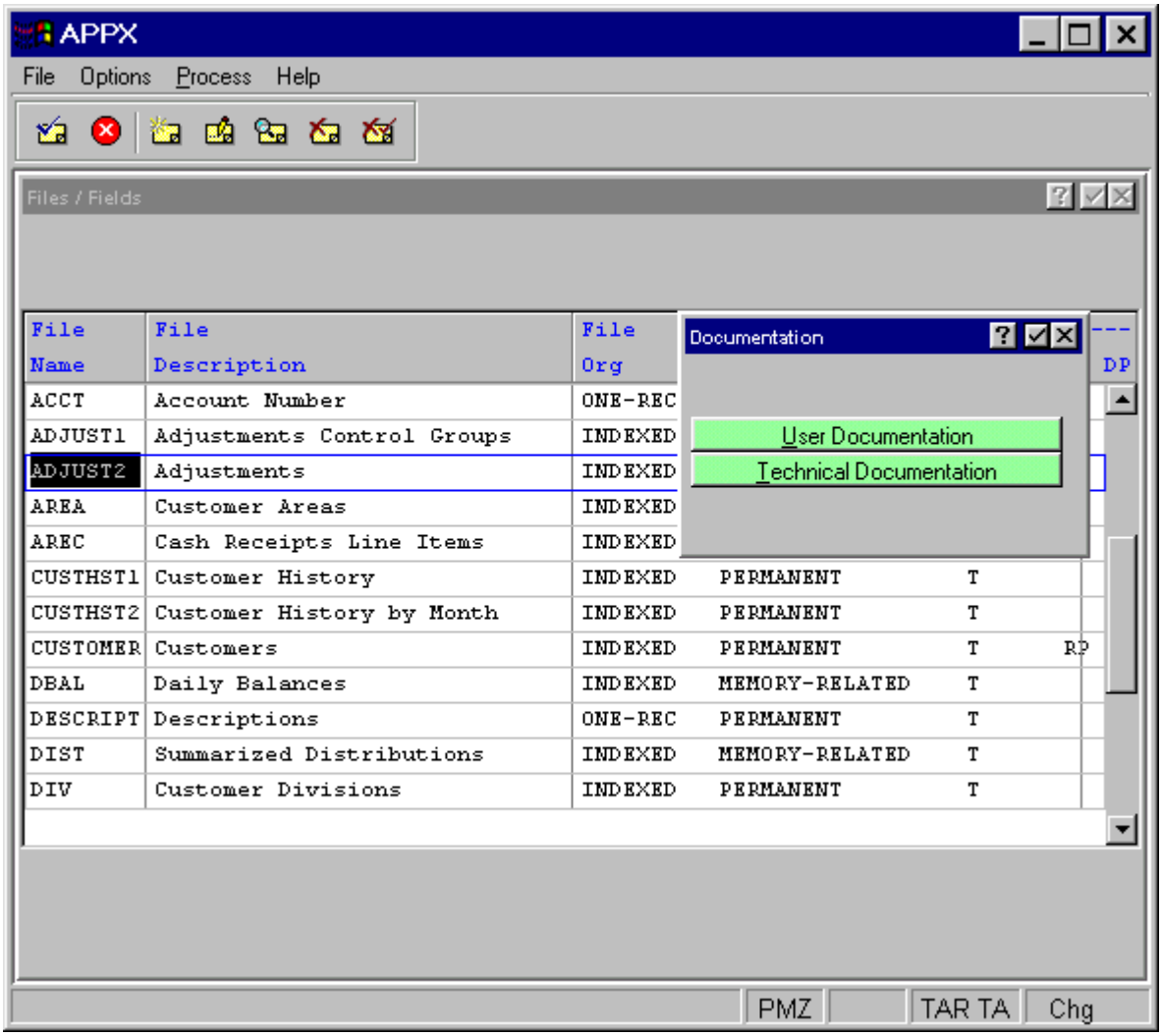


Figure 2-2-5. Documentation Overlay

Figure 2-2-6 shows the User Documentation screen. Upon completion, you are returned to the Documentation overlay, where you can access the Technical Documentation screen.

User Documentation screen

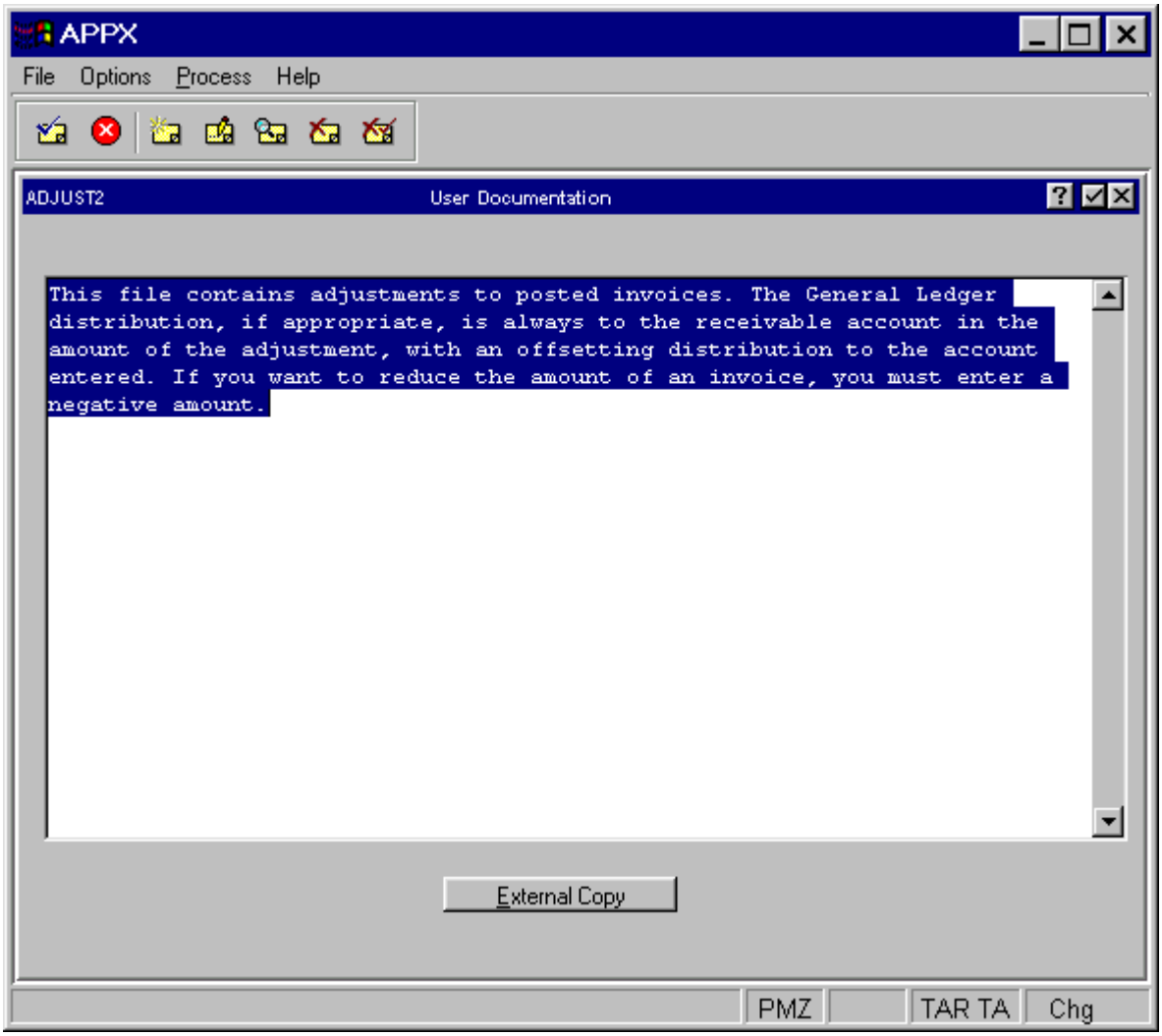


Figure 2-2-6

Integration with Database Management and System Administration

APPX's database management and system administration facilities are both integrated with the data dictionary. Database management uses many of the dictionary's specifications to create and manage APPX files. System administration establishes the security codes and profiles referenced by the dictionary.

Database Management and the Data Dictionary

Some of the database management facility is system- or platform-specific, while application design is system-independent. From the integration of the two facilities, you can develop an APPX application without regard to the target platform, yet operate the application to take maximum advantage of the system it operates on.

The database management facility is most tightly integrated with those dictionary components that are database-dependent. The term *database-dependent,* or database-related, refers to what is unique to a specific database (as defined by database management).

In the APPX dictionary, permanent disk-based files are database-dependent. Each database that accesses a permanent, disk-based file actually accesses a separate copy of the file. Unstored files, temporary files, work fields, and memory files are database-independent. In this case, APPX maintains a single file that is accessed by all databases.

For database-dependent files, a number of dictionary specifications provide information and defaults for, or impose restrictions on, the file creation utility in database management. These specifications include *File Name, Organization, Variable Length?, Compress Records?,* and *Create by Database Management?* For disk-based temporary files (database-independent), database management refers to the dictionary specifications for file size information.

The dictionary is also integrated with database management's restructure utility. APPX uses this utility to automatically restructure files to reflect such changes as the length of a field or a change in storage type. The flag that indicates a file requires restructuring is maintained by the data dictionary when it is compiled or recompiled. Refer to the *System Management Manual* for a discussion of the restructure utility.

System Administration and the Data Dictionary

The principal interface between the system administration facility and the data dictionary is security. Security profiles are established in system administration, and serve to restrict access to database-dependent components of APPX, such as the permanent, disk-based files in the dictionary. These profiles are identified by a profile name and a database ID. They are therefore linked to a specific database, permitting you to assign different restrictions to test, development, and production databases, for example.

As indicated in the *Database Security* section, you implement security in the dictionary by assigning security codes to domains, files, fields, and work fields, which act to restrict users' rights based upon their system administration security profiles.

CHAPTER 2-3

Domains

Introduction to Domains

Domains are generic definitions, or patterns, for fields and work fields that share a set of common attributes. These attributes include such things as the physical space a field occupies, the set of values it contains, and the way the field is presented to a user.

APPX recognizes the following nine types of domains, with variations in the screen specifications presented once domain type has been established on the Domains screen.

- *Alpha* domains are patterns for fields comprised of letters, numbers, and special characters (-, #, or &, for example), where the primary characteristic is field length.
- *Format* domains are like alpha domains except that an edit mask is the primary characteristic.
- *System* domains are patterns for fields that contain data provided by the system, often referred to as audit fields.
- *Text* domains are patterns for fields comprised of blocks of text.
- *Token* domains are like alpha domains except they have specific values that are stored as corresponding numbers.
- *Numeric* domains are patterns for fields restricted to number, decimal (.), and sign (-) entries.
- *Date* domains are patterns for fields that contain dates and/or times, from a hundredth of a second through a century.
- *Logic* domains are patterns for fields with possible values of yes ☑ or no ☐, or ? to represent an unknown value.
- *Same As* domains are patterns for fields that share the same characteristics as a domain in another application in your database.

Domains Screen

The Domains screen contains general attributes that are common to all domains. The *Type* you enter on this screen for a domain governs the subsequent screens that APPX displays to further define the domain. Domain specifications establish values for all fields that reference the domain and cannot be overridden. Therefore, it may be desirable to leave some attributes blank (such as descriptive name or column headings) to allow for making individual entries at the field level.

You access the Domains screen, shown in *Figure 2-3-1*, by selecting DOMAINS from the Database Management menu overlay.

Domains screen

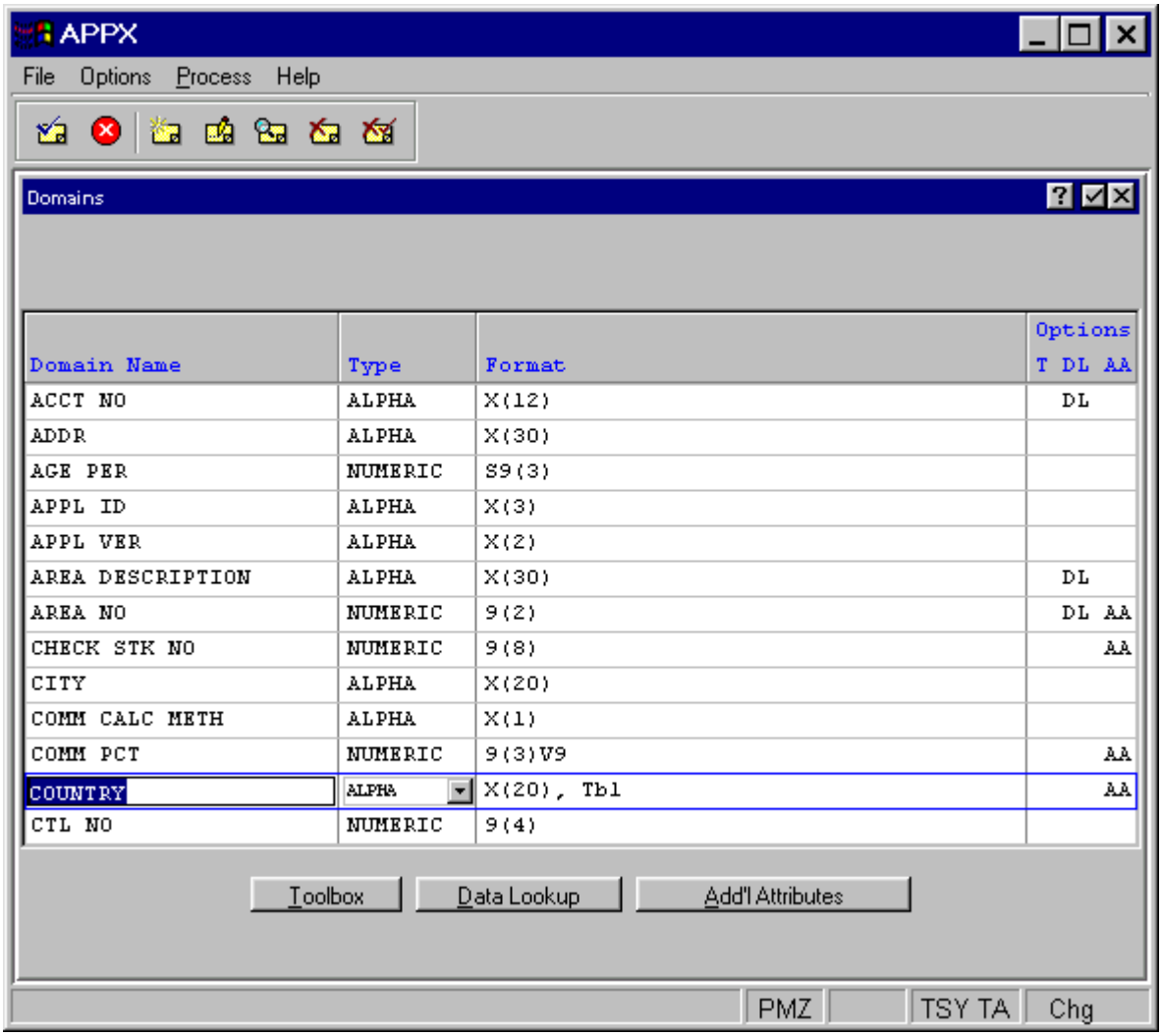


Figure 2-3-1

The Domains screen contains the following fields:

Domain Name

Uniquely identifies the domain within an APPX application. Fields assume the attributes of a domain by referencing the domain's *Domain Name*. (22 uppercase alphanumeric characters, restricted to numbers, letters, and spaces).

- The name may represent a specific field that references the domain, such as an Account Number domain, or a category of fields, such as an Amount domain that is referenced by a number of different fields, including AP Invoice Amount, Sales, AR Invoice Amount, Total Purchases, and Order Amount.

Type

Categorizes the domain according to its specific content or function in APPX. The specifications presented on subsequent screen overlays depend on the *Type* entered here, although most types share some common specifications. The domain *Type* options are:

- ***Alpha***, for field patterns that can contain numbers, letters, and special characters (such as -, #, or &).
- ***Format***, for field patterns where edit mask is the primary characteristic.
- ***System***, for field patterns that contain data provided by the system, often referred to as audit fields.
- ***Text***, for field patterns comprised of blocks of text.
- ***Token***, for field patterns with specific values that are stored as corresponding numbers.
- ***Numeric***, for field patterns restricted to number, decimal (.), and sign (-) entries.
- ***Date***, for field patterns that contain dates and/or times, from a hundredth of a second through a century.
- ***Logic***, for field patterns with possible values of yes ☑ or no ☐, or ? to represent an unknown value.
- ***Same As***, for field patterns that share the same characteristics as a domain in another application in your database.
- If you enter a partial value, APPX selects the option with the next highest value. For example, an entry of *ALPHA* or *A* is equivalent to entering *ALPHA*. An entry of *C* references the next highest value: *DATE*. Each option is discussed in detail in the corresponding section.

Format

Displays a symbolic representation of the domain for ease of reference. It is generated by APPX according to the detail specifications entered for the domain and cannot be edited.

Options

Are indicators that display whenever documentation (T), data lookup (DL), or additional attributes (AA) have been defined for this domain.

Three options are available from the Domains screen as described in the corresponding sections:

- Toolbox
- Data Lookup
- Additional Attributes

Domains Toolbox Option

The toolbox overlay appears on the screen when you invoke the TOOLBOX option from the Domains screen. The overlay contains a number of features such as DOCUMENTATION MANAGEMENT and SECURITY MANAGEMENT that will assist you when creating domains. The options listed in the Toolbox overlay are described briefly below. See *Chapter 3-3, Process Definition* for a more detailed explanation of the these options.

Documentation

When you select the DOCUMENTATION option from the Toolbox overlay, you will first see the Documentation overlay which allows you to select between user or technical documentation. Once you have entered your documentation, you are returned to the Domains screen.

Security

The SECURITY option allows you to identify the APPX security code to which a user must have rights in order to access the domain. The security code must be four numeric characters. If blank, there are no restrictions.

Fields/Domains Cross Reference

The Fields/Domains cross-reference option enables you to determine the location and frequency of a selected domain, file, field, application ID, process or ILF statement within a specific database.

Change ALL References

This feature tells APPX to find and replace all references to a certain domain in an application.

View Audit Info

This feature allows you to see who created the currently selected inquiry process and who changed the currently selected inquiry process most recently. It also gives the date of creation and the date it was last changed. Note that this option (`OPTION 98`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Invoke a Process

This feature is available to facilitate testing. It allows you to invoke any process; the currently selected inquiry process using the Test Database defined in system administration is the default. Note that this option (`OPTION 99`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Domains Data Lookup

The DATA LOOKUP option from the Domains screen accesses the Data Lookup overlay, shown in *Figure 2-3-2*, where you can identify an application, file, and key field that can be accessed by using the value in any field referencing this domain. The key field must be a primary or alternate key to the file. This entry establishes the *Data Lookup* specifications for all fields that reference the domain. Upon completion, you are returned to the Domains screen.

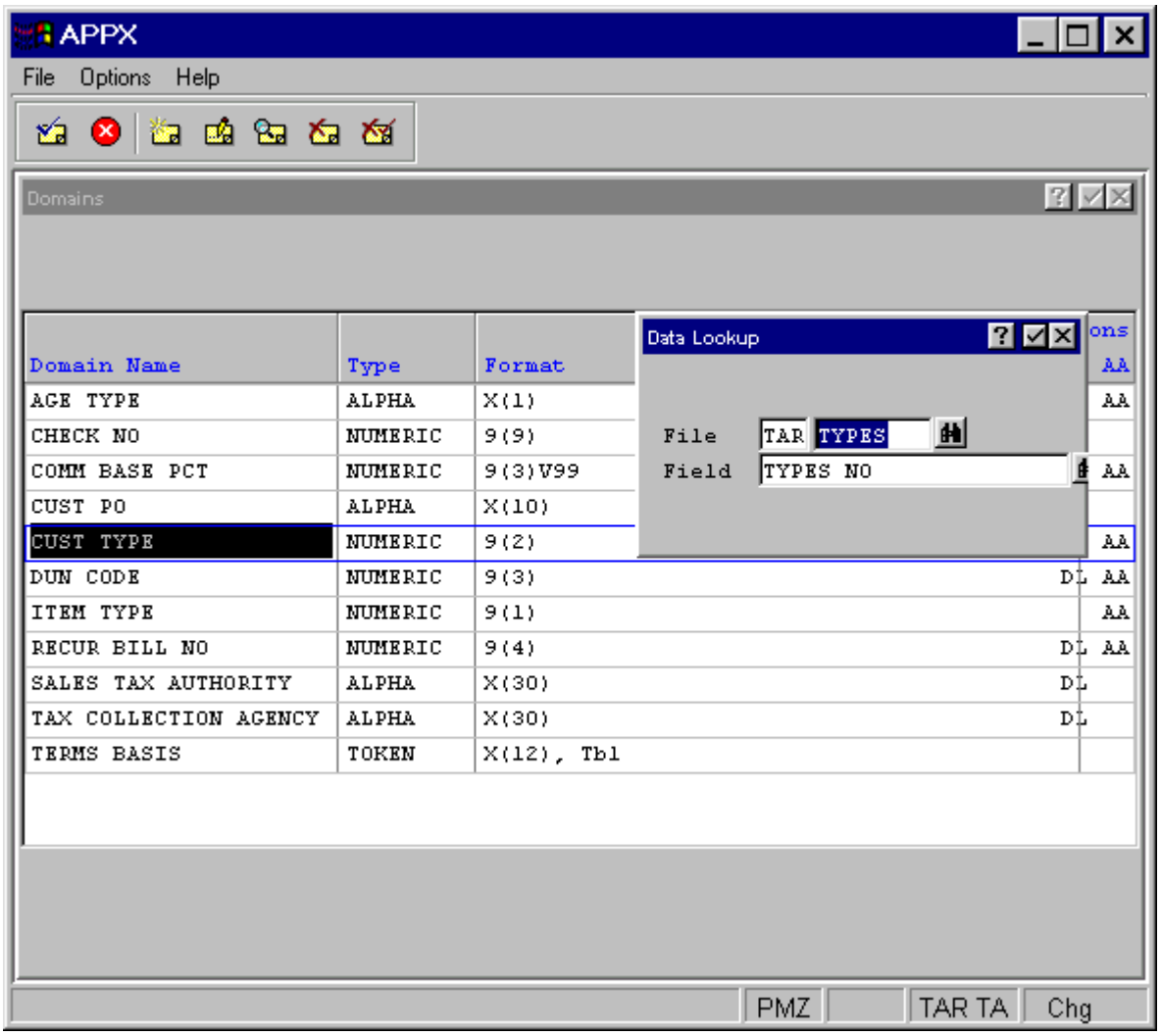


Figure 2-3-2. Domains Data Lookup Overlay

Domains Additional Attributes

The ADDITIONAL ATTRIBUTES option from the Domains screen accesses an Additional Attributes overlay, from which you can enter additional parameters for the domain. The specific overlay varies by domain type. Each overlay is presented later in this chapter according to the respective type of domain. Upon completion, you are returned to the Domains screen.

Common Domain Specifications

When you select ENTER from the Domains screen title bar, the Domains specifications overlay displays. The following two entries, *Descriptive* and *Column Hdgs*, are common to the specifications overlay for all types of domains other than *Same As* domains:

- ***Descriptive***, if entered, establishes the *Descriptive* for all fields referencing the domain. However, since domain attributes cannot be overridden, you may wish to leave *Descriptive* blank here to allow for individual entry at the field level. Because *Descriptive* is longer and can be more flexibly formatted, it is less cryptic and easier to understand than *Field Name*. The system uses *Descriptive* when communicating with users wherever possible. It is used, for example, when the value of a field displays to a user on a standard image, such as the *Select Access Path* display or an APPX-generated subheading.
- ***Column Hdgs***, if entered, establishes up to two lines of column heading text for all fields referencing this domain. However, since domain attributes cannot be overridden, you may wish to leave *Column Hdgs* blank here to allow for individual entry at the field level. For cosmetic reasons, the length of a heading should not exceed the length of the field itself. The APPX output process uses column heading text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. Refer to Output Process Specifications, Output Standard Column Heading Image, and Output Frame Specifications in *Chapter 3-7* for more information on automatically generated column headings and frame classes in output functions.

Alpha Domains

An alpha domain can contain letters, numbers, and special characters, such as *#, $,* and *&*. Examples include names, addresses, descriptions, and certain account numbers or serial numbers.

The Alpha Domains overlay, shown in *Figure 2-3-3*, allows you to add Alpha specific attributes to those already defined for domains. The alpha domain overlay displays after you complete the entries on the domains screen if the domain *Type* you select is *ALPHA*.

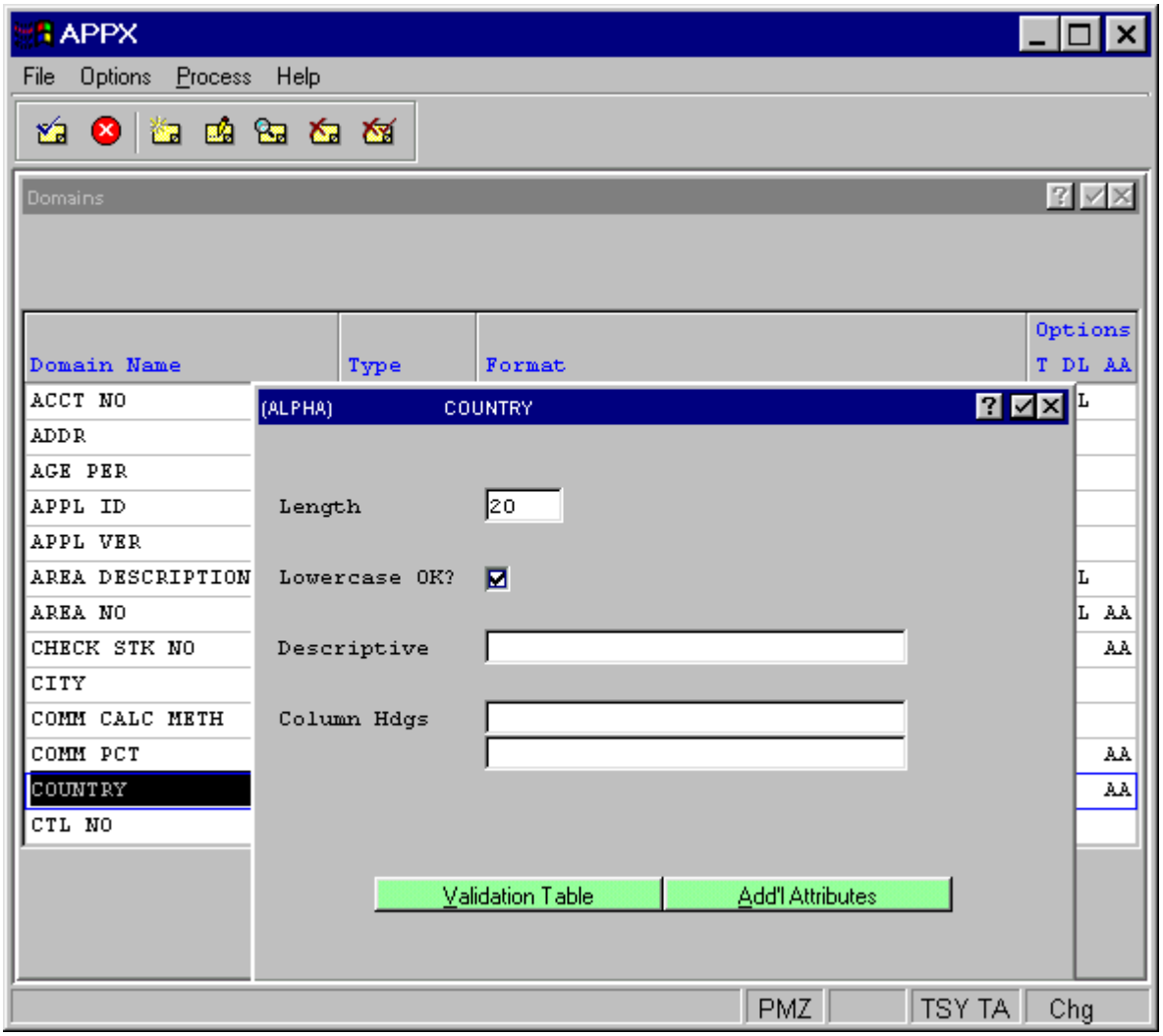


Figure 2-3-3. Alpha Domains Overlay

The Alpha Domains overlay contains the following fields:

Length

Designates the number of character positions, or length, of this domain.

Lowercase OK?

Indicates whether or not a modifiable item on an interactive screen can be entered in upper- and lowercase letters. Click the checkbox to toggle between "yes" ☑ and "no" ☐ values. If "no" ☐, the entry is stored and displayed exclusively as uppercase, regardless of how a user enters the information. This entry does not affect ILF statements. A lowercase value established with a statement displays as lowercase, even if this specification prohibits lowercase entries. The default is "yes" ☑.

Descriptive

Provides a longer, more easily understood alternative to *Domain Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Domain Name* is used. Because *Descriptive* is less cryptic and can be more flexibly formatted than *Domain Name*, it is easier to understand. As a result, the system uses *Descriptive* when communicating with users. *Descriptive* is used, for example, when the value of a domain displays to a user on a standard image, such as the *Select Access Path* display or an APPX-generated subheading.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

For Alpha domain type, the following options are available as described in the corresponding sections:

- Validation Table
- Additional Attributes

Alpha Domain Validation Table

The VALIDATION TABLE option from the Alpha Domains overlay accesses the Validation Table overlay, shown in *Figure 2-3-4*. A validation table provides a more powerful alternative to the use of range checks for data validation when there is a known set of values. Through the list ▼ feature, it provides the user with visibility of all the possible field values as well as the ability to select the desired value.

If defined for the domain, it establishes the validation table for all fields that reference the domain and cannot be modified at the field level. Conversely, if not defined for the domain, a validation table cannot be defined for any field that references the domain.

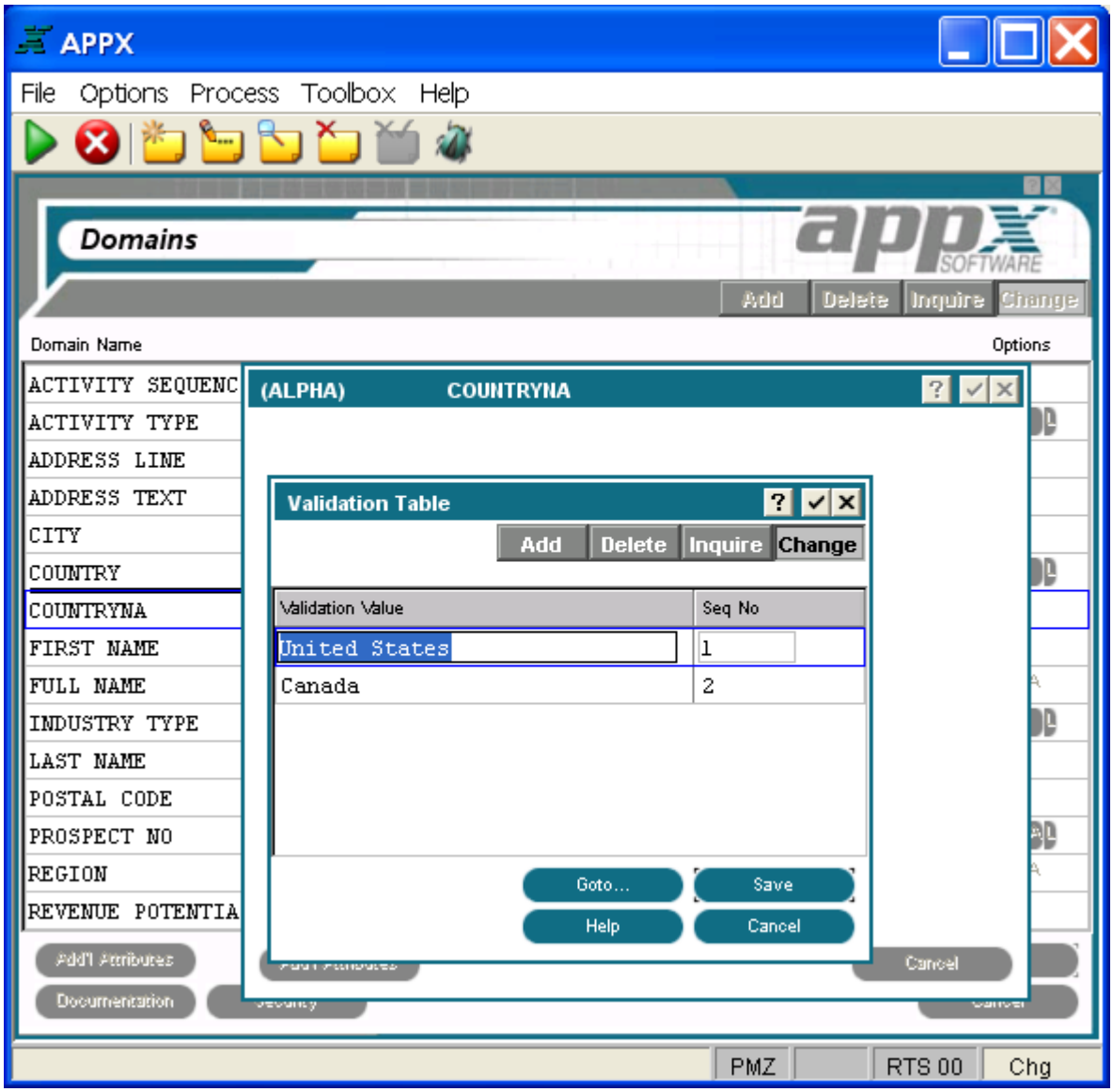


Figure 2-3-4. Alpha Domain Validation Table Overlay

You can establish a validation table for any Alpha domain in the data dictionary. The entries in the table are case-sensitive. If you enter a validation value in all capital letters, an application user must enter the value in all capital letters.

Validation table operations are database-independent. They access a designer-defined table in application design, not data contained in a file in a database, for the range of possible values. As a result, validation does not vary from database to database. It is automatically available to, and exactly the same for, every database that is associated with an application. All validation values are available for review and selection by the end user through the list feature.

The Validation Table overlay contains the following fields:

Validation Value

Provides a valid field value. It is case-sensitive and is used for online review, selection, and validation. Note that validation values newly added to the data dictionary are not available to a current APPX user until the user exits and re-enters APPX.

Seq No

Provides an identifying number for the associated validation value. It is defaulted by APPX and can be overridden, but it must be unique within a table.. It determines the order in which validation values are presented, both for selection by the user and for editing in application design. APPX assigns the first validation value a number of *1*, with additional values assigned a number that is equal to the highest existing sequence number plus *1*. APPX does not require that sequence numbers be contiguous within a table.

Alpha Domain Additional Attributes

From the Alpha Domains overlay, you have an ADDITIONAL ATTRIBUTES option. The Alpha domain Additional Attributes overlay is shown in *Figure 2-3-5*.

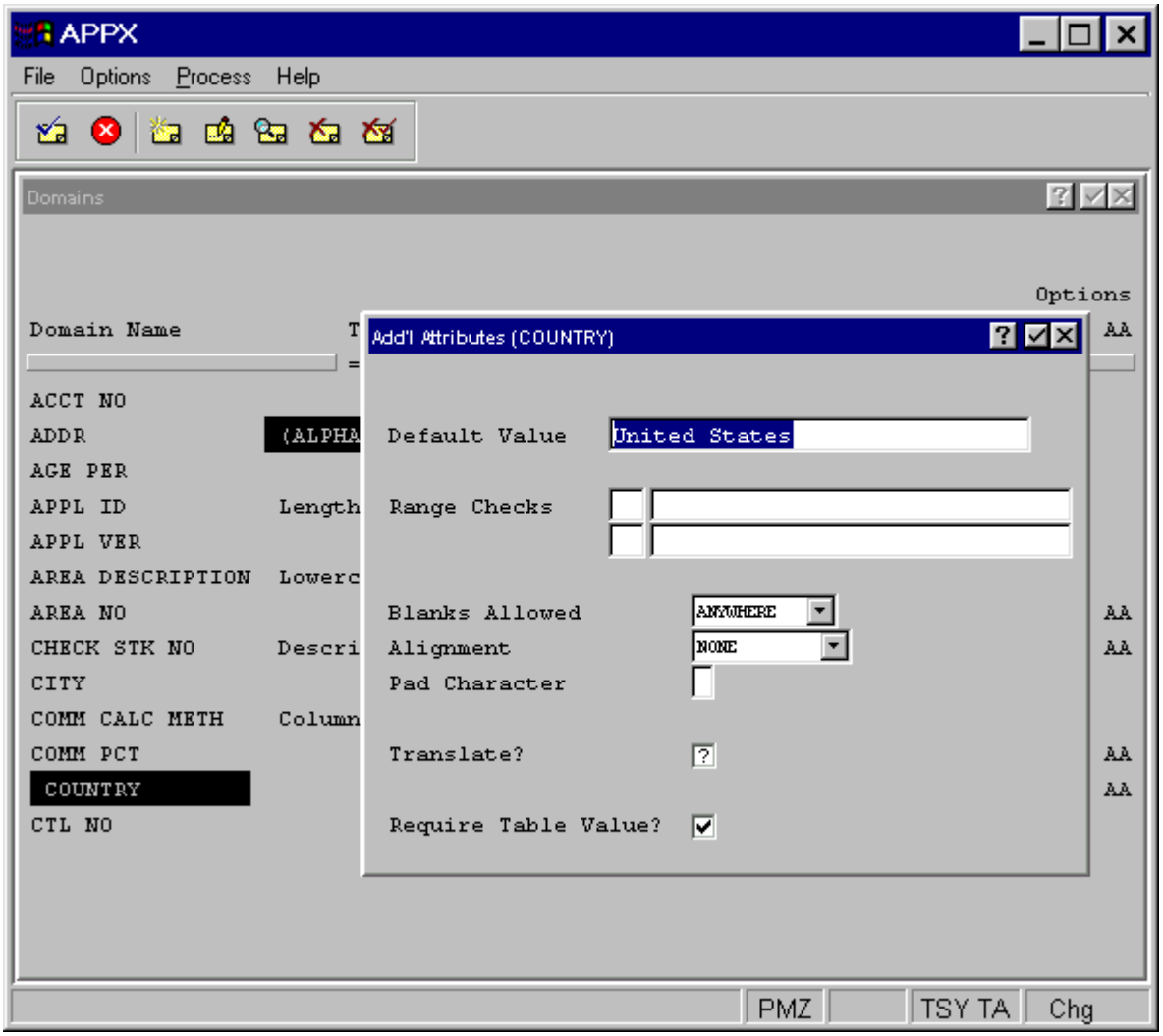


Figure 2-3-5. Alpha Domain Additional Attributes Overlay

The Alpha Domain Additional Attributes Overlay contains the following fields:

Default Value

Establishes a modifiable starting value for all image items whenever they are first displayed in add mode in an input image, unless modified by a user. If blank, there is no starting value displayed on the image.

Range Checks

Defines a series of allowable values that APPX uses for online data validation. The first component represents the relation. The options are:

- *EQ* (equal to)
- *NE* (not equal to)
- *GT* (greater than)
- *LT* (less than)

- ***GE*** (greater than or equal to)
- ***LE*** (less than or equal to)
- ***IN*** (includes)

The second component represents the value assigned to the relation. If you enter two relations, APPX infers an AND relationship.

Blanks Allowed

Indicates whether or not blanks are permitted and, if so, where they can be placed. The *Blanks Allowed* options are:

- ***NONE*** — The field cannot contain blanks.
- ***TRAILING*** — Blanks are permitted only when positioned to the right of the far right-hand (non-constant) character in a field. Trailing constant characters are treated as trailing blanks for the purpose of applying this restriction.
- ***ANYWHERE*** — Blanks can be entered in any position within a field.

The default is *ANYWHERE*.

Alignment

Determines whether or not data in a field is aligned for storage and/or display and, if so, the type of alignment. The *Alignment* options are:

- ***NONE*** — Data is not aligned for either storage or display purposes.
- ***LEFT*** — Data is aligned with the far left-hand character position in a field.
- ***JUSTIFIED*** — Data is aligned with both the far left-and right-hand character positions in a field, and distributed to occupy the entire field. Where necessary, data is padded with spaces.
- ***RIGHT*** — Data is aligned with the far right-hand character position in a field.
- ***CENTERED*** — Data displays in the center of a field.

The default is *NONE*. *Figure 2-3-6* shows the different alignment options.

Figure 2-3-6. Alignment Options

This type is aligned flush with the left margin.	This type is aligned flush with the right margin.	This type is centered within the left and right margins.	This type is aligned flush with both margins.
Left	Right	Centered	Justified

Pad Character

Indicates the character to be used instead of leading and trailing spaces in a field.

Translate?

Indicates if the value of fields based on this domain should be translated into a designated language if APPX's Language Translation feature is being used. To qualify for translation, an alpha domain must have a validation table specified; otherwise, this option cannot be selected. Click the checkbox to toggle between "yes" ☑, "no" ☐, and unknown ? values. If the translation feature is not in use, retain the default value of ?.

Require Table Value?

Indicates if fields based on this domain require one of the validation table values to be entered. To force the user to enter one of the defined values, click the checkbox to toggle it to "yes" ☑. If a "no" ☐ value is selected, the validation table shown when the user selects the list feature, is advisory only and the user can enter a value not listed in the table. If there is no validation table defined, this option does not apply and cannot be selected. If *Translate?* is ☑, this option should also be ☑.

Format Domains

A format domain is like an alpha domain, except that its primary characteristic is an edit mask.

The Format Domain overlay, shown in *Figure 2-3-7*, displays after you complete the entries on the Domains screen if the domain *Type* you select is *FORMAT*.

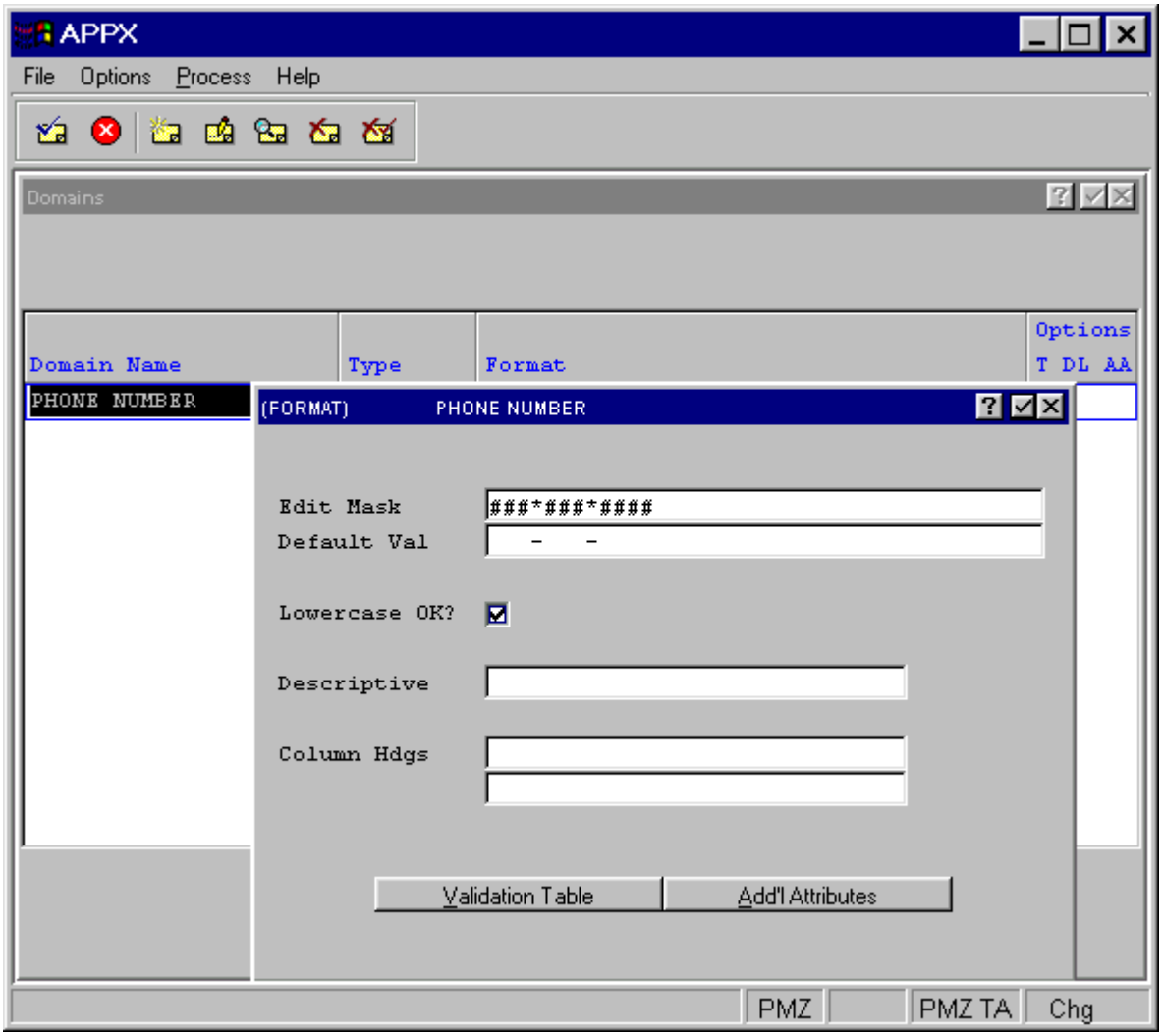


Figure 2-3-7. Format Domain Overlay

The Format Domain overlay contains the following fields:

Edit Mask

Establishes format requirements for data validation and also determines how the field is stored and displayed. A mask character representing the allowable values is entered for each position in the field. The mask characters are:

- # (numbers only)
- @ (letters only)
- & (numbers and letters only)
- ! (any characters)
- * (use default value character) — The valid character for this position in a mask is defined by referencing the same position in the default value.
- For example, you can mask a phone number field with the following entry: ###*###*####.

Default Value

Establishes a starting value for all referenced fields whenever they are first displayed in add mode in an input image, unless modified by a user. If blank, there is no starting value displayed on the image. The positioning of constant characters in the default value must correspond to the positions of *s, if any, in the edit mask. For example, a possible default value for the phone number mask indicated in *Edit Mask* earlier is: _ _ _ - _ _ _ - _ _ _ _ (the underlines represent blanks).

Lowercase OK?

Indicates whether or not a modifiable item on an interactive screen can be entered in upper- and lowercase letters. If "no" ☐, the entry is stored and displayed exclusively as uppercase, regard-less of how a user enters the information. This entry does not affect statements. A lowercase value established with a statement displays as lowercase, even if this specification prohibits lowercase entries. The default is "yes" ☑.

Descriptive

Provides a longer, more easily understood alternative to *Domain Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Domain Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a domain itself. Refer to Standard Column Heading Image for additional information.

For Format domain type, the following options are available as described in the corresponding sections:

- Validation Table
- Additional Attributes

Format Domain Validation Table

The VALIDATION TABLE option from the Format Domains overlay accesses the Validation Table overlay, shown in *Figure 2-3-8*. A validation table provides a more powerful alternative to the use of range checks for data validation when there is a known set of values. Through the list ▼ feature, it provides the user with visibility of all the possible field values as well as the ability to select the desired value.

If defined for the domain, it establishes the validation table for all fields that reference the domain and cannot be modified at the field level. Conversely, if not defined for the domain, a validation table cannot be defined for any field that references the domain.

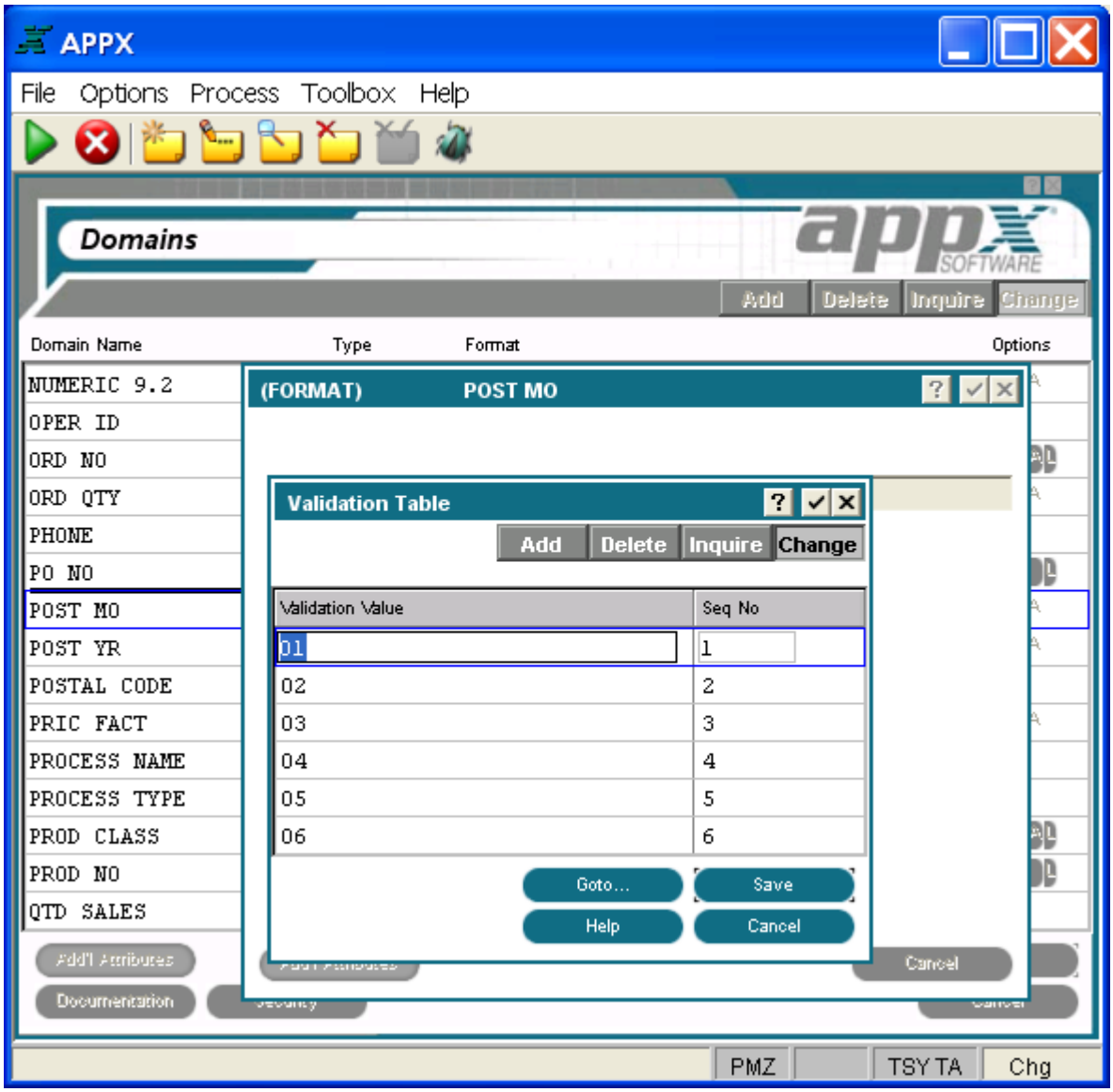


Figure 2-3-8. Format Domain Validation Table Overlay

You can establish a validation table for any Format domain in the data dictionary. The entries in the table are case-sensitive. If you enter a validation value in all capital letters, an application user must enter the value in all capital letters.

Validation table operations are database-independent. They access a designer-defined table in application design, not data contained in a file in a database, for the range of possible values. As a result, validation does not vary from database to database. It is automatically available to, and exactly the same for, every database that is associated with an application. All validation values are available for review and selection by the end user through the list ▼ feature.

The Validation Table overlay contains the following fields:

Validation Value

Provides a valid field value. It is case-sensitive and is used for online review, selection, and validation. Note that validation values newly added to the data dictionary are not available to a current APPX user until the user exits and re-enters APPX.

Seq No

Provides an identifying number for the associated validation value. It is defaulted by APPX and can be overridden, but it must be unique within a table.. It determines the order in which validation values are presented, both for selection by the user and for editing in application design. APPX assigns the first validation value a number of *1*, with additional values assigned a number that is equal to the highest existing sequence number plus *1*. APPX does not require that sequence numbers be contiguous within a table.

Format Domain Additional Attributes

From the Format domain overlay, you have an ADDITIONAL ATTRIBUTES option. The Format domain Additional Attributes overlay is shown in *Figure 2-3-9*.

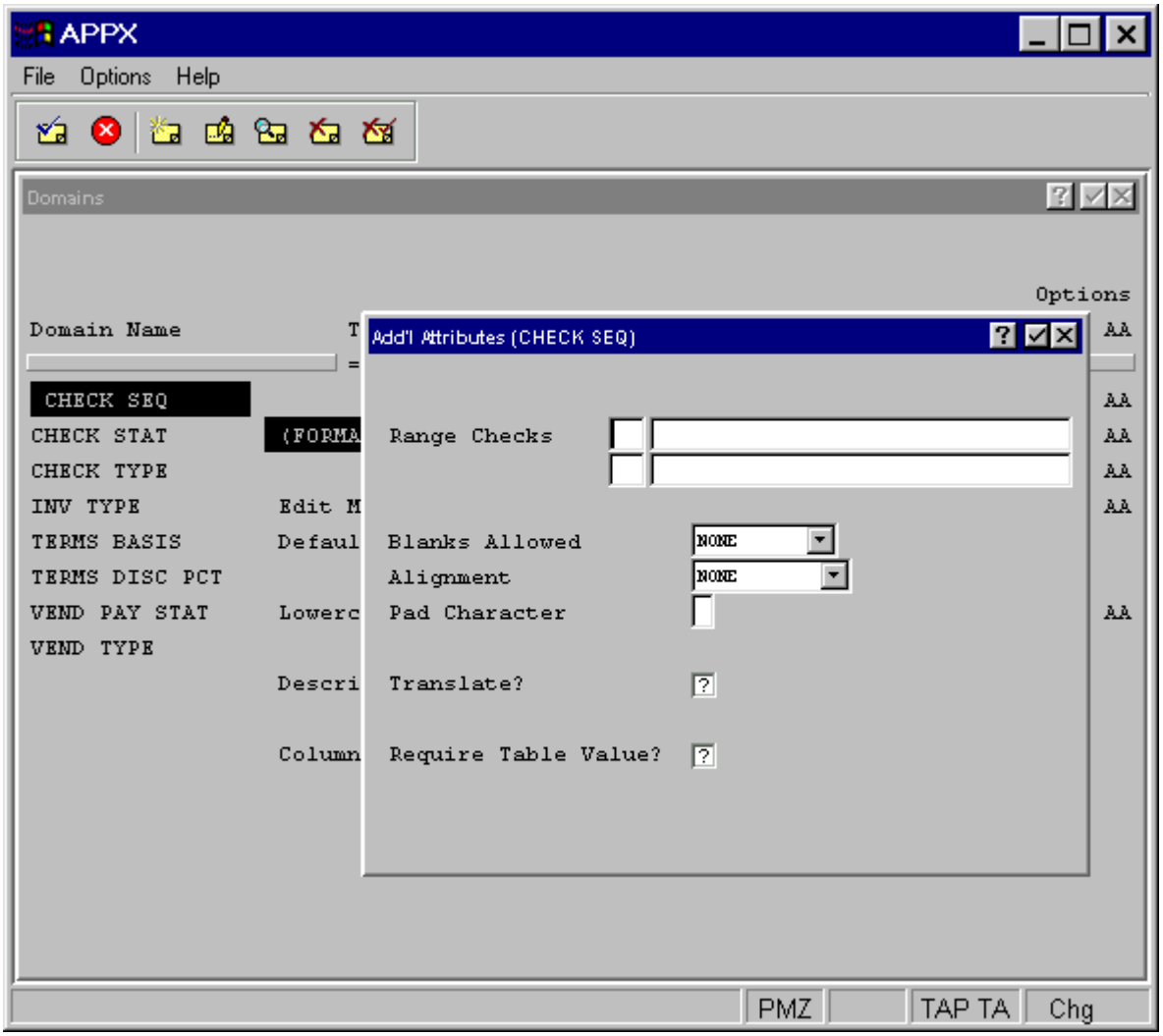


Figure 2-3-9. Format Domain Additional Attributes Overlay

The Format Domain Additional Attributes Overlay contains the following fields:

Range Checks

Defines a series of allowable values that APPX uses for online data validation. The first component represents the relation. The options are ***EQ*** (equal to), ***NE*** (not equal to), ***GT*** (greater than), ***LT*** (less than), ***GE*** (greater than or equal to), ***LE*** (less than or equal to), ***IN*** (includes).

- The second component represents the value assigned to the relation. If you enter two relations, APPX infers an AND relationship.

Blanks Allowed

Indicates whether or not blanks are permitted and, if so, where they can be placed. The *Blanks Allowed* options are ***NONE, TRAILING,*** and ***ANYWHERE***. The default is *ANYWHERE*.

Alignment

Determines whether or not data in a field is aligned for storage and/or display and, if so, the type of alignment. The *Alignment* options are ***NONE, LEFT, JUSTIFIED, RIGHT*** and ***CENTERED.*** The default is *NONE*.

Pad Character

Indicates the character to be used instead of leading and trailing spaces in a field.

Translate?

Indicates if the value of fields based on this domain should be translated into a designated language if APPX's Language Translation feature is being used. To qualify for translation, a format domain must have a validation table specified; otherwise, this option cannot be selected. Click the checkbox to toggle between "yes" ☑, "no" ☐, and unknown ? values. If the translation feature is not in use, retain the default value of ?.

Require Table Value?

Indicates if fields based on this domain require one of the validation table values to be entered. To force the user to enter one of the defined values, click the checkbox to toggle it to "yes" ☑. If a "no" ☐ value is selected, the validation table shown when the user selects the list feature, is advisory only and the user can enter a value not listed in the table. If there is no validation table defined, this option does not apply and cannot be selected. If *Translate?* is ☑, this option should also be ☑.

System Domains

A system domain, often referred to as an audit domain, displays data that is provided by the system.

The system domain overlay for dates, shown in *Figure 2-3-10*, allows you to add system specific attributes to those already defined for domains. The system domain specifications overlay displays after you complete the entries on the domains screen if the domain type you enter is *system*.

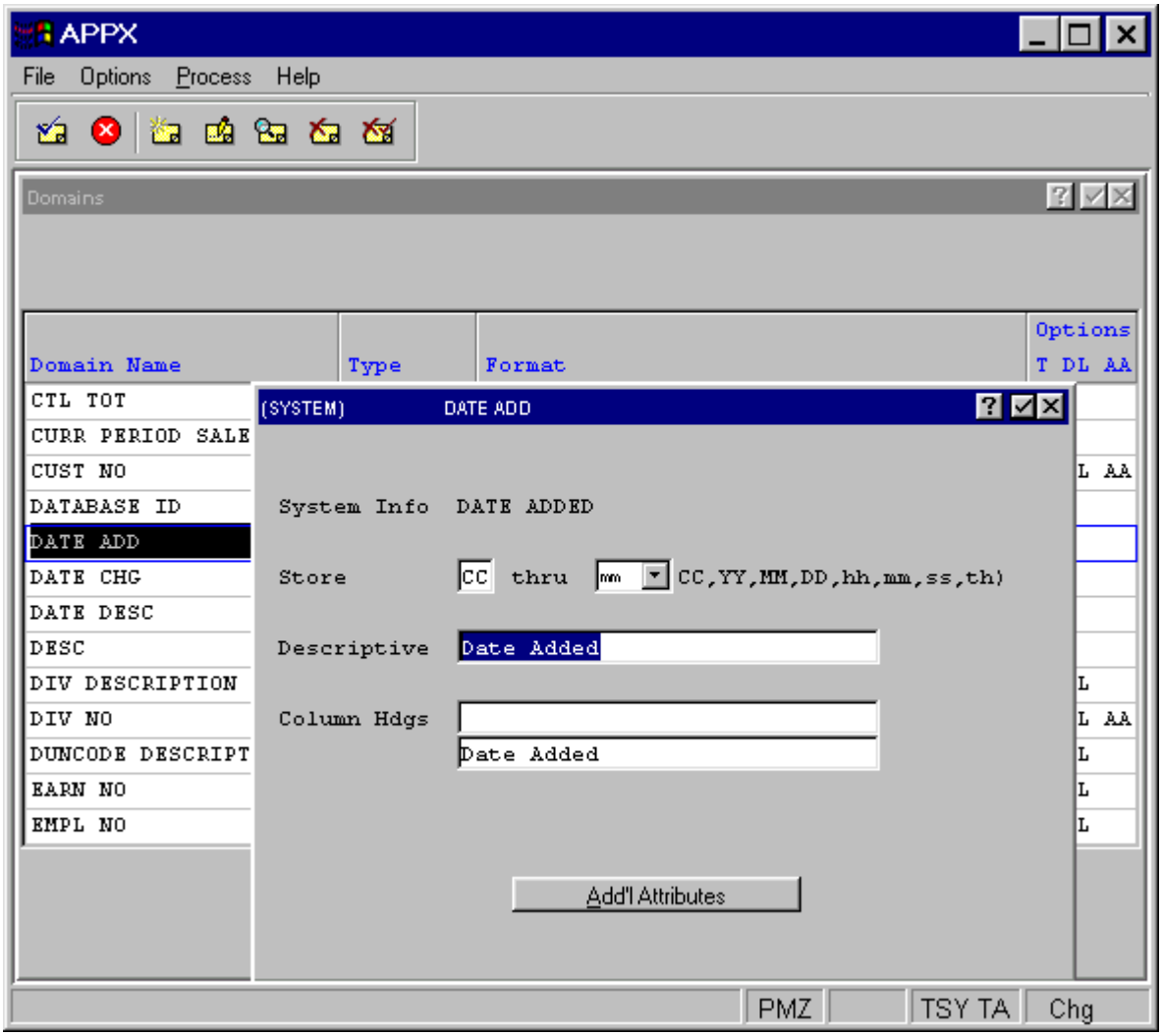


Figure 2-3-10. System Domain Overlay

The System Domain Specifications overlay contains the following fields:

System Info

Determines the option that is to be displayed on the field specifications screen when defining a system field. The *System Info* options are ***USER ADDED***, which identifies the user ID of the operator who added the record; ***USER CHANGED***, which identifies the user ID of the operator who modified the record. ***DATE ADDED***, which identifies the date that the record was added; and ***DATE CHANGED***, which identifies the date the record was last modified.

Select ENTER to display the remaining fields:

Store thru

Is displayed for System Date fields only. Select the upper and lower limits for this date/time field.

Descriptive

Provides a longer, more easily understood alternative to *Domain Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Domain Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a domain itself. Refer to Standard Column Heading Image for additional information.

System Domain Additional Attributes

The ADDITIONAL ATTRIBUTES option is only available if the DATE ADDED or DATE CHANGED type is selected. There is only one field on this overlay:

Storage type

Denotes how the system date field should be stored. Your choices are:

- ***BINARY*** — The date is stored in binary format.

PACKED DECIMAL

Stores two decimal digits for each byte.

ALPHA

Stores the external representation of date.

The default is *PACKED DECIMAL*.

Text Domains

A text domain allows a user to enter a block of text on the data entry screen.

The text domain overlay, shown in *Figure 2-3-11*, allows you to add text specific attributes to those already defined for domains. The text domain specifications overlay displays after you complete the entries on the domains screen if the domain *Type* you select is *TEXT*.

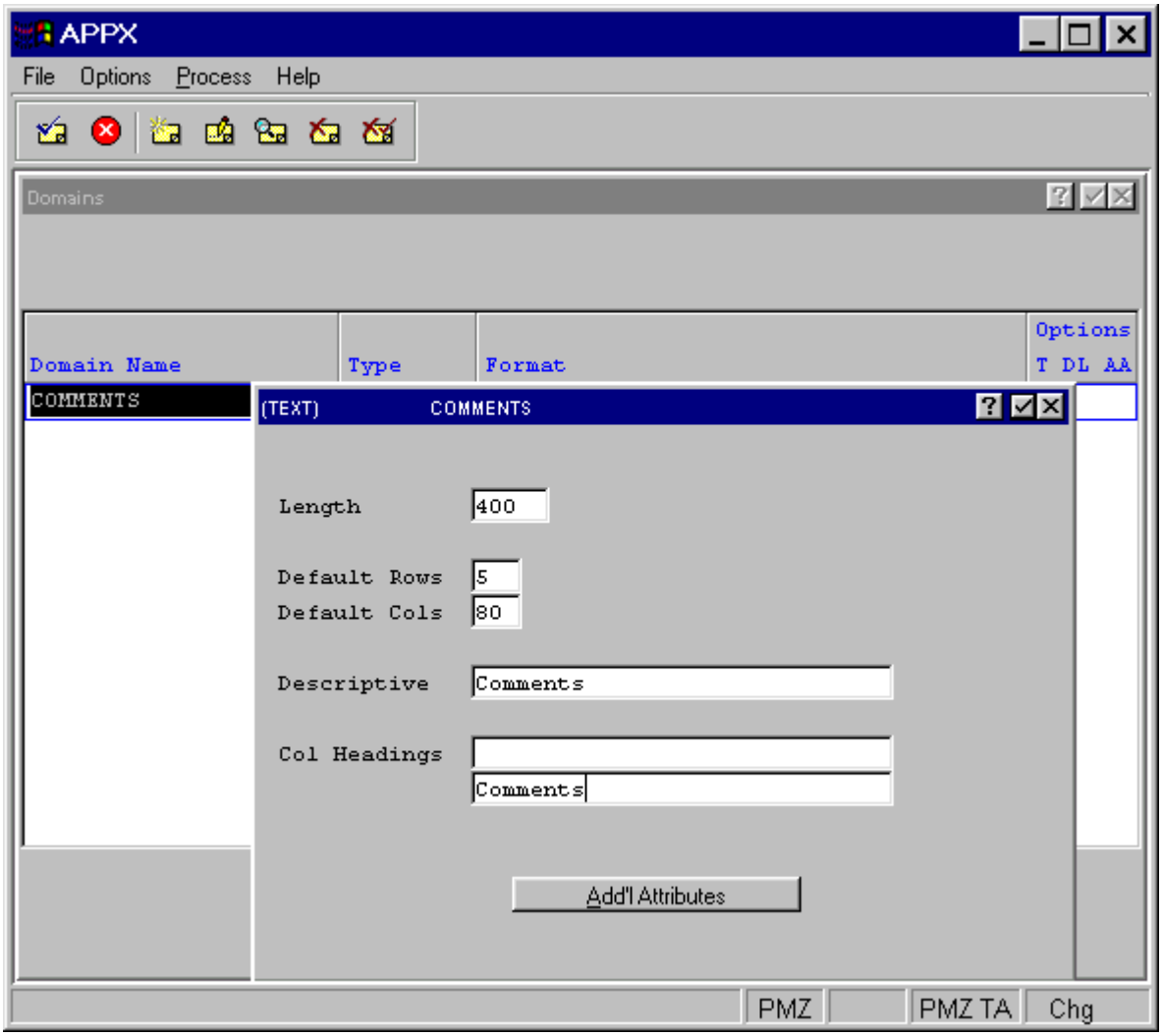


Figure 2-3-11. Text Domain Overlay

The Text Domain Specifications overlay contains the following fields:

Length

Designates the number of character positions, or length, of this field.

Default Rows/Cols

Determines the default number of rows and columns to be used when specifying a field on an image.

Descriptive

Provides a longer, more easily understood alternative to *Domain Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Domain Name* is used.

Col Headings

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a domain itself. Refer to Standard Column Heading Image for additional information.

For Text domain type, the following option is available as described in the corresponding section:

- Additional Attributes

Text Domain Additional Attributes

From the Text domain overlay, you have an ADDITIONAL ATTRIBUTES option. The Text domain Additional Attributes overlay is shown in *Figure 2-3-12*.

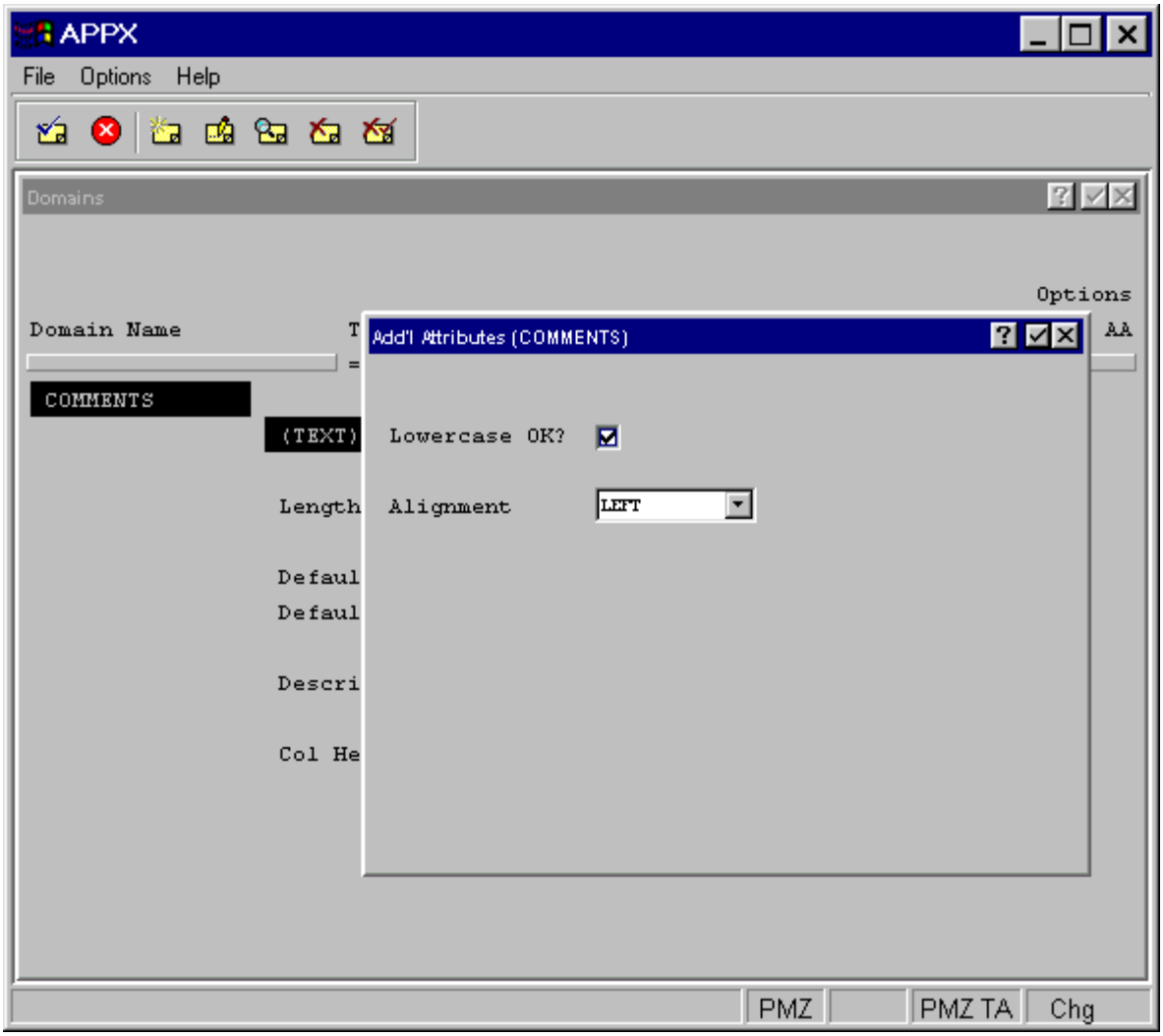


Figure 2-3-12. The Text Domain Additional Attributes Overlay

The Text Domain Additional Attributes Overlay contains the following fields:

Lowercase OK?

Indicates whether or not a modifiable item on an interactive screen can be entered in lowercase letters. If “no” ☐, the entry is stored and displayed exclusively as uppercase, regardless of how a user enters the information. A lowercase value established with a statement displays as lowercase, even if this specification prohibits it. The default is “yes” ☑.

Alignment

Determines the alignment of data in a field. The *Alignment* options are ***NONE, LEFT, JUSTIFIED, RIGHT*** and ***CENTERED.*** This effects the display and storage. The default is *NONE*.

Token Domains

A token domain is like an alpha domain but has specific values that are stored as corresponding numbers.

The token domain overlay, shown in *Figure 2-3-13*, allows you to add token specific attributes to those already defined for domains. The token domains specification overlay displays after you complete the entries on the domains screen if the domain *Type* you select is *TOKEN*.

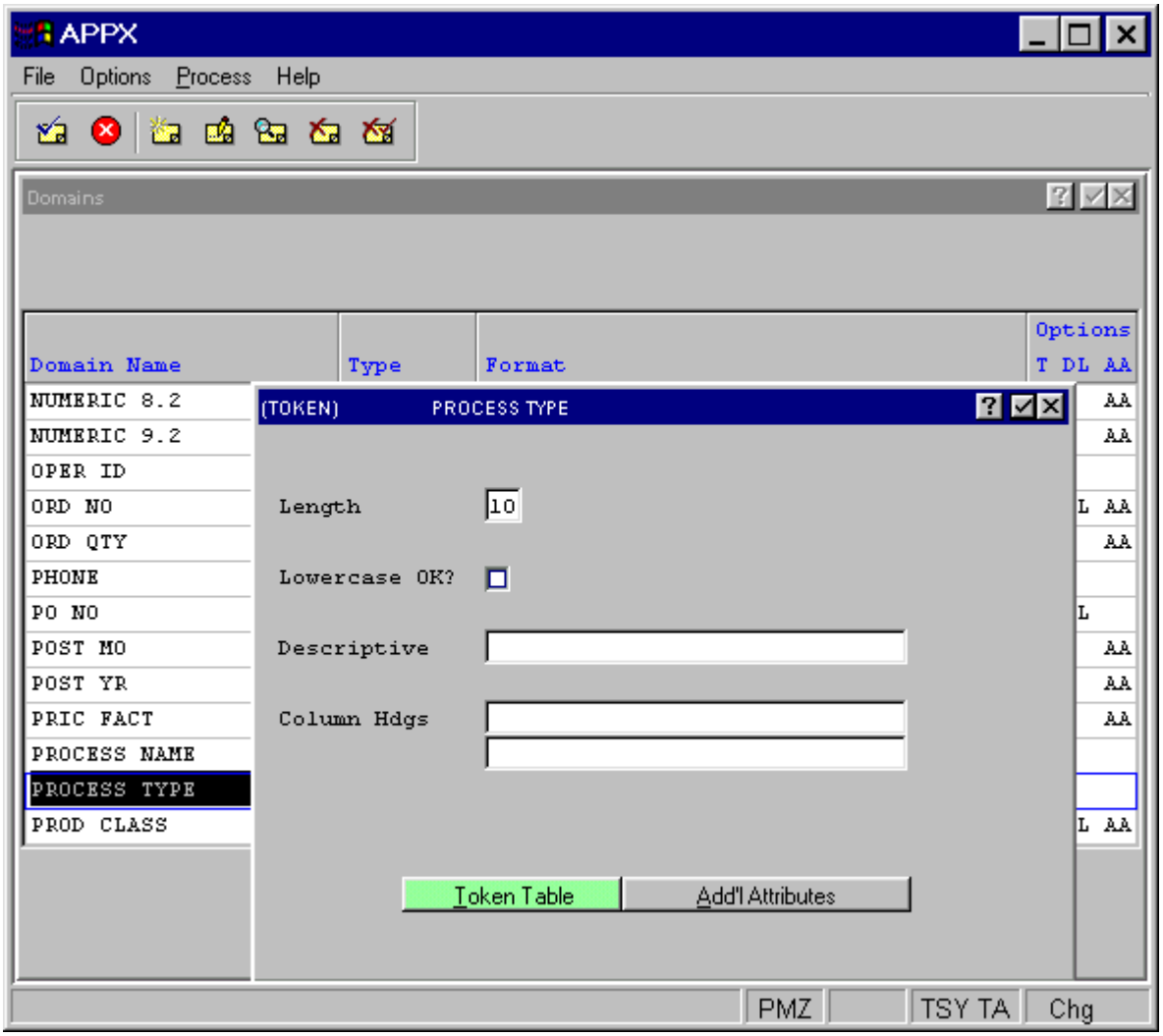


Figure 2-3-13. Token Domain Overlay

The Token Domain Specifications overlay contains the following fields:

Length

Designates the number of character positions, or length, of this domain.

Lowercase OK?

Indicates whether or not a modifiable item on an interactive screen can be entered in upper- and lowercase letters. If "no" ☐, the entry is stored and displayed exclusively as uppercase, regardless of how a user enters the information. This entry does not affect statements. A lowercase value established with a statement displays as lowercase, even if this specification prohibits lowercase entries. The default is "yes" ☑.

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

For Token domain type, the following options are available as described in the corresponding sections:

- Token Table
- Additional Attributes

Token Domain Token Table

From the Token Domain overlay, the TOKEN TABLE option accesses the Token Table overlay shown in *Figure 2-3-14* below. A token table provides an alternative to the use of range checks for data token.

You can establish a token table for any Token domain in the data dictionary. The entries in the table are case-sensitive. If you enter a token value in all capital letters, an application user must enter the value in all capitals.

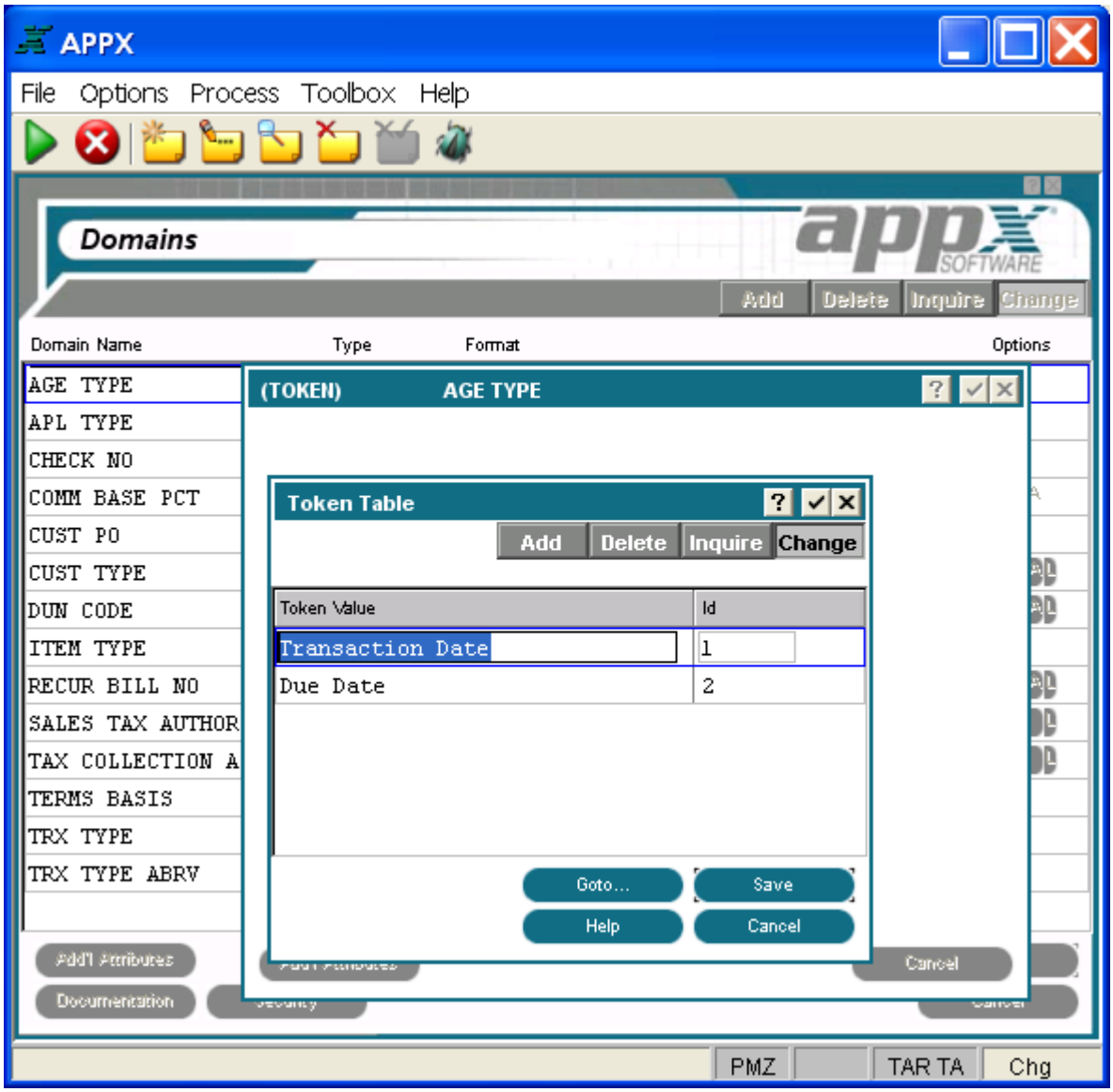


Figure 2-3-14. Token Domain Token Table Overlay

Token table operations are database-independent. They access a designer-defined table in application design, not data contained in a file in a database, for the range of possible values. As a result, a token does not vary from database to database. It is automatically available to, and exactly the same for, every database that is associated with an application.

The Token Table overlay contains the following fields:

Token Value

Is the alphanumeric value associated with a token ID. APPX automatically translates between token values and ID's whenever necessary. This entry is case-sensitive. If entered in capital letters, an application user must also enter the value in capital letters. Also, token values newly added to the data dictionary are not available to a current APPX user until the user exits and re-enters APPX.

ID

Displays a unique number that is defaulted by APPX. This specification establishes the binary value that APPX stores in place of the associated token value. APPX assigns the first token value an ID of *1*, with additional values assigned an ID that is equal to the highest existing ID plus *1*. The default can be overridden, but must be unique within a table. APPX does not require that ID's be contiguous within a table.

Token Domain Additional Attributes

From the Token domain overlay, the ADDITIONAL ATTRIBUTES option accesses the Token domain Additional Attributes overlay shown in *Figures 2-3-15*.

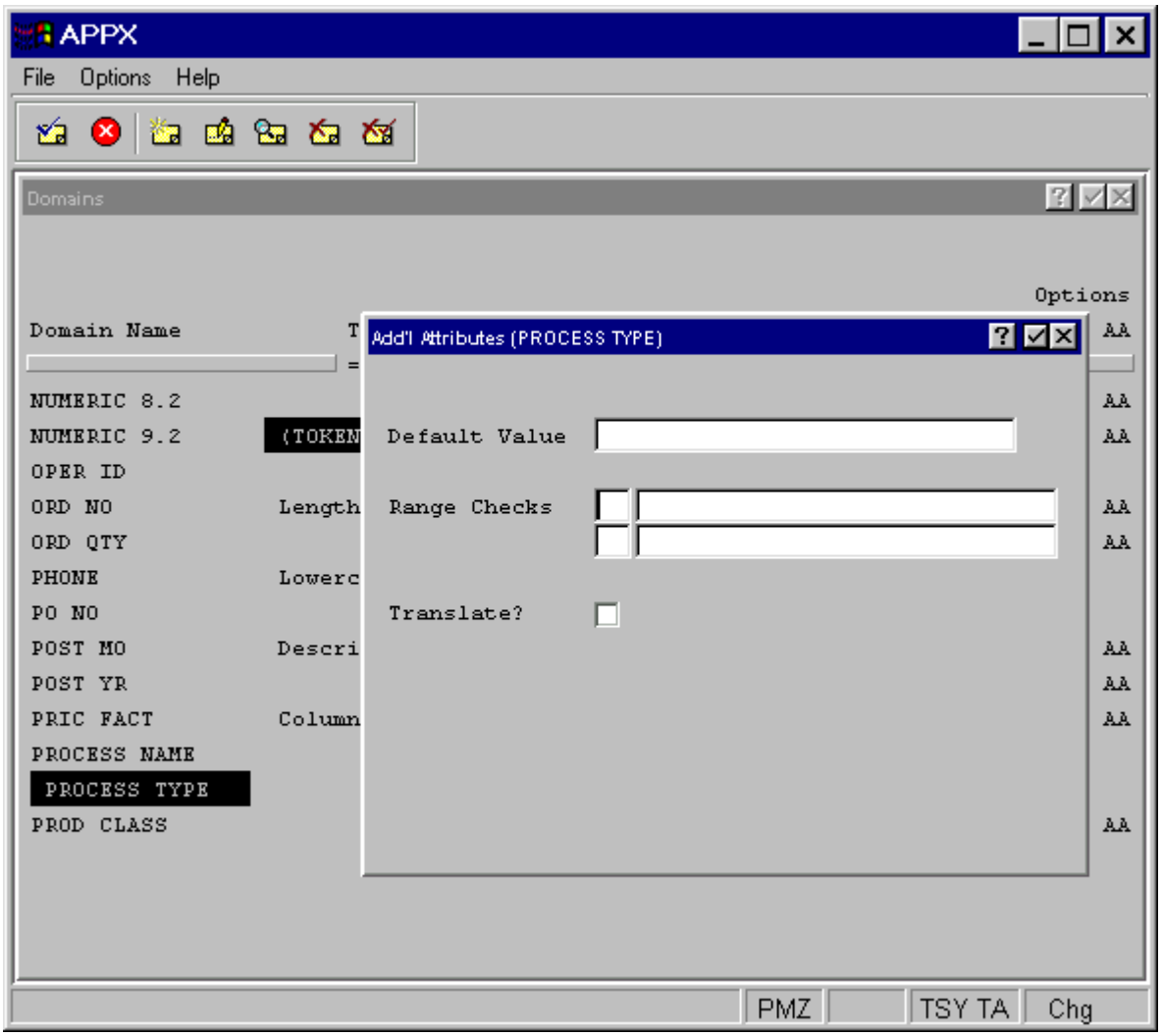


Figure 2-3-15. Token Domain Additional Attributes Overlay

The Token Domain Additional Attributes Overlay contains the following fields:

Default Value

Establishes a modifiable starting value for all image items whenever they are first displayed in add mode in an input image, unless modified by a user. If blank, there is no starting value displayed on the image.

Range Checks

Is not used for token fields.

Translate?

Indicates if the value of fields based on this domain should be translated into a designated language if APPX's Language Translation feature is being used. Click the checkbox to toggle between "yes" ☑, "no" ☐, and unknown ? values. If the translation feature is not in use, retain the default value of ?.

Numeric Domains

A *numeric* domain is restricted to number, decimal, and sign (-) entries. Examples include amounts, prices, and quantities.

The Numeric Domains overlay, shown in *Figure 2-3-16*, displays after you complete the entries on the Domains specifications screen if the *Type* you select is *NUMERIC*.

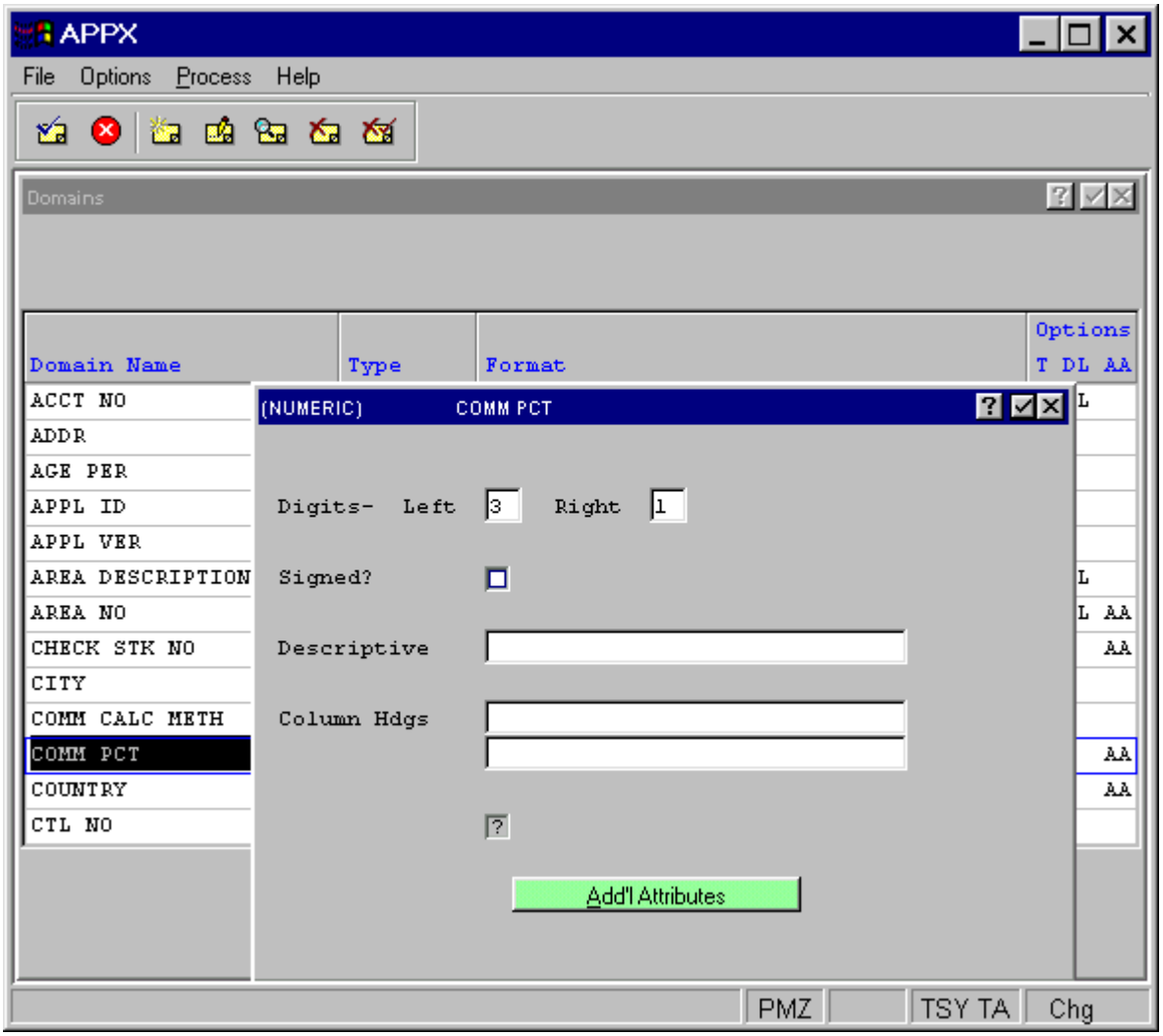


Figure 2-3-16. Numeric Domains Overlay

The Numeric domains overlay contains the following fields:

- ***Digits — Left.*** Enter the number of digits to the left of the decimal.
- ***Digits — Right.*** Enter the number of digits to the right of the decimal.

Signed?

Specifies whether the numbers entered in this domain are positive or negative. A "no" ☐ would ensure that only positive numbers are entered. The default is "no".

Descriptive

Provides a longer, more easily understood alternative to *Domain Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Domain Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a domain itself. Refer to Standard Column Heading Image for additional information.

Auto Seq?

Click the checkbox to toggle between "yes" ☑ if this numeric domain represents a sequence number with a value that should be maintained by APPX and "no" ☐ if it does not.

For Numeric domain type, the following option is available as described in the corresponding section:

- Additional Attributes

Numeric Domain Additional Attributes

From the Numeric Domains overlay, the ADDITIONAL ATTRIBUTES option accesses the Additional Attributes overlay shown in *Figure 2-3-17* below.

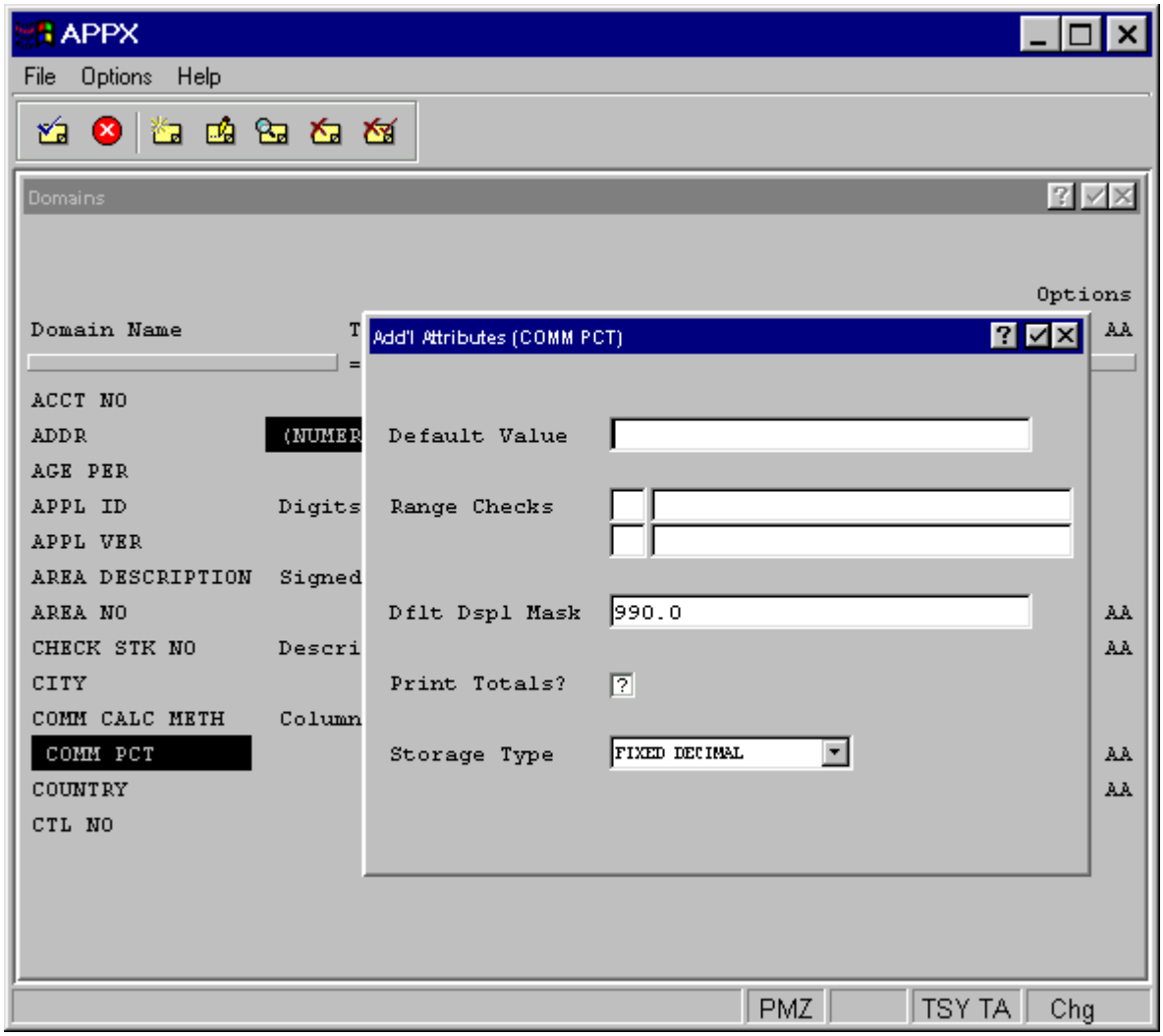


Figure 2-3-17 . Numeric Domain Additional Attributes Overlay

The numeric domains Additional Attributes overlay contains the following fields:

Default Value

Establishes a value for all referenced fields whenever they first display an input image in add mode, unless modified by a user. If blank, the starting value is assumed to be the null value.

Range Checks

Defines a series of allowable values that APPX uses for online data validation. The first component represents the relation. The options are ***EQ*** (equal to), ***NE*** (not equal to), ***GT*** (greater than), ***LT*** (less than), ***GE*** (greater than or equal to), ***LE*** (less than or equal to), ***IN*** (includes).

- The second component represents the value assigned to the relation. If you enter two relations, APPX infers an AND relationship.

Dflt Dspl Msk

Controls how fields based on this domain appear on an APPX image to a user (unless overridden at the image item level). If blank, APPX uses a display of non-fixed leading spaces with one significant digit.

- Digits to the left of a decimal are ***leading digits***.
- Digits to the right of a decimal are ***trailing digits***.

Leading zeros

Refer to zeros that are to the left of the far left-hand leading digit.

Trailing zeros

Refer to zeros that are to the right of the far right-hand trailing digit.

- There are six types of characters that you can use on a numeric mask: digit placeholders, decimals, group characters, sign characters, currency symbols, and alignment characters, as described in a later section.

Print Totals?

Indicates whether fields based on this domain should be automatically totaled on reports. Click the checkbox to toggle between "yes" and "no" . This entry is used as a default for the auto-total specification in output and inquiry process definition.

Storage Type

Determines the internal storage format for all fields that reference this numeric domain. The *Storage Type* options are:

- ***BINARY*** — The number is stored in binary form. The high order bit determines the sign: on or *1* for positive, and off or *0* for negative. The decimal position is established by the *Decimals — Left/Right* entries.

FIXED DECIMAL

Employs a packed decimal format that stores two decimal digits for each byte with the exception of the last bit, which is the sign bit. The decimal position is established by the *Decimals — Left/Right* entries.

FLOATING DECIMAL

Employs a combination of binary and packed decimal formats that always requires a total of nine bytes of storage. The first 15 digits are stored as packed decimal with two decimal digits for each byte (for a total storage requirement of eight bytes). The final byte (binary) determines the maximum number of digits that currently appears to the right of the decimal.

- The default is *FIXED DECIMAL*. For a comprehensive discussion of these options, see Numeric Storage Types.

Numeric Storage Types

There are three storage types available for numeric domains, numeric fields, and numeric work fields. These are discussed in detail below.

- ***BINARY*** — The number is stored in binary form. The high order bit determines the sign: on or *1* for positive, and off or *0* for negative. The decimal position is established by the *Decimals — Left/Right* entries. Binary storage is appropriate when the associated data has a fixed decimal position, and a maximum of 10 significant digits for non-decimal values (9999999999) or nine digits for decimal values (9999999.99). Dollar amounts, for example, where you know there are always two digits to the right of the decimal, are particularly suited for this type of storage.
- Binary storage minimizes storage space requirements and maximizes performance. There are no restrictions imposed on the mathematical operations that you can perform on binary data.
- The internal storage length for binary fields is either one, two, or four bytes. The specific storage length depends upon the maximum number of digits in a field, whether or not a field is signed, and range checks, if any.
- To calculate the storage length for a binary field, you determine the smallest of the signed and unsigned ranges in *Table 2-3-1* that contains all the possible values for the field, including any range check values. For example, a field with possible values from *0* to *255* requires two bytes of storage rather than one byte if it includes a range check of LE 256.

Table 2-3-1. Internal Storage Lengths for Binary Fields

Storage	Range of Signed Numerics	Range of Unsigned Numerics
One Byte	-128 to +127	0 to 255
Two Bytes	-32,768 to +32,767	0 to 65,535
Four Bytes	-2,147,483,648 to +2,147,483,647	0 to 4,294,967,295

- Note that ranges with maximum and minimum values of two digits or less always require one byte of storage. Ranges consisting of values in excess of five digits always require four bytes.

FIXED DECIMAL

Employs a packed decimal format that stores two decimal digits for each byte with the exception of the last bit, which is the sign bit. The decimal position is established by the *Decimals — Left/Right* entries.

- Fixed decimal storage is appropriate for fields with lengths greater than the lengths that are permitted for binary storage, and for which a constant decimal position can be assigned. The maximum field length is 31 significant digits.
- This option requires more bytes of internal storage than a binary field, but less than a variable decimal field. To calculate the internal storage length of a fixed decimal field, apply the following formula and round up to the nearest byte:
- (Total Digits + 1) / 2

- For example, a field with 10 digits requires six bytes of storage: *(10 + 1) / 2,* rounded up to the nearest byte.

FLOATING DECIMAL

Employs a combination of binary and packed decimal formats that always requires a total of nine bytes of storage. Although the field definition allows for 15 digits to the left of the decimal point plus 15 digits to the right of the decimal point, only the first 15 digits of the actual number are stored. These 15 digits are stored as packed decimal with two decimal digits for each byte (for a total storage requirement of eight bytes). The final byte (binary) determines the maximum number of digits that currently appears to the right of the decimal.

- Floating decimal storage is appropriate for decimal fields without a consistent decimal position. These fields are limited to a maximum of 15 significant digits. There are no restrictions imposed on the mathematical operations that you can perform on floating decimal fields.
- This option requires more bytes of internal storage for any given number of significant digits than either of the other options, or a consistent nine bytes. These nine bytes are composed of eight bytes stored in a packed decimal format (for the 15 digits), and a one-byte binary value that stores the current location of the decimal point (0 to 15).

The default is *FIXED DECIMAL.* Actual storage requirements are determined by the *Storage Type, Decimals — Left/Right, Signed?,* and *Range Checks* specifications.

The attributes of each of the three storage types, binary, fixed decimal, and floating decimal, are summarized in *Table 2-3-2* below.

Table 2-3-2. A Comparison of Storage Types for Numeric Domains/Fields

Attribute	Binary	Fixed Decimal	Floating Decimal
Decimal Position	Fixed	Fixed	Variable
Storage Requirements	Minimum	Intermediate	Maximum
Digit Limitation	10 (or 9 with decimal)	31	15 significant digits
Performance	Best	Intermediate	Worst

Six Types of Characters Used in Numeric Masks

There are six types of characters that can be used in a numeric display mask - digit placeholders, sign characters, currency symbols, decimals characters, group characters, and alignment characters. These characters are described below:

Digit Placeholders - There are four ***digit placeholder*** characters, each representing a potential significant digit in a numeric field. The choice of one placeholder over another impacts the appearance of any zeroes and blanks in a field.

- **0** - represents a character position that always displays. It either contains the numeric value in the data or, if the data is blank in that position, a *0* as a placeholder. If the number of trailing digits is variable, *0*s are used to designate the minimum number of digits to display.
- **8** - represents a character position that contains either a numeric value in the data or, if the data is zero in a leading or trailing position, a space. An *8* converts a 0 to a fixed space. The number *8* is used to guarantee decimal or currency sign alignment while eliminating leading and trailing *0*s.
- **9** - represents a character position which, if leading or trailing and zero, is eliminated from a left- or right-justified mask. In other words, a *9* converts a *0* to a non-fixed space. The number *9* (as opposed to the number *8*) is used to conserve space when the alignment of decimals and currency signs is not a concern.
- ***** - represents a character position that, if leading and zero, is replaced by an asterisk. Asterisks are primarily used to ensure that a number is not altered on a printout, such as an accounts payable check.

Sign Characters - There are six ***sign character*** types that ***APPX*** uses to identify positive and negative numbers. Only one type can be used in a single mask. You can enter the characters anywhere on a mask (leading or trailing). The bracket characters "( )" or "< >" however, must be positioned to surround the digit placeholders.

- **+** - displays as a + if a number is positive, and is replaced by a - if the number is negative.
- **–** - only displays if a number is negative. If the number is positive, it is replaced with a fixed or non-fixed space, depending upon its position relative to the alignment character.
- **( )** - surround negative numbers. If the number is positive, it is replaced with a fixed or non-fixed space, depending upon its position relative to the alignment character.
- **<>** - surround negative numbers. If the number is positive, it is replaced with a fixed or non-fixed space, depending upon its position relative to the alignment character.
- **DR** - identifies a negative number in a field that normally contains a credit balance. If a number is positive, it is replaced by a CR.
- **CR** - identifies a negative number in a field that normally contains a debit balance. If a number is positive, it is replaced by a DR.

Currency Symbols - The ***currency symbol,*** entered as a *$,* is used to identify numbers stated in terms of a specific currency. The maximum length for a currency symbol is three characters, and the maximum number of currency symbols in a mask is two, one leading and one trailing.

Decimal Characters - A ***decimal character*** is entered into a mask as a "." ("," internationally) to separate the leading and trailing digits in a number. For variable decimal numbers, the position of the decimal indicates the minimum number of trailing digits.

Group Characters - A ***group character*** is entered as a “,” ("." internationally) to separate digits into groupings of thousands.

Alignment characters - Alignment characters are optional, justification-control characters that are represented by an underscore (_). They separate fixed leading or trailing portions of a mask from justifiable leading or trailing portions. Alignment characters indicate that data cannot be shifted farther than the position that is represented by their placement on a mask. All characters placed before a leading underscore and after a trailing underscore maintain a fixed position. (Leading underscores precede digit placeholders, and trailing underscores follow.) The character position that is occupied by an underscore is eliminated when a field displays.

Mask Characters Influenced by Values in System Administration

The following three characters displayed in numeric masks are influenced by system administration values:

Currency symbol. When displayed to a user, the $ is replaced with the specified currency symbol, and displays in the position (either leading or trailing) that is specified for the symbol. For example, a U.S. dollar sign (*$*) always displays in a leading position. A British pound sign (£) also displays in a leading position.

For masks with a single currency symbol, a leading sign displays as the far left-hand character on a mask. A trailing symbol displays as the far right-hand character of a mask. If other non-digit characters are included in a display, such as sign characters, the currency symbol is presented outside of these characters; for example: £*<750>*. To fine-tune the placement of leading and trailing symbols, you must use both leading and trailing currency symbols on a mask.

For masks with both leading and trailing currency symbols, the symbol displays in a specific, designer-designated, leading or trailing position. The designer assumes control over the appearance of the symbol relative to other non-digit characters in a field. The designer can, for example, place currency symbols inside the < > sign character. The symbol that displays (leading or trailing) depends on the system administration specification. The other symbol is ignored.

Decimal point. When displayed to a user, the “.” is replaced with the decimal character that is appropriate for the language in use.

Group character. When displayed to a user, the “,” is replaced with the group character that is appropriate for the language in use.

Refer to the *System Management Manual* for additional information.

Text in Numeric Masks

In addition to the six types of characters described earlier that can be included in a numeric display mask, a mask can include text. For example, it may be desirable to insert the text “percent” into a numeric mask to the right of the digit placeholders.

Formats Excluded by Numeric Masking Facility

APPX’s numeric masking facility was designed to provide maximum flexibility. You can define a mask in any format that is not specifically excluded below.

Masks must include at least one digit placeholder, and no more than 15 digit placeholders.

Each digit placeholder character, if used, must appear in the following sequence in the mask:

98*0.089.

*9*s, if used, must define the outside boundaries for digits in a mask. Leading *9*s must be placed to the left of all other digit placeholders, and trailing *9*s must be placed to the right of all other digit placeholders.

99880.08999 Correct, 98980.08998 Incorrect

*8*s, if used, must be placed between any *9*s and the *s or *0*s. Leading *8*s must be placed to the right of any *9*s, and to the left of any *s or *0*s. Trailing *8*s must be placed to the left of ant *9*s, and to the right of *0*s.

9,980.089 Correct, 9,890.098 Incorrect, 9,*80.089 Incorrect

*s are leading digit placeholders only. If used, they must be placed between any leading *9*s or *s and any leading *0*s in the mask.

*0*s, if used, must define the inside boundaries for digits in a mask. Leading *0*s must be placed to the right of all other digit placeholders, and trailing *0*s must be placed to the left of all other digit placeholders.

The only characters that can be placed between any two digit placeholders are the decimal and group characters, or another digit placeholder.

A mask can contain only one decimal, and the decimal must be followed by a digit placeholder.

A group character, if included in a mask, must be positioned to the left of any decimal and used to separate groups of three digit placeholders. For example, APPX considers an entry of *9,99* to be text rather than digit placeholders and a group character because the group character is not associated with a group of three-digit placeholders.

The paired sign indicators “( )” or “< >” must be placed outside of the far right- and left-hand digit placeholders in a mask.

(99,990.009) Correct, <99,990.009> Correct, 9(9,990.009) Incorrect

Text, if entered, must be placed outside of the far right- and left-hand digit placeholders in a mask.

Date Domains

A date domain contains date and/or time information. This information may, at your option, range from a century through a hundredth of a second.

Examples of date/time domains include invoice dates, order dates, elapsed time data, date/time stamps used to uniquely identify records.

The Date domains overlay, shown in *Figure 2-3-18*, permits you to add date/time-specific attributes to those already defined for the domain. It automatically displays after you complete the entries in the Domains specifications screen if the type you selected is *DATE*.

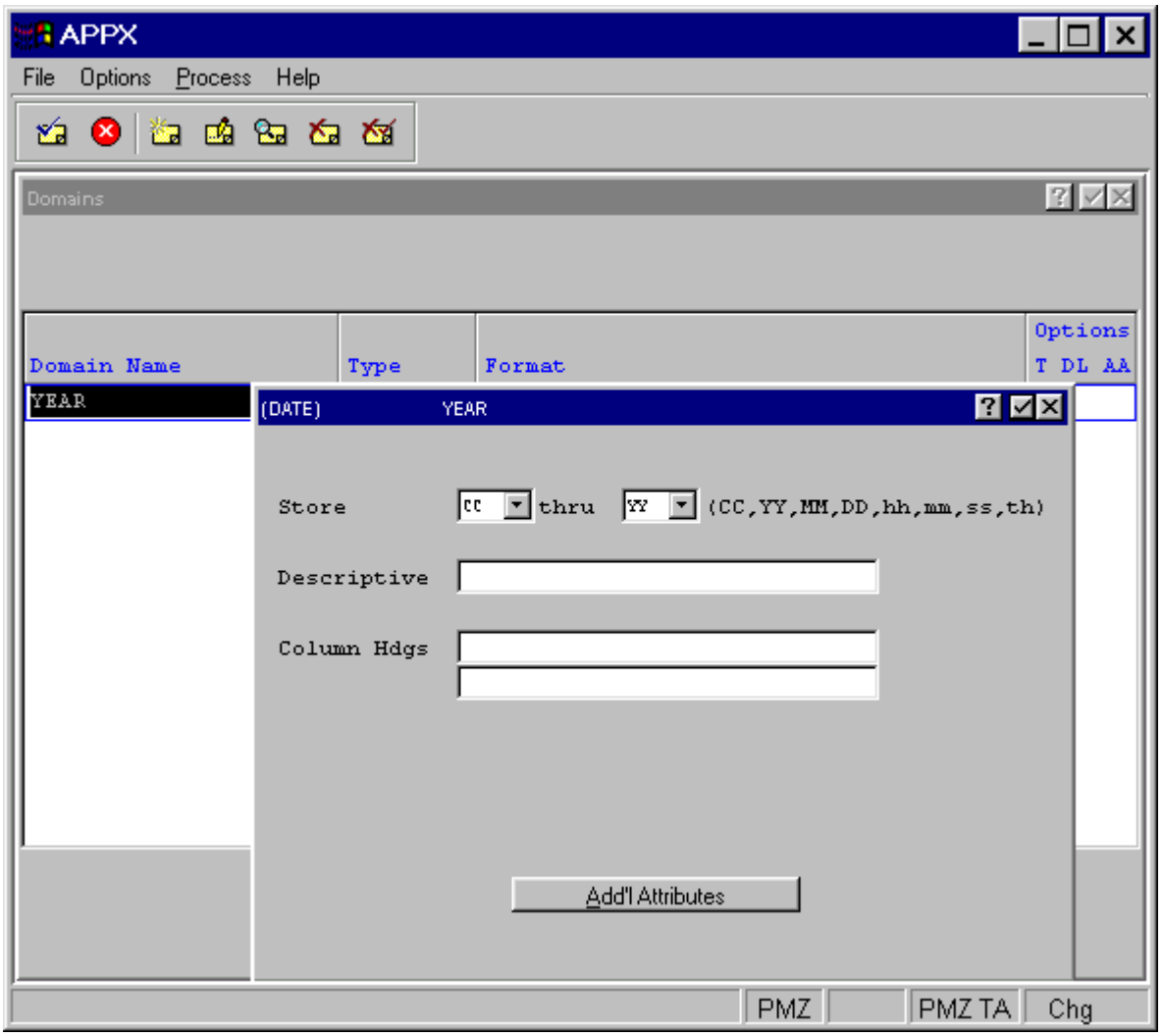


Figure 2-3-18. Date Domain Overlay

The Date domains overlay contains the following fields:

Store thru

Designates the parts of a date that are physically stored. Valid entries are CC (Century), YY (Year), MM (Month), DD (Day), hh (Hour), mm (Minute), ss (Second), th (hundredth of a second).

Descriptive

Provides a longer, more easily understood alternative to *Domain Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Domain Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a domain itself. Refer to Standard Column Heading Image for additional information.

The Date domains Additional Attributes overlay contains the following fields:

Default Value

See definition for Default Value under Numeric Domains.

Range Checks

See definition for Range Checks under Numeric Domains.

Dflt Dspl Msk

Controls how fields based on this domain appear to a user on an APPX image, unless overridden at the image item level. If left blank, the date mask specified for the application (in Application File Maintenance in System Administration) is used or, if that is also blank, the date mask specified in System Parameters is used. Refer to the Display Mask (Date/Time) section in Chapter 3-14: Using the Character Based Image Editor for additional information about this entry.

Storage Type

Determines the internal storage format of the date field. The *Storage Type* options for Date domains are:

- ***BINARY*** — The date is stored in binary format.

PACKED DECIMAL

Stores two decimal digits for each byte.

ALPHA

Stores the external representation of data.

- The default is ***PACKED DECIMAL***.

Logic Domains

A *logic* domain has possible values of "yes" ☑, "no" ☐, or unknown ?. Examples include various application parameters such as whether or not to post transactions on an accrual basis, and control domains that govern the execution of subroutines based upon an *exists/does not exist* condition.

The Logic domains overlay, shown in *Figure 2-3-19*, permits you to add logic-specific attributes to those already defined for the domain. It automatically displays after you complete the entries on the Domains specifications screen if the *Type* you select is *LOGIC*.

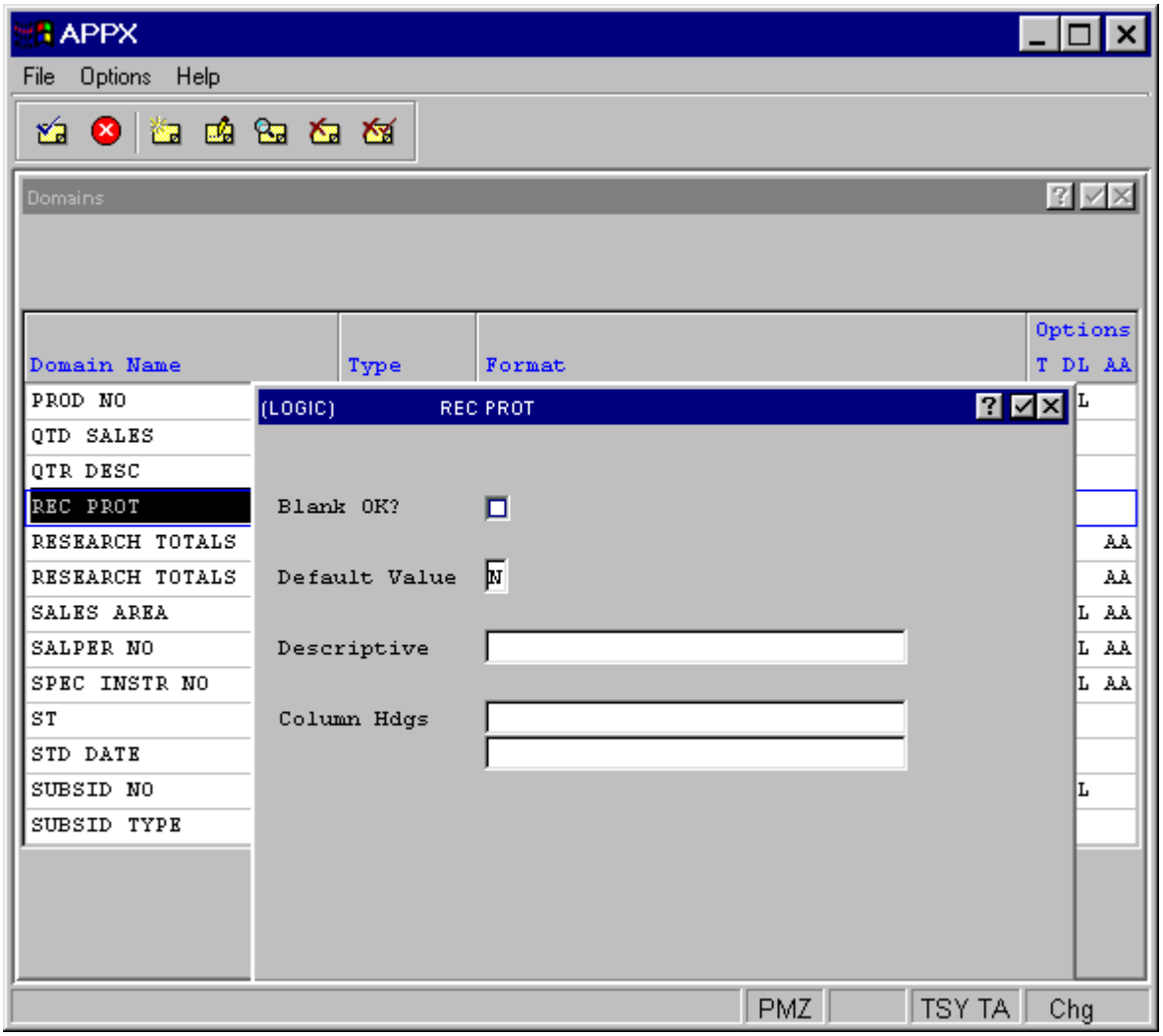


Figure 2-3-19. Logic Domain Overlay

The Logic domains overlay contains the following attributes:

Blank OK?

Indicates, for information only, whether a user should be permitted to leave a field referencing this domain blank on an input image. (Whether or not such a field can actually be left blank is controlled by the image item specification 'Required?'.) The default is “no” ☐. Click the checkbox to toggle between “yes” ☑ and “no” ☐ values.

Default Value

Establishes a starting value for image items whenever they first display in add mode on an input image, unless modified by a user. If blank, the starting value that displays on the image is unknown or ?. The *Default Value* options are ***blank*** (default value of *unknown* or ?), ***Y*** (default value of *yes* or ☑), and ***N*** (default value of *no* or ☐). The default is blank.

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

Same As Domains

A *Same As* domain shares the same characteristics as a domain in another application. This allows you to define fields throughout your database with the same characteristics while ensuring that all fields in an application reference a local domain.

The *Same As* domain overlay, shown in *Figure 2-3-20,* allows you to identify a domain in another application within your database that has the same characteristics required for a domain in your current application. It automatically displays after you complete the entries on the Domains specifications screen if the *Type* you select is *SAME AS.*

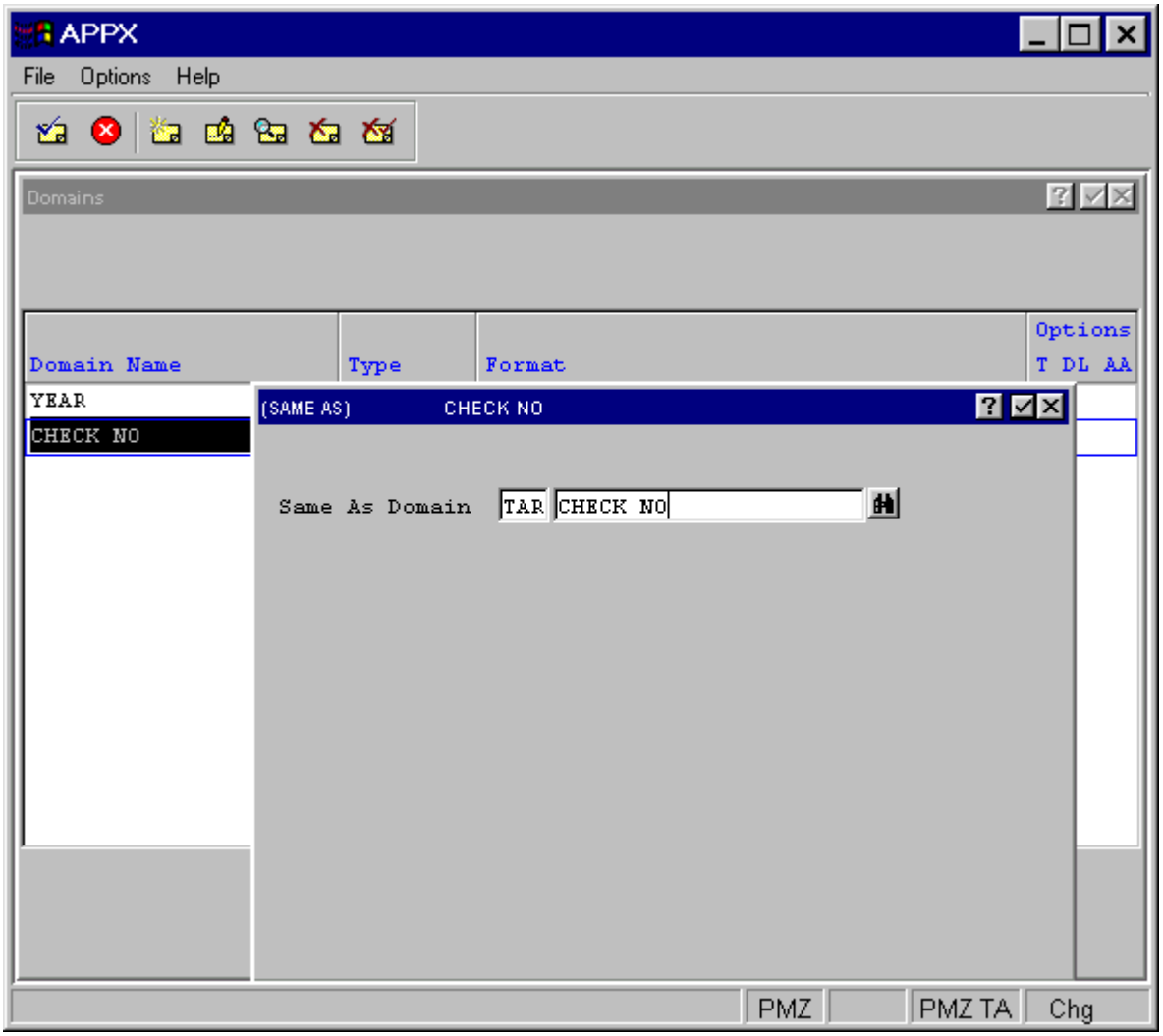


Figure 2-3-20. Same As Domain Overlay

The *Same As* domains overlay contains the following field:

Same As Domain

Identifies a domain in another application within your database that has the same characteristics required for the local domain.

CHAPTER 2-4

Files and Fields

Introduction to Files and Fields

Files are data storage structures for APPX fields (individual items of alphanumeric, numeric, date/time, and logic information in an APPX database). *Fields* represent the smallest units of data storage in APPX applications. Fields are grouped together as records stored within files. The record layout for a file is determined by the fields defined for that file within the data dictionary

The Files option from the Database Management Menu is used to define or maintain the specifications and documentation for an application's files and fields. The specifications on the Files/Fields screens represent a combination of the specifications that define the file storage characteristics. File specifications include *File Name, Description, Organization,* and *File Type.*

Once specifications for a file have been completed, screens for fields within that file can be accessed. The actual screens that display depend upon whether or not you select an option and the specifications you enter at various stages in the definition process.

Files

The Files/Fields screen, shown in *Figure 2-4-1* below, displays when you select the FILES option from the Database Management Menu. The Files/Fields screen presents a scrolling display of file specifications. Once specifications for a file have been entered, you can access the Fields screen for that file by selecting ENTER .

Files/Fields screen

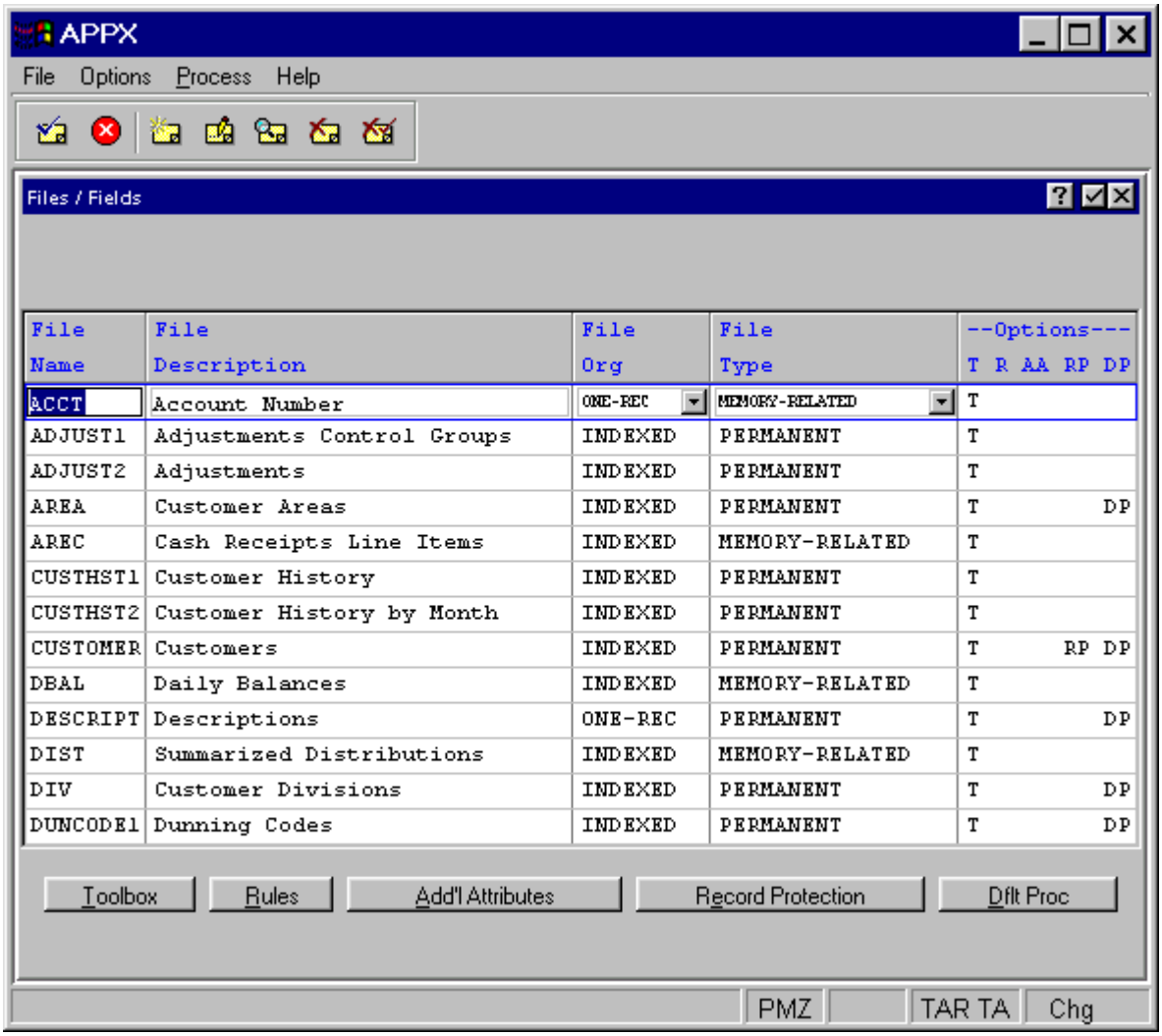


Figure 2-4-1

The Files/Fields screen contains the following fields:

File Name

Uniquely identifies each file in a database. *File Name* also serves as the name of the current record area.

Description

Provides a longer, more flexibly formatted alternative to the technical *File* name. If blank, APPX uses the *File* name as the *Description*. Because *Description* is less cryptic and easier to understand than *File Name*, APPX uses it when communicating with users wherever possible. An example is the construction of error and warning messages. If a user selects an input process for a file that was not created (Balance, for example), the system might display the message "Account Balance file does not exist." We suggest that you use a plural entry for *Description* and avoid appending the word "file".

- · ***File Org*** (organization) determines the way records are stored in and retrieved from a file. The *Org* options are:

ONE-REC

Files are consecutive files that are limited to one record.

CONSEQ

Files are organized according to the relative sequence that records are added to a file. Records are retrieved by relative position.

INDEXED

Files are organized according to the content of specified fields, or key fields. Records are retrieved by key path.

- The default is *INDEXED*. Refer to the File Organization section in *Chapter 2-1* for more information.
- ***File Type*** — There are two broad categories of file type: *permanent* and *temporary*. Permanent files are stored on disk and remain there whether the user is logged on or not. Temporary files can be available for a process and its subprocesses, for a set of related processes, or for an entire session. Temporary files can be stored on disk or in memory, or not stored at all. In general, memory files are faster and can improve performance. There are eight distinct file types:

PERMANENT

Files are stored on disk and remain there whether the user is logged on or not.

DISK-SUBPROCESS

Files are temporary, stored on disk, and deleted when the main process ends.

DISK-RELATED

Files are temporary, stored on disk, and deleted when the process that called the related processes (usually a job process) ends.

DISK-DETACHED

Files are temporary, stored on disk, and remain available until the user logs off.

MEMORY-SUBPROCESS

Files are temporary, stored in memory, and deleted when the main process ends.

MEMORY-RELATED

Files are temporary, stored in memory, and deleted when the process that called the related processes (usually a job process) ends.

MEMORY-DETACHED

Files are temporary, stored in memory, and remain available until the user logs off.

WORKING-STORAGE

Files are temporary, not stored at all, and deleted when the process that is using them ends. For instance, working-storage files can be used to pass file-related information from an update process to a related subroutine. Refer to the Working-Storage Files section in Chapter 1-3 for more information.

- Chapter 1-5: Interprocess Communication provides a complete discussion of detached, related, and subprocess files and processes. Also, refer to the File Storage Options and Temporary versus Permanent Files sections of *Chapter 2-1* for additional information.

Options

Are indicators that are present whenever documentation (T), rules (R), additional attributes (AA), record protection (RP), or default process (DP) have been defined for this file.

The Files/Fields screen contains the following options as described in the corresponding sections:

- Toolbox
- Rules
- Additional Attributes
- Record Protection
- Default Process

Files Toolbox Option

The toolbox overlay appears on the screen when you invoke the TOOLBOX option from the Files/Fields screen. The overlay, shown in *Figure 2-4-2*, contains a number of features such as Documentation Management and Security Management that will assist you when creating files and fields.

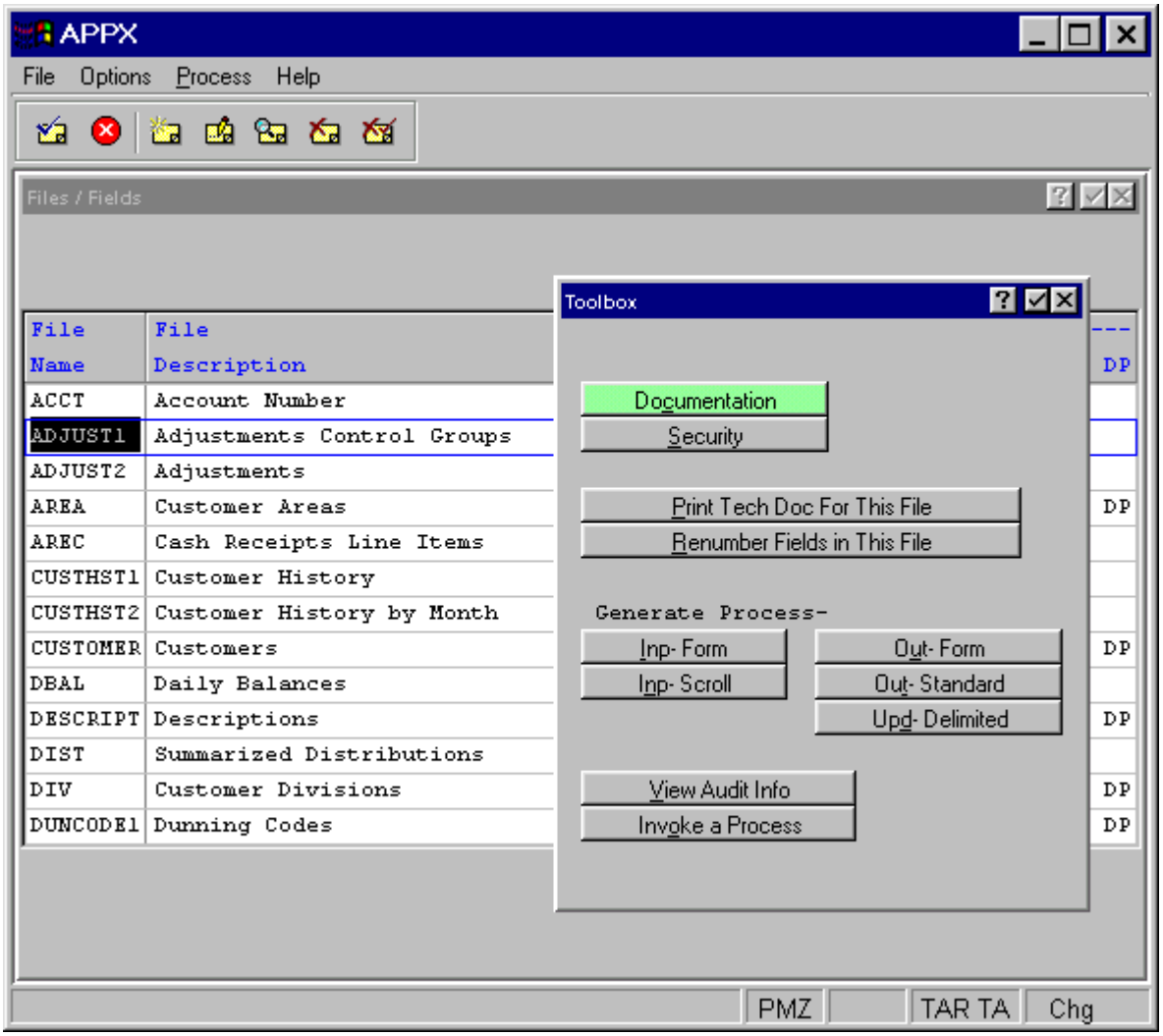


Figure 2-4-2. Files Toolbox Overlay

The options listed in the Toolbox overlay are described below.

Documentation

When you select the DOCUMENTATION option from the Toolbox overlay, you will first see the Documentation overlay which allows you to select between user or technical documentation. Once you have entered your documentation, you are returned to the files/fields screen.

Security

The SECURITY option allows you to identify the APPX security code to which a user must have rights in order to access the file. The security code must be four numeric characters. If blank, there are no restrictions.

Print Tech Doc for this File

The PRINT TECH DOC FOR THIS FILE option tells APPX to print the technical specifications for the currently selected file. When you select this option, a print disposition screen displays which prompts you to define your print job. Select ENTER to send print job and you will be returned to the files/fields screen. If you printed to screen, select END to return to the files/fields screen.

Renumber Fields for this File

The RENUMBER FIELDS for this File option restores the numbering of the fields in the currently selected file to the default numbering convention in APPX. The first field is defined as *100*, and additional fields are assigned a number equal to the highest existing field number, plus *100*. When APPX finishes renumbering, you are returned to the files/fields screen.

Generate Process Inp - Form

The GENERATE PROCESS INP - FORM option automatically creates an input process using the fields in the currently selected file. The resulting input process will have one field per row. Synonyms and Substrings are ignored. After you select this option, a screen displays that prompts you to name the new process. Then select ENTER and APPX generates the new input process, which can be modified as needed just like any other APPX process. See Chapter 3-6: Input Processes for more information about creating and modifying input processes.

Generate Process Inp - Scroll

The GENERATE PROCESS INP - SCROLL option automatically creates a scrolling input process using the fields in the currently selected file. Fields will be placed from left to right on a scrolling frame. Synonyms, Substrings, and System fields are ignored. When you select this option, a screen displays that prompts you to name the new process. Then select ENTER and APPX generates the new input process, which can be modified as needed just like any other APPX process. See Chapter 3-6: Input Processes for more information about creating and modifying input processes.

Generate Process Out - Form

The GENERATE PROCESS OUT - FORM option automatically creates an output process using the fields in the currently selected file. The resulting output process will have one field per row. Synonyms and Substrings are ignored. After you select this option, a screen displays that prompts you to name the new process. Then select ENTER and APPX generates the new output process, which can be modified as needed just like any other APPX process. See Chapter 3-7: Output Processes for more information about creating and modifying output processes.

Generate Process Out - Standard

The GENERATE PROCESS OUT - STANDARD option automatically creates a scrolling output process using the fields in the currently selected file. Fields will be placed from left to right on a scrolling frame. Synonyms, Substrings, and System fields are ignored. When you select this option, a screen displays that prompts you to name the new process. Then select `ENTER` and APPX generates the new output process, which can be modified as needed just like any other APPX process. See Chapter 3-7: Output Processes for more information about creating and modifying output processes.

Generate Process Upd - Delimited

The GENERATE PROCESS UPD - DELIMITED option automatically creates an update process using the fields in the currently selected file. The process is then used to create exportable text files with fields separated by commas. Fields will be placed from left to right. Synonyms, Substrings, and System fields are ignored. When you select this option, a screen displays that prompts you to name the new process. Then select `ENTER` and APPX generates the new update process, which can be modified as needed just like any other APPX process. See Chapter 3-8: Update Processes for more information about creating and modifying update processes.

View Audit Info

This feature allows you to see who created the currently selected file and who changed it most recently. It also gives the date of creation and the date it was last changed. Note that this option (`OPTION` `98`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox. See *Chapter 3-3, Process Definition* for more information about the VIEW AUDIT INFO option.

Invoke a Process

This feature is available to facilitate testing. It allows you to invoke any process. Note that this option (`OPTION` `99`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox. See *Chapter 3-3, Process Definition* for a more detailed explanation of the Invoke a Process facility.

Files Rules Option

The Rules option from the Files/Fields screen accesses the File Level Event Points for Rules overlay shown in *Figure 2-4-3* below. This overlay provides options to access the ILF Editor for APPX event points. If statements are defined in an event point, the respective text on the screen is highlighted. This option allows you to define ILF code that is executed whenever the designated file is accessed or updated in any APPX process.

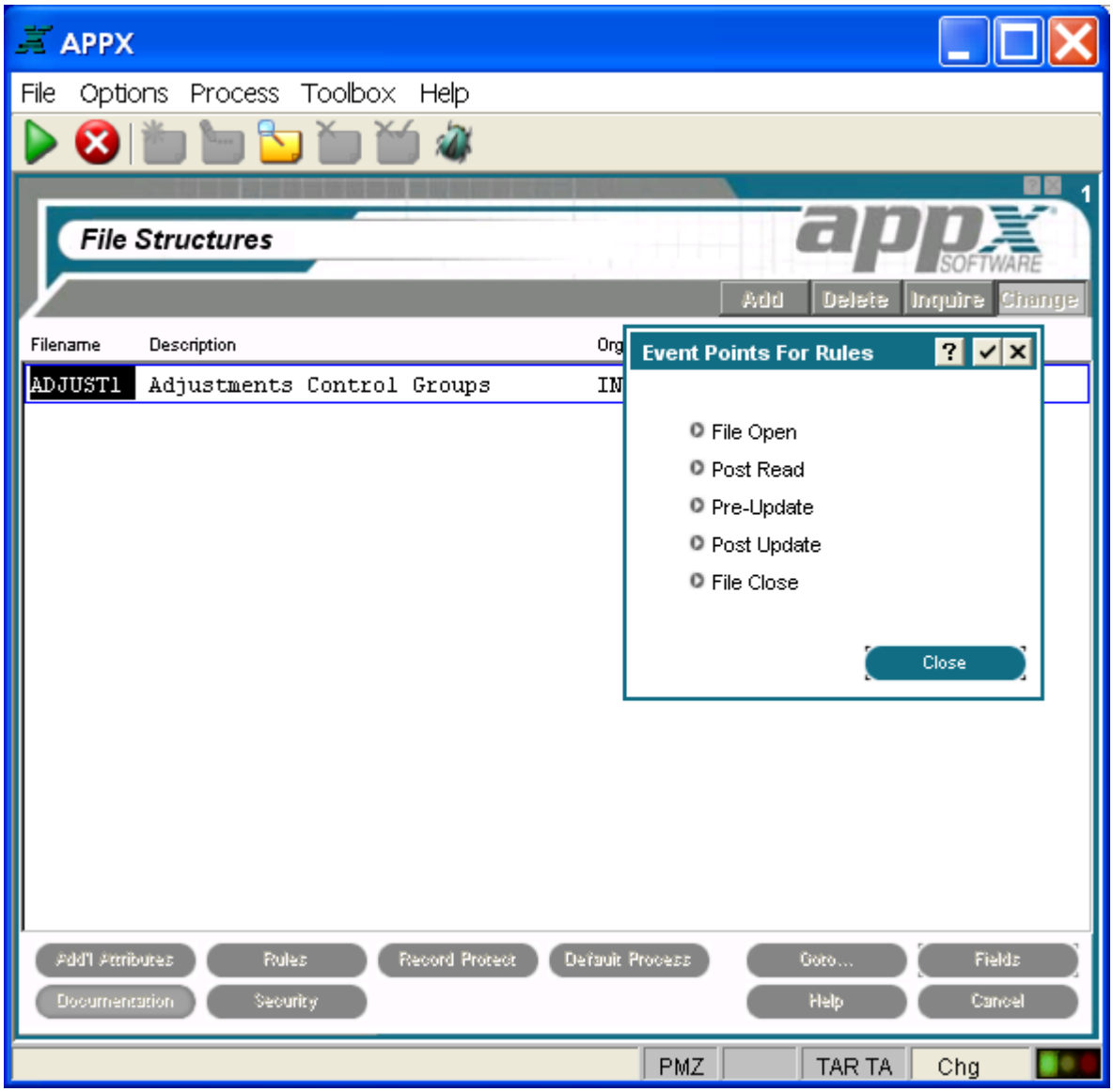


Figure 2-4-3. File-Level Event Points for Rules Overlay

The File Level Event Points for Rules overlay contains the following options:

File Open

FILE OPEN accesses the ILF editor for the File Open event point. Upon completion, you are returned to the Event Points overlay. This code is executed whenever the file is opened by a process.

Post Read

POST READ accesses the ILF editor for the *Post Read* event point. Upon completion, you are returned to the Event Points overlay. This code is executed immediately following the execution of a read statement on the file (READ, READNEXT, or BEG READ) or automatic read of the file when designated as a PCF file.

Pre-Update

PRE-UPDATE accesses the ILF editor for the *Pre-Update* event point. Upon completion, you are returned to the Event Points overlay. This code is executed immediately prior to the issuance of an update statement on the file (WRITE, REWRITE or DELETE) or automatic update of the file when designated as a PCF file.

Post Update

POST UPDATE accesses the ILF editor for the *Post Update* event point. Upon completion, you are returned to the Event Points overlay. This code is executed immediately following the execution of an update statement on the file (WRITE, REWRITE or DELETE) or automatic update of the file when designated as a PCF file.

File Close

FILE CLOSE accesses the ILF editor for the *File Close* event point. Upon completion, you are returned to the Event Points overlay. This code is executed whenever the file is closed by a process.

Using file-level event points localizes complex update rules into a single location, assuring that your code executes for every update. Since these event points execute within the context of a file-level process, they share very little data with other processes. In fact, the only thing shared between a file-level process and the "foreground" process is the PCF record buffer. For example, in the *Post Read* event point, you have access to the contents of the record just retrieved. If changes are made in the record buffer, those changes are reflected in the foreground process. In the *Pre-Update* event point, you have access to the PCF record buffer as delivered by the foreground process. File-level event points use their own copies of work variables so even work fields are not shared with other processes.

To create file level event points, simply select the option corresponding to the desired execution point. See the Event Points and File-Level Event Points sections for further information.

Files Additional Attributes Option

The files Additional Attributes overlay, shown in Figure 2-4-3, displays when you invoke the ADDITIONAL ATTRIBUTES option from the Files/Fields screen. From this overlay, you can enter additional parameters for the file. Upon completion, you are returned to the Files/Fields screen.

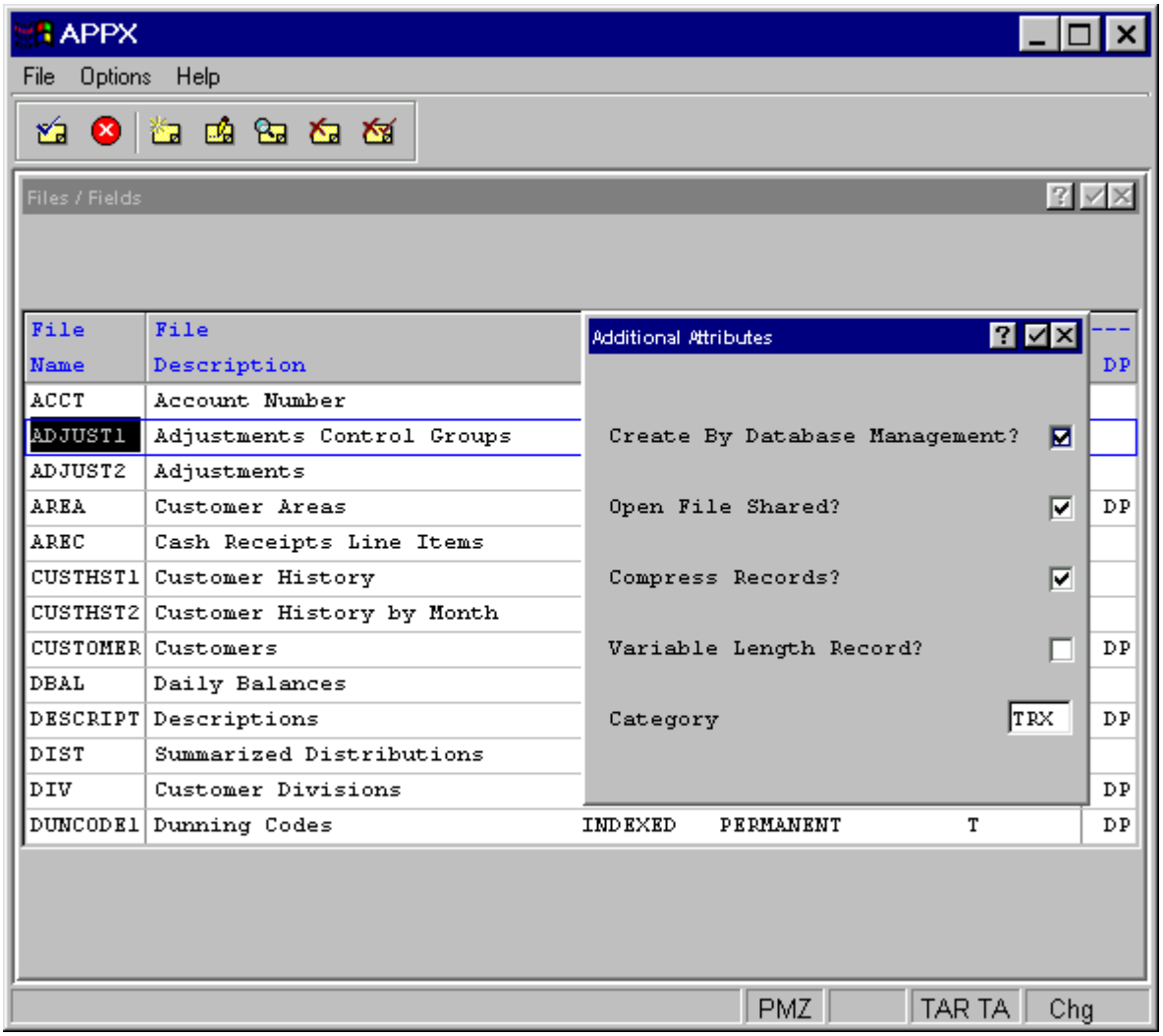


Figure 2-4-3. Files Additional Attributes Overlay

The files Additional Attributes overlay contains some or all of the following fields, depending upon the file type:

Create By Database Management?

Indicates how a permanent file is to be created, either with the create utility in database management, or by the designer with statements. The specification does not apply to temporary files. The Create by Database Management? options are:

- ☑ A file is to be created automatically by the create utility in database management.
- ☐ You assume exclusive control over when a file is created. It is created when an APPX process executes a CREATE statement. The system, however, includes this file on display screens in database management, permitting the database manager to establish defaults, such as the number of records in the file.
- The default is “yes” ☑.

Open File Shared?

Indicates whether or not you want this file to be shared between users. The *Open File Shared?* options are:

- ☑ The file is to be shared between users.
- ☐ The file is to be used exclusively by the user that first opens the file. This exclusivity continues until the user closes the file.
- The default is “yes” ☑.

Compress Records?

Indicates whether or not the records in a file are to be compressed, when possible. Compression optimizes the use of external storage devices by eliminating empty space in a record. Whether or not records can be compressed, and the way they are compressed, depends upon the operating system. In some systems, compressed records in consecutive files may not be modifiable. Refer to your system manual for additional information. The *Compress Records?* options are "yes" ☑, which tells APPX to compress when possible, and "no" ☐, which tells APPX not to compress. The default is "yes" ☑.

Variable Length Record?

Indicates whether or not the record storage length for a file varies from record to record. Select “yes” ☑ for variable length storage in which APPX allocates the exact amount of space required to store each field in a record (the fixed length), less the trailing null portion of the record. When a record is read from such a file, APPX restores the truncated, null fields. On some systems, variable length records in sequential files may not be modifiable.

- Select “no” ☐ for fixed length storage in which APPX allocates a predetermined amount of space for each record, regardless of whether or not the entire allocation is necessary. This length is equal to the sum of the lengths of the fields in record. The default for this field is “no”. Click the checkbox to toggle between “yes” ☑ and “no” ☐ values.
- Files with variable length records undergo a “quick restructure” when fields have been added at the end of the record. The restructure process must still be run, but only the file header is processed, not each individual record. This dramatically decreases restructure time when comparing the restructure time for a file with fixed-length records.

Category

Is used by utilities that automatically generate applications.

Files Record Protection Option

The Record Protection overlay, shown in *Figure 2-4-4*, displays when you select the RECORD PROTECTION option from the Files/Fields screen. The features in this overlay allow you to prevent users from accessing, modifying or deleting records or record keys.

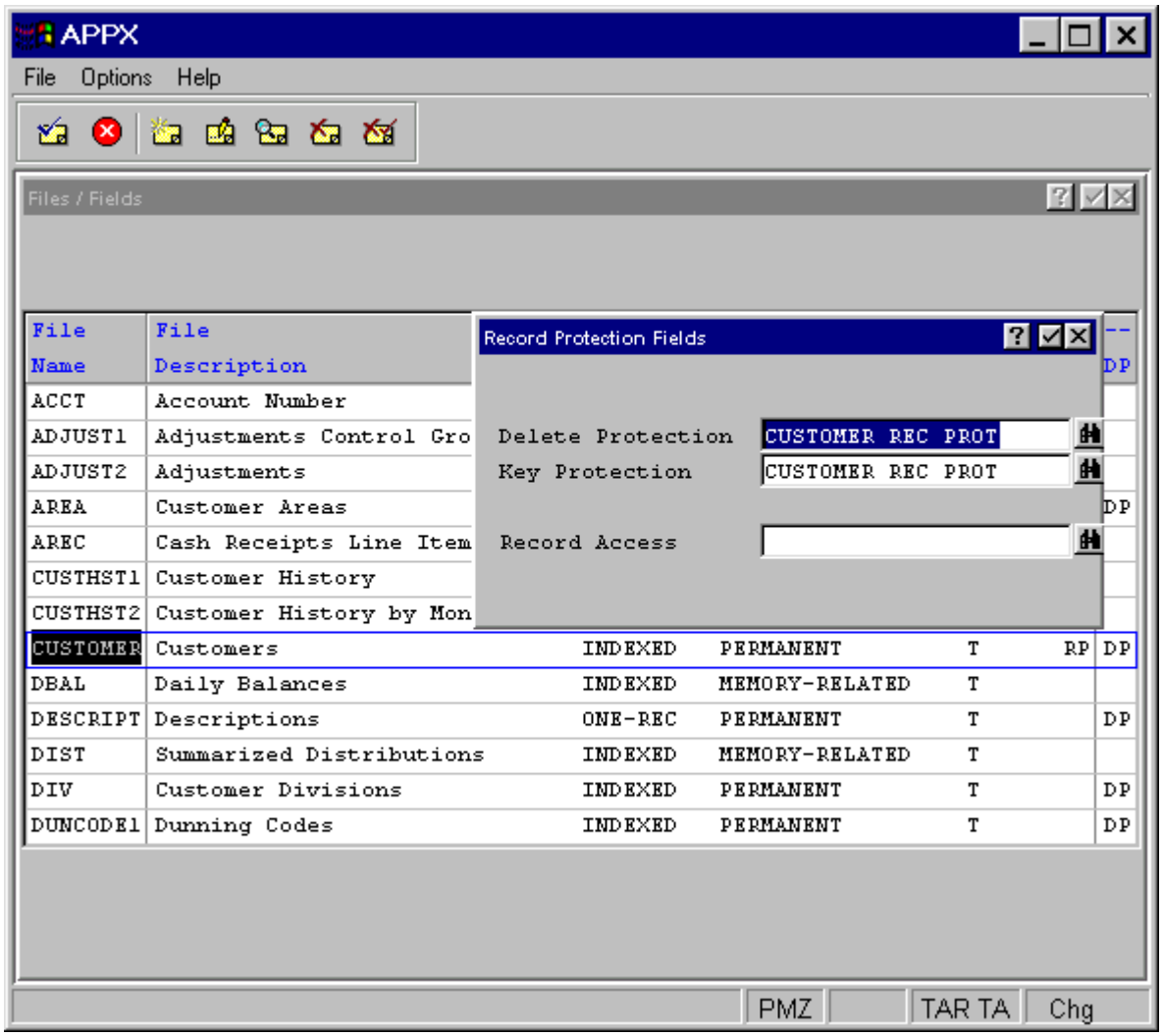


Figure 2-4-4. Files Record Protection Overlay

These fields appear on the Record Protection Overlay:

- The ***Delete Protection*** field allows you to designate a logic or numeric field for record delete protection. A logic field with a value of "yes" ☑, or a numeric field with a non-zero value, signals to APPX to prevent the record it belongs to from being deleted during data entry. The same field can be designated as both Delete Protection and Key Protection. Protection is only provided within an input process during Live Operations (see Operational Phase). This field can be used to prevent the creation of orphan records by prohibiting the deletion of records referenced by other records anywhere in a database. You must explicitly maintain the value in this field.

- The ***Key Protection*** field allows you to designate a logic or numeric field for record key protection. A logic field with a value of "yes" ☑, or a numeric field with a non-zero value, signals to APPX to prevent the primary key for that record from being changed during data entry. The same field can be designated as both Delete Protection and Key Protection.. Protection is only provided within an input process during Live Operations (see Operational Phase). This field can be used to maintain the integrity of a database by prohibiting the modification of keys referenced by other records anywhere in the database. You must explicitly maintain the value in this field.
- The ***Record Access*** field allows you to designate a four-character alphanumeric field for security code. All record accesses upon this file within APPX processes check the contents of this field for access class. If the user does not have rights to the security code stored in this field for the desired record, APPX responds as if the record is not on file. The value in this field must be maintained by the designer and must be four numeric characters. If blank, there are no restrictions.

Files Default Process Option

The Default Input Process overlay appears on the screen when you invoke the DFLT PROC option from the Files/Fields screen. From this overlay, shown in *Figure 2-4-5,* you can optionally identify the APPX input process that is used to maintain this file. Upon completion, you are returned to the Files/Fields screen.

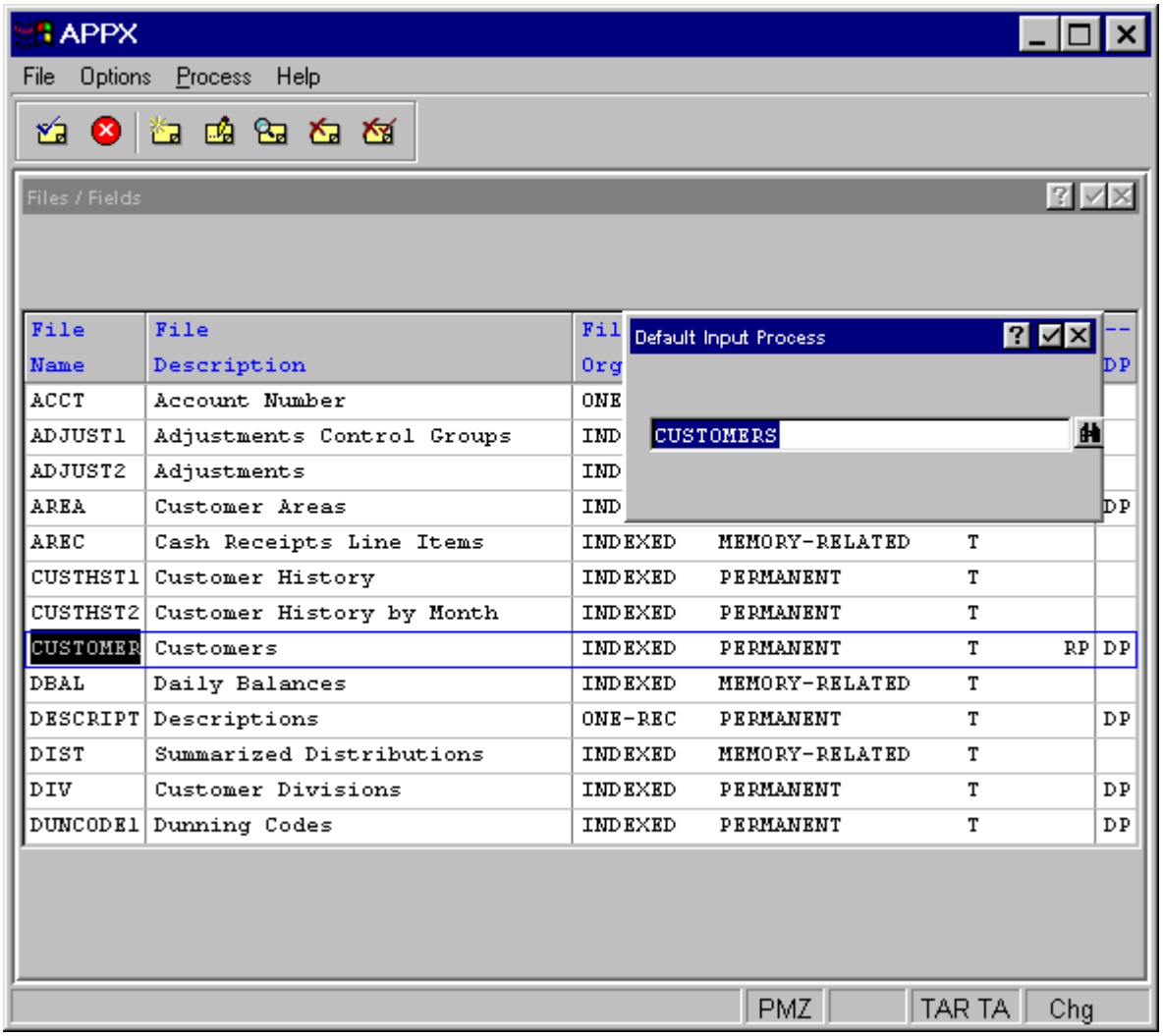


Figure 2-4-5. Files Default Input Process Overlay

Default Input Process is a navigational specification that links a field on an image to the input process that optionally executes from the *Scan* screen.

For example, assume that a user is entering a customer number on a sales order screen. The user has forgotten the appropriate customer number, and uses the SCAN option to display all of the customers and their numbers. At this point, the user discovers a problem with the customer information. If you defined a default process for the customer file, the user can elect to add a missing customer record or maintain an outdated customer record without ending the Sales Orders entry process. If you did not define a default process, the user must end Sales Orders entry, end the Sales Orders application, select the Accounts Receivable application, and select the Customer File Maintenance process to accomplish the same maintenance activity. Refer to the *User Manual* for a discussion of the SCAN option.

Fields

The Fields specifications screen, shown in *Figure 2-4-6*, establishes attributes that are common to all fields in APPX, regardless of their type. This screen automatically displays after you complete the entries in the Files/Fields specifications screen by selecting ENTER .

Fields screen

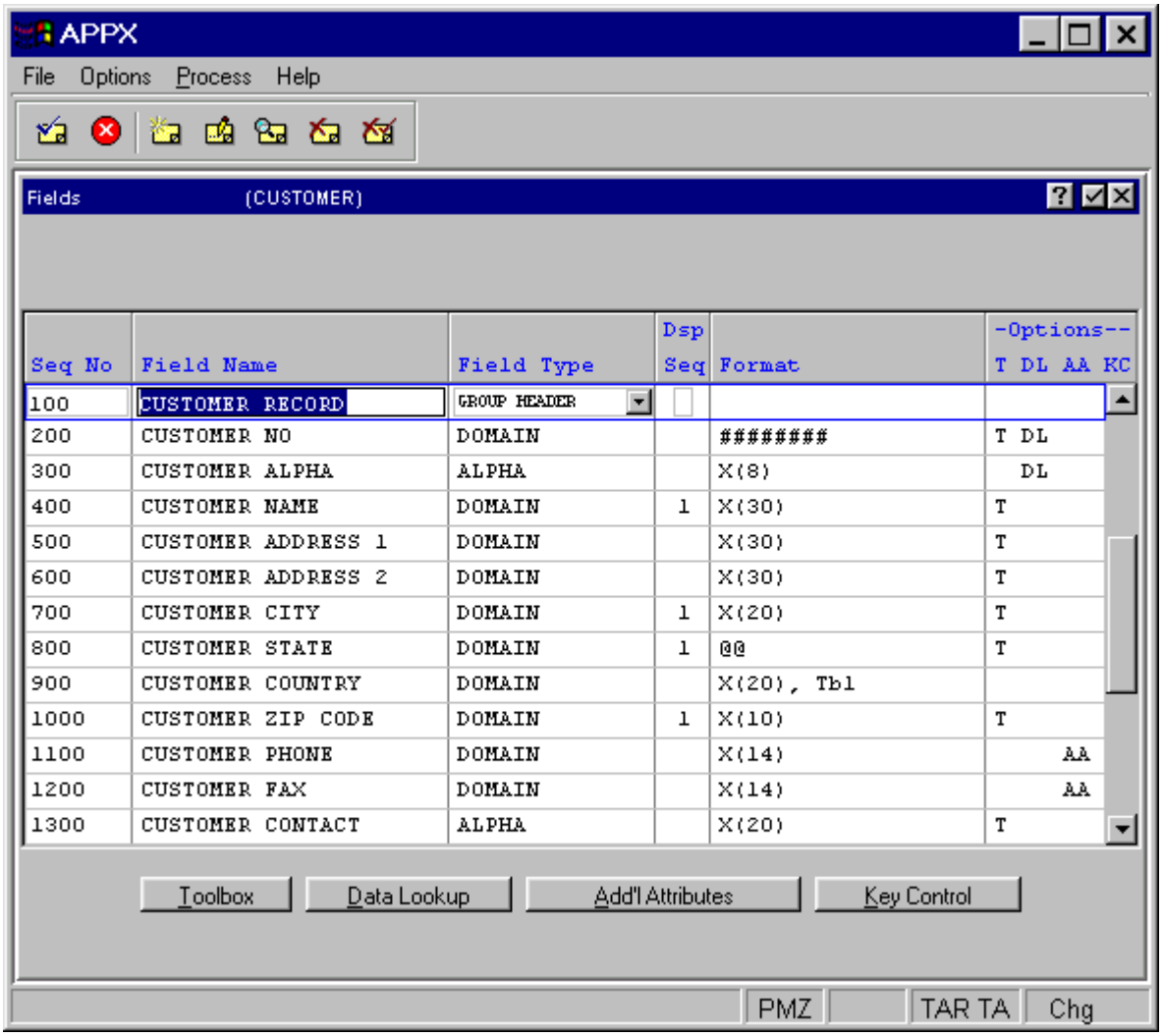


Figure 2-4-6

The *Fields* screen contains the following fields:

- ***Seq No*** displays a sequence number that is defaulted by APPX. This default sequence number can be overridden, but must be unique within a file. Sequence numbers determine the order of fields in the record structure. The first field that is defined in add mode is assigned the sequence number *100*, and additional fields are assigned a number that is equal to the highest existing sequence number, plus *100*.

- The sequence number of zero is reserved by APPX and cannot be assigned to a data dictionary field.
- Key numbers are assigned to primary and alternate keys based upon a key field's sequence number. Refer to the Special Uses for Synonym Fields section for a discussion of the use of synonym fields to alter key numbers.
- The sequence number is of particular significance to synonym and substring fields, and to group fields (group headers and group trailers). Synonym and substring fields must have sequence numbers higher than the sequence numbers of the fields they reference. All fields included in a group field must have sequence numbers that fall between the number of the group header and the group trailer or *Thru Field Name* field.
- ***Field Name*** specifies a unique identifier for each field in a record. As shown in the sample screen above, a common convention is to begin each field name with the name of the file.
- ***Field Type*** categorizes fields according to their function within APPX. The specifications that are presented on subsequent screen overlays depends on the *Field Type* entered here, although most share some common specifications. The *Field Type* options are:
- ***Domain***, which assumes the attributes that are defined for an alphanumeric, date, logic, or numeric domain.
- ***Alpha***, for fields that can contain numbers, letters, and special characters (such as -, #, or &).
- ***Format***, for fields that can contain an edit mask.
- ***System***, for fields, often referred to as audit fields, that display data that is provided by the system.
- ***Text***, for text fields.
- ***Token***, for fields with specific values that are stored as corresponding numbers.
- ***Numeric***, for fields restricted to number, decimal (.), and sign (-) entries.
- ***Date***, for fields that contain dates and/or times, from a hundredth of a second through a century.
- ***Logic***, for fields with possible values of yes ☑, no ☐, or ? to represent an unknown value.
- ***Synonym***, which provides an alternative name for referencing an existing APPX field.
- ***Substring***, which represents a discrete portion of a longer alphanumeric field.
- ***Group Header***, which marks the beginning and end of a series of fields in a group field or concatenation.
- ***Group Trailer***, a placeholder that optionally marks the final field in a concatenation, permitting future expansion of the group.
- If you enter a partial value, APPX selects the option with the next highest value. For example, an entry of *SUB* or simply *S* is equivalent to entering *SUBSTRING*. An entry of *A* references the next highest value: *ALPHA*. Each field type is described in detail later in this chapter.
- ***Dsp Seq***, or *display sequence*, if present, indicates this is a *descriptive field* and also determines the order of the field's appearance relative to other descriptive fields in the file. *Descriptive fields* are fields whose value is important in helping users identify specific records. For example, customer name, contact name, and telephone number fields may all be appropriate descriptive fields for a customer file. They are used in APPX-generated subheadings and on data lookup screens (such as

the Scan display when the user selects the SCAN option). Fields with duplicate display sequence numbers are displayed according to the field sequence numbers. If *Dsp Seq* is blank, the field is not treated as a descriptive field.

- ***Options*** are indicators that are present whenever documentation (T), data lookup (DL), additional attributes (AA), or key control (KC) have been defined for this field.

The Fields screen contains the following options as described in the corresponding sections:

- Toolbox
- Data Lookup
- Additional Attributes (this option is described separately by field type)
- Key Control

Fields Toolbox Option

The Fields Toolbox overlay, shown in *Figure 2-4-7*, displays when you select the TOOLBOX option from the Fields screen.

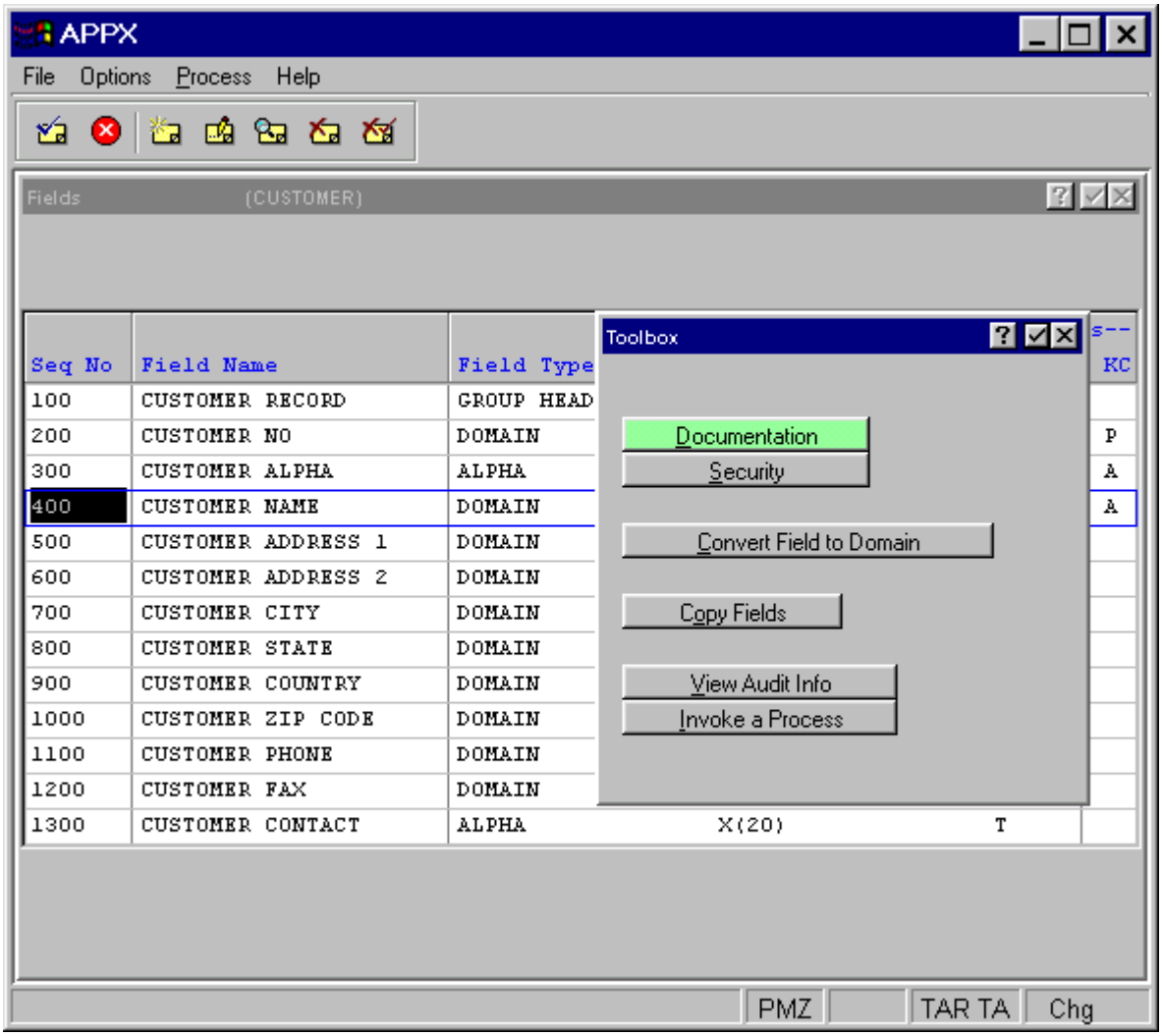


Figure 2-4-7. Fields Toolbox Overlay

The following options are available from the Fields Toolbox overlay:

Documentation

When you select the DOCUMENTATION option from the Toolbox overlay, you will first see the Documentation overlay which allows you to select between user or technical documentation. Once you have entered your documentation, you are returned to the fields screen.

Security

The SECURITY option allows you to identify the APPX security code to which a user must have rights in order to access the field. The security code must be four numeric characters. If blank, there are no restrictions.

Convert Field to Domain

This feature allows you to create a domain using the exact specifications of a designated field. When you select this option, you must first name the new domain. Then select the CREATE DOMAIN option and let APPX define a new domain using the characteristics of the currently selected field.

Copy Fields

This feature allows you to copy a field or sequence of fields into the data dictionary for the selected file. The selected fields to copy may be from the current file or another file. See the Fields Toolbox Copy Fields Option section for details.

View Audit Info

This feature allows you to see who created the currently selected field and who changed it most recently. It also gives the date of creation and the date it was last changed. Note that this option (`OPTION 98`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Invoke a Process

This feature is available to facilitate testing. It allows you to invoke any process. Note that this option (`OPTION 99`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Fields Toolbox Copy Fields Option

The Copy Fields overlay, shown in *Figure 2-4-8* below, displays when you select the COPY FIELDS option from the Fields Toolbox overlay. This feature allows you to copy a field or sequence of fields into the data dictionary for the selected file. The selected fields to copy may be from the current file or another file.

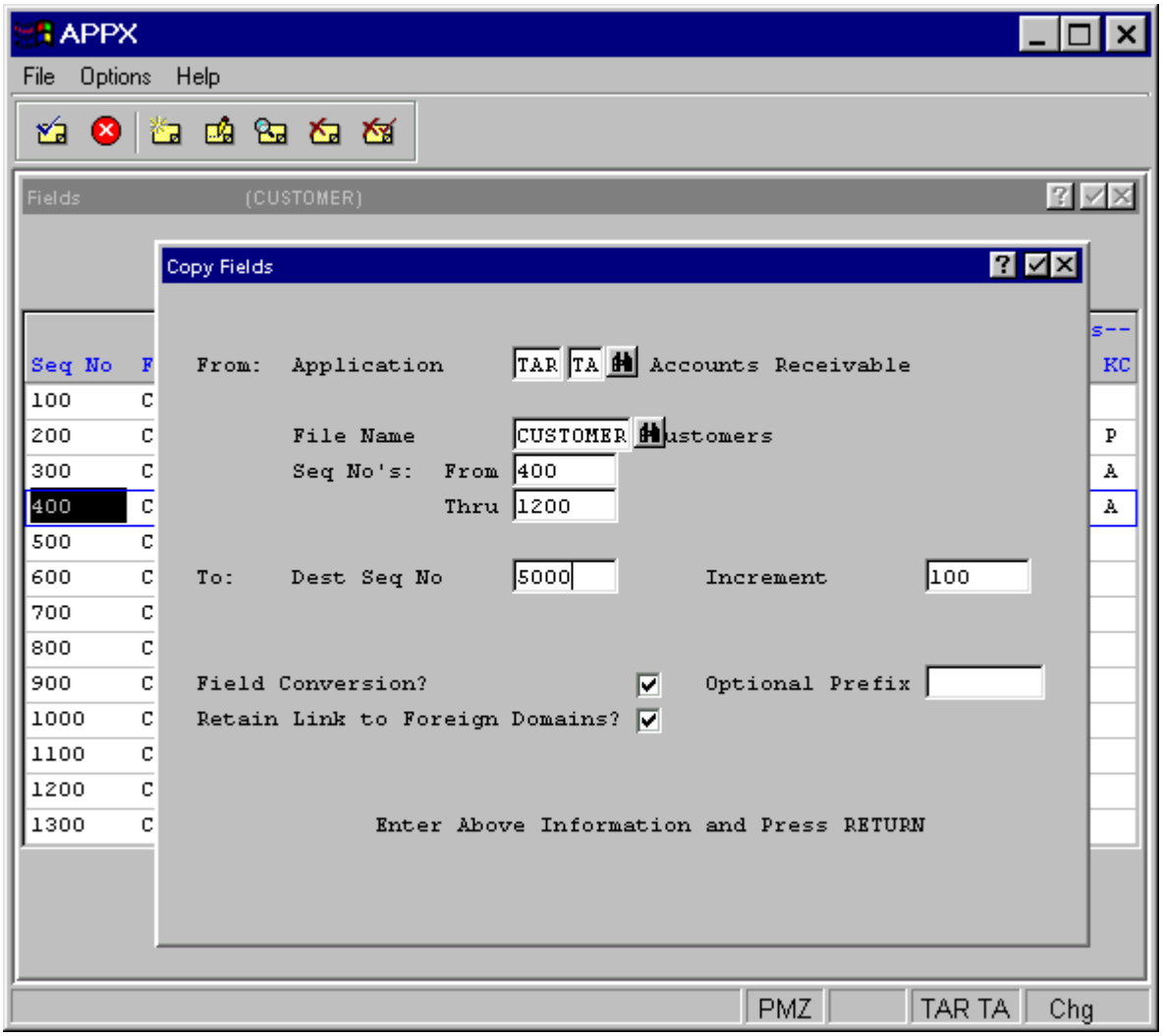


Figure 2-4-8. Copy Fields Overlay

To copy a field or fields into the current file's data dictionary, you must enter:

- *Application* ID and version corresponding to the location of the file you want to copy fields from.

File Name

Of the file you want to copy fields from.

Seq No's

Of the fields you want to copy. If you want to copy a single field, leave the "thru" sequence number blank.

Dest Seq No

Is the beginning sequence number for the new fields (or the single sequence number if you are copying just one field).

Increment

Is the increment to be used in determining successive sequence numbers if you are copying more than one field (a value of 100 will create sequence numbers 2000, 2100, 2200 . . . if you entered 2000 as your destination sequence number).

Field Conversion?

To control whether or not the field name should be converted to conform to APPX coding standards. Field names should begin with the name of the file that the field name is being copied into. Select the checkbox to toggle between "yes" ☑ and "no" ☐. This option should not be used if APPX coding standards have not been used in this record structure.

Optional Prefix

Is used in conjunction with *Field Conversion?* to substitute the destination field name with a different name for the field name prefix.

Retain Link To Foreign Domains?

Controls whether or not references to foreign domains (domains outside of the current application) will be retained. For example, if fields from application "AAA" are being transferred to application "BBB" then all references to "AAA" domains are considered foreign domains. The default value of this field is "yes". Select the checkbox to toggle between "yes" ☑ and "no" ☐. If "yes" is selected, references to foreign domains will be retained in the transferred fields. If "no" is selected, foreign domains will be "localized" or transferred to the current application. Domains already existing in the current application will not be overwritten but may be transferred separately if desired.

Fields Data Lookup Option

The DATA LOOKUP option from the Fields screen accesses the Fields Data Lookup Overlay, shown in *Figure 2-4-9* below. If the field can be used as a key to another file, you may wish to specify the file and key field for data lookup purposes. Data Lookup specifications for a field enable use of the SCAN option to look up a value for the field on an input image or on a record selection screen in a query. These specifications are defaulted as the Data Lookup values whenever the field is defined on an image. In addition, APPX uses this key path to access descriptive information when generating a standard subheading image for the field in an output process. Upon completion of Data Lookup entries, you are returned to the Fields screen.

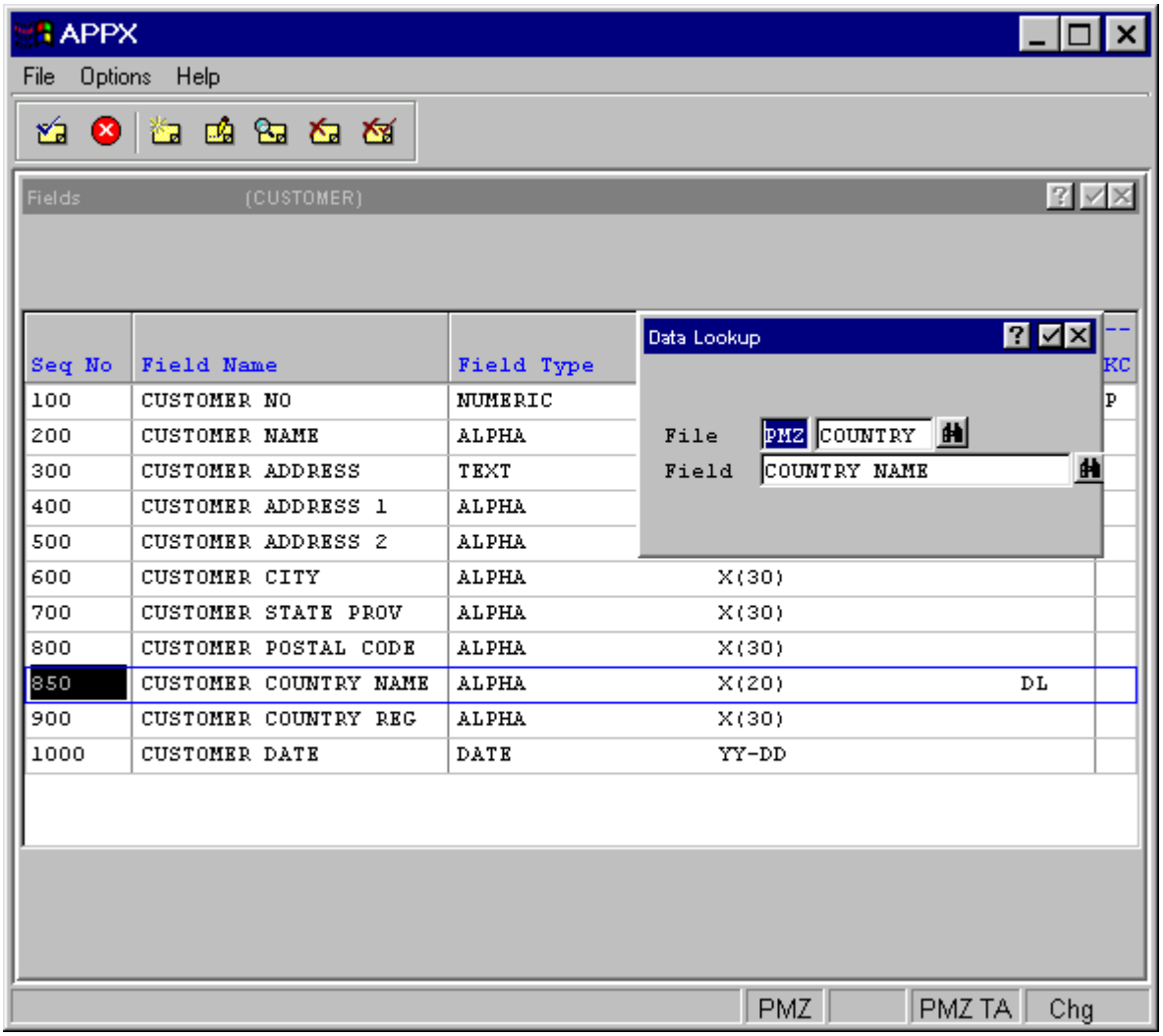


Figure 2-4-9. Fields Data Lookup Overlay

The Fields Data Lookup overlay contains the following fields:

File

Identifies the application ID and file name of the file to be accessed for data lookup.

Field

Identifies a primary or alternate key to the data lookup file. In addition, APPX uses a key field to access related information when generating a standard subheading image in an output process. The user can then look up and select the contents of this field by use of the SCAN option when entering data in input or query processes for the original field.

Fields Additional Attributes

The ADDITIONAL ATTRIBUTES option from the Fields screen accesses an Additional Attributes overlay, from which you can enter additional parameters for the field. The specific overlay varies by field type. Each overlay is presented later in this chapter according to the respective type of field. Upon completion, you are returned to the Fields screen.

Fields Key Control Option

KEY CONTROL accesses the fields Key Control overlay, shown in *Figure 2-4-10*, from which you can define attributes about key fields. This option applies only to key fields (both primary and alternate) for indexed files. For non-key fields, this option is ignored unless a field is part of a group field that is itself a key to an indexed file. Upon completion, you are returned to the Fields screen.

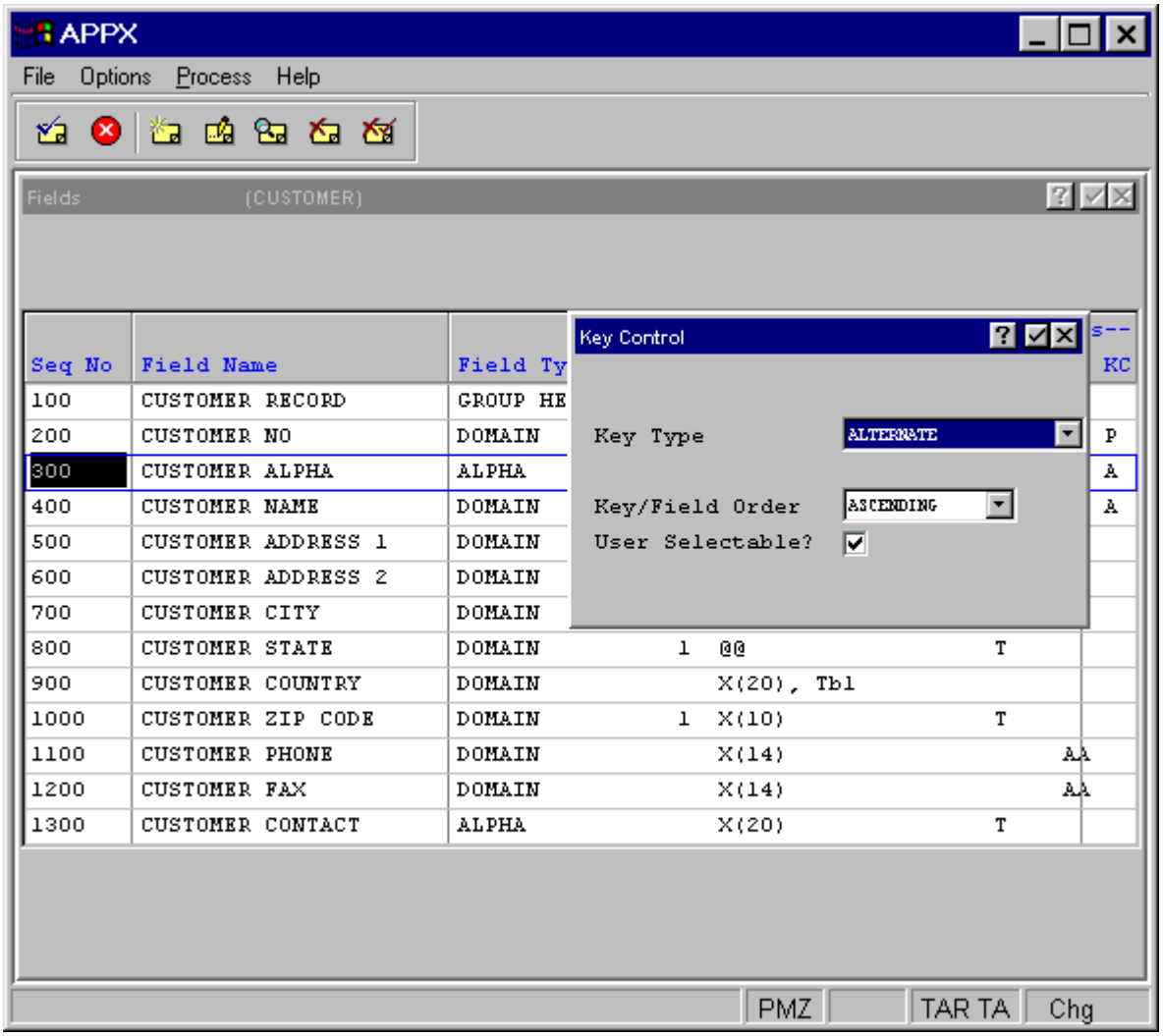


Figure 2-4-10. Fields Key Control Overlay

The fields Key Control overlay contains the following fields:

Key Type

Indicates whether or not a field is a key field and, if yes, the type of key. A group trailer field cannot be a key field. The *Key Type* options can be scanned and consist of:

PRIMARY

Key fields are the default key paths for a file. APPX requires one (and only one) primary key for each indexed file. All records can be accessed by a primary key path (must be unique).

ALTERNATE

Key fields optionally provide alternative key paths to a file. Duplicate values and up to 16 alternate keys are permitted in APPX.

ALTERNATE/UNIQUE

Key fields optionally provide alternative key paths to a file. Duplicate values are not allowed.

ALTERNATE/FORCED

Key fields are the same as ALTERNATE key fields except that duplicate keys are retreived in primary key order.

- The default is blank.

Key/Field Order

Specifies the order of the key or key segment.

User Selectable?

Determines whether or not a user can switch to a key path with the *Select Access Path* option from within an input process. The *User Selectable?* options are “yes” , a user can select the key path; or “no” where a user cannot select the key path. Its use is restricted to internal access (statements). The default is “yes”.

Common Fields Specifications

When you select ENTER from the Fields screen, a Fields specifications overlay displays. The specific Fields specifications overlay that appears depends upon the type of field that is currently selected. However, there are some specifications that are shared by most field types (either on the specifications overlay or the additional attributes overlay). These include the following as described in the corresponding sections:

- Occurrence

For information about type-specific Fields Specifications overlays, refer to the following sections which discuss each type in detail:

- Domain Fields
- Alpha Fields
- Format Fields
- System Fields
- Text Fields
- Token Fields
- Numeric Fields
- Date Fields
- Logic Fields
- Synonym Fields
- Substring Fields
- Group Header Fields
- Group Trailer Fields

Occurrence

Occurrence determines the number of times a field is present in a record. If blank, APPX assumes a value of 1. This field may contain up to five numeric characters and is restricted to a maximum of 32,767.

A multiple-occurrence field is simply an alternative to defining multiple fields in a file. For example, a balance record in a General Ledger application may contain an ending balance field that is repeated 12 times, one for each month in the year. You could define 12 fields, January Ending Balance, February Ending Balance, March Ending Balance, and so on, or you could define one ending balance field with 12 occurrences.

APPX considers each occurrence as a separate field, uniquely identified by the combination of application ID, field name, and occurrence number. Each occurrence of a multiple-occurrence field is no different from a single-occurrence field in that:

- APPX allocates unique physical storage for each occurrence.
- You can paint a specific occurrence on an image.
- Statements operate on specific occurrences of a field.

Unlike separate fields, however, if a multiple-occurrence field is included in a group, all occurrences become part of the group. It is not possible to selectively include specific occurrences in a group, while excluding other occurrences. However, see Special Uses for Group Fields for some creative ways to manipulate groups and occurrences.

There are two primary advantages of using multiple-occurrence fields. Multiple-occurrence fields:

- Minimize the effort that is required to define the data dictionary. You enter a single field definition rather than a separate field definition for each occurrence.
- Minimize the effort that is required to manipulate data using APPX statements.

You can exert less control over multiple-occurrence fields than you can over single-occurrence fields because you:

- Must include all occurrences in a group.
- Cannot define a synonym for a multiple-occurrence field.
- Cannot establish a multiple-occurrence substring.

Domain Fields

A domain-type field assumes the attributes that are defined for any type of domain in the same or an outside application. One of the Domain-Type Field Specifications overlays automatically displays after you complete the entries on the Fields specifications screen if the *Field Type* you select is *DOMAIN*. *Figure 2-4-11* below displays if the domain type is *alpha, format, text, token, date,* or *logic*. *Figure 2-4-15* displays if the domain type is *system*. *Figure 2-4-16* displays if the domain type is *numeric*. If the domain type is *same as*, the domain type of the root domain determines which screen is displayed next.

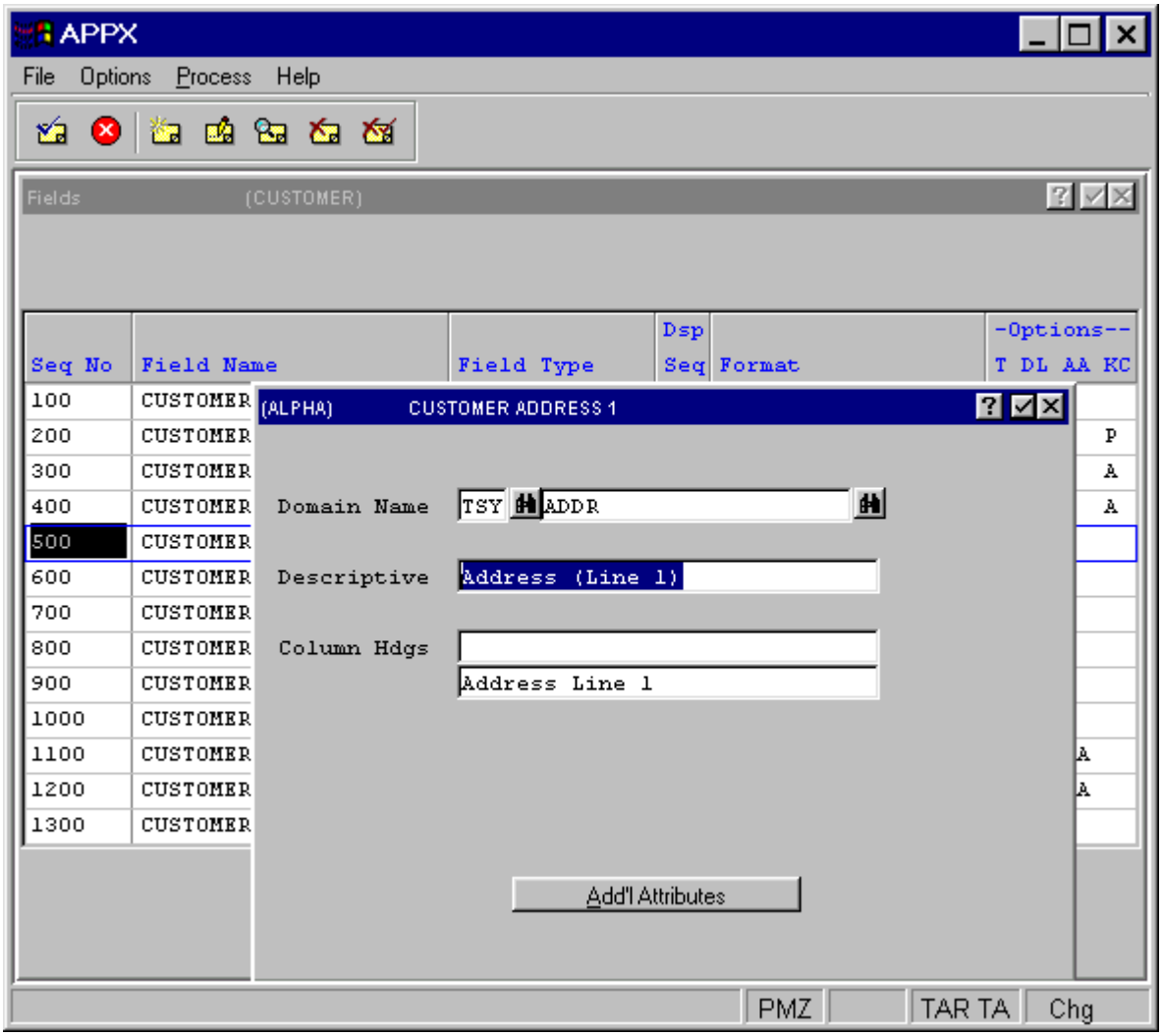


Figure 2-4-11. Domain Field Specification Overlay (Alpha Domain)

The following fields are contained on the Domain-Type Field Specifications overlay for Alpha, Format, Text, Token, Date, and Logic domains:

Domain Name

Links a domain-type field to a specific domain in the same or another APPX application.

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used. Because *Descriptive* is less cryptic and can be more flexibly formatted than *Field Name*, it is easier to understand, and so the system uses *Descriptive* when communicating with users. *Descriptive* is used, for example, when the value of a field displays to a user on a standard image, such as the *Select Access Path* display or an APPX-generated subheading. We suggest that you use a singular entry for *Descriptive,* and avoid appending the word “field”.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

The Domain-Type Field Specification overlays for Alpha, Format, Text, Token, Date, and Logic domains contain the following option described in the corresponding section:

- Additional Attributes

Domain Fields Additional Attributes Option

ADDITIONAL ATTRIBUTES accesses the domain-type field Additional Attributes overlay, from which you can enter additional parameters for this domain-type field. Upon completion, you are returned to the Domain Fields specifications overlay.

The Domain-Type Field Additional Attributes overlay for *alpha*, *text*, *token* and *date* Domains is shown in *Figure 2-4-12*.

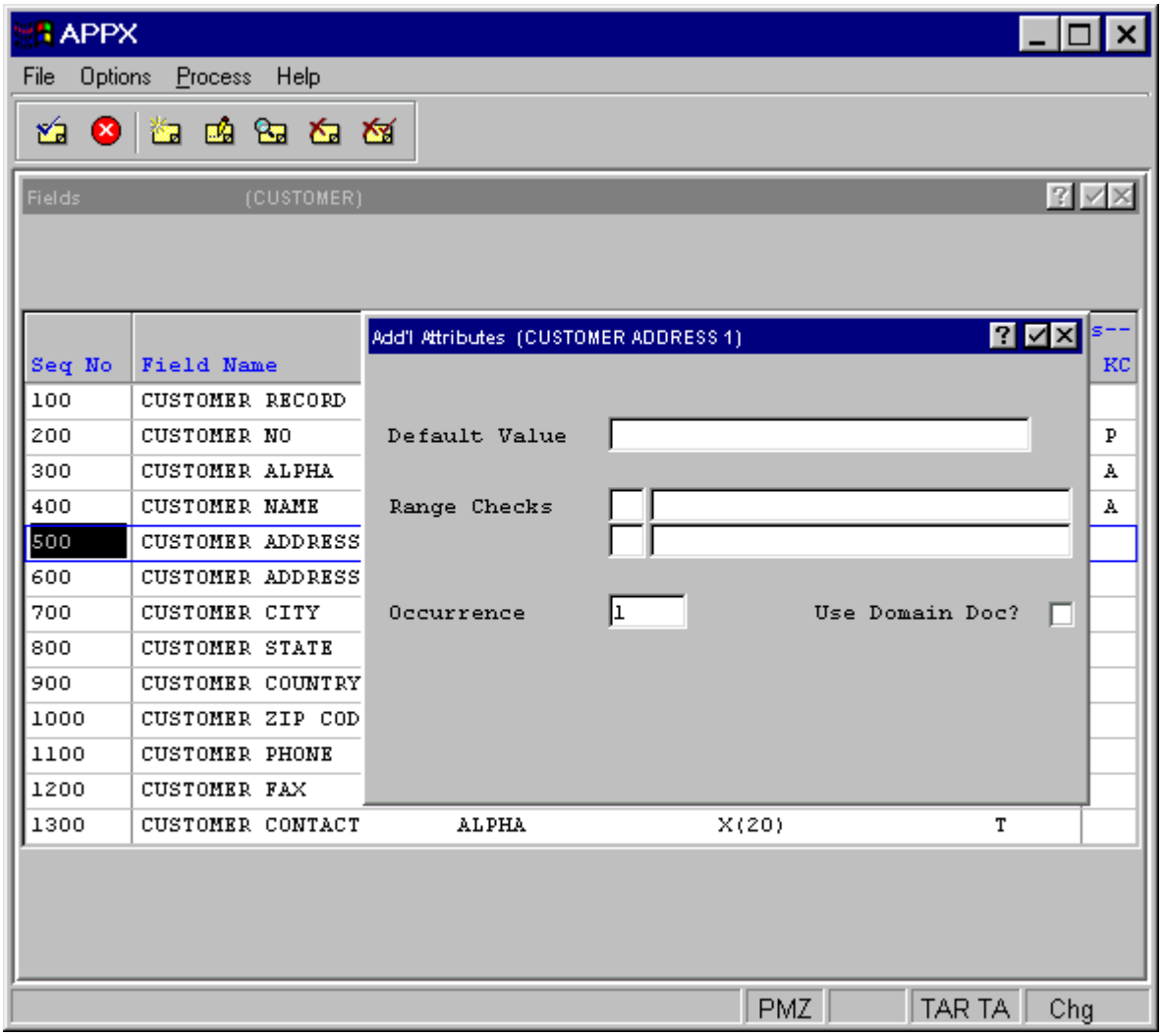


Figure 2-4-12. Domain Field Additional Attributes Overlay for Alpha, Text, Token and Date Domains

The Domain-Type Field Additional Attributes overlay for *format* Domains is shown in *Figure 2-4-13*.

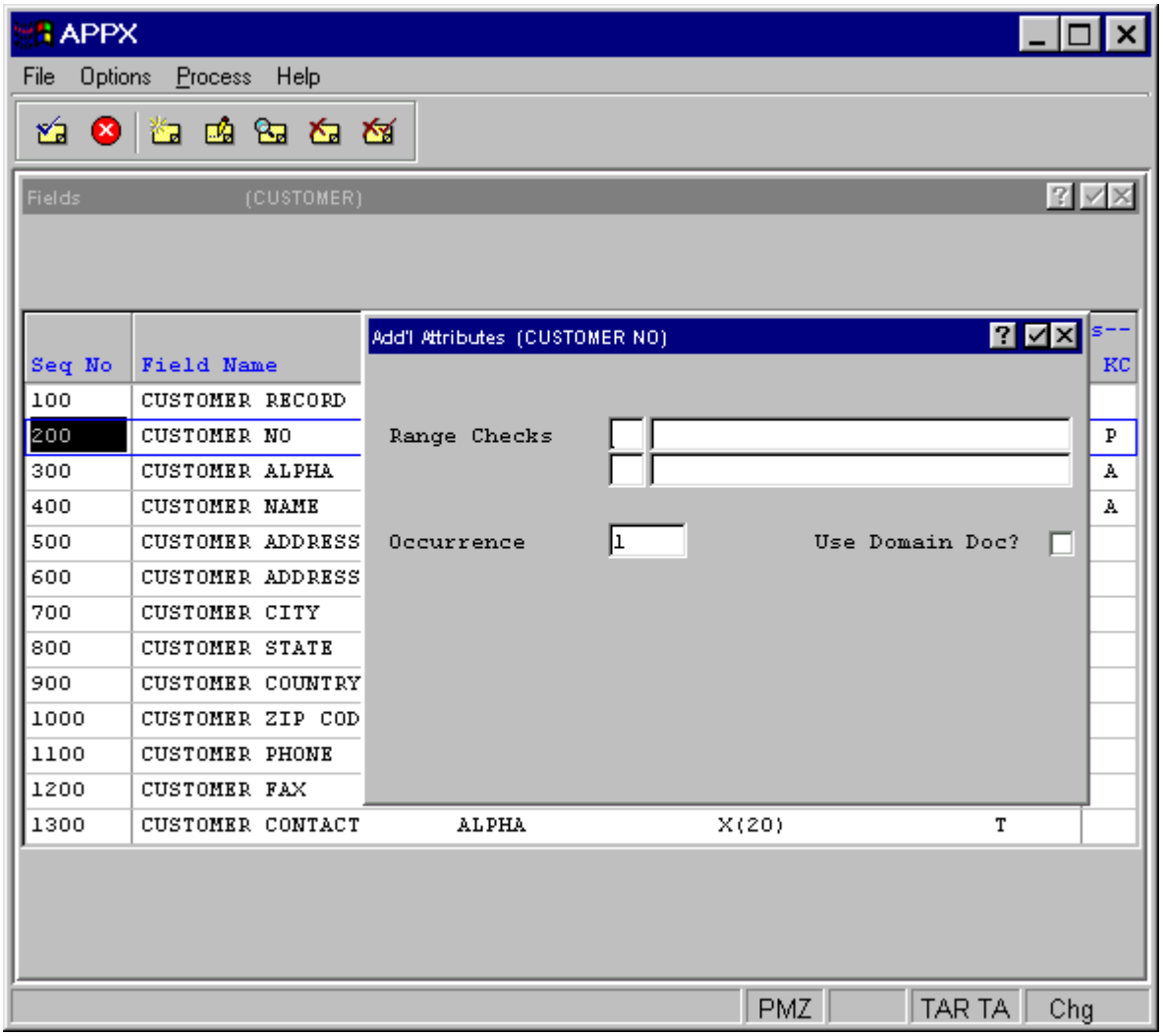


Figure 2-4-13. Domain Field Additional Attributes Overlay for Format Domains

The Domain-Type Field Additional Attributes overlay for *logic* Domains is shown in *Figure 2-4-14*.

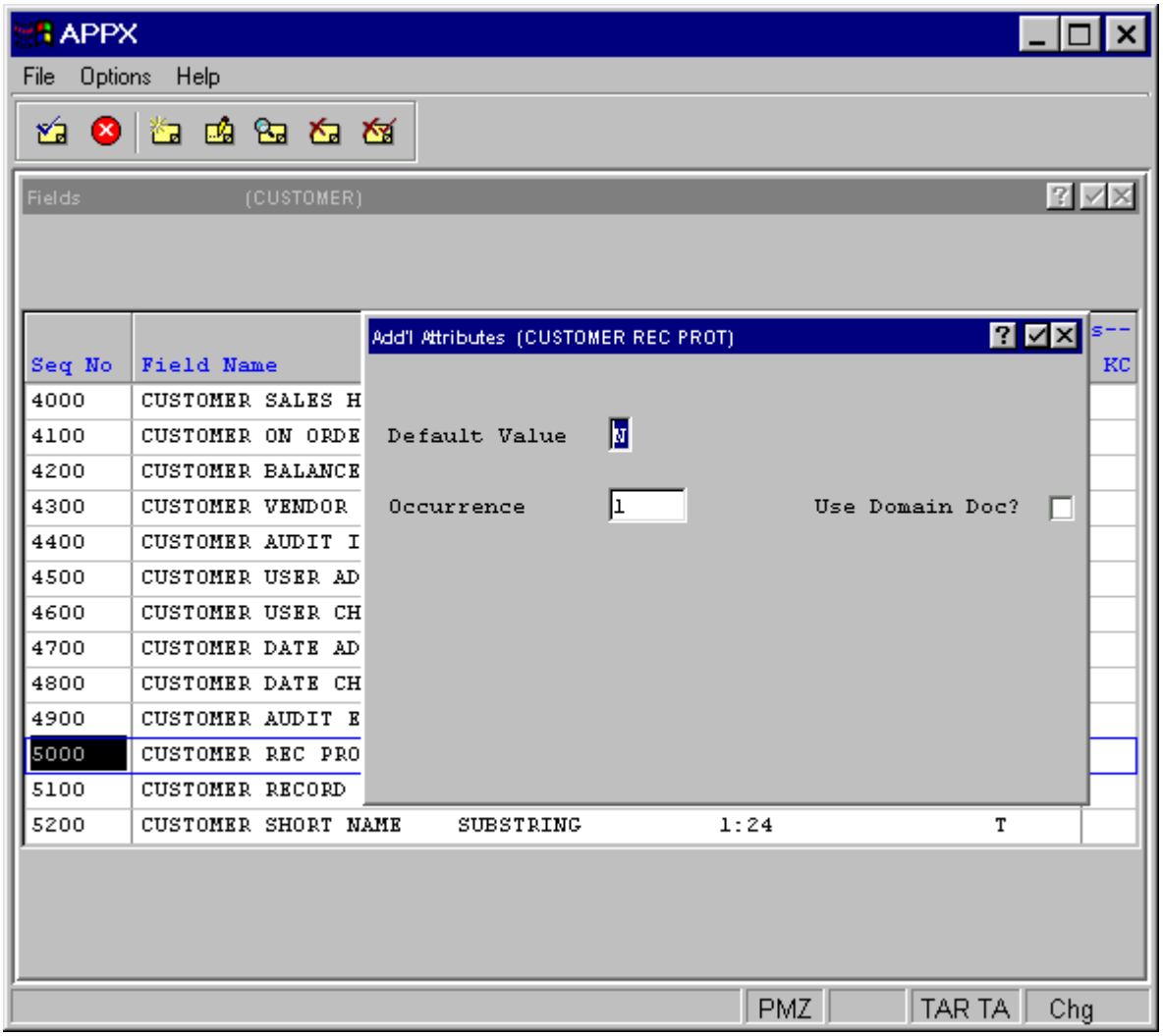


Figure 2-4-14. Domain Field Additional Attributes Overlay for Logic Domains

Some or all of the following Fields are contained on the Domain-Type Field Additional Attributes overlays for *alpha*, *format, logic, text*, *token* and *date* Domains:

Default Value

Establishes a modifiable starting value for all referenced fields whenever they are first displayed in add mode in an input image, unless modified by a user. If blank, there is no starting value displayed on the image.

Range Checks

Defines a series of allowable values that APPX uses for online data validation. The first component represents the relation. The options are ***EQ*** (equal to), ***NE*** (not equal to), ***GT*** (greater than), ***LT*** (less than), ***GE*** (greater than or equal to), and ***LE*** (less than or equal to). The second component represents the value that is assigned to the relation. If you enter two relations, APPX infers an AND relationship.

Occurrence

Determines the number of times a field is present in a record. If blank, APPX assumes a value of 1. This field may contain up to five numeric characters and is restricted to a maximum of 32,767. See the Occurrence section for a more comprehensive discussion of this field.

Use Domain Doc?

Toggle the checkbox between "yes" ☑ if you want APPX to use the documentation from the domain as Help text and "no" ☐ if you plan to enter Help text unique to this field.

Domain Fields (System Domains)

The Domain-type field specification overlay, shown in *Figure 2-4-15*, displays after you select ENTER from the fields screen if the *Field Type* you select is *DOMAIN* and the domain type is *system*.

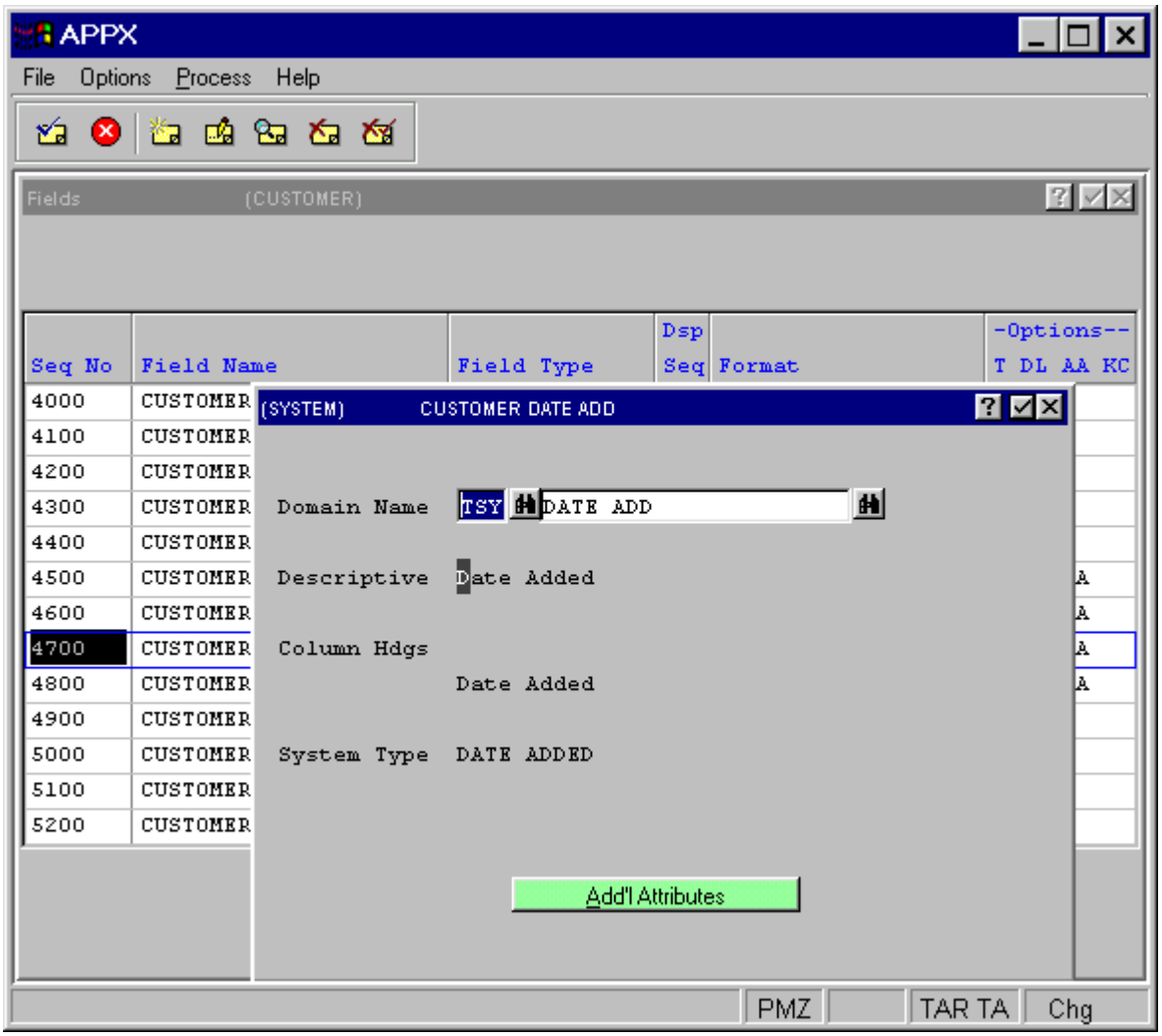


Figure 2-4-15. Domain Field Specification Overlay (System Domain)

The following fields are contained on the Domain Field Specifications overlay for *system* Domains:

Domain Name

Links a domain-type field to a specific domain in the same or another APPX application.

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used. Refer to the definition of *Descriptive* in the *Domain Fields* section for more information.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

- ***System Type*** — An System domain can be used to identify the user ID of the operator who adds or modifies a record in the database. The *System Type* options are ***USER ADDED***, which identifies the user ID of the operator who added the record; ***USER CHANGED***, which identifies the user ID of the operator who modified the record; ***DATE ADDED***, which identifies the date that the record was added; and ***DATE CHANGED***, which identifies the date the record was last modified.

The ADDITIONAL ATTRIBUTES option accesses the Additional Attributes overlay which contains only one field:

Use Domain Doc?

Toggle the checkbox between "yes" ☑ if you want APPX to use the documentation from the domain as Help text and "no" ☐ if you plan to enter Help text unique to this field.

Domain Fields (Numeric Domains)

The Domain type field specification overlay, shown in *Figure 2-4-16*, displays after you select ENTER from the fields screen if the *Field Type* you select is *DOMAIN* and the domain type is *numeric*.

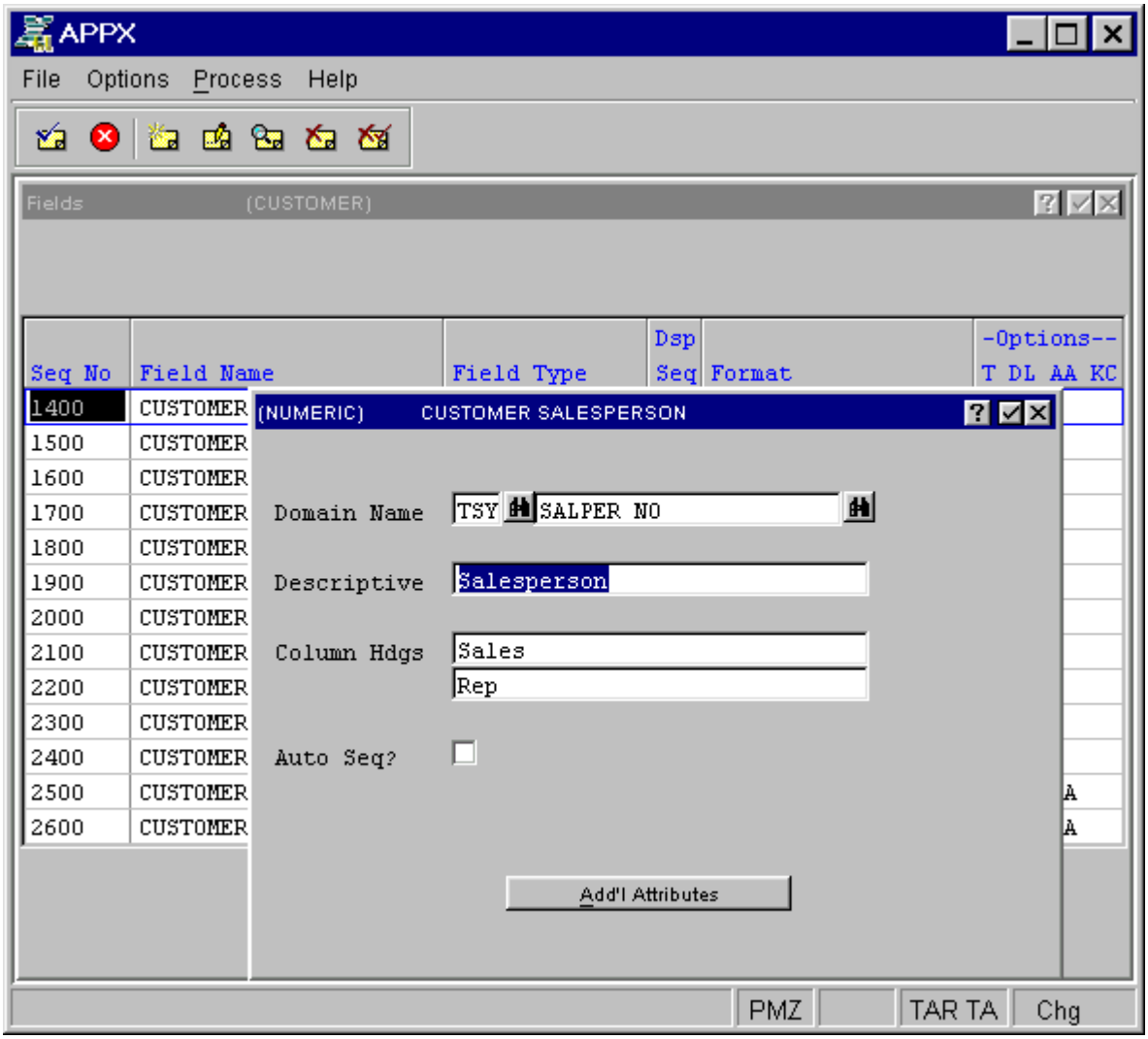


Figure 2-4-16. Domain Field Specification Overlay (Numeric Domain)

The following fields are contained on the Domain-Type Field Specifications overlay for *numeric* Domains:

Domain Name

Links a domain-type field to a specific domain in the same or another APPX application.

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

Auto Seq?

Select "yes" ☑ if this numeric field represents a sequence number with a value that should be maintained by APPX.

One option, ADDITIONAL ATTRIBUTES, is also available on this screen. This option accesses the Additional Attributes overlay for *numeric* Domain Fields shown in *Figure 2-4-17.* Upon completion, you are returned to the numeric Domain-type field specifications overlay.

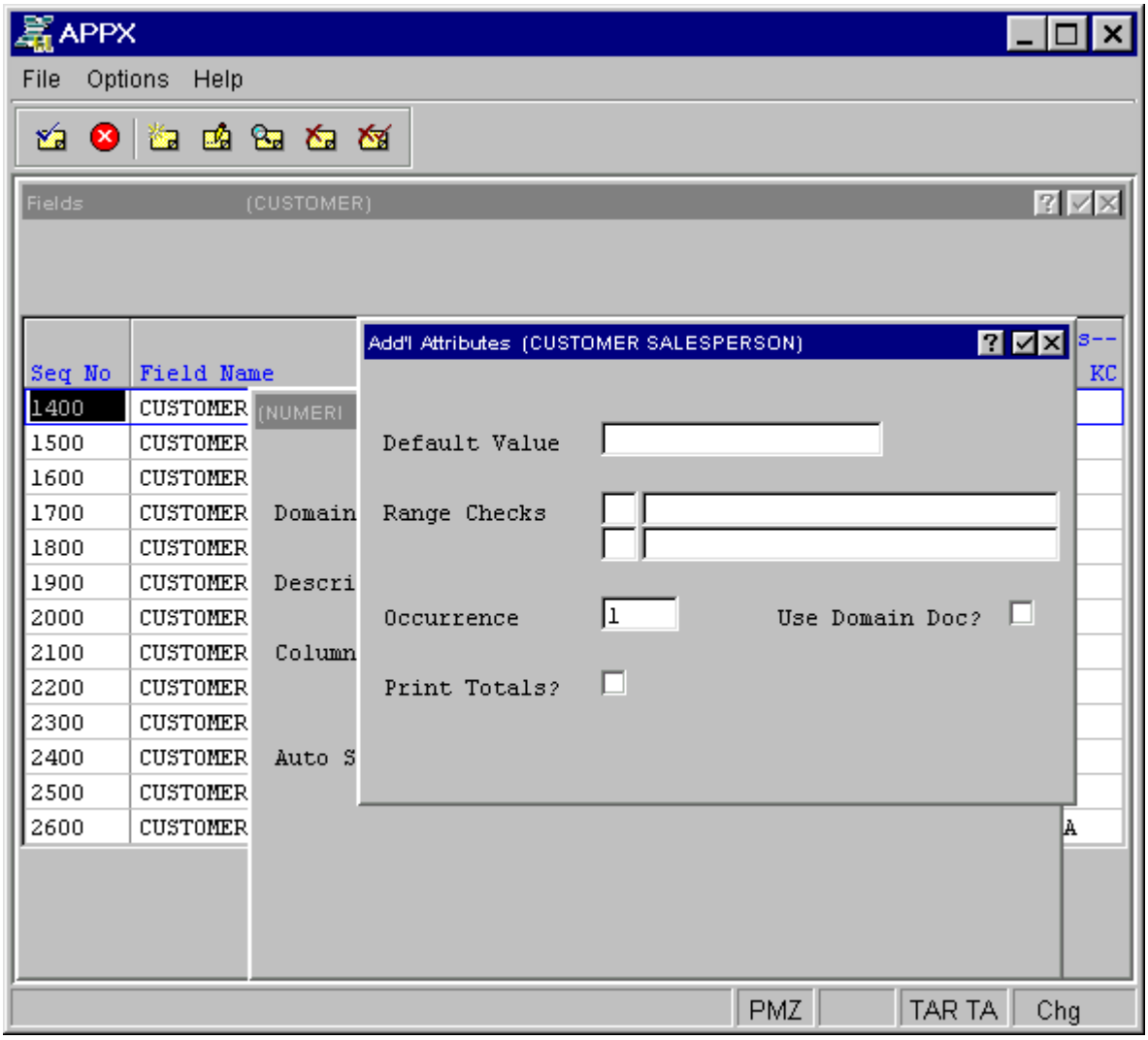


Figure 2-4-17. Domain Field Additional Attributes Overlay for Numeric Domains

The Additional Attributes overlay for *numeric* Domain Fields contains the following:

Default Value

Establishes a modifiable starting value for all referenced fields whenever they are first displayed in add mode in an input image, unless modified by a user. If blank, there is no starting value displayed on the image.

Range Checks

Defines a series of allowable values that APPX uses for online data validation. The first component represents the relation. The options are ***EQ*** (equal to), ***NE*** (not equal to), ***GT*** (greater than), ***LT*** (less than), ***GE*** (greater than or equal to), and ***LE*** (less than or equal to). The second component represents the value that is assigned to the relation. If you enter two relations, APPX infers an AND relationship.

Occurrence

Determines the number of times a field is present in a record. If blank, APPX assumes a value of 1. This field may contain up to five numeric characters and is restricted to a maximum of 32,767. See the Occurrence section for a more comprehensive discussion of this field.

Use Domain Doc?

Click the checkbox to toggle between "yes" ☑ if you want APPX to use the documentation from the domain as Help text and "no" ☐ if you plan to enter Help text unique to this field.

Print Totals?

Click the checkbox to toggle between "yes" ☑ and "no" ☐ to indicate if this field should be automatically totaled on reports.

Alpha Fields

An Alpha field can contain numbers, letters, and special characters, such as *#, $,* and *&*. Examples include names, addresses, certain account numbers or serial numbers, and text.

The Alpha fields overlay, shown in *Figure 2-4-18*, allows you to add Alpha specific attributes to those already defined for fields. The alpha field overlay displays after you complete the entries on the Fields specifications screen if the *Field Type* you select is *ALPHA*.

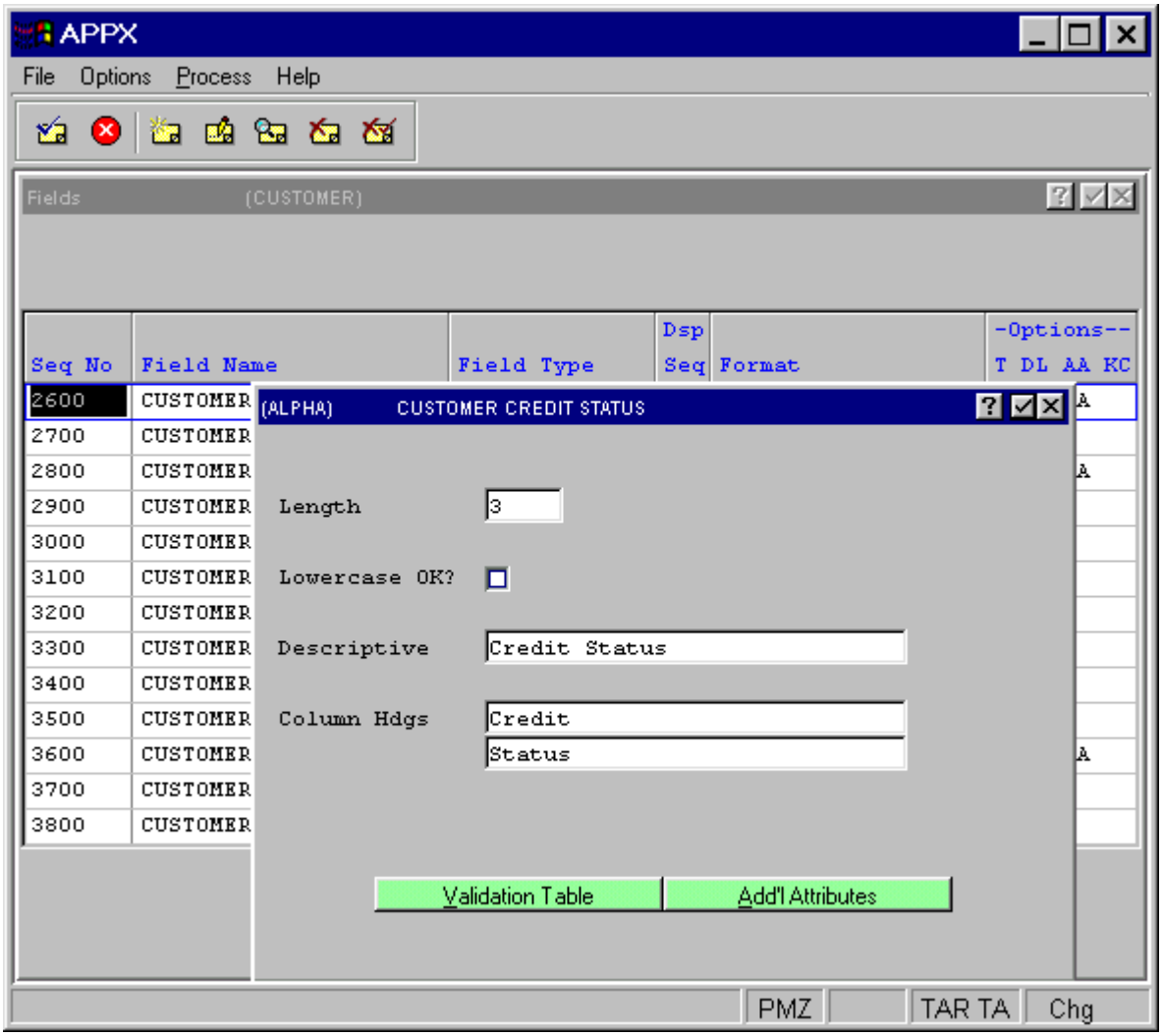


Figure 2-4-18. Alpha Fields Overlay

The Alpha Fields overlay contains the following fields:

Length

Designates the number of character positions, or length, of this field up to 32,767 (32K) characters.

Note: APPX does not support fields (or records) longer than 32K characters. If you map an APPX file to a pre-existing RDBMS table, you may find that the table includes a column larger than 32K; such columns are often referred to as memo columns. APPX offers limited support for memo columns. You can access a memo column by defining (in the APPX data dictionary) an ALPHA X(***n***) field in the corresponding position.

When you read a record that includes a memo column, APPX will only read the first ***n*** characters and will truncate any remaining data. If you read a record *with hold* and APPX would have to truncate a memo field, the read will fail. Reading a record *with hold* implies that you may try to update the record and APPX will try to prevent you from losing data through truncation. If you need to delete a record that would include truncated data, use partial-record I/O to avoid truncation. You can also use *partial-record I/O* to update other fields in the record. In general, you should avoid reading memo columns except in those cases where you actually need the data.

Lowercase OK?

Indicates whether or not a modifiable item on an interactive screen can be entered in upper- and lowercase letters. If 'no" ☐, the entry is stored and displayed exclusively as uppercase, regardless of how a user enters the information. This entry does not affect statements. A lowercase value established with a statement displays as lowercase, even if this specification prohibits lowercase entries.

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used. Refer to the definition of *Descriptive* in the Common Domain Specifications section of *Chapter 2-3* for more information.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

The Alpha Fields overlay provides two options which are described in the corresponding sections:

- VALIDATION TABLE
- ADDITIONAL ATTRIBUTES.

Alpha Fields Validation Table

The VALIDATION TABLE option from the Alpha Fields overlay accesses the Validation Table overlay, shown in *Figure 2-4-19*. A validation table provides a more powerful alternative to the use of range checks for data validation when there is a known set of values. Through the list feature, it provides the user with visibility of all the possible field values as well as the ability to select the desired value.

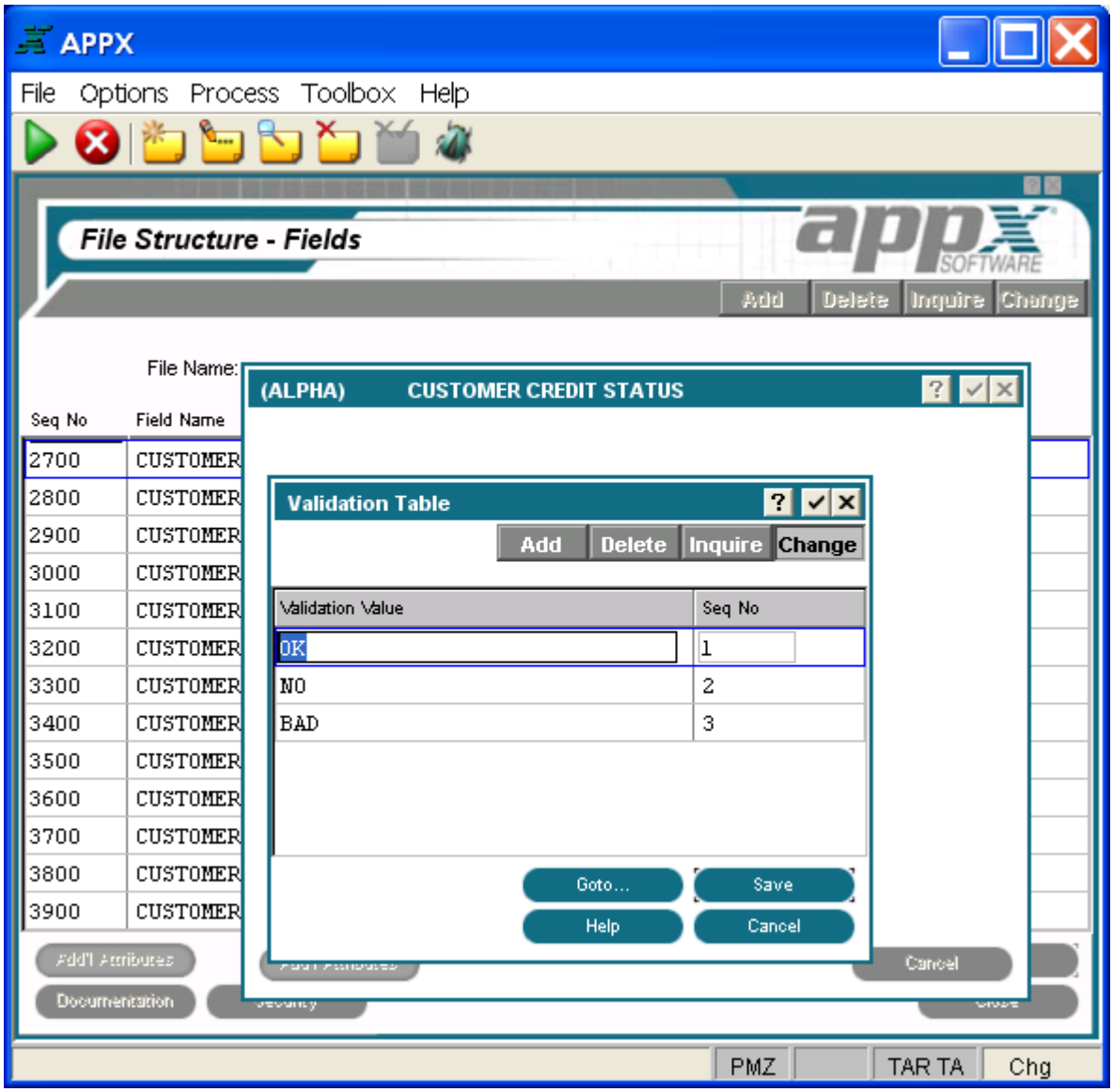


Figure 2-4-19. Alpha Fields Validation Table Overlay

You can establish a validation table for any Alpha field in the data dictionary. The entries in the table are case-sensitive. If you enter a validation value in all capital letters, an application user must enter the value in all capital letters.

Validation table operations are database-independent. They access a designer-defined table in application design, not data contained in a file in a database, for the range of possible values. As a result, validation does not vary from database to database. It is automatically available to, and exactly the same for, every database that is associated with an application. All validation values are available for review and selection by the end user through the list feature.

The Validation Table overlay contains the following fields:

Validation Value

Provides a valid field value. It is case-sensitive and is used for online review, selection, and validation. Note that validation values newly added to the data dictionary are not available to a current APPX user until the user exits and re-enters APPX.

Seq No

Provides an identifying number for the associated validation value. It is defaulted by APPX and can be overridden, but it must be unique within a table.. It determines the order in which validation values are presented, both for selection by the user and for editing in application design. APPX assigns the first validation value a number of *1*, with additional values assigned a number that is equal to the highest existing sequence number plus *1*. APPX does not require that sequence numbers be contiguous within a table.

Alpha Fields Additional Attributes

From the Alpha field overlay, you have an ADDITIONAL ATTRIBUTES option. The Alpha field Additional Attributes overlay is shown in *Figure 2-4-20*.

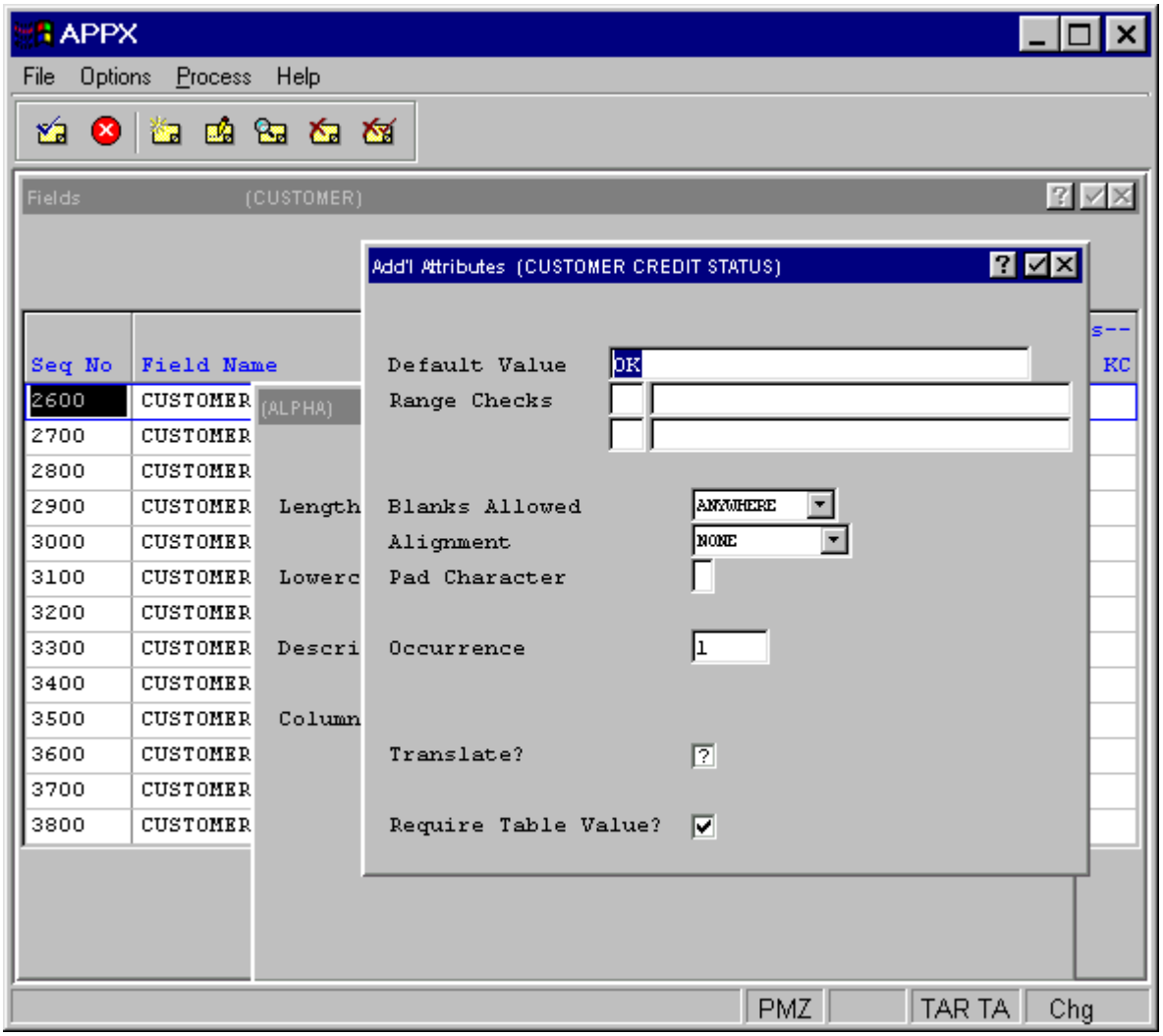


Figure 2-4-20. Alpha Field Additional Attributes Overlay

The Alpha Field Additional Attributes Overlay contains the following fields:

Default Value

Establishes a modifiable starting value for all image items whenever they are first displayed in add mode in an input image, unless modified by a user. If blank, there is no starting value displayed on the image.

Range Checks

Defines a series of allowable values that APPX uses for online data validation. The first component represents the relation. The options are ***EQ*** (equal to), ***NE*** (not equal to), ***GT*** (greater than), ***LT*** (less than), ***GE*** (greater than or equal to), and ***LE*** (less than or equal to). The second component represents the value that is assigned to the relation. If you enter two relations, APPX infers an AND relationship.

Blanks Allowed

Indicates whether or not blanks are permitted and, if so, where they can be placed. The *Blanks Allowed* options are ***NONE, TRAILING,*** and ***ANYWHERE***. The default is *ANYWHERE*.

Alignment

Determines whether or not data in a field is aligned for storage and/or display and, if so, the type of alignment. The *Alignment* options are ***NONE, LEFT, JUSTIFIED, RIGHT*** and ***CENTERED.*** The default is *NONE*.

Pad Character

Indicates the character to be used instead of leading and trailing spaces in a field.

Occurrence

Determines the number of times a field is present in a record. If blank, APPX assumes a value of 1. This field may contain up to five numeric characters and is restricted to a maximum of 32,767. See the Occurrence section for a more comprehensive discussion of this field.

Translate?

Indicates if the value of the alpha field should be translated into a designated language if APPX's Language Translation feature is being used. To qualify for translation, an alpha field must have a validation table defined; otherwise, this option cannot be selected. Click the checkbox to toggle between "yes" ☑, "no" ☐, and unknown ? values. If the translation feature is not in use, retain the default value of ?.

Require Table Value?

Indicates whether or not to require one of the validation table values to be entered. To force the user to enter one of the defined values, click the checkbox to toggle it to "yes" ☑. If a "no" ☐ value is selected, the validation table shown when the user selects the list feature, is advisory only and the user can enter a value not listed in the table. If there is no validation table defined, this option does not apply and cannot be selected. If *Translate?* is ☑, this option should also be ☑.

Format Fields

A format type field is like an alpha field except that its primary characteristic is an edit mask.

The Format Field overlay, shown in *Figure 2-4-21*, displays after you complete the entries on the Fields specifications screen if the *Field Type* you select is *FORMAT*.

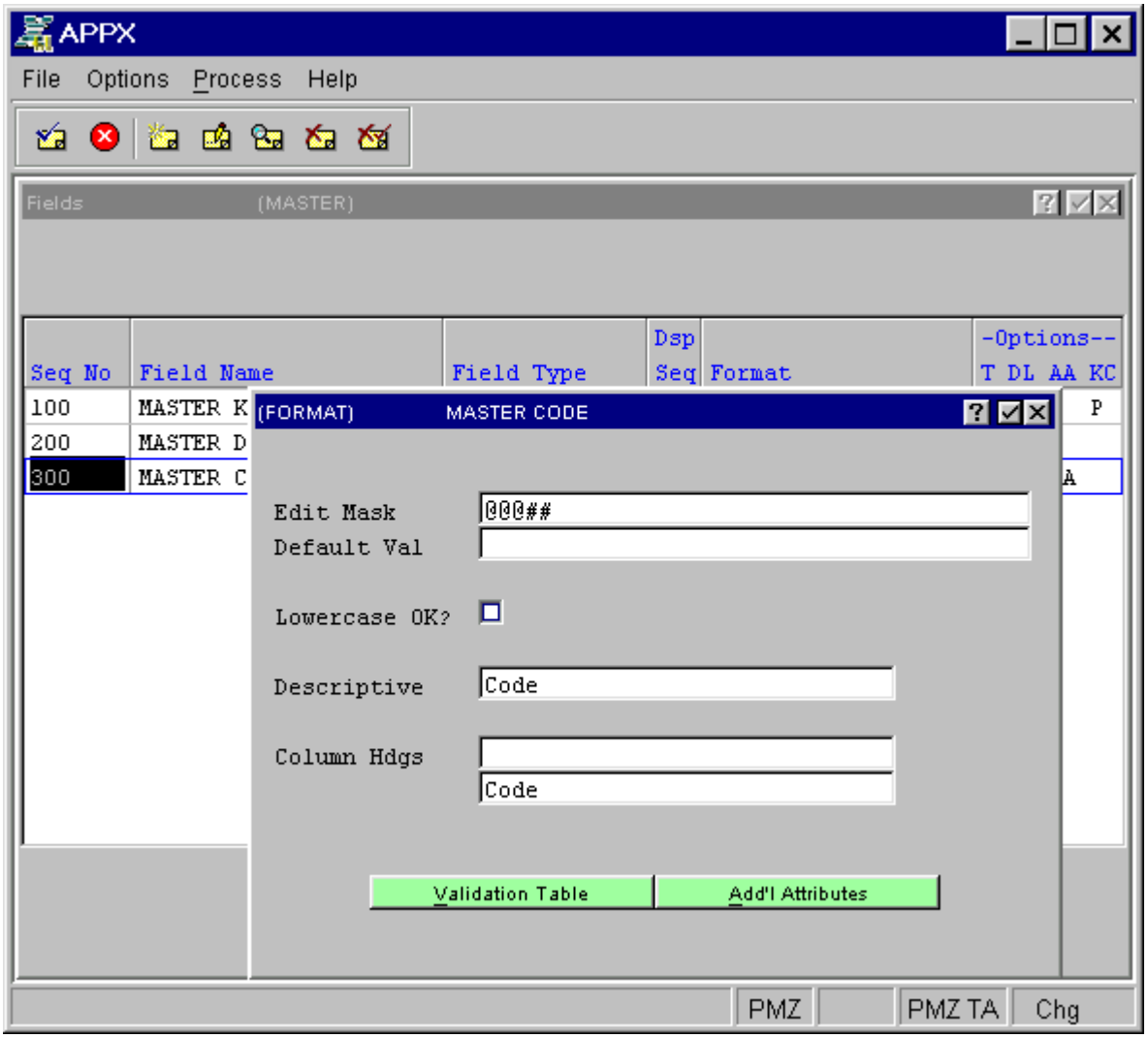


Figure 2-4-21. Format Field Overlay

The Format Field overlay contains the following fields:

Edit Mask

Establishes format requirements for data validation. A mask character representing the allowable values is entered for each position in a field. The mask characters are:

- # (numbers only)
- @ (letters only)
- & (numbers and letters only)
- ! (any characters)
- * (use default value character) — The valid character for this position in a mask is defined by referencing the same position in the default value.
- For example, you can mask a phone number field with the following entry: ###*###*####.

Default Value

Establishes a starting value for all referenced fields whenever they are first displayed in add mode in an input image, unless modified by a user. If blank, there is no starting value displayed on the image. The positioning of constant characters in the default value must correspond to the positions of *s, if any, in the edit mask. For example, a possible default value for the phone number mask indicated in *Edit Mask* earlier is: _ _ _ - _ _ _ - _ _ _ _ (the underlines are blanks).

Lowercase OK?

Indicates whether or not a modifiable item on an interactive screen can be entered in upper- and lowercase letters. If "no" ☐, the entry is stored and displayed exclusively as uppercase, regardless of how a user enters the information. This entry does not affect statements. A lowercase value established with a statement overrides a "no" value in this field and displays as lowercase.

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

The Format Field Specifications overlay provides two options which are described in the following sections: VALIDATION TABLE and ADDITIONAL ATTRIBUTES.

Format Fields Validation Table

The VALIDATION TABLE option from the Format Fields overlay accesses the Validation Table overlay, shown in *Figure 2-4-22*. A validation table provides a more powerful alternative to the use of range checks for data validation when there is a known set of values. Through the list ▼ feature, it provides the user with visibility of all the possible field values as well as the ability to select the desired value.

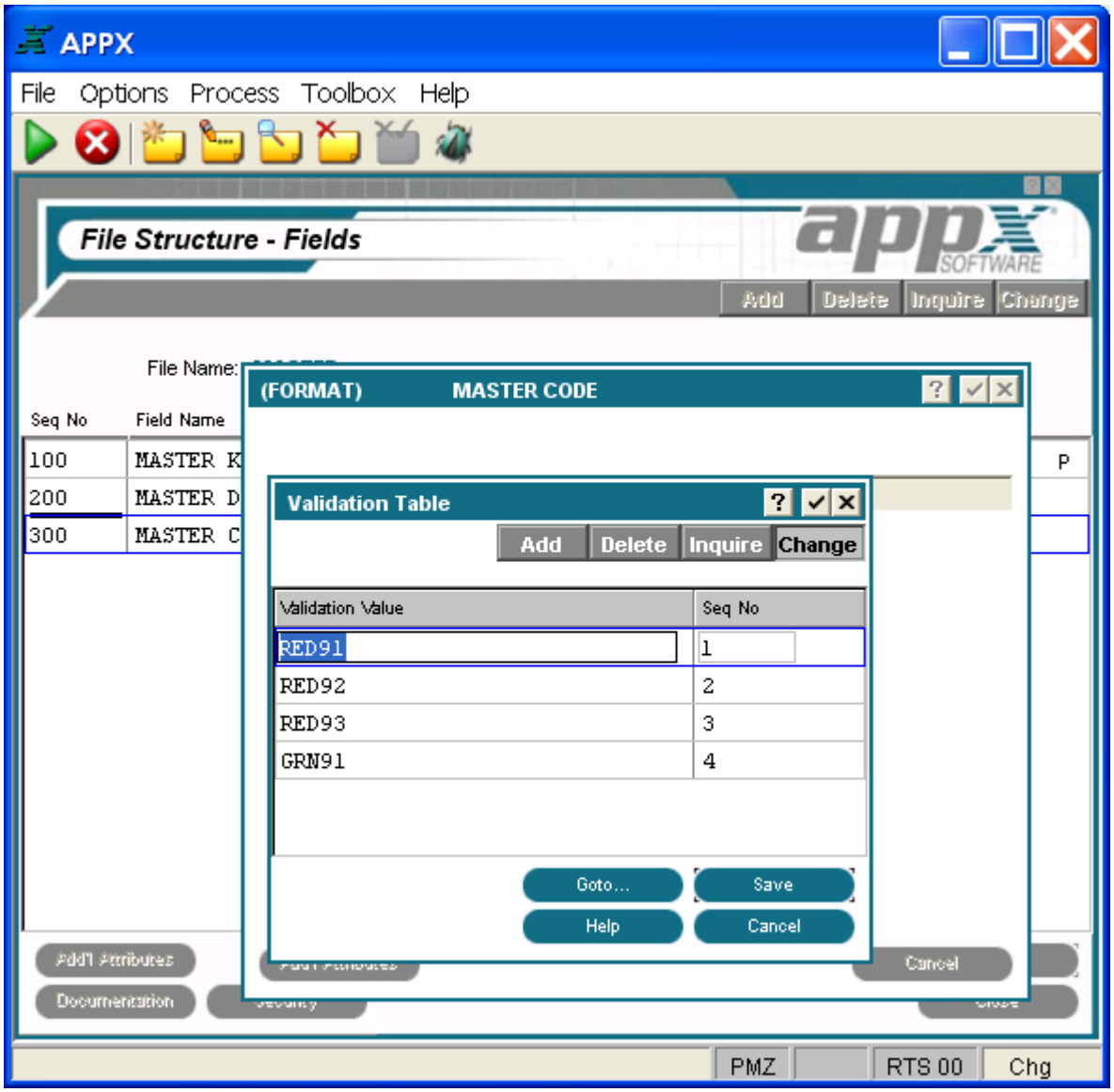


Figure 2-4-22. Format Fields Validation Table Overlay

You can establish a validation table for any Format field in the data dictionary. The entries in the table are case-sensitive. If you enter a validation value in all capital letters, an application user must enter the value in all capital letters.

Validation table operations are database-independent. They access a designer-defined table in application design, not data contained in a file in a database, for the range of possible values. As a result, validation does not vary from database to database. It is automatically available to, and exactly the same for, every database that is associated with an application. All validation values are available for review and selection by the end user through the list feature.

The Validation Table overlay contains the following fields:

Validation Value

Provides a valid field value. It is case-sensitive and is used for online review, selection, and validation. Note that validation values newly added to the data dictionary are not available to a current APPX user until the user exits and re-enters APPX.

Seq No

Provides an identifying number for the associated validation value. It is defaulted by APPX and can be overridden, but it must be unique within a table.. It determines the order in which validation values are presented, both for selection by the user and for editing in application design. APPX assigns the first validation value a number of *1*, with additional values assigned a number that is equal to the highest existing sequence number plus *1*. APPX does not require that sequence numbers be contiguous within a table.

Format Fields Additional Attributes

From the Format field overlay, you have an ADDITIONAL ATTRIBUTES option. The Format field Additional Attributes overlay is shown in *Figure 2-4-23*.

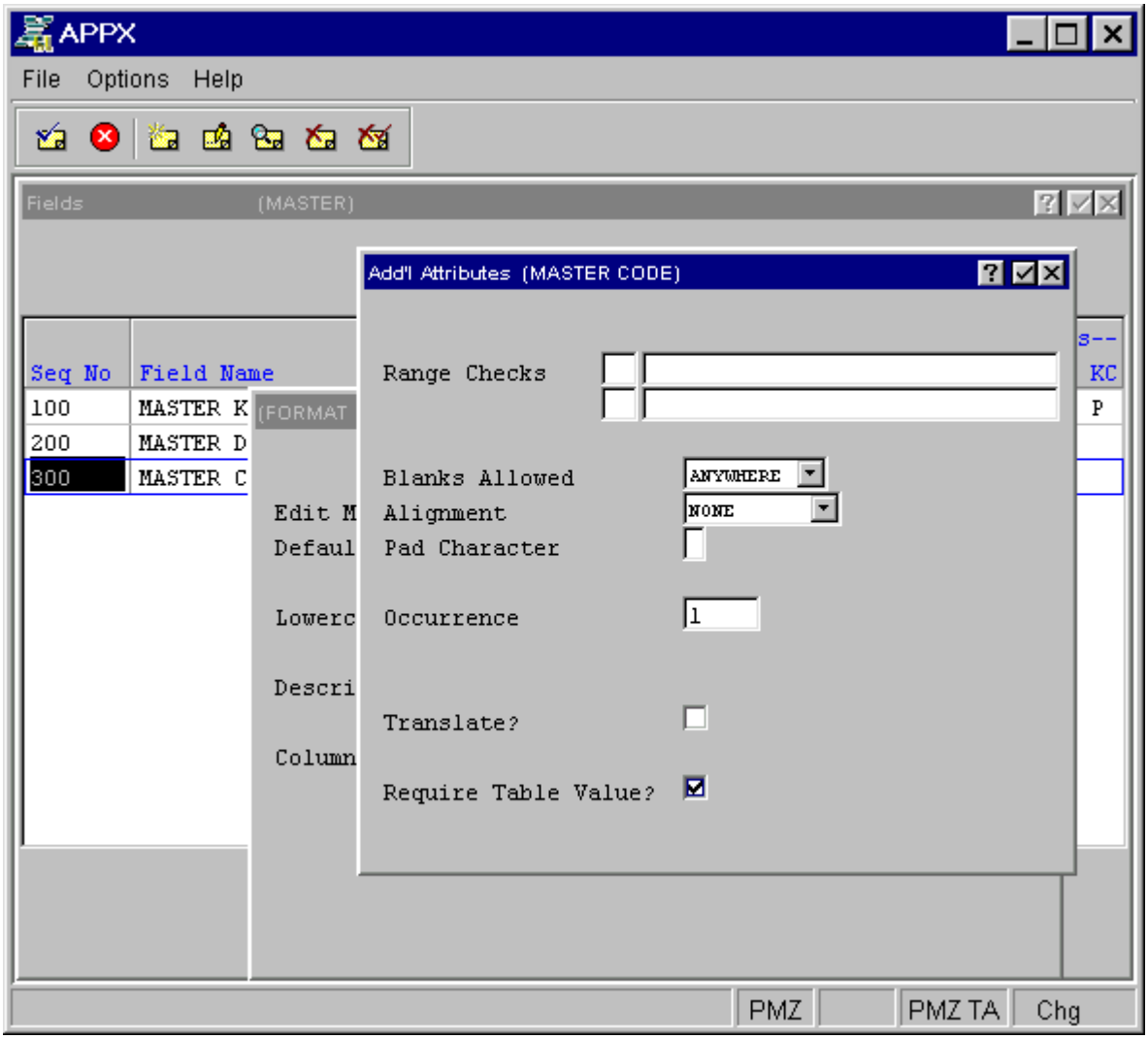


Figure 2-4-23. Format Field Additional Attributes Overlay

The Format Field Additional Attributes Overlay contains the following fields:

Range Checks

Defines a series of allowable values that APPX uses for online data validation. The first component represents the relation. The options are ***EQ*** (equal to), ***NE*** (not equal to), ***GT*** (greater than), ***LT*** (less than), ***GE*** (greater than or equal to), and ***LE*** (less than or equal to). The second component represents the value that is assigned to the relation. If you enter two relations, APPX infers an AND relationship.

Blanks Allowed

Indicates whether or not blanks are permitted and, if so, where they can be placed. The *Blanks Allowed* options are ***NONE, TRAILING,*** and ***ANYWHERE*** (the default).

Alignment

Determines whether or not data in a field is aligned for storage and/or display and, if so, the type of alignment. The *Alignment* options are ***NONE, LEFT, JUSTIFIED, RIGHT*** and ***CENTERED.*** The default is *NONE*.

Pad Character

Indicates the character to be used instead of leading and trailing spaces in a field.

Occurrence

Determines the number of times a field is present in a record. If blank, APPX assumes a value of 1. This field may contain up to five numeric characters and is restricted to a maximum of 32,767. See the Occurrence section for a more comprehensive discussion of this field.

Translate?

Indicates if the value of the format field should be translated into a designated language if APPX's Language Translation feature is being used. To qualify for translation, a format field must have a validation table defined; otherwise, this option cannot be selected. Click the checkbox to toggle between "yes" ☑, "no" ☐, and unknown ? values. If the translation feature is not in use, retain the default value of ?.

Require Table Value?

Indicates whether or not to require one of the validation table values to be entered. To force the user to enter one of the defined values, click the checkbox to toggle it to "yes" ☑. If a "no" ☐ value is selected, the validation table shown when the user selects the list feature, is advisory only and the user can enter a value not listed in the table. If there is no validation table defined, this option does not apply and cannot be selected. If *Translate?* is ☑, this option should also be ☑.

System Fields

A system field, often referred to as an audit field, displays data that is provided by the system.

The System Field overlay, shown in *Figure 2-4-24*, allows you to add system specific attributes to those already defined for fields. The System Field overlay displays after you complete the entries on the Fields specifications screen if the *Field Type* you select is *SYSTEM*.

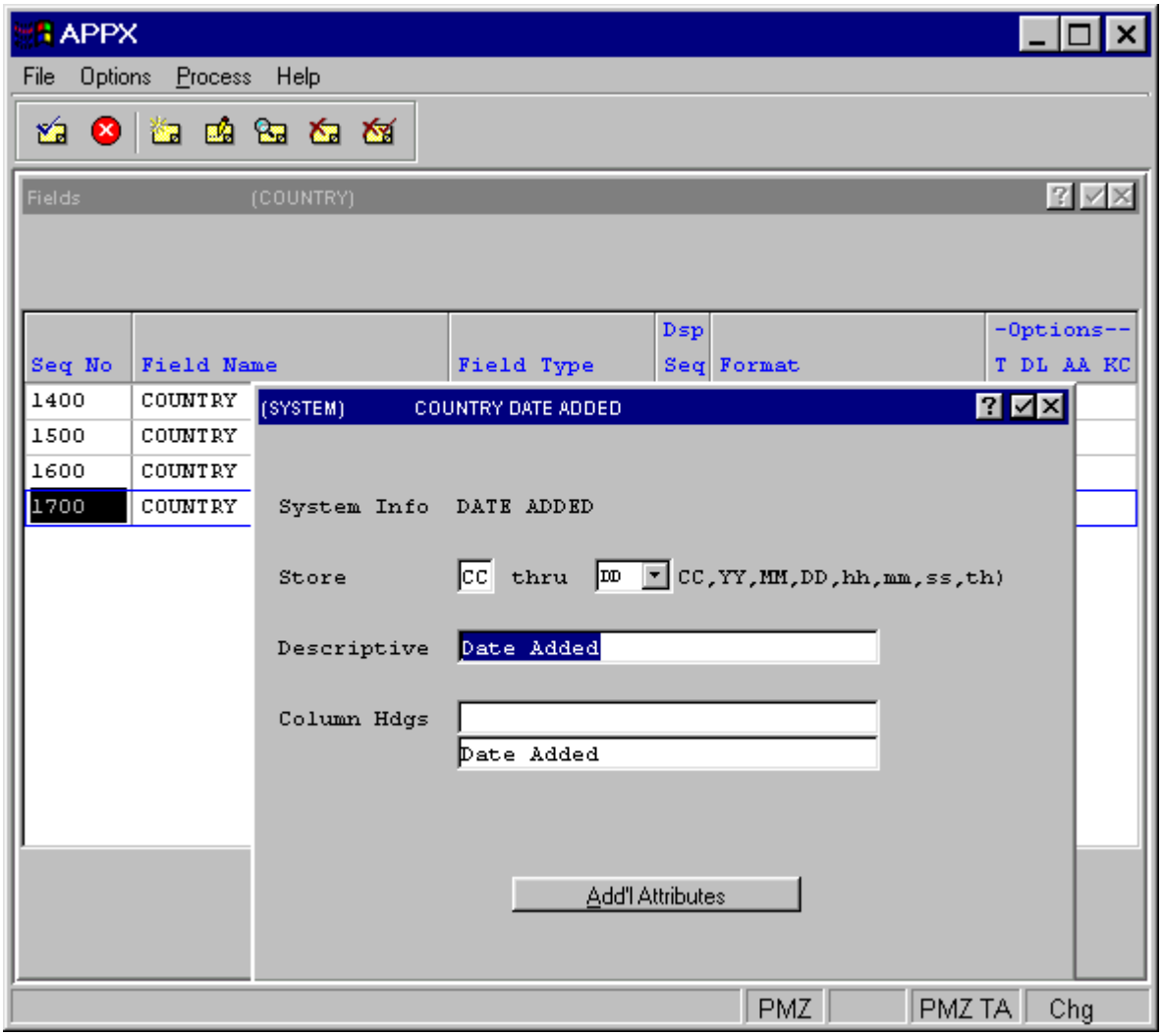


Figure 2-4-24. System Field Overlay

The System Field overlay contains the following fields:

System Info

Indicates the type of system field. Select from **DATE ADDED, DATE CHANGED, USER ADDED** and **USER CHANGED**. After selecting enter, the remaining fields are displayed.

Store thru

Is displayed for system date fields only and indicates the upper and lower components of this date/time field.

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

Text Fields

A text field allows a user to enter a block of text on the data entry screen.

The text field overlay, shown in *Figure 2-4-25*, allows you to add text specific attributes to those already defined for fields. The text field specifications overlay displays after you complete the entries on the fields screen if the *Field Type* you select is *TEXT*.

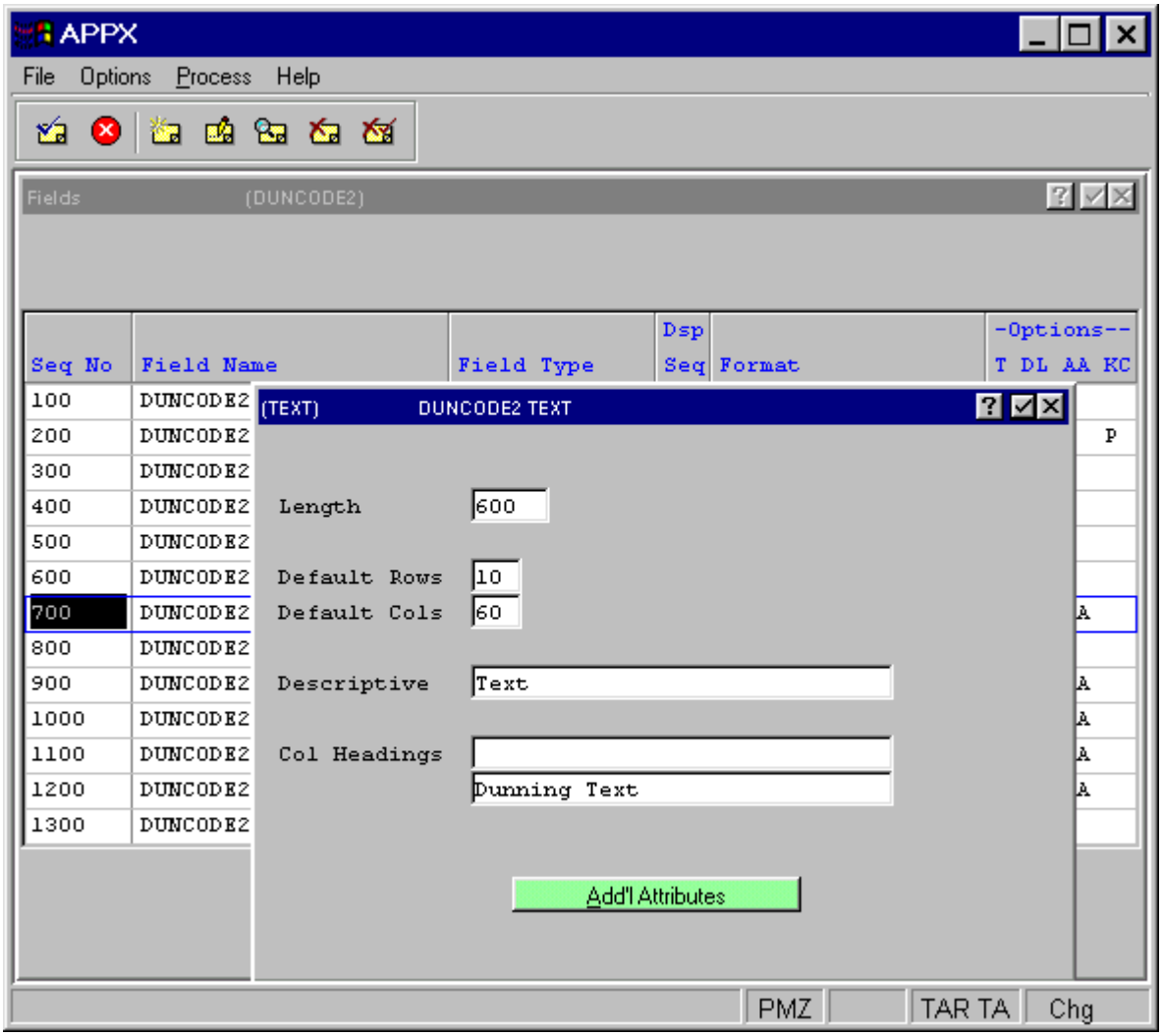


Figure 2-4-25. Text Field Overlay

Length

Designates the number of character positions, or length, of this field.

Default Rows/Cols

Determines the default number of rows and columns to be used when specifying a field on an image.

Descriptive

Provides a longer, more easily understood alternative to *Domain Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Domain Name* is used.

Col Headings

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a domain itself. Refer to Standard Column Heading Image for additional information.

For Text field type, the following option is available as described in the corresponding section:

- Additional Attributes

Text Fields Additional Attributes

From the Text field overlay, you have an ADDITIONAL ATTRIBUTES option. The Text field Additional Attributes overlay is shown in *Figure 2-4-26.*

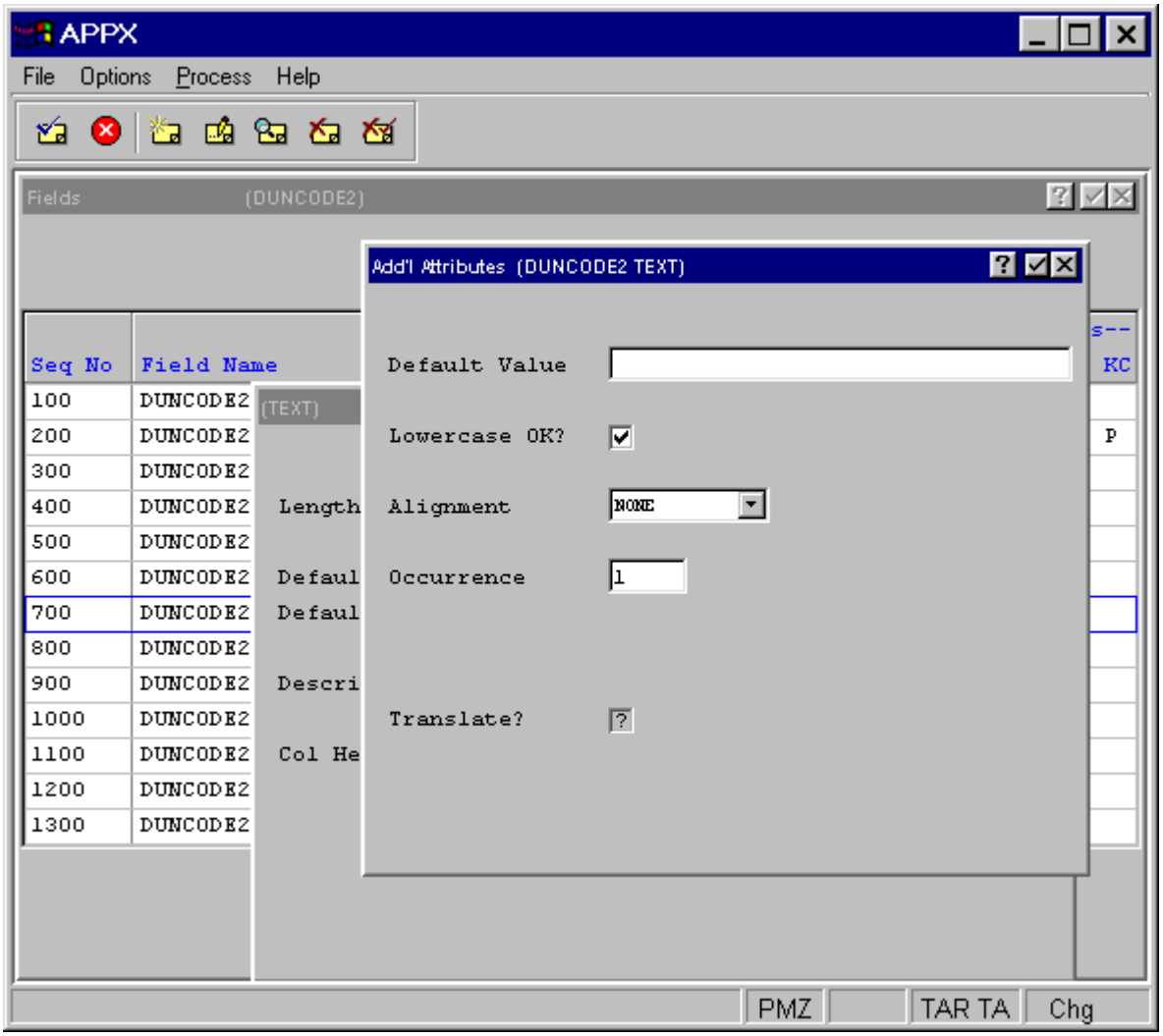


Figure 2-4-26. Text Field Additional Attributes Overlay

The Text Field Additional Attributes Overlay contains the following fields:

Default Value

Indicates the value that should be placed into this field when adding new records to a file. This value can be changed by the user, if desired.

Lowercase OK?

Indicates whether or not a modifiable item on an interactive screen can be entered in upper- and lowercase letters. If "no" ☐, the entry is stored and displayed exclusively as uppercase, regardless of how a user enters the information. This entry does not affect statements. A lowercase value established with a statement displays as lowercase, even if this specification prohibits lowercase entries. The default is "yes" ☑.

Alignment

Determines whether or not data in a field is aligned for storage and/or display and, if so, the type of alignment. The *Alignment* options are ***NONE, LEFT, JUSTIFIED, RIGHT*** and ***CENTERED.*** The default is *NONE*.

Occurrence

Determines the number of times a field is present in a record. If blank, APPX assumes a value of 1. This field may contain up to five numeric characters and is restricted to a maximum of 32,767. See the Occurrence section for a more comprehensive discussion of this field.

Translate?

Normally indicates if the value of the field should be translated into a designated language if APPX's Language Translation feature is being used. However, this option is not available for a text file field (unlike a text work field) and cannot be selected.

Token Fields

A token field is like an alpha field but it has specific values that are stored as corresponding numbers.

The token field overlay, shown in *Figure 2-4-27*, allows you to add token specific attributes to those already defined for fields. The token fields specification overlay displays after you complete the entries on the fields screen if the *Field Type* you select is *TOKEN*.

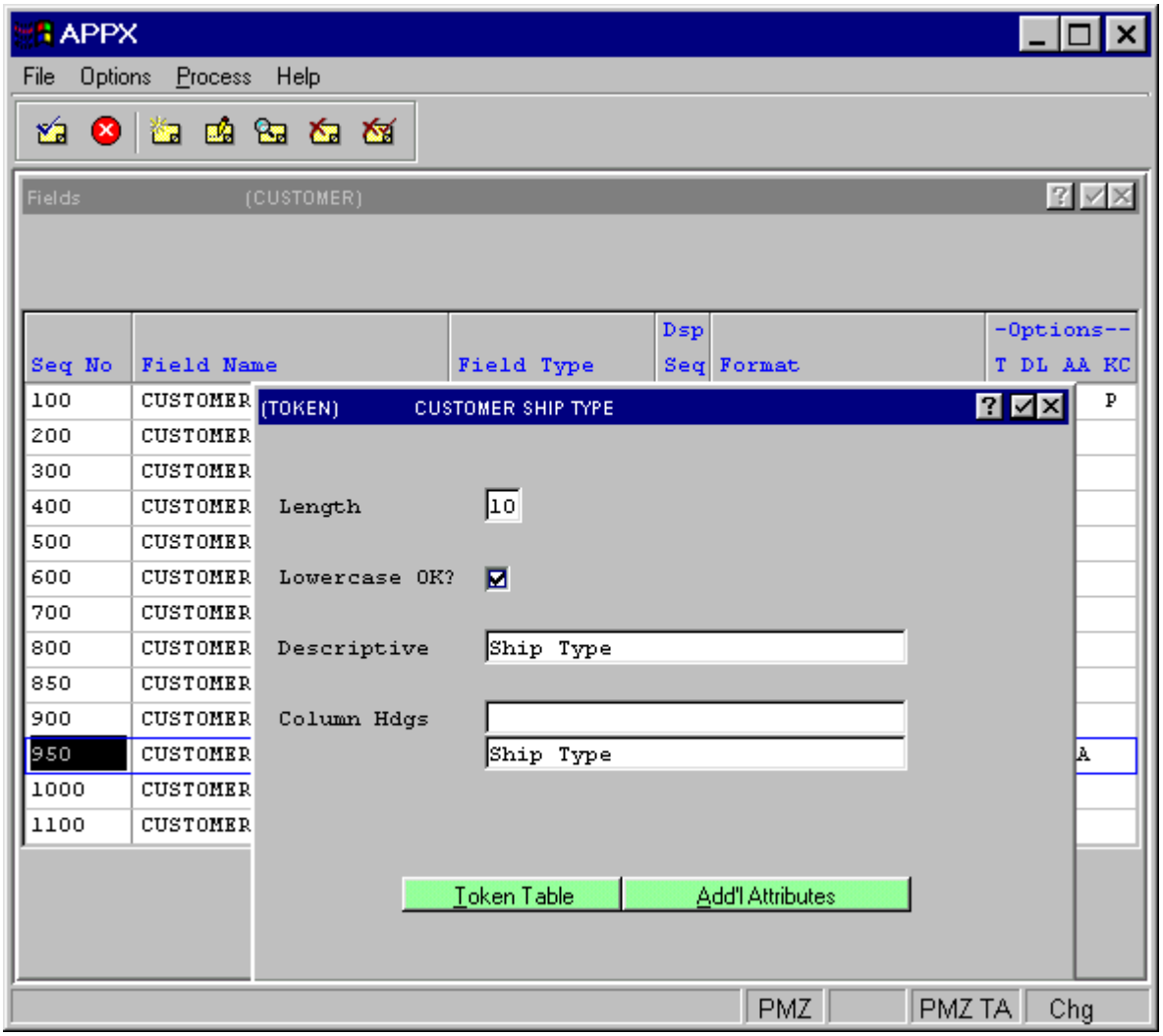


Figure 2-4-27. Token Field Overlay

The Token Field Specifications overlay contains the following fields:

Length

Designates the number of character positions, or length, of this field.

Lowercase OK?

Indicates whether or not a modifiable item on an interactive screen can be entered in upper- and lowercase letters. If "no" ☐, the entry is stored and displayed exclusively as uppercase, regardless of how a user enters the information. This entry does not affect statements. A lowercase value established with a statement displays as lowercase, even if this specification prohibits lowercase entries. Click the checkbox to toggle between "yes" ☑ and "no" ☐.

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

For Token field type, the following options are available as described in the corresponding sections:

- Token Table
- Additional Attributes

Token Fields Token Table

TOKEN TABLE accesses the Token Table overlay, shown in *Figure 2-4-28*. A token table provides an alternative to the use of range checks for data token.

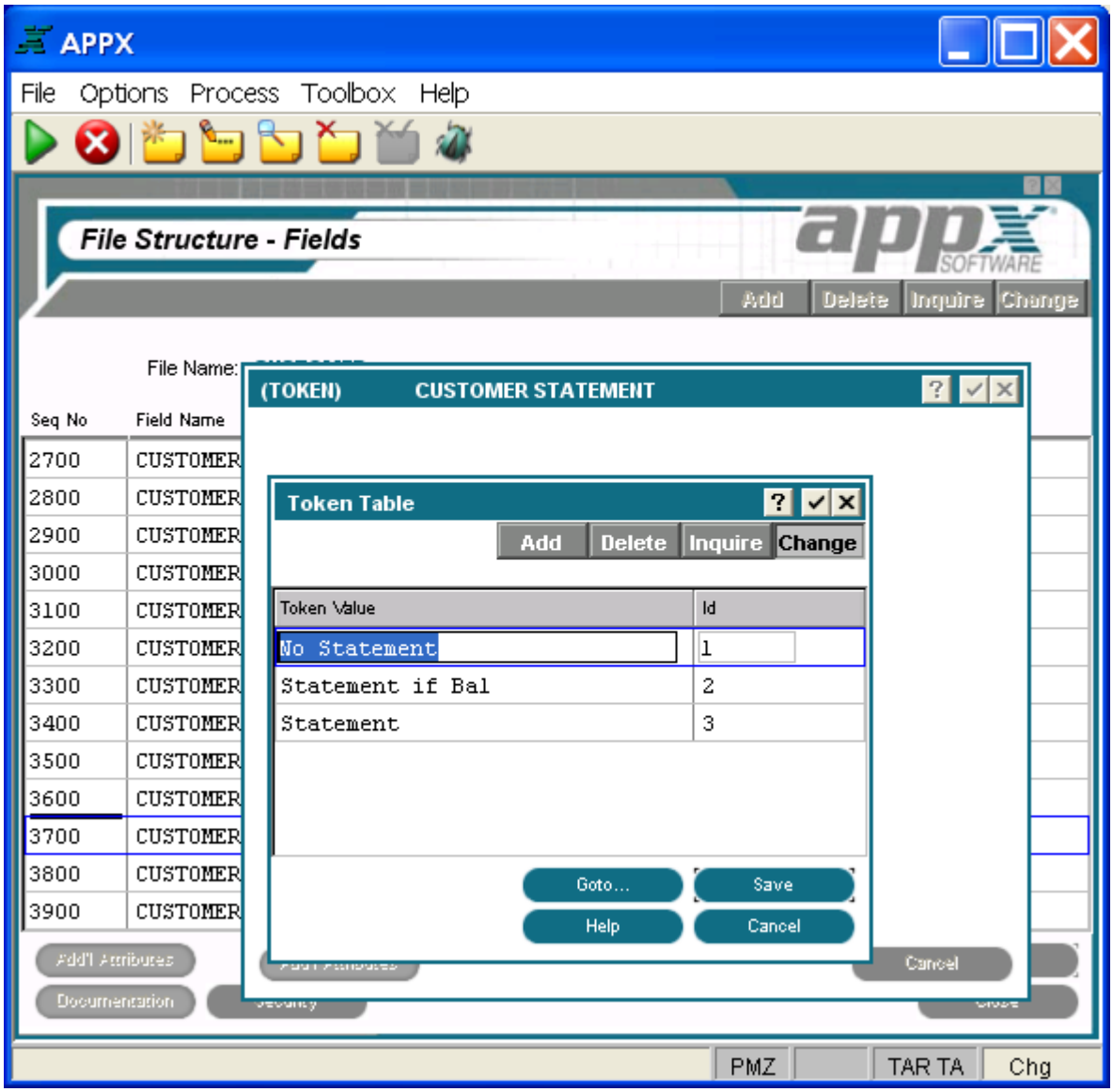


Figure 2-4-28. Token Fields Token Table Overlay

You can establish a token table for any Token field in the data dictionary. The entries in the table are case-sensitive. If you enter a token value in all capital letters, an application user must enter the value in all capital letters.

Token table operations are database-independent. They access a designer-defined table in application design, not data contained in a file in a database, for the range of possible values. As a result, token does not vary from database to database. It is automatically available to, and exactly the same for, every database that is associated with an application.

The Token Table overlay contains the following fields:

Token Value

Is the alphanumeric value associated with the token ID. APPX automatically translates between token values and ID's whenever necessary. This entry is case-sensitive. If entered in capital letters, an application user must also enter the value in capital letters.

ID

Displays a unique number that is defaulted by APPX. This specification establishes the binary value that APPX stores in place of the associated token value. APPX assigns the first token value an ID of *1*, with additional values assigned an ID that is equal to the highest existing ID plus *1*. The default can be overridden, but must be unique within a table. APPX does not require that ID's be contiguous within a table.

Token Fields Additional Attributes

From the Token field overlay, you have an ADDITIONAL ATTRIBUTES option. The Token field Additional Attributes overlay is shown in *Figure 2-4-29*.

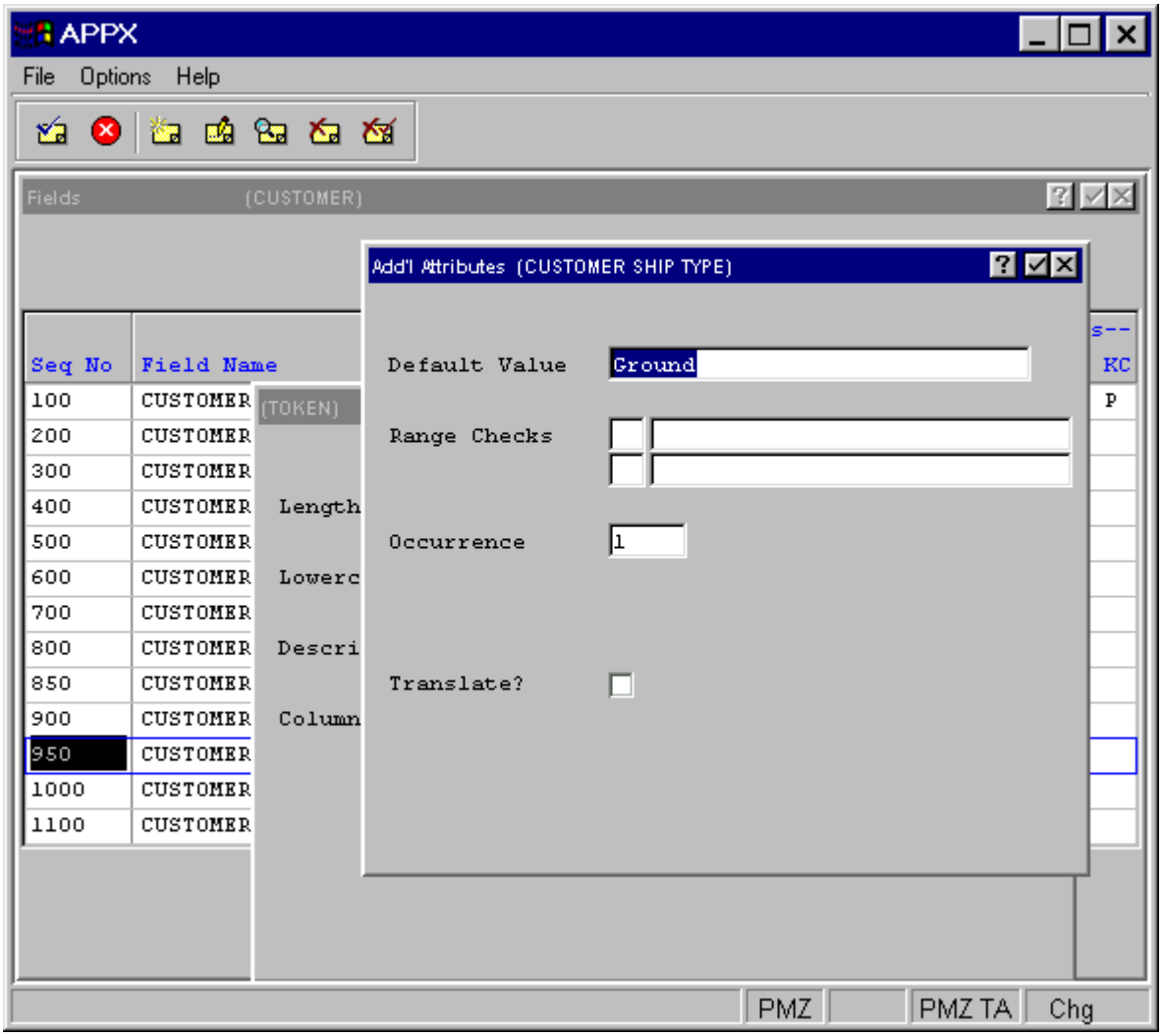


Figure 2-4-29. Token Field Additional Attributes Overlay

The Token Field Additional Attributes Overlay contains the following fields:

Default Value

Establishes a modifiable starting value for all image items whenever they are first displayed in add mode in an input image, unless modified by a user. If blank, there is no starting value displayed on the image.

Range Checks

Defines a series of allowable values that APPX uses for online data validation. The first component represents the relation. The options are ***EQ*** (equal to), ***NE*** (not equal to), ***GT*** (greater than), ***LT*** (less than), ***GE*** (greater than or equal to), and ***LE*** (less than or equal to). The second component represents the value that is assigned to the relation. If you enter two relations, APPX infers an AND relationship.

Occurrence

Determines the number of times a field is present in a record. If blank, APPX assumes a value of 1. This field may contain up to five numeric characters and is restricted to a maximum of 32,767. See the Occurrence section for a more comprehensive discussion of this field.

Translate?

Indicates if the value of the token field should be translated into a designated language if APPX's Language Translation feature is being used. Click the checkbox to toggle between "yes" ☑, "no" ☐, and unknown ? values. If the translation feature is not in use, retain the default value of ?.

Numeric Fields

A *numeric* field is restricted to number, decimal, and sign (-) entries. Examples include amounts, prices, and quantities.

The Numeric fields overlay, shown in *Figure 2-4-30*, permits you to add numeric-specific attributes to those already defined for the field. It automatically displays after you complete the entries on the Fields specifications screen if the *Field Type* you select is *NUMERIC*.

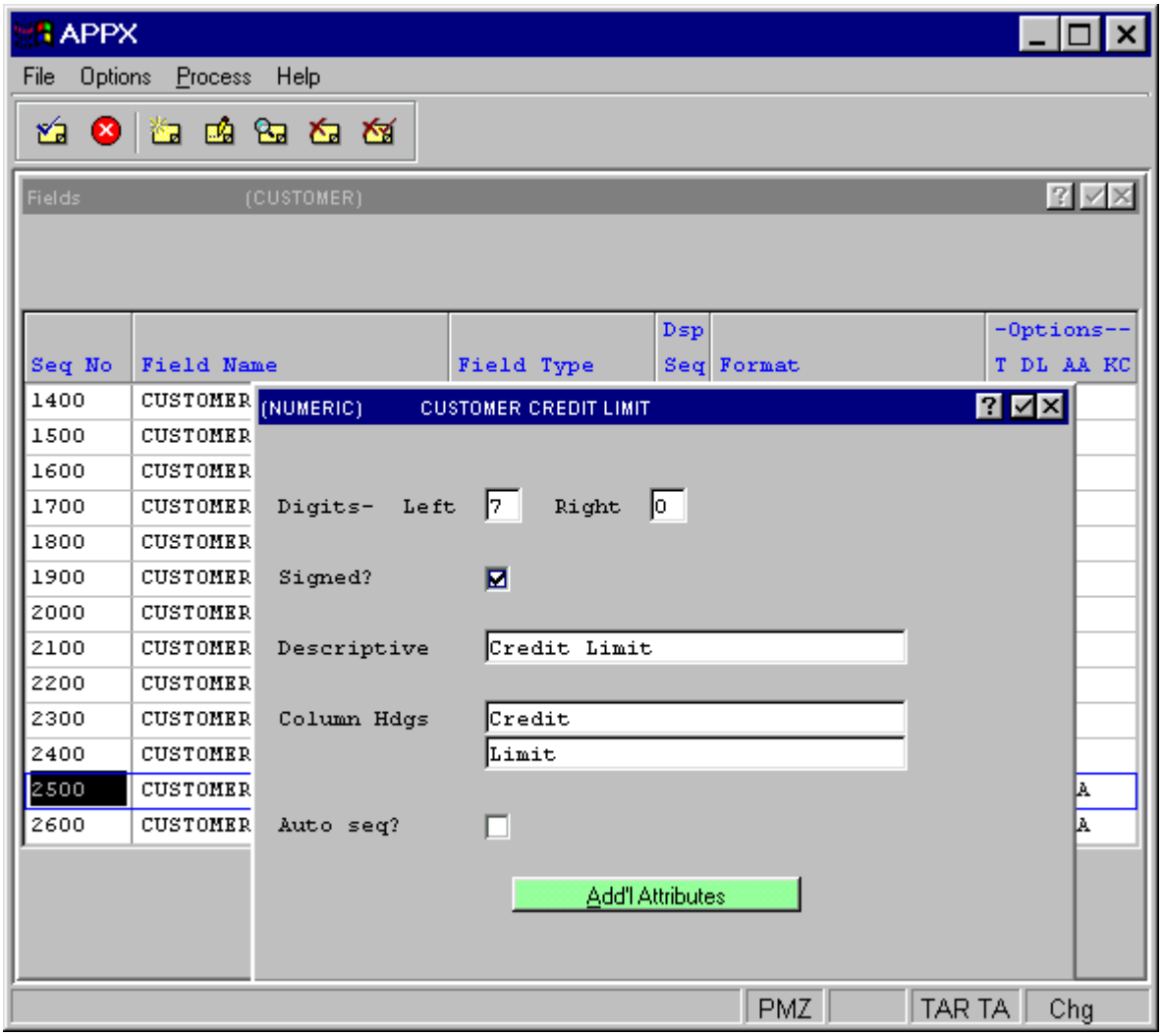


Figure 2-4-30. Numeric Fields Overlay

The Numeric fields overlay contains the following fields:

- ***Digits — Left.*** Enter the number of digits to the left of the decimal.
- ***Digits — Right.*** Enter the number of digits to the right of the decimal.

Signed?

Specifies whether the numbers entered in this field are positive or negative. “No” ☐ would ensure that only positive numbers are entered. The default is “no” ☐ if no digits have been defined to the right of the decimal, "yes" ☑ if digits have been defined to the right of the decimal. Click the checkbox to toggle between “yes” and “no”.

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

Auto Seq?

Select "yes" ☑ if this numeric field represents a sequence number with a value that should be maintained by APPX. Click the checkbox to toggle between "yes" ☑ and "no" ☐. Note that *Auto Seq?* only appears if the numeric field has no decimal places.

From the Numeric fields overlay, you can access the ADDITIONAL ATTRIBUTES option, shown in *Figure 2-4-31*.

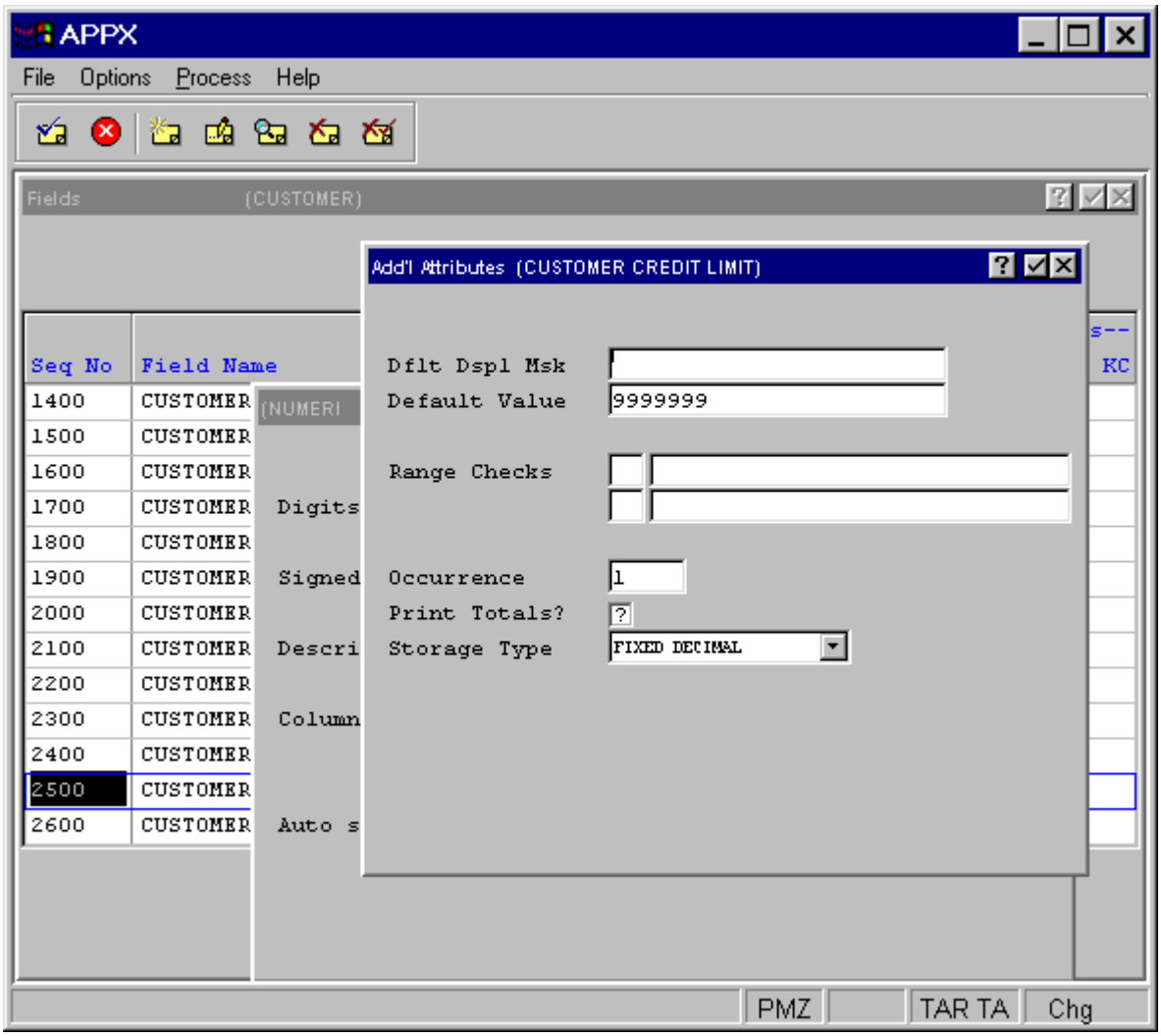


Figure 2-4-31. Numeric Fields Additional Attributes Overlay

The numeric fields Additional Attributes overlay contains the following fields:

Dflt Display Msk

Designates the format that the data displays to a user in an APPX image. This entry establishes the *Display Mask* default for all image items that reference the numeric field. If blank, APPX uses a display of nonfixed leading spaces with one significant digit. Refer to the *Numeric Domains* section (near *Figure 2-3-16)* in *Chapter 2-3* for additional information about this entry.

Default Value

Establishes a starting value for all image items whenever they first display in add mode in an input image, unless modified by a user. If blank, the starting value is assumed to be the null value.

Range Checks

Defines a series of allowable values that APPX uses for online data validation. The first component represents the relation. The options are ***EQ*** (equal to), ***NE*** (not equal to), ***GT*** (greater than), ***LT*** (less than), ***GE*** (greater than or equal to), and ***LE*** (less than or equal to). The second component represents the value that is assigned to the relation. If you enter two relations, APPX infers an AND relationship.

Occurrence

Determines the number of times a field is present in a record. If blank, APPX assumes a value of 1. This field may contain up to five numeric characters and is restricted to a maximum of 32,767. See the Occurrence section for a more comprehensive discussion of this field.

Print Totals?

Select “yes” ☑ to indicate that this field should be automatically totaled on reports. This field is used as a default for the auto-total specification in output and inquiry process definition. Click the checkbox to toggle between “yes” ☑ and “no” ☐ values.

Storage Type

Determines the internal storage format for data. The *Storage Type* options are:

- ***BINARY*** — The number is stored in binary form. The high order bit determines the sign: on or *1* for positive, and off or *0* for negative. The decimal position is established by the *Decimals — Left/Right* entries. The maximum field length is 10 digits for non-decimal numbers or 9 digits for decimal numbers.

FIXED DECIMAL

Employs a packed decimal format that stores two decimal digits for each byte with the exception of the last bit, which is the sign bit. The decimal position is established by the *Decimals — Left/Right* entries. The maximum field length is 31 digits.

FLOATING DECIMAL

Employs a combination of binary and packed decimal formats that always requires a total of nine bytes of storage. Although the field definition allows for 15 digits to the left of the decimal point plus 15 digits to the right of the decimal point, only the first 15 digits of the actual number are stored. These 15 digits are stored as packed decimal with two decimal digits for each byte (for a total storage requirement of eight bytes). The final byte (binary) determines the maximum number of digits that currently appears to the right of the decimal.

- The default is *FIXED DECIMAL*. For a comprehensive discussion of these options, see Numeric Storage Types in *Chapter 2-3*.

Date Fields

A date field contains date and/or time information. This information may, at your option, range from a century through a hundredth of a second. A field definition may also include separator characters (such as */*, -, or *:*) and text, including the day of the week and the name of the month.

Examples of date/time fields include invoice dates, order dates, elapsed time data, date/time stamps used to uniquely identify records, and audit trail fields such as date-added and date-changed.

The Date fields overlay, shown in *Figure 2-4-32*, permits you to add date/time-specific attributes to those already defined for the field. It automatically displays after you complete the entries in the Fields specifications screen if the *Type* you select is *DATE*.

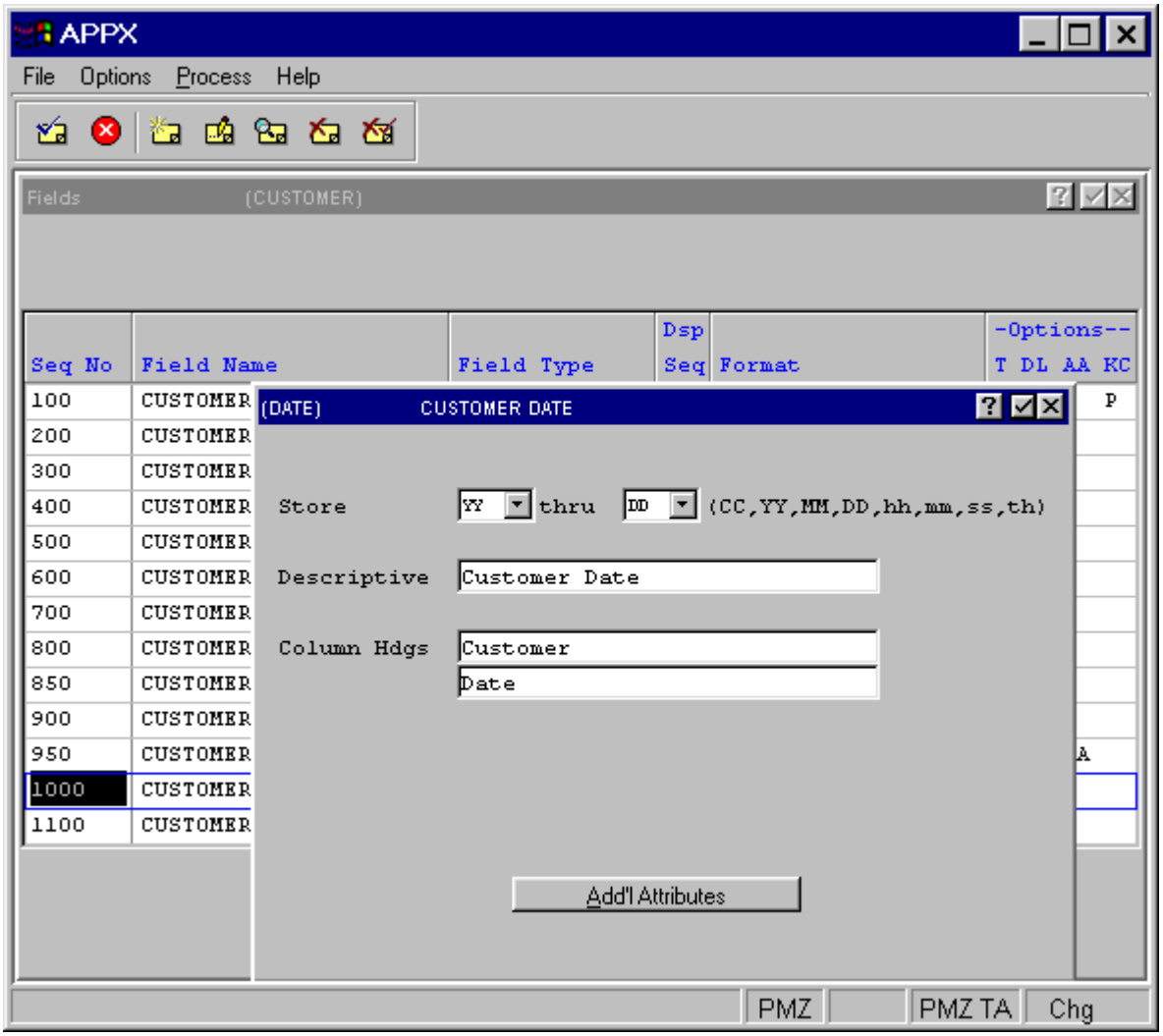


Figure 2-4-32. Date Fields Overlay

The Date fields overlay contains the following fields:

Store thru

Designates the parts of a date that are physically stored. Valid entries can be selected from a list by clicking the list button to the right of the field. Valid entries are:

CC	(Century)
YY	(Year)
MM	(Month)
DD	(Day)
hh	(Hour)
mm	(Minute)
ss	(Second)

th	(Tick (hundredth of a second))

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

You can access the ADDITIONAL ATTRIBUTES option, shown in *Figure 2-4-33*, from the Date fields overlay.

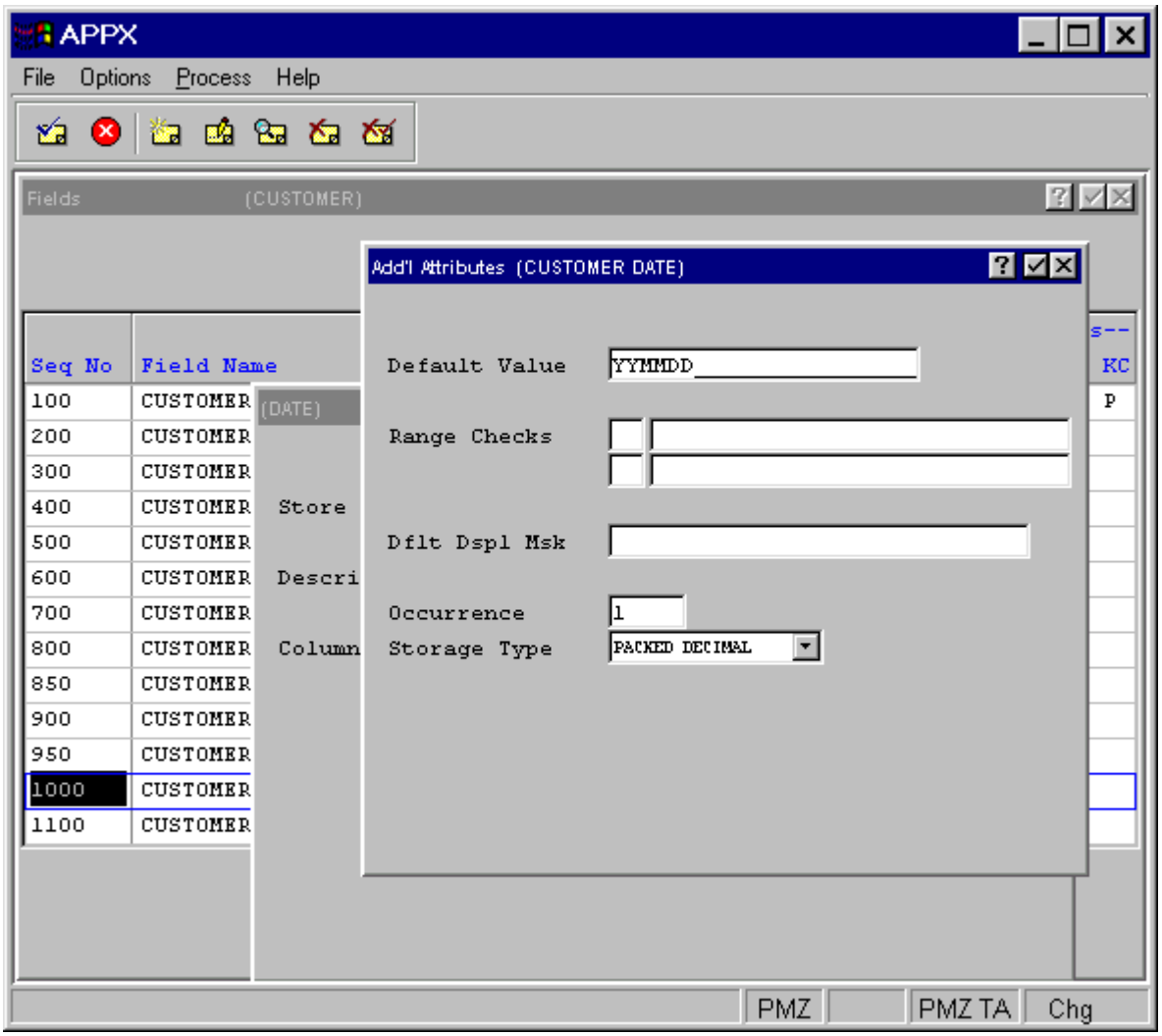


Figure 2-4-33. Date Fields Additional Attributes Overlay

The date fields Additional Attributes overlay contains the following fields:

Default Value

Establishes a starting value for image items whenever they are first displayed in add mode on an input image, unless modified by a user. If blank, there is no starting value displayed on the image.

Range Checks

Defines a series of allowable values that APPX uses for online data validation. The first component represents the relation. The options are ***EQ*** (equal to), ***NE*** (not equal to), ***GT*** (greater than), ***LT*** (less than), ***GE*** (greater than or equal to), and ***LE*** (less than or equal to). The second component represents the value that is assigned to the relation. If you enter two relations, APPX infers an AND relationship.

Dflt Dspl Msk

Controls how this field appears to a user on an APPX image, unless overridden at the image item level. If left blank, the date mask specified for the application (in Application File Maintenance in System Administration) is used or, if that is also blank, the date mask specified in System Parameters is used. Refer to the Display Mask (Date/Time) section in Chapter 3-14: Using the Character Based Image Editor for additional information about this entry.

Occurrence

Determines the number of times a field is present in a record. If blank, APPX assumes a value of 1. This field may contain up to five numeric characters and is restricted to a maximum of 32,767. See the Occurrence section for a more comprehensive discussion of this field.

Storage Type

Determines the internal storage format of the date field. The *Storage Type* options are:

- ***BINARY*** — The date is stored in binary format.

PACKED DECIMAL

Stores two decimal digits for each byte.

ALPHA

Stores the external representation of data.

- The default is *PACKED DECIMAL*.

Logic Fields

A *logic* field is a checkbox with possible values of "yes" , "no" , or "unknown" . Examples include various application parameters such as whether or not to post transactions on an accrual basis, and control fields that govern the execution of subroutines based upon an *exists/does not exist* condition.

The Logic fields overlay, shown in *Figure 2-4-34*, permits you to add logic-specific attributes to those already defined for the field. It automatically displays after you complete the entries on the Fields specifications screen if the *Field Type* you select is *LOGIC*.

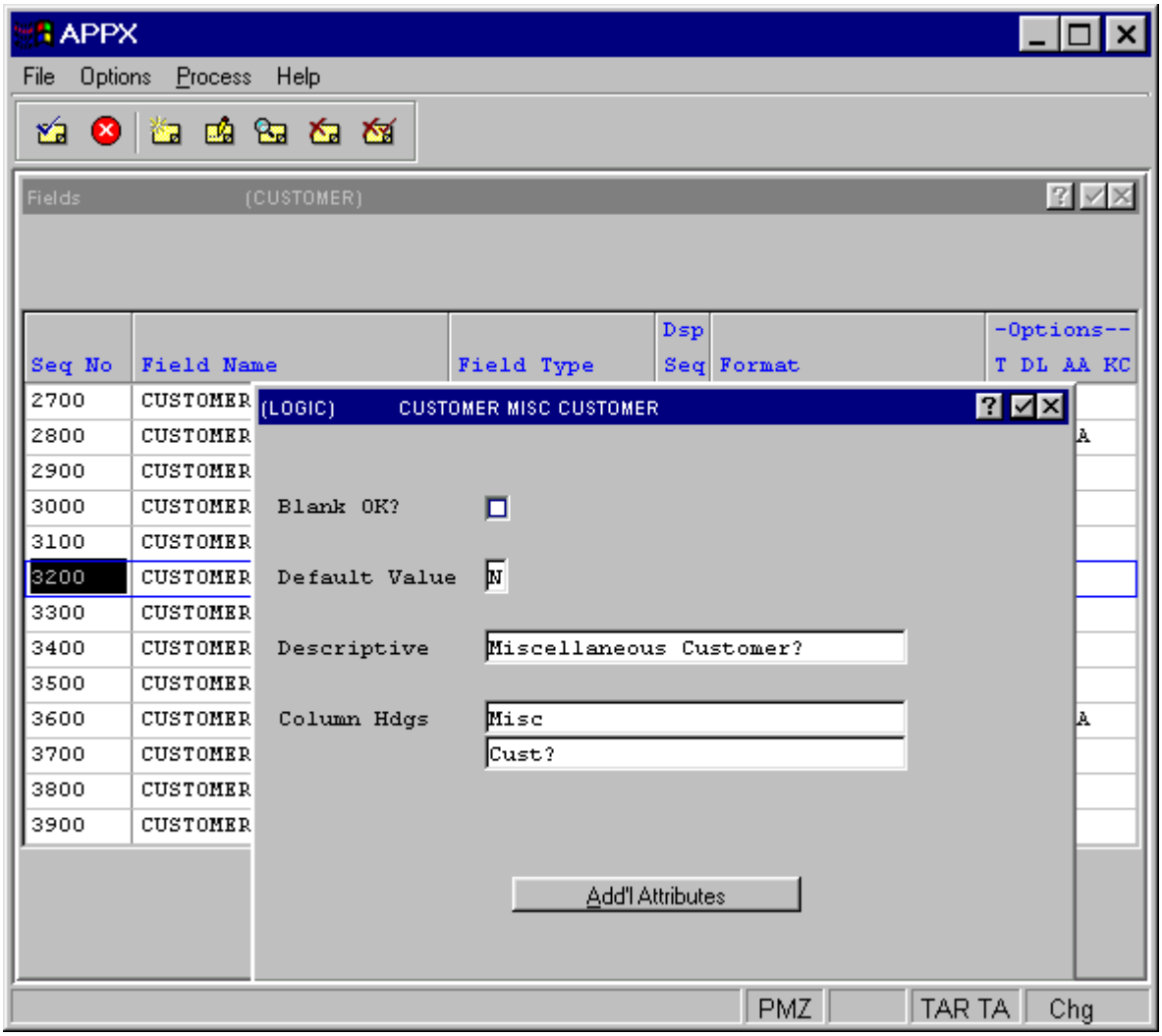


Figure 2-4-34. Logic Fields Overlay

The Logic fields overlay contains the following attributes:

Blank OK?

Indicates, for information only, whether a user should be permitted to leave this field blank on an input image. (Whether or not the field can actually be left blank is controlled by the image item specification 'Required?'.) The default is “no" ☐. Click the checkbox to toggle between “yes” ☑ and “no” ☐.

Default Value

Establishes a starting value for image items whenever they first display in add mode on an input image, unless modified by a user. If blank, the starting value that displays on the image is unknown or ?. The *Default Value* options are ***blank*** (default value of *unknown* or ?), *Y* (default value of *yes* or ☑), and *N* (default value of *no* or ☐). The default is blank.

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

The Logic fields overlay offers one option described in the following section: ADDITIONAL ATTRIBUTES.

Logic Fields Additional Attributes

You can access the Additional Attributes overlay, shown in *Figure 2-4-35*, from the Logic fields overlay.

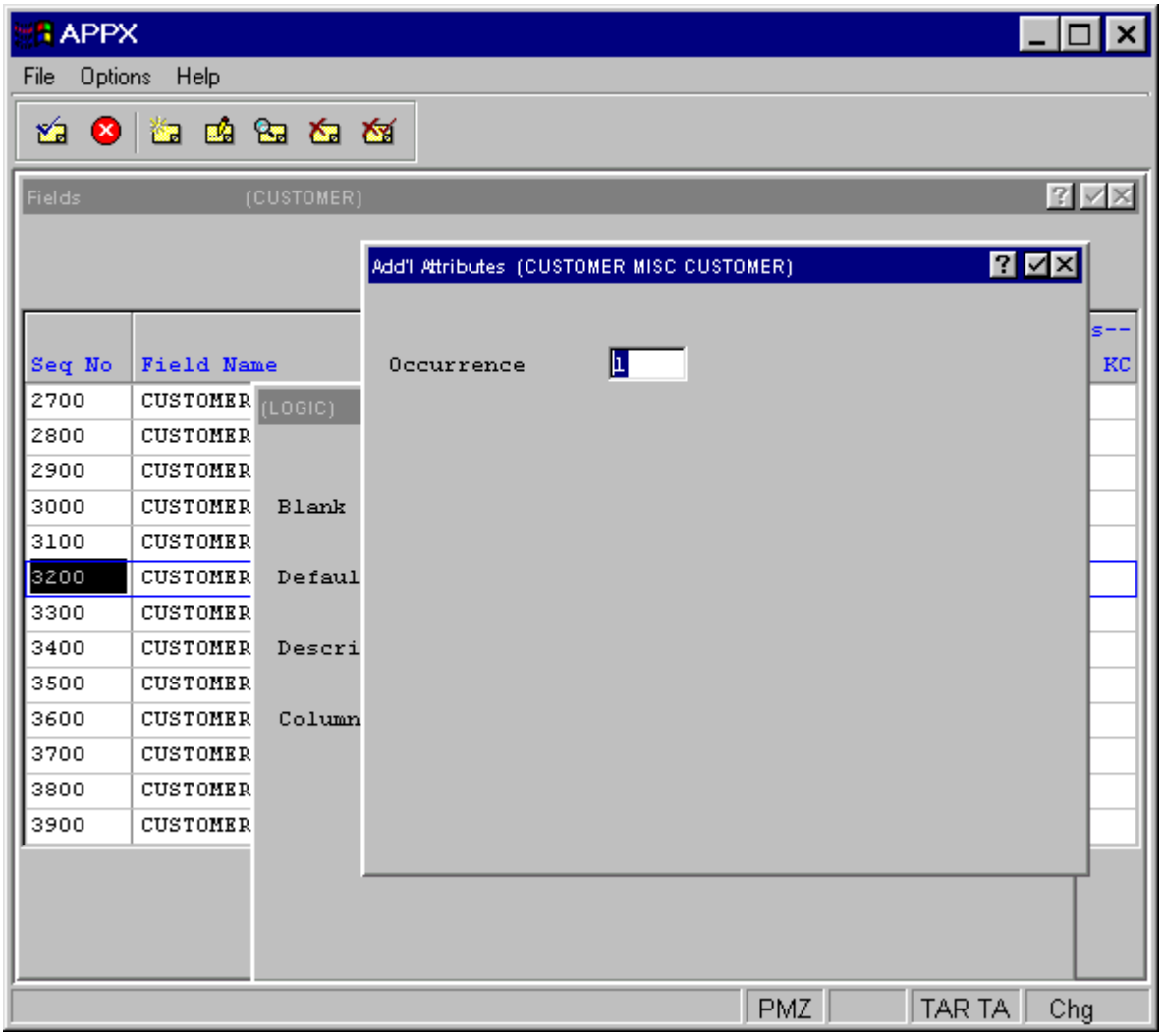


Figure 2-4-35. Logic Fields Additional Attributes Overlay

The logic field Additional Attributes overlay contains the following field:

Occurrence

Determines the number of times a field is present in a record. If blank, APPX assumes a value of 1. This field may contain up to five numeric characters and is restricted to a maximum of 32,767. See the Occurrence section for a more comprehensive discussion of this field.

Synonym Fields

A synonym field provides an alternative name for referencing an existing APPX field in the same record. By using synonym fields, you can reference the same data under more than one field name, and with more than one field sequence number.

You can only establish synonyms for physical fields. Synonyms cannot reference substrings, group headers, or group trailers. The sequence number of a synonym field must be greater than the sequence number of the field it references. In addition, a field that is referenced by a synonym field must be a single-occurrence field in the same record.

The Synonym specifications overlay, shown in *Figure 2-4-36*, automatically displays after you complete the entries on the Fields specifications screen if the *Field Type* you select is *SYNONYM*.

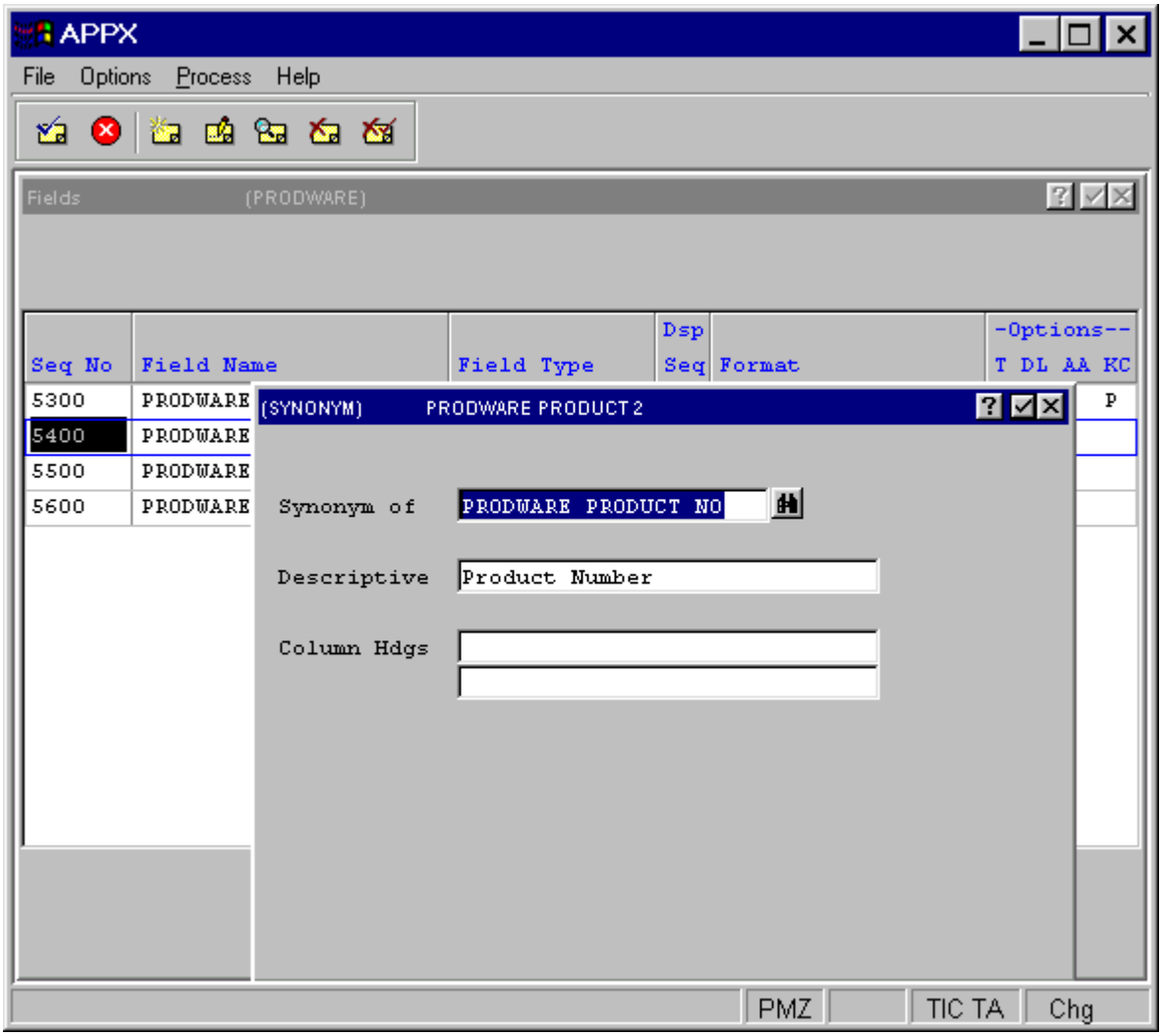


Figure 2-4-36. Synonym Fields Overlay

The Synonym fields overlay contains the following fields:

Synonym of

Identifies the physical field that is referenced by the synonym. The referenced field cannot be a substring, group trailer, group header, or another synonym field, and is limited to a single occurrence. The sequence number for this field must be smaller than the sequence number that is assigned to the synonym.

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

Special Uses for Synonym Fields

Because synonym fields permit you to access data in physical fields with alternative field names and sequence numbers, they provide additional capabilities. These capabilities include such things as resequencing key numbers to match those assigned by non-APPX databases, creating noncontiguous or overlapping group fields, and providing an alternative key order for online data retrieval.

Resequencing key numbers

The APPX dictionary permits you to establish one primary key and up to 16 alternate keys to a file. APPX automatically assigns key number *0* to a primary key when the data dictionary is processed. Alternate keys are assigned key numbers based upon their position in a record (their sequence number). The alternate key field with the lowest sequence number is key number *1*, the alternate key field with the next lowest sequence number is key number *2*, and so on. You cannot change the assigned key numbers. If you subsequently modify the sequence numbers of key fields, the assigned key numbers are not changed.

If an APPX data dictionary is created for an application that was developed in another third or fourth generation language, the APPX-assigned key numbers may not match the key numbers that were assigned during development. Synonym fields can be used to resequence key number assignments, forcing a match between APPX's key numbers and the key numbers that were assigned by the development software.

Creating noncontiguous groups

A group field in APPX must be comprised of fields with contiguous sequence numbers. You cannot include a field with a noncontiguous sequence number in a group field.

If for any reason you cannot change the sequence number of a noncontiguous field to satisfy this requirement, you can use a synonym field. Simply define a synonym for the noncontiguous field, and assign the synonym a sequence number that falls between the sequence number of the group header field and group trailer field. The synonym automatically becomes part of the group.

A synonym field occupies the same physical space and accesses the same data as the (noncontiguous) field it references. A synonym's sequence number, however, is contiguous with that of other fields in the group, satisfying the group requirement.

Creating overlapping group fields

A group field cannot incorporate a field that is part of another group unless it incorporates all fields from the other group. In other words, groups can be nested but they cannot overlap.

You can circumvent this restriction by defining a synonym for a field that participates in more than one group. For example, suppose you create group A, which includes a customer number and an invoice number field. You wish to define group B, which also includes the customer number. You must first define a synonym of the customer number, and include the synonym field in group B.

A synonym field occupies the same physical space and accesses the same data as the field it references. However, because APPX considers the synonym to be a separate field, group A and group B do not share a common field and are therefore not considered as overlapping.

Providing an alternative key order for online retrieval

Many times, a group is designated as a key, solely to establish an order or sequence for online data retrieval. Under some circumstances, however, it may be necessary to provide for online retrieval of the same information by the same fields, but in different orders. (Remember that the retrieval order is deermined by the relative sequence numbers of fields within a group, and the ascending/descending specifications for those fields.)

Take, for example, a history file that contains a customer number and an annual activity amount for each customer. In one instance, you may wish to retrieve records based on customer activity, sequenced in descending order, from highest-volume customer to lowest. In another, you may wish to reverse the sequence. To provide for online retrieval in both an ascending and descending sequence, you need a synonym field.

Now, assume your application includes a Products by Customer file that identifies the range of products each customer is authorized to purchase. In one instance, you are required to retrieve, in product number order, all the products a specific customer is authorized to buy. Such an operation requires a group field key consisting of customer number and product number, in that order. In another instance, your user needs to retrieve all the customers who are authorized to buy a specific product. This operation requires a group field key consisting of product number and customer number, in that order. To establish this second key, incorporating the same fields as the first key but in a different sequence, you must first create synonyms of the customer number and product number fields. For these synonym fields, the customer synonym's sequence number must be larger than that of the product synonym.

Substring Fields

A *substring* field is a discrete portion of a longer alphanumeric field (specifically, an alpha, format, or text field or a domain-type field based on an alpha, format or text domain). For example, assume that a six-character account number field is coded so that the first four characters identify the type of account and the final two characters identify a geographic location. Processes that require the location value can access a substring field defined as the final two characters of the account number field.

A substring is defined for those instances where it is of benefit to access both a field in its entirety and a discrete segment of the field, depending upon the specific requirements of the operation or display. If you have not defined a substring, your only option is to manipulate the contents of the field itself with statements.

You can only define substrings for alphanumeric physical fields in the same record. They cannot reference synonyms, group headers, or group trailers, and a substring cannot be included as part of a group field or be referenced by a synonym field. In addition, the sequence number of a substring field must be greater than the sequence number of the alphanumeric field it references. Substring fields are limited to a single occurrence.

A substring occupies the same physical space as the alphanumeric field it is a part of. Its data is a subset of the data contained in that alphanumeric field. Any value that is established in the alphanumeric field is automatically established in the substring. The substring's initial value is also a subset of the alphanumeric field's initial value.

You can think of a substring as the inverse of a group field. The relationship between a substring and its alphanumeric field is similar to the relationship between a component field and its group. Whether you choose to define a field and various substrings, or a group and various component fields, impacts the appearance of fields on APPX images. A substring loses many of the edit characteristics of the alphanumeric field it references when it is painted on an image. Group fields do not. APPX requires you to paint the components of a group on an image, not the group itself, thereby ensuring that all the edit characteristics are maintained.

The Substring fields specifications overlay, shown in *Figure 2-4-37*, automatically displays after you complete the entries on the Fields specifications screen if the *Field Type* you select is *SUBSTRIN*G.

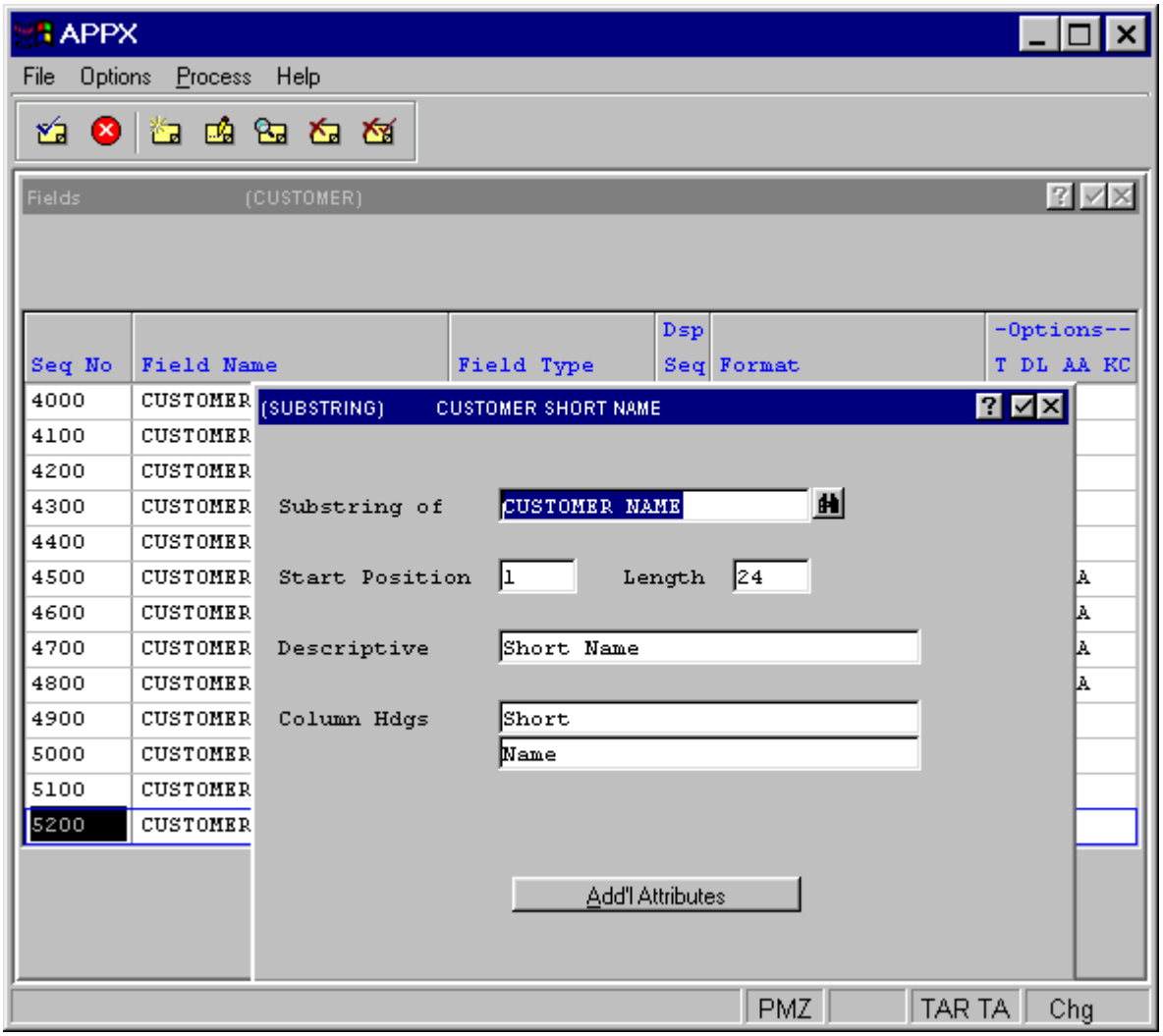


Figure 2-4-37. Substring Fields Overlay

The Substring fields overlay contains the following fields:

Substring of

Identifies the alphanumeric field that is referenced by the substring. The field referenced must be an alpha, format, or text field or a domain-type field based on an alpha, format, or text domain. The sequence number for this field must be smaller than the sequence number that is assigned to the substring.

Start Position

Determines the character position in the referenced field that represents the first character of the substring. For example, if phone number is defined as a format field with edit mask ###*###*####, the start position for a substring for the phone number exchange would be 5.

Length

Designates the number of consecutive characters that comprise a substring field. The length of a substring cannot exceed the length of the alphanumeric field it references. This means that the sum of the value that is entered in this specification and the value in *Start Position*, less one, cannot exceed the length of the referenced field.

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

Group Fields

A *group field* (often referred to simply as a group) is a series of one or more fields with contiguous sequence numbers, which can be referenced by a single field name. The group field's primary purpose is to simplify data assignments and comparisons between two sets of fields with matching contents, formats, and storage types. You can only reference a group field in a statement or within selection criteria, or specify it as a key.

You create a group by defining a group header and, optionally, a group trailer field.

- A ***group header field*** marks the boundaries of a series of contiguous fields in a group, and provides a field name to reference the group. A specification entry on the group header overlay, *Thru Field Name*, also defines the last field in a group. The fields in a group are those with field sequence numbers that fall between the sequence numbers of the group header field and this last field. Any future additions to the group must have sequence numbers greater than the group header field and less than the *Thru Field Name*. You cannot add fields to the end of the group without modifying the *Thru Field Name* specification unless you use a group trailer field.
- A ***group trailer field*** is an optional placeholder to mark the position of the final field in a group. It simplifies group maintenance by providing a streamlined mechanism to allow for future additions to the end of a group. You enter a group trailer field (rather than the final field in a group) in a *Thru Field Name* specification on the group header. When you use a group trailer to mark the end of a group, you can add new fields at the end without changing the *Thru Field Name* specification. It also provides a visual clue as to the end of the group when scanning the Data Dictionary. If a field's sequence number is between the group header and group trailer, it belongs to the group.

When defining or using a group field, you must satisfy the following requirements:

- Fields that comprise a group must have contiguous sequence numbers. This restriction should not be taken to imply that fields must be contiguous in terms of physical storage. Refer to the section entitled Special Uses for Synonym Fields for additional information.
- A group can be nested (entirely contained within another group). For example, assume your GL account number is a group field (main account, sub account, division, and branch, perhaps). Further assume that your application contains a key field that is composed of the account number and a year (another group field). The account number group is nested within the account number/year group.
- A group cannot partially overlap the fields in another group. In other words, two groups cannot share a component field without sharing all the component fields in one of the groups. You cannot, for example, define one group consisting of a year and an account number, and another using the same account number and an amount.
- A group can be disjointed (entirely outside of any other group). If you define a synonym for the account number field in the previous example and use it to establish the second group, the two groups are disjointed. Because of the use of a synonym, the groups do not contain a common field.
- A group cannot include a substring field.
- A group must itself be a single occurrence, and must include all occurrences of any multiple-occurrence component fields.
- The final field in a group (or group trailer field, if used) must have a sequence number that is greater than that of the group header field.
- A group cannot be painted as an item on an image. Each component field must be painted individually. For example, you cannot paint an account number group on an input screen. You must, instead, individually paint its component main account, sub account, division, and branch fields on the image.

Group Header Fields

A *group header field* defines a concatenation of fields that form a group and can be referenced together by a single field name (see the Group Fields section for a discussion of group characteristics). For example, an Account group header field may consist of three components, a Primary Account, a Corporate Division, and a Corporate Branch location, each of which are individual fields in APPX. The Account group header provides an alternative to referencing each component when you require all three for a specific process. A group header field occupies no physical space in a file.

The Group Header fields specifications overlay, shown in *Figure 2-4-38*, automatically displays after you complete entries on the Fields specifications screen if the *Field Type* you select is *GROUP HEADER*.

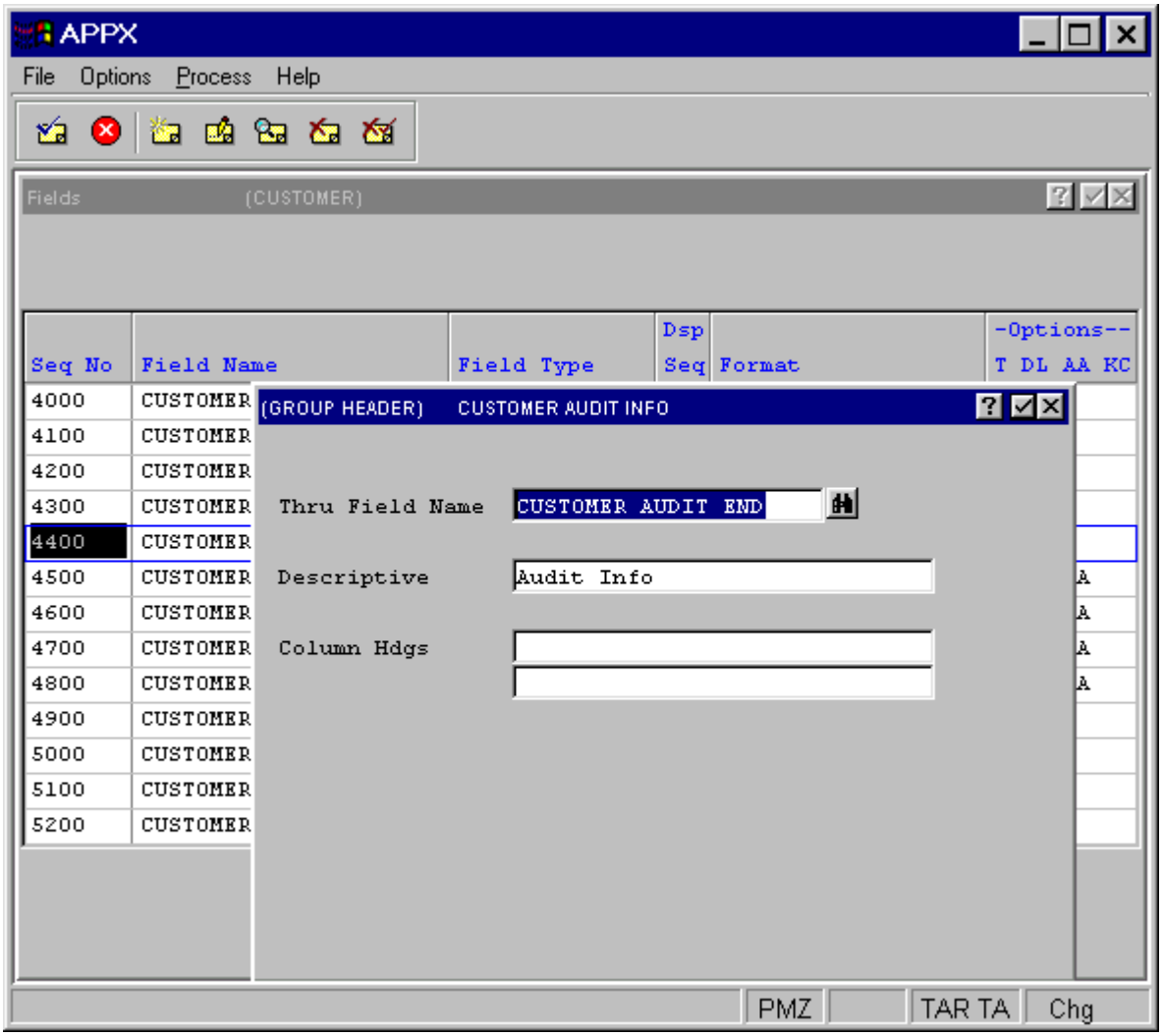


Figure 2-4-38. Group Header Fields Overlay

The Group Header fields overlay contains the following fields:

Thru Field Name

Designates the name of the final field that is included in a group field or concatenation. This field may be an actual field, or a placeholder (group trailer) field. By using a group trailer in this specification, you simplify future expansion at the end of a group field. You can add fields with sequence numbers between those of the group header and group trailer fields to a file, and they become part of the group without your having to update this specification.

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

Group Trailer Fields

A *group trailer field* provides an optional way to allow for future additions to the end of a group field. (A group field is a concatenation of fields that can be referenced together by a single field name: the group header field. See the Group Fields section for a discussion of group characteristics.) All fields within a group must have a sequence number that falls between the number you defined for the group header field and that of the field you entered in the *Thru Field Name* specification.

A group trailer simplifies group field maintenance by providing a placeholder to mark the end of a group. You enter a group trailer field (rather than the final field in a group) in the *Thru Field Name* specification on the group header field. In the future, any field you add with a sequence number between that of the group header and the group trailer becomes part of the group. It is not necessary to modify the *Thru Field Name* specification to include the new field.

For example, assume your account number is a group field that is currently composed of a main account, sub account, and division. If you do not use a group trailer, you must identify the division as the final field in the group (with the *Thru Field Name* specification). If you add a branch component to the end of the account number, you must change the *Thru Field Name* specification to list the branch, rather than the division, as the final field in the group. If instead, you had used a group trailer as the final field in the group, you could add a branch to the group simply by creating the branch field with a sequence number that is greater than that of the division and smaller than that of the group trailer. You would not be required to alter the *Thru Field Name* specification.

As with a group header field, the group trailer occupies no physical space in the file. There are no specifications that are unique to a group trailer field.

Special Uses for Group Fields

Because a group is treated as a single Format string, there are advantages to manipulating a group rather than its component fields. With care, you can achieve results that would not otherwise be possible. For example, group fields can be used to:

- Access discrete portions (substrings) of other fields. For example, assume that you want to divide a four-byte binary number into four individual bytes (to access and manipulate each byte individually). You cannot define a substring field for the number because it is a numeric field, and APPX restricts substrings to alphanumeric fields. You can, however, define a one-byte numeric field

with four occurrences, a group field for the four-byte field, and another group field for the one byte/four-occurrence field. You set the value of the one-byte/four-occurrence group field equal to the value in the four-byte group field. Once the numeric values are copied into the one-byte group field, you can manipulate an individual occurrence (equal to a single byte in the original binary number).

- Simplify statements definition. By using a group, you can manipulate two or more fields with a single statement, rather than a statement for each field. You can even manipulate entire records with a single statement. For example, if you have two files with record structures that contain the same 15 fields, and you combine all 15 into a single group field, you can compare the values in any two records by comparing the values in their respective group fields. If you do not define a group, an IF, SET, or some other statement must be repeated 15 times, once for each field on the record.

CHAPTER 2-5

Work Fields

Introduction to Work Fields

Work fields are temporary fields that are not maintained in a file, but can be manipulated by processes in APPX (usually through the use of statements). They are used to display values that are not stored, such as a calculated line item amount (quantity multiplied by price) or a constant value like π (pi). A work field value is available in memory to a user.

The Work Fields screen, shown in *Figure 2-5-1*, displays attributes that are common to all work fields, regardless of type (alphanumeric or date, for example). You access this screen by selecting the WORK FIELDS option from the Database Management menu overlay.

Work Fields screen

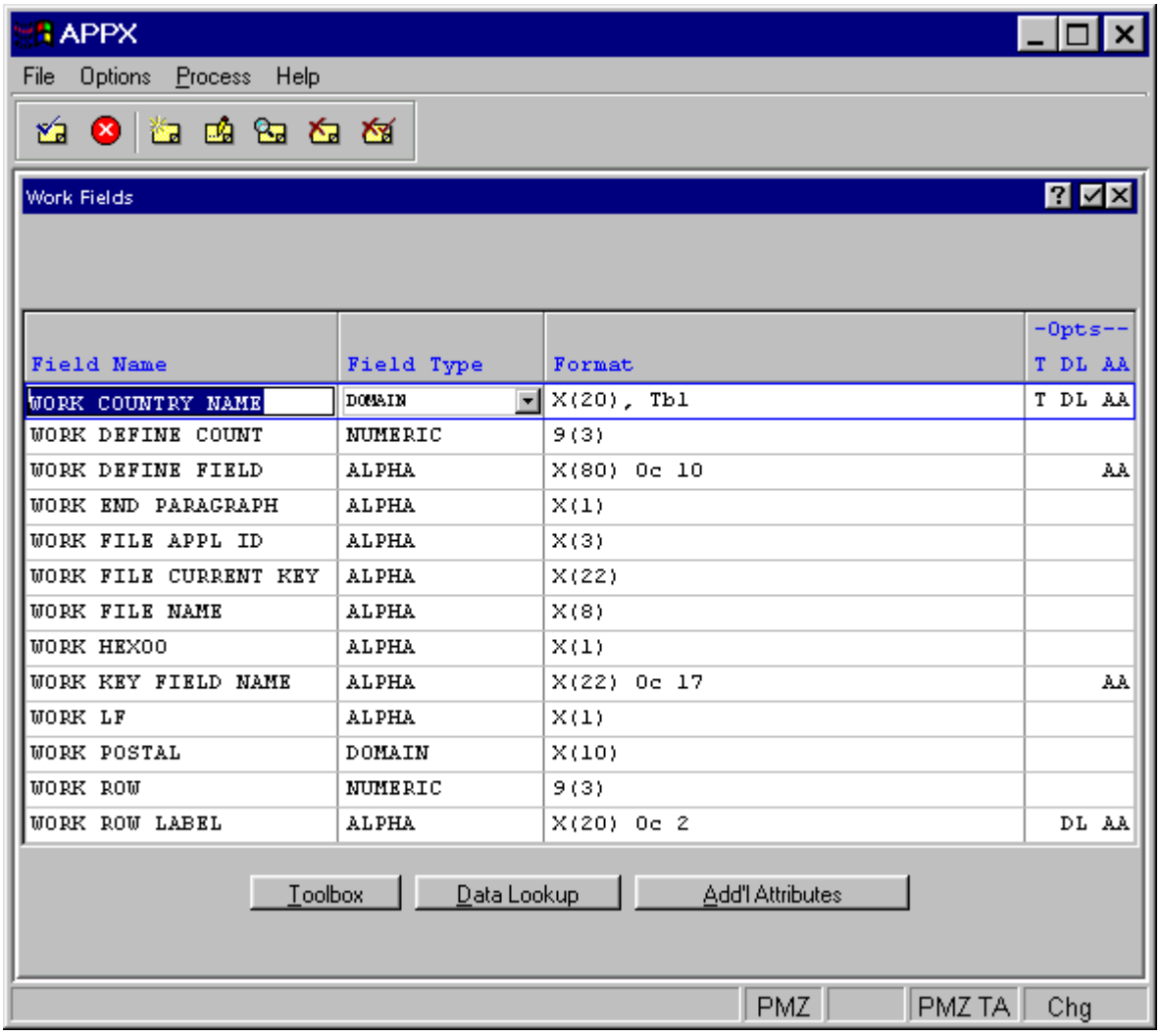


Figure 2-5-1

The Work Fields screen contains the following fields:

Field Name
Uniquely identifies individual work fields within an APPX application. Work fields are referenced by their *Field Name* throughout application design.

Field Type
Categorizes fields according to their function within APPX. The specifications that are presented on subsequent screen overlays depend on the *Field Type* entered here. The *Field Type* options are:

Domain
Assumes the attributes that are defined for an alpha, date, logic, or numeric work field.

Alpha
Work fields may contain numbers, letters, and special characters (-, #, or &, for example).

Format
Work fields may contain an edit mask.

Text
Work fields may contain text.

Token
Work fields use specific values that are stored as corresponding numbers.

Numeric
Work fields are restricted to number, decimal (.), and sign (-) entries.

Date
Work fields may contain dates and/or times, from a hundredth of a second through a century.

Logic
Work fields have possible values of yes ☑, no ☐, or ? to represent an unknown value.

There are three options available from the Work Fields screen: Toolbox, Data Lookup, and Additional Attributes.

Work Fields Toolbox Option

The toolbox overlay appears on the screen when you invoke the TOOLBOX option from the work fields screen. The overlay, shown in Figure 2-5-2, contains a number of features such as Documentation Management and Security Management that will assist you when creating work fields.

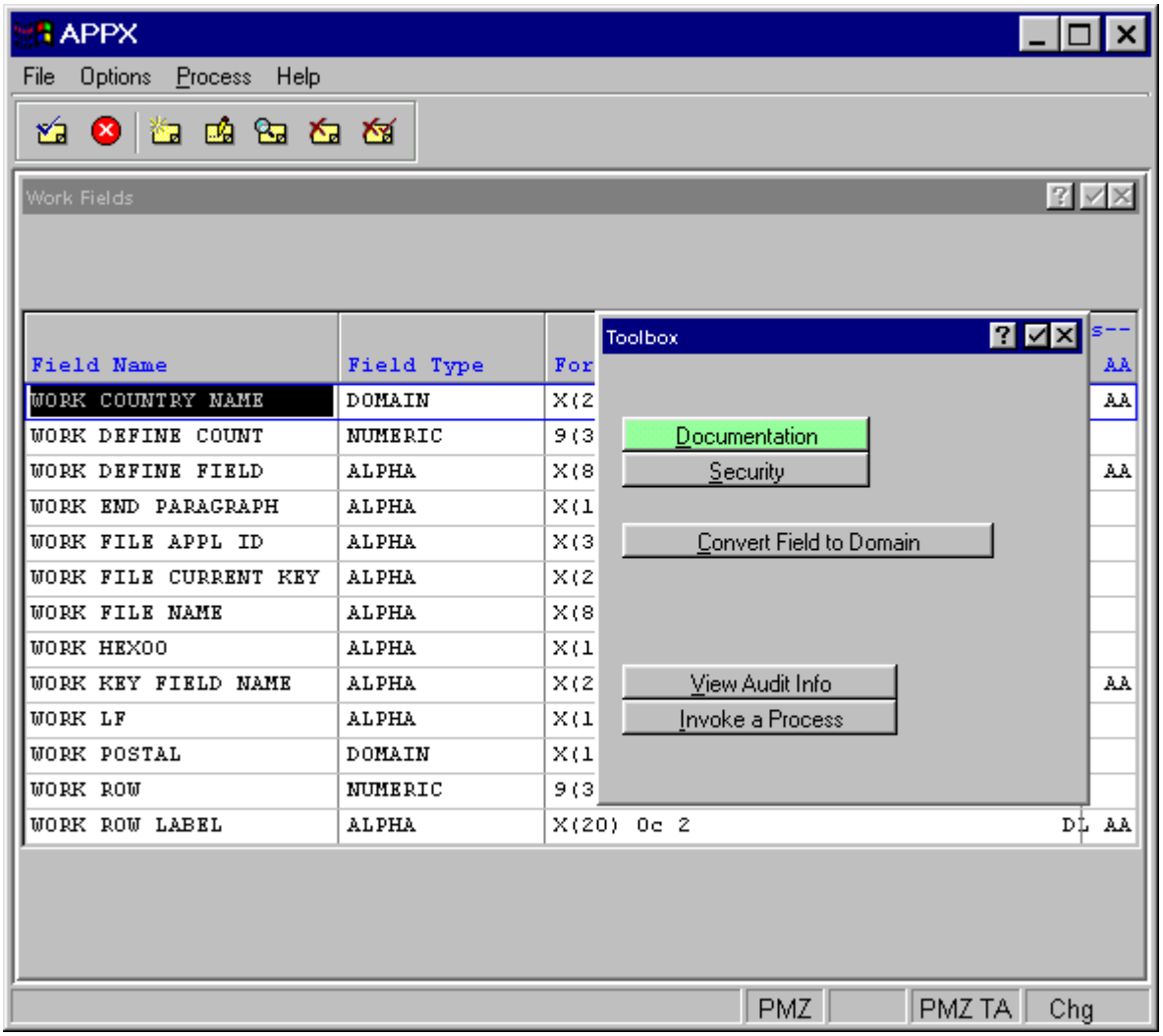


Figure 2-5-2. Work Fields Toolbox Overlay

The options listed in the Toolbox overlay are described below.

Documentation

When you select the DOCUMENTATION option from the Toolbox overlay, you will first see the Documentation overlay which allows you to select between user or technical documentation. Once you have entered your documentation, you are returned to the work fields screen.

Security

The SECURITY option allows you to identify the APPX security code to which a user must have rights in order to access the field. The security code must be four numeric characters. If blank, there are no restrictions.

Convert Field to Domain

This feature allows you to create a domain using the exact specifications of a designated field. When you select this option, you must first name the new domain. Then select the CREATE DOMAIN option and let APPX define a new domain using the characteristics of the currently selected field.

View Audit Info

This feature allows you to see who created the currently selected work field and who changed it most recently. It also gives the date of creation and the date it was last changed. Note that this option (`OPTION 98`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox. See *Chapter 3-3, Process Definition* for more information about the VIEW AUDIT INFO option.

Invoke a Process

This feature is available to facilitate testing. It allows you to invoke any process. Note that this option (`OPTION 99`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox. See *Chapter 3-3, Process Definition* for a more detailed explanation of the Invoke a Process facility.

Work Fields Data Lookup Option

DATA LOOKUP accesses the Data Lookup Overlay, shown in *Figure 2-5-3*, where you can identify the application, file, and key field that can be accessed by users when selecting the SCAN option. The field you specify must be a primary or alternate key to the file you specified. Upon completion, you are returned to the Work Fields screen.

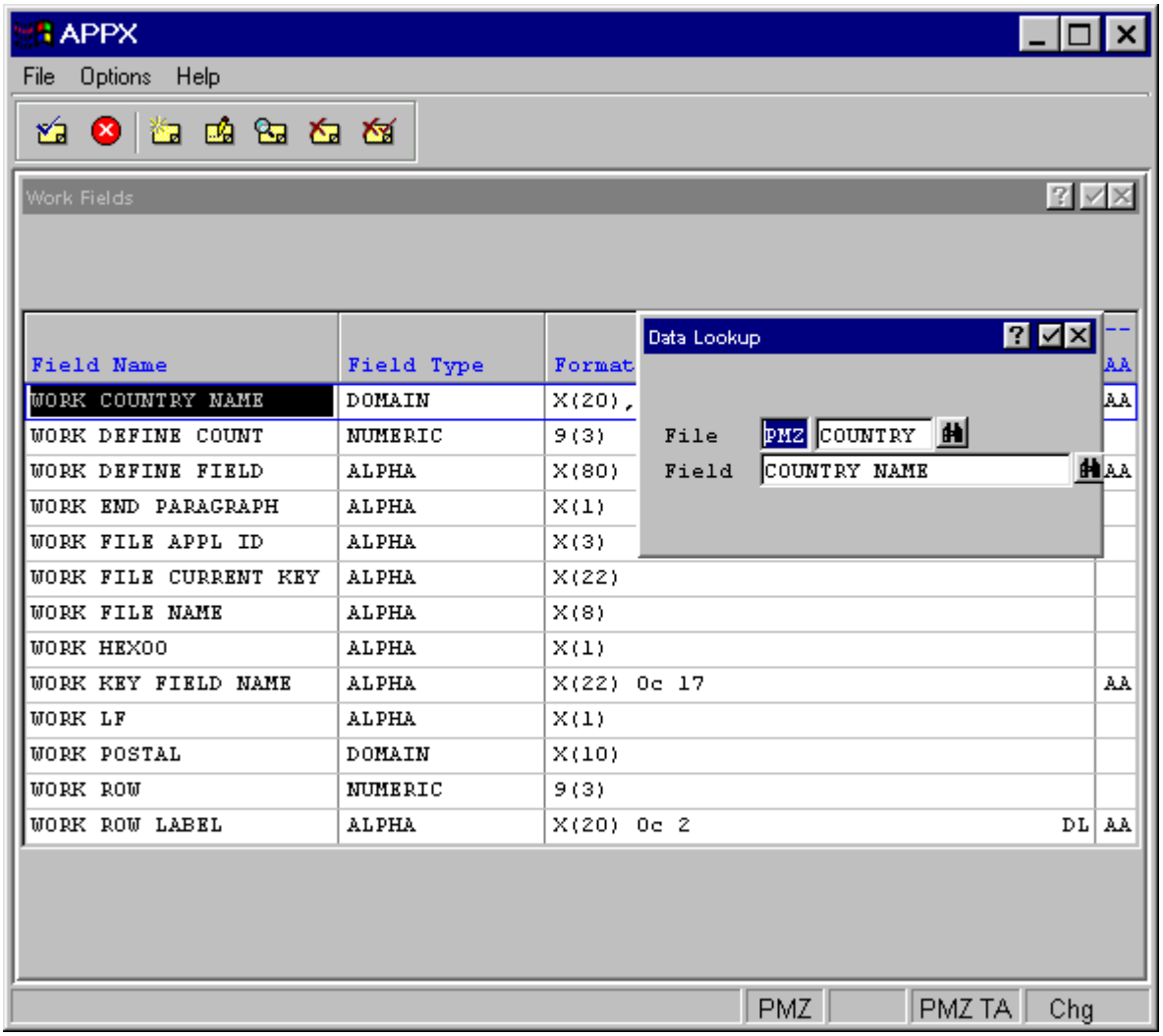


Figure 2-5-3. Work Fields Data Lookup Overlay

Work Fields Additional Attributes Option

ADDITIONAL ATTRIBUTES accesses the work fields Additional Attributes overlay, from which you can enter additional parameters for this work field. The specific parameters vary according to the type of work field, and are presented later in the sections pertaining to each type. Upon completion, you are returned to the Work Fields screen.

Common Work Fields Specifications

When you select ENTER ☑ from the Work Fields screen, the work field specifications overlay displays. This overlay displays differently depending on the type of work field that is currently selected. For information about type-specific Work Field Specifications overlays, refer to the following sections which discuss each type in detail.

Domain Work Fields

A domain-type work field assumes the attributes that are defined for any type of domain in the same or an outside application. To use an assumed value from a domain definition, leave the specification blank. The Domain-Type Work field Specifications overlays, such as the ones shown in *Figures 2-5-4* and *2-5-8,* automatically display after you complete the entries on the Work fields specifications screen if the *Field Type* you select is *DOMAIN. Figure 2-5-4* displays if the domain type is *alpha, format, text, token, date,* or *logic. Figure 2-5-8* displays if the domain type is *NUMERIC.*

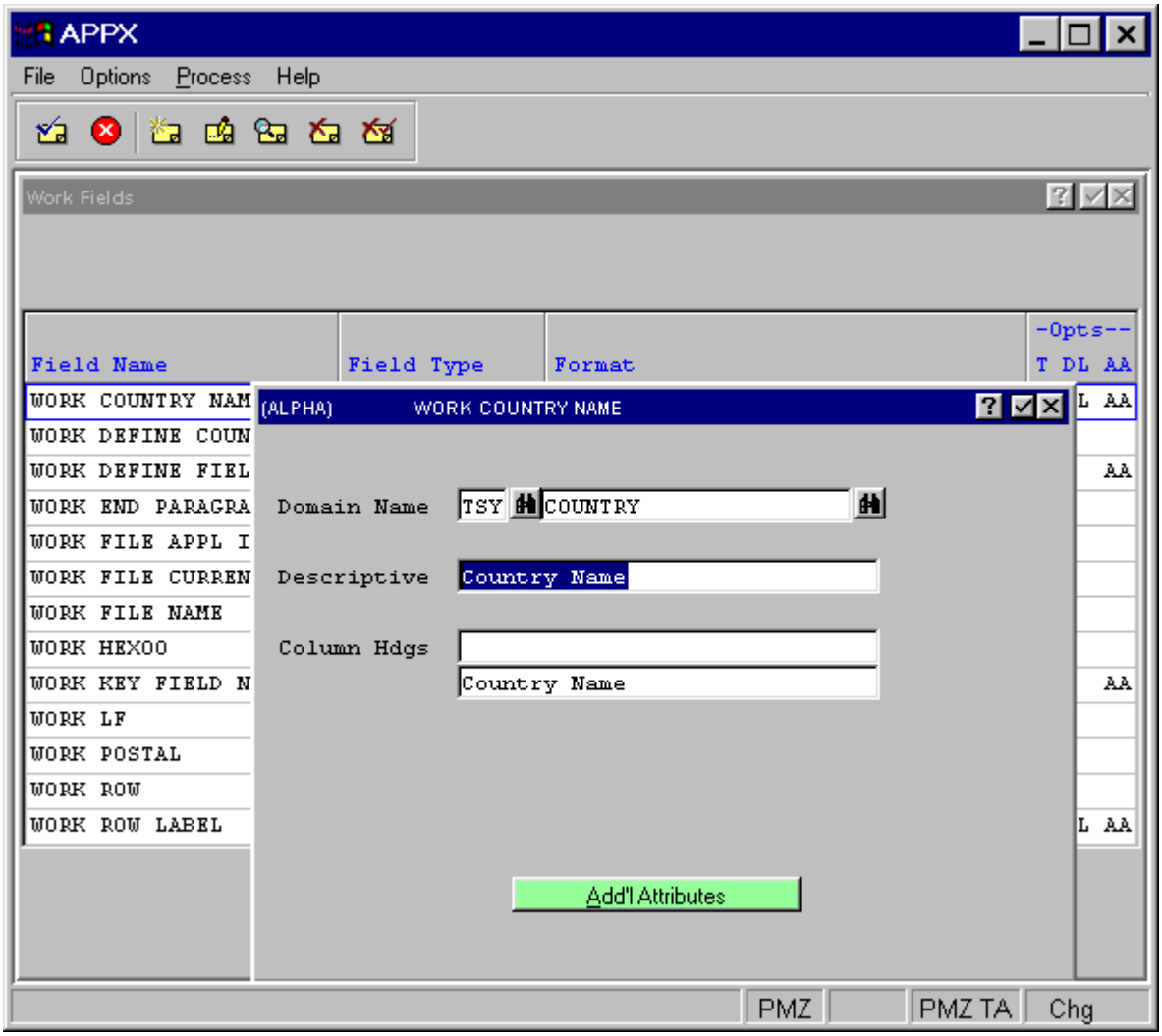


Figure 2-5-4. Domain Work Field Specification Overlay (Alpha Domain)

The following fields are contained on the Domain Type Work field Specifications overlay for Alpha, Format, Text, Token, Date, and Logic Domain-Type Work fields:

Domain Name

Links a domain-type work field to a specific domain in the same or another APPX application.

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used. Because *Descriptive* is less cryptic and can be more flexibly formatted than *Field Name*, it is easier to under-stand. As a result, the system uses *Descriptive* when communicating with users. *Descriptive* is used, for example, when the value of a field displays to a user on a standard image, such as the *Select Access Path* display or an APPX-generated subheading. We suggest that you use a singular entry for *Descriptive,* and avoid appending the word "field".

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

The Domain Type Work field Specification overlays for Alpha, Format, Text, Token, Date, and Logic Domains contain the option ADDITIONAL ATTRIBUTES described in the next section.

Domain Work Fields Additional Attributes

ADDITIONAL ATTRIBUTES accesses the domain-type work field Additional Attributes overlay, from which you can enter additional parameters for this domain-type work field. The Domain-Type Work field Additional Attributes overlay for *alpha*, *text*, *token* and *date* Domains is shown in *Figure 2-5-5.* The Domain-Type Work field Additional Attributes overlay for *format* Domains is shown in *Figure 2-5-6*. The Domain-Type Work field Additional Attributes overlay for *logic* Domains is shown in *Figure 2-5-7*. Upon completion, you are returned to the Domain-type work field specifications overlay.

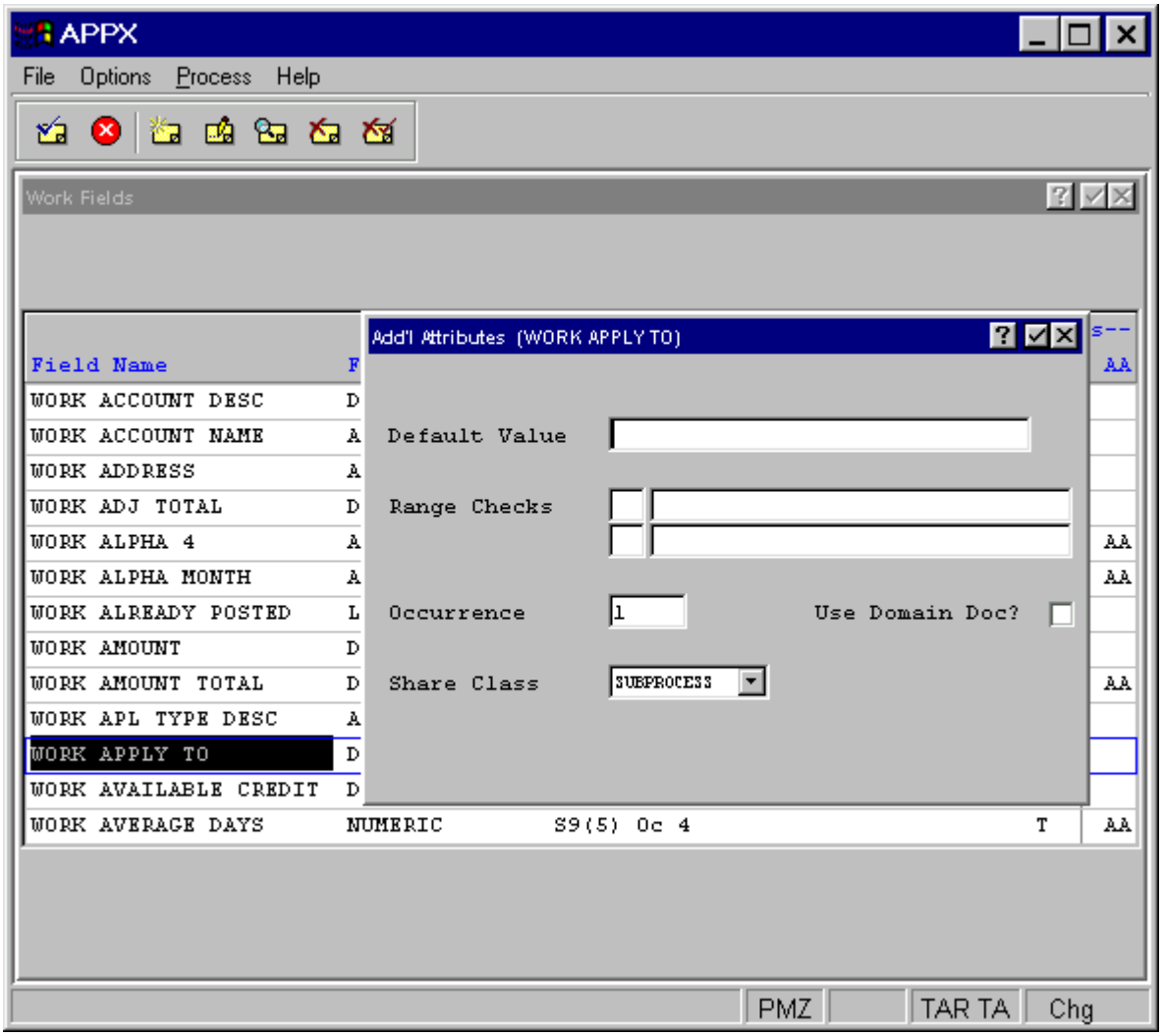


Figure 2-5-5. Domain Work Field Additional Attributes Overlay for Alpha, Text, Token and Date Domains

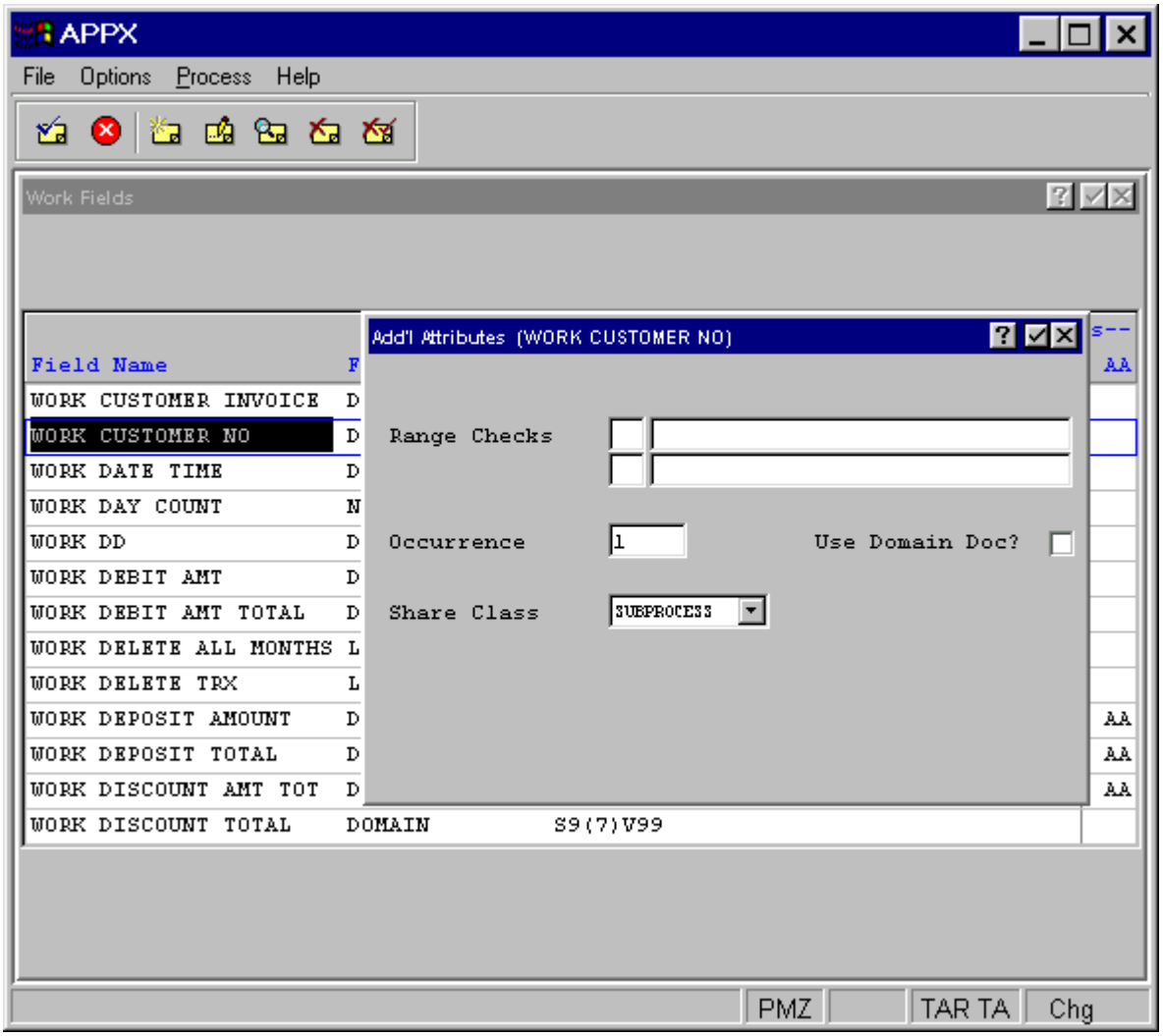


Figure 2-5-6. Domain Work Field Additional Attributes Overlay for Format Domains

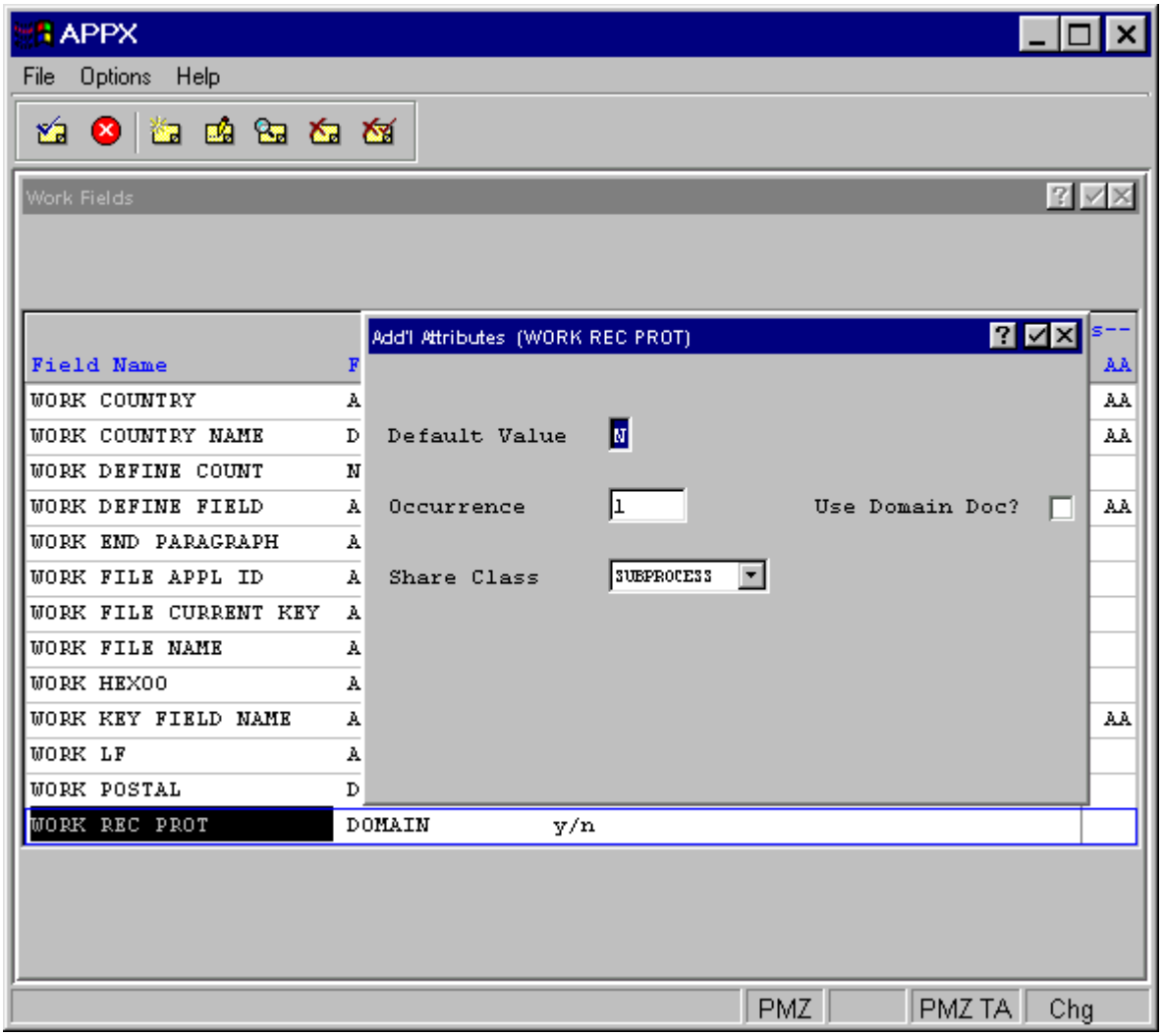


Figure 2-5-7. Domain Work Field Additional Attributes Overlay for Logic Domains

Some or all of the following Work fields are contained on the Domain-Type Work field Additional Attributes overlays for *alpha*, *format, logic, text*, *token* and *date* Domain types:

Default Value

Establishes a modifiable starting value for all referenced work fields whenever they are first displayed in add mode in an input image, unless modified by a user. If blank, there is no starting value displayed on the image.

Occurrence

Determines the number of times a field is present in a record. If blank, APPX assumes a value of 1. This field may contain up to five numeric characters and is restricted to a maximum of 32,767. See the Occurrence section for a more comprehensive discussion of this field.

Range Checks

Defines a series of allowable values that APPX uses for online data validation. The first component represents the relation. The options are ***EQ*** (equal to), ***NE*** (not equal to), ***GT*** (greater than), ***LT*** (less than), ***GE*** (greater than or equal to), and ***LE*** (less than or equal to). The second component represents the value that is assigned to the relation. If you enter two relations, APPX infers an AND relationship.

Use Domain Doc?

Click the checkbox to toggle between "yes" ☑ if you want APPX to use the documentation from the domain as Help text and "no" ☐ if you plan to enter Help text unique to this field.

Share Class

Defines at what level the values contained in this work field can be shared. The *Share Class* options are:

- ***SUBPROCESS*** — The values can be shared between a process and its subprocesses.
- ***RELATED*** — The values can be shared among related processes in a job.
- ***DETACHED*** — The values are available until the user logs off.
- ***NOT SHARED*** — The values are available only for the current process.
- The default is *SUBPROCESS*.

Domain Work Fields (Numeric Domain)

The Domain type work field specification overlay, shown in *Figure 2-5-8*, displays after you select ENTER ☑ from the work fields screen if the *Field Type* you select is *DOMAIN* and the domain type is *NUMERIC*.

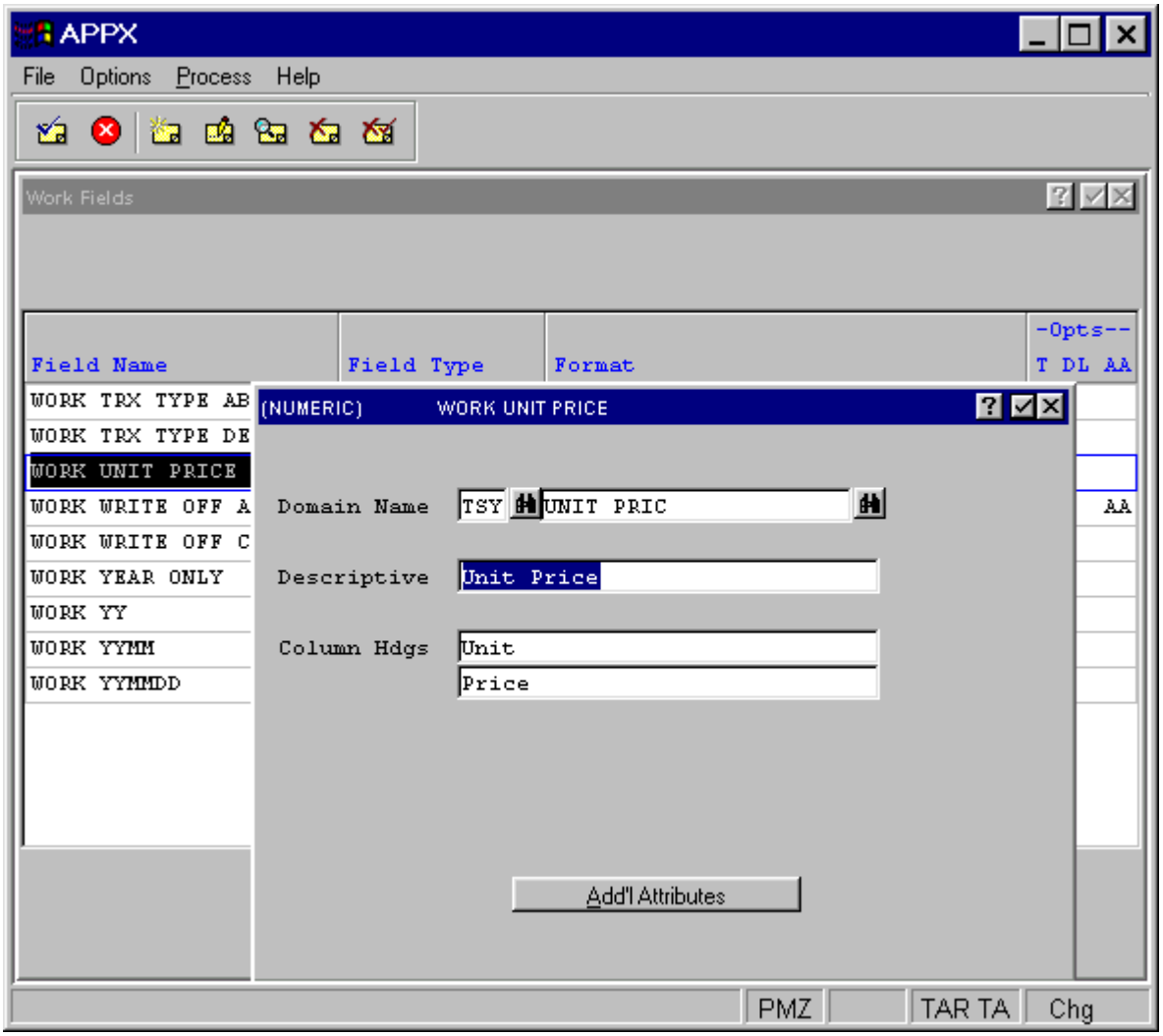


Figure 2-5-8. Domain Work Field Specification Overlay (Numeric Domain)

The following fields are contained on the specifications overlay for Numeric Domain-Type Work fields:

Domain Name

Links a domain-type work field to a specific domain in the same or another APPX application.

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

The Domain Type Work field Specification overlay for Numeric Domains contains the following option as described in the corresponding section:

- Additional Attributes

Domain Work Fields (Numeric Domain) Additional Attributes

The Domain-Type Work field Additional Attributes overlay for *numeric* Domains is shown in *Figure 2-5-9.* Upon completion, you are returned to the numeric Domain-type work field specifications overlay.

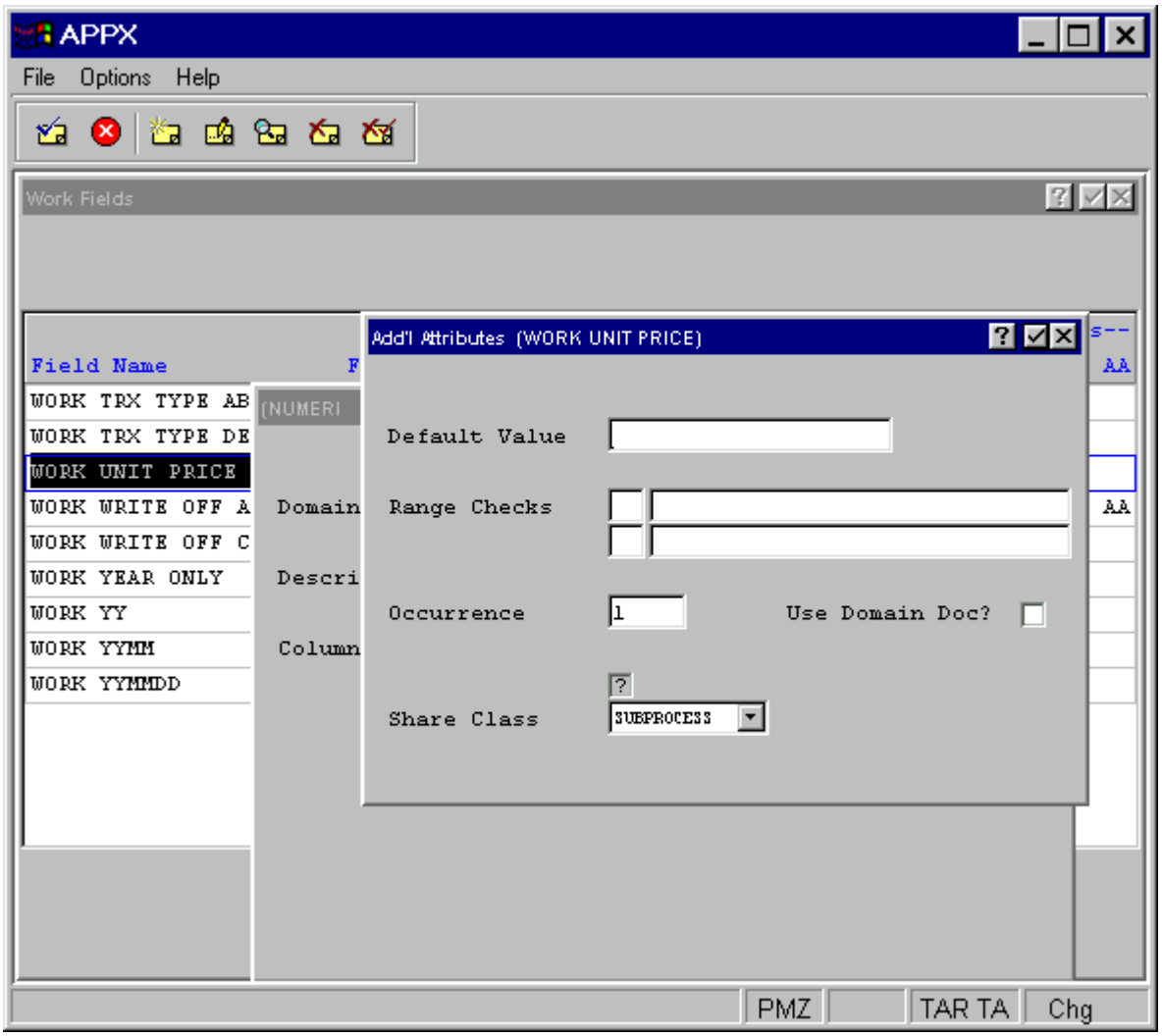


Figure 2-5-9. Domain Work Field Additional Attributes Overlay for Numeric Domains

The Additional Attributes overlay contains the following fields:

Default Value

Establishes a modifiable starting value for all referenced fields whenever they are first displayed in add mode in an input image, unless modified by a user. If blank, there is no starting value displayed on the image.

Range Checks

Defines a series of allowable values that APPX uses for online data validation.. The first component represents the relation. The options are ***EQ*** (equal to), ***NE*** (not equal to), ***GT*** (greater than), ***LT*** (less than), ***GE*** (greater than or equal to), and ***LE*** (less than or equal to). The second component represents the value that is assigned to the relation. If you enter two relations, APPX infers an AND relationship.

Occurrence

Determines the number of times a field is present in a record. If blank, APPX assumes a value of 1. This field may contain up to five numeric characters and is restricted to a maximum of 32,767. See the Occurrence section for a more comprehensive discussion of this field.

Share Class

Defines at what level the values contained in this work field can be shared. The *Share Class* options are:

- ***SUBPROCESS*** — The values can be shared between a process and its subprocesses.
- ***RELATED*** — The values can be shared among related processes in a job.
- ***DETACHED*** — The values are available until the user logs off.
- ***NOT SHARED*** — The values are available only for the current process.
- The default is *SUBPROCESS*.

Alpha Work Fields

An Alpha work field can contain numbers, letters, and special characters, such as *#, $,* and *&.* Examples include names, addresses, certain account numbers or serial numbers, and text.

The Alpha work field overlay, shown in *Figure 2-5-10*, displays after you complete the entries on the work fields screen if the work *Field Type* you select is *ALPHA*.

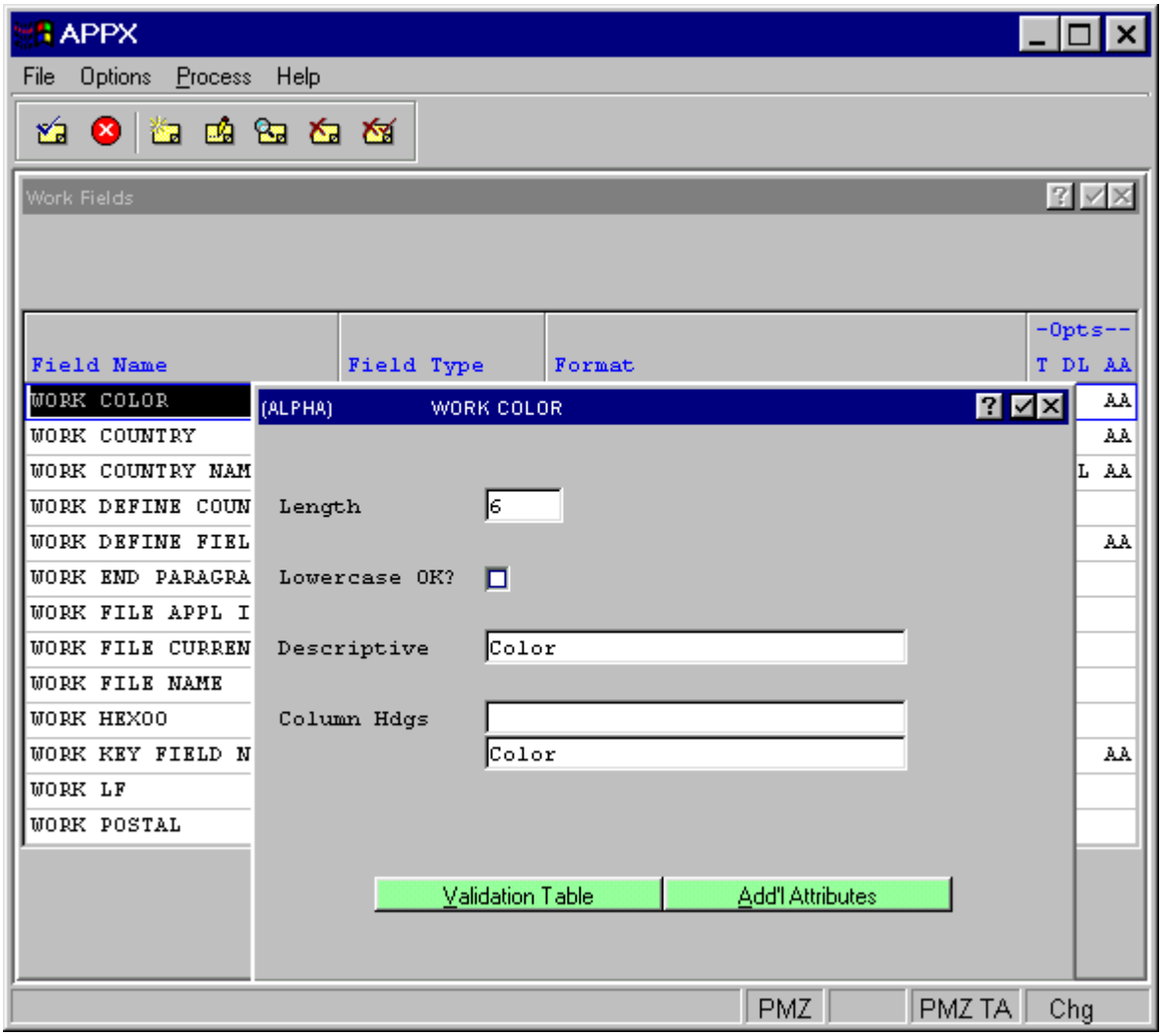


Figure 2-5-10. Alpha Work Fields Overlay

The Alpha Work field Specifications overlay contains the following fields:

Length

Designates the number of character positions, or length, of this field.

Lowercase OK?

Indicates whether or not a modifiable item on an interactive screen can be entered in upper- and lowercase letters. If "no" ☐, the entry is stored and displayed exclusively as uppercase, regardless of how a user enters the information. This entry does not affect statements. A lowercase value established with a statement displays as lowercase, even if this specification prohibits lowercase entries. Click the checkbox to toggle between "yes" ☑ and "no" ☐.

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

For Alpha work field type, the following options are available as described in the corresponding sections:

- Validation Table
- Additional Attributes

Alpha Work Fields Validation Table

The VALIDATION TABLE option from the Alpha Work Fields overlay accesses the Validation Table overlay, shown in *Figure 2-5-11*. A validation table provides a more powerful alternative to the use of range checks for data validation when there is a known set of values. Through the list feature, it provides the user with visibility of all the possible field values as well as the ability to select the desired value.

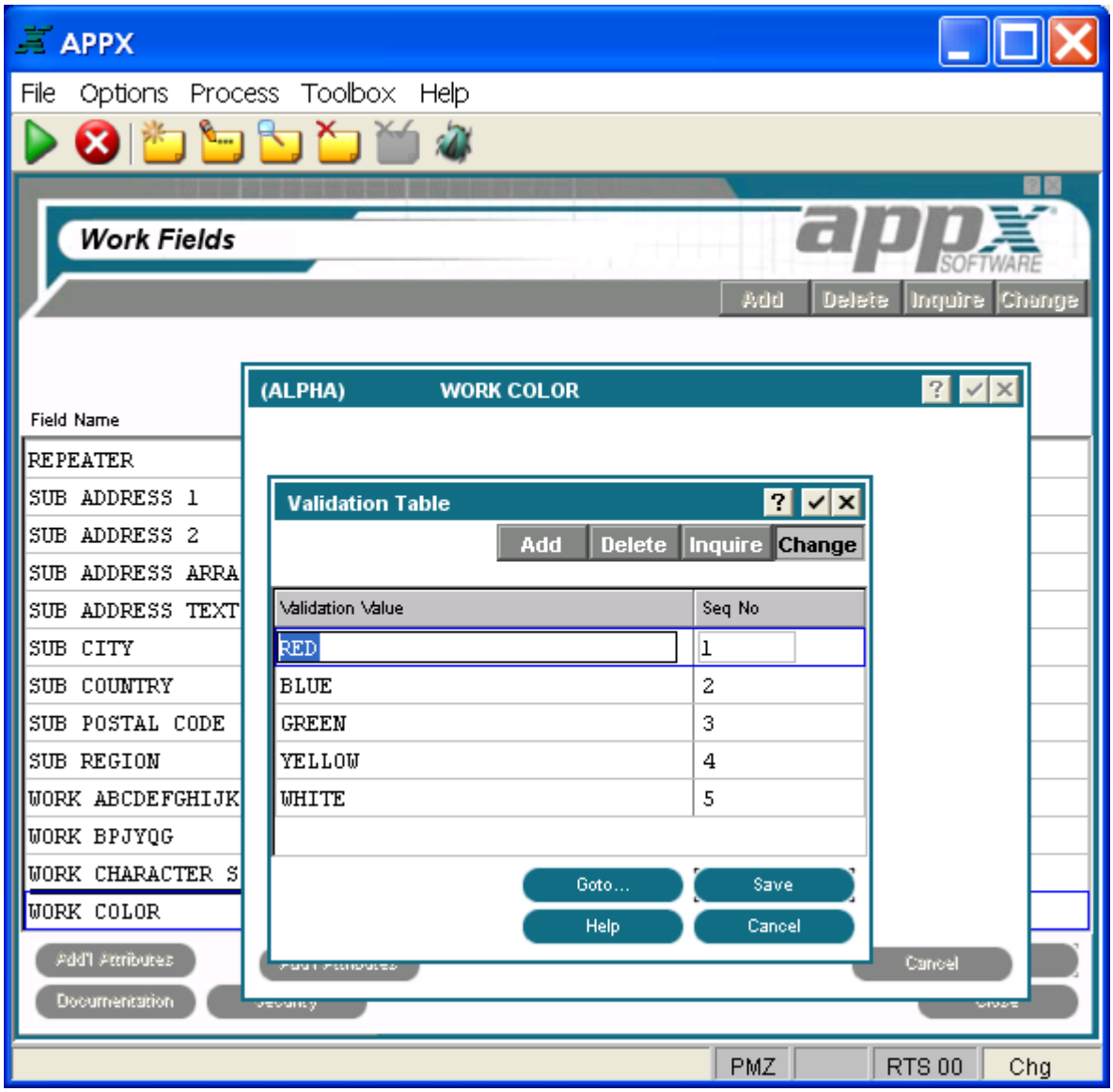


Figure 2-5-11. Alpha Work Fields Validation Table Overlay

You can establish a validation table for any Alpha work field in the data dictionary. The entries in the table are case-sensitive. If you enter a validation value in all capital letters, an application user must enter the value in all capital letters.

Validation table operations are database-independent. They access a designer-defined table in application design, not data contained in a file in a database, for the range of possible values. As a result, validation does not vary from database to database. It is automatically available to, and exactly the same for, every database that is associated with an application. All validation values are available for review and selection by the end user through the list ▼ feature.

The Validation Table overlay contains the following fields:

Validation Value

Provides a valid field value. It is case-sensitive and is used for online review, selection, and validation. Note that validation values newly added to the data dictionary are not available to a current APPX user until the user exits and re-enters APPX.

Seq No

Provides an identifying number for the associated validation value. It is defaulted by APPX and can be overridden, but it must be unique within a table.. It determines the order in which validation values are presented, both for selection by the user and for editing in application design. APPX assigns the first validation value a number of *1*, with additional values assigned a number that is equal to the highest existing sequence number plus *1*. APPX does not require that sequence numbers be contiguous within a table.

Alpha Work Fields Additional Attributes

From the Alpha work field overlay, you have an ADDITIONAL ATTRIBUTES option. The Alpha work field Additional Attributes overlay is shown in *Figures 2-5-12*.

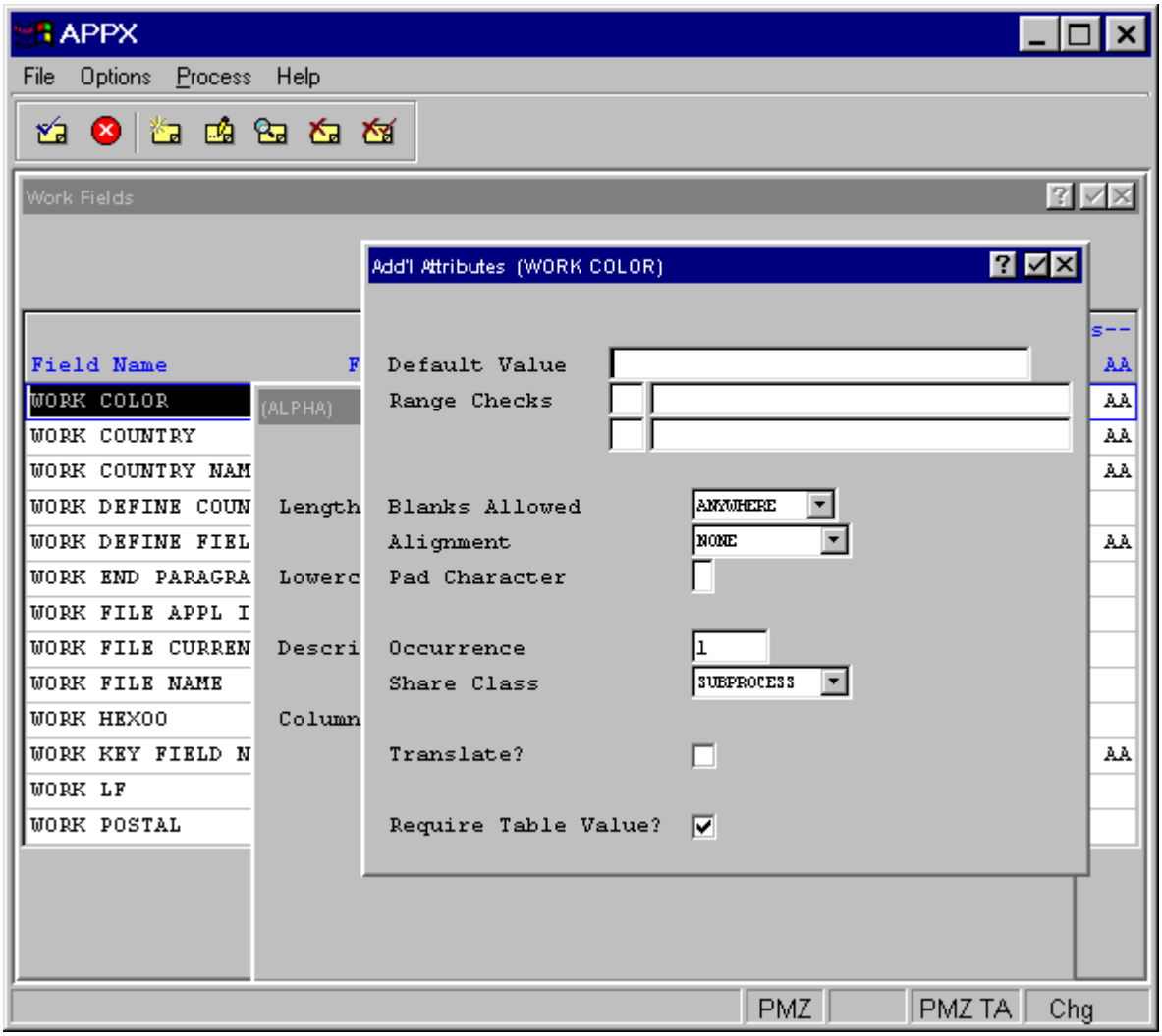


Figure 2-5-12. Alpha Work Field Additional Attributes Overlay

The Alpha Work field Additional Attributes Overlay contains the following fields:

Default Value

Establishes a modifiable starting value for all image items whenever they are first displayed in add mode in an input image, unless modified by a user. If blank, there is no starting value displayed on the image.

Range Checks

Defines a series of allowable values that APPX uses for online data validation. The first component represents the relation. The options are ***EQ*** (equal to), ***NE*** (not equal to), ***GT*** (greater than), ***LT*** (less than), ***GE*** (greater than or equal to), and ***LE*** (less than or equal to). The second component represents the value that is assigned to the relation. If you enter two relations, APPX infers an AND relationship.

Blanks Allowed

Indicates whether or not blanks are permitted and, if so, where they can be placed. The *Blanks Allowed* options are ***NONE, TRAILING,*** and ***ANYWHERE***. The default is *ANYWHERE*.

Alignment

Determines whether or not data in a field is aligned for storage and/or display and, if so, the type of alignment. The *Alignment* options are ***NONE, LEFT, JUSTIFIED, RIGHT*** and ***CENTERED.*** The default is *NONE*.

Pad Character

Indicates the character to be used instead of leading and trailing spaces in a field.

Occurrence

Determines the number of times a field is present in a record. If blank, APPX assumes a value of 1. This field may contain up to five numeric characters and is restricted to a maximum of 32,767. See the Occurrence section for a more comprehensive discussion of this field.

Share Class

Defines at what level the values contained in this work field can be shared. The *Share Class* options are:

- ***SUBPROCESS*** — The values can be shared between a process and its subprocesses.
- ***RELATED*** — The values can be shared among related processes in a job.
- ***DETACHED*** — The values are available until the user logs off.
- ***NOT SHARED*** — The values are available only for the current process.
- The default is *SUBPROCESS*.

Translate?

Indicates if the value of the work field should be translated into a designated language if APPX's Language Translation feature is being used. Since this option does not require a validation table for a work field (unlike a file field), it is the designer's responsibility to ensure that a translatable work field without a validation table is read only. Click the checkbox to toggle between "yes" ☑, "no" ☐, and unknown ? values. If the translation feature is not in use, retain the default value of ?.

Require Table Value?

Indicates whether or not to require one of the validation table values to be entered. To force the user to enter one of the defined values, click the checkbox to toggle it to "yes" ☑. If a "no" ☐ value is selected, the validation table shown when the user selects the list feature, is advisory only and the user can enter a value not listed in the table. If there is no validation table defined, this option does not apply and cannot be selected. If there is a validation table and *Translate?* is ☑, this option should also be ☑.

Format Work Fields

A format work field is like an alpha work field except that its primary characteristic is an edit mask.

The Format Work Field overlay, shown in *Figure 2-5-13*, displays after you complete the entries on the Work Fields screen if the work *Field Type* you select is *FORMAT*.

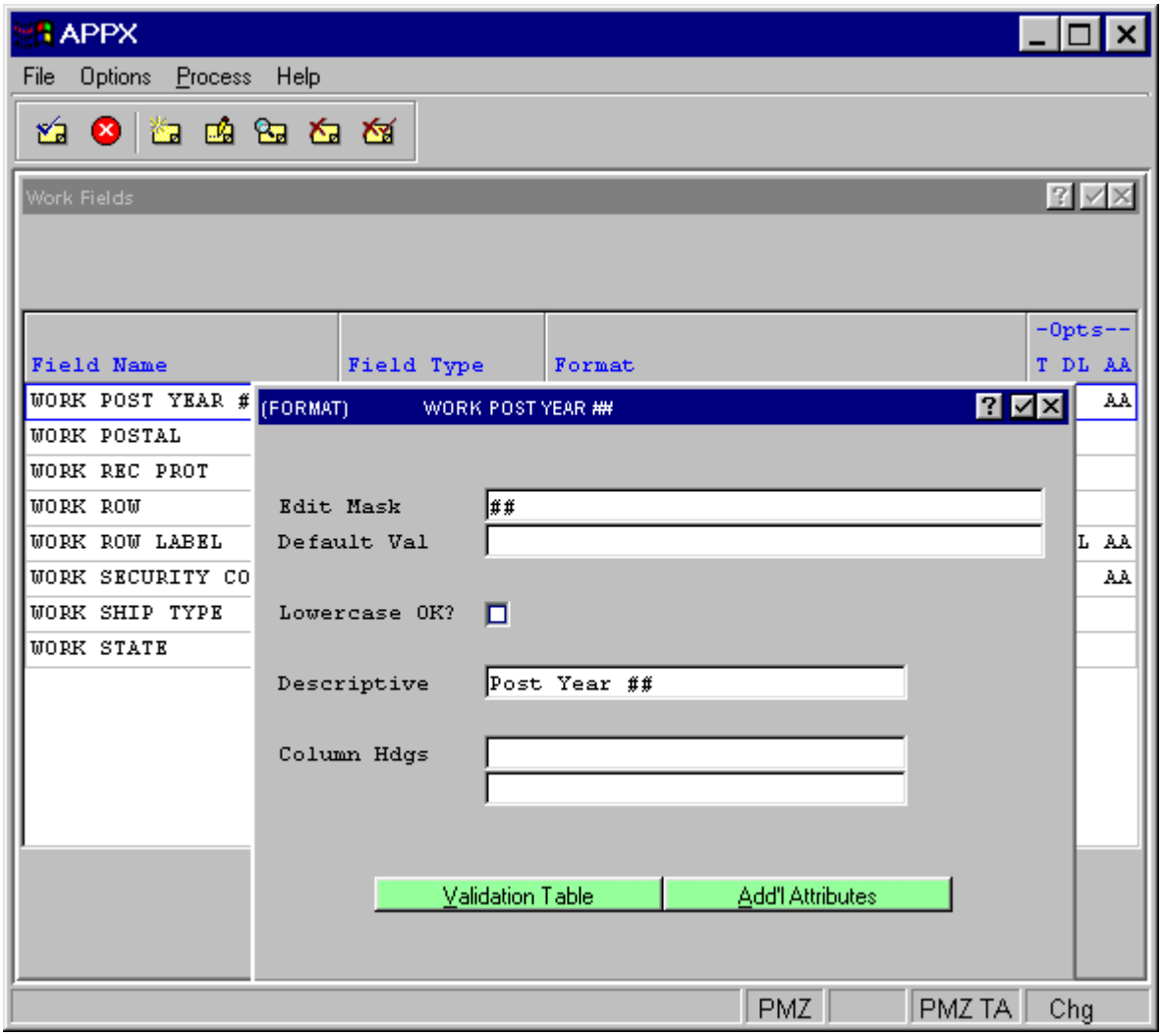


Figure 2-5-13. Format Work Field Overlay

The Format Work Field overlay contains the following fields:

Edit Mask

Establishes format requirements for data validation. A mask character representing the allowable values is entered for each position in a field. The mask characters are:

- # (numbers only)
- @ (letters only)

- & (numbers and letters only)
- **!** (any characters)
- * (use default value character) — The valid character for this position in a mask is defined by referencing the same position in the default value.
- For example, you can mask a phone number field with the following entry: ###*###*####.

Default Value

Establishes a starting value for all referenced fields whenever they are first displayed in add mode in an input image, unless modified by a user. If blank, there is no starting value displayed on the image. The positioning of constant characters in the default value must correspond to the positions of *s, if any, in the edit mask. For example, a possible default value for the phone number mask indicated in *Edit Mask* earlier is: _ _ _ - _ _ _ - _ _ _ _ (the underlines are blanks).

Lowercase OK?

Indicates whether or not a modifiable item on an interactive screen can be entered in upper- and lowercase letters. If "no" ☐, the value is stored and displayed exclusively as uppercase, regardless of how a user enters the information. This entry does not affect statements. A lowercase value established with a statement overrides a "no" entry in this field and displays as lowercase. Click the checkbox to toggle between "yes" ☑ and "no" ☐.

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

For Format work field type, the following options are available as described in the corresponding sections:

- Validation Table
- Additional Attributes

Format Work Fields Validation Table

THE VALIDATION TABLE option from the Format Work Fields overlay accesses the Validation Table overlay, shown in *Figure 2-5-14*. A validation table provides a more powerful alternative to the use of range checks for data validation when there is a known set of values. Through the list feature, it provides the user with visibility of all the possible field values as well as the ability to select the desired value.

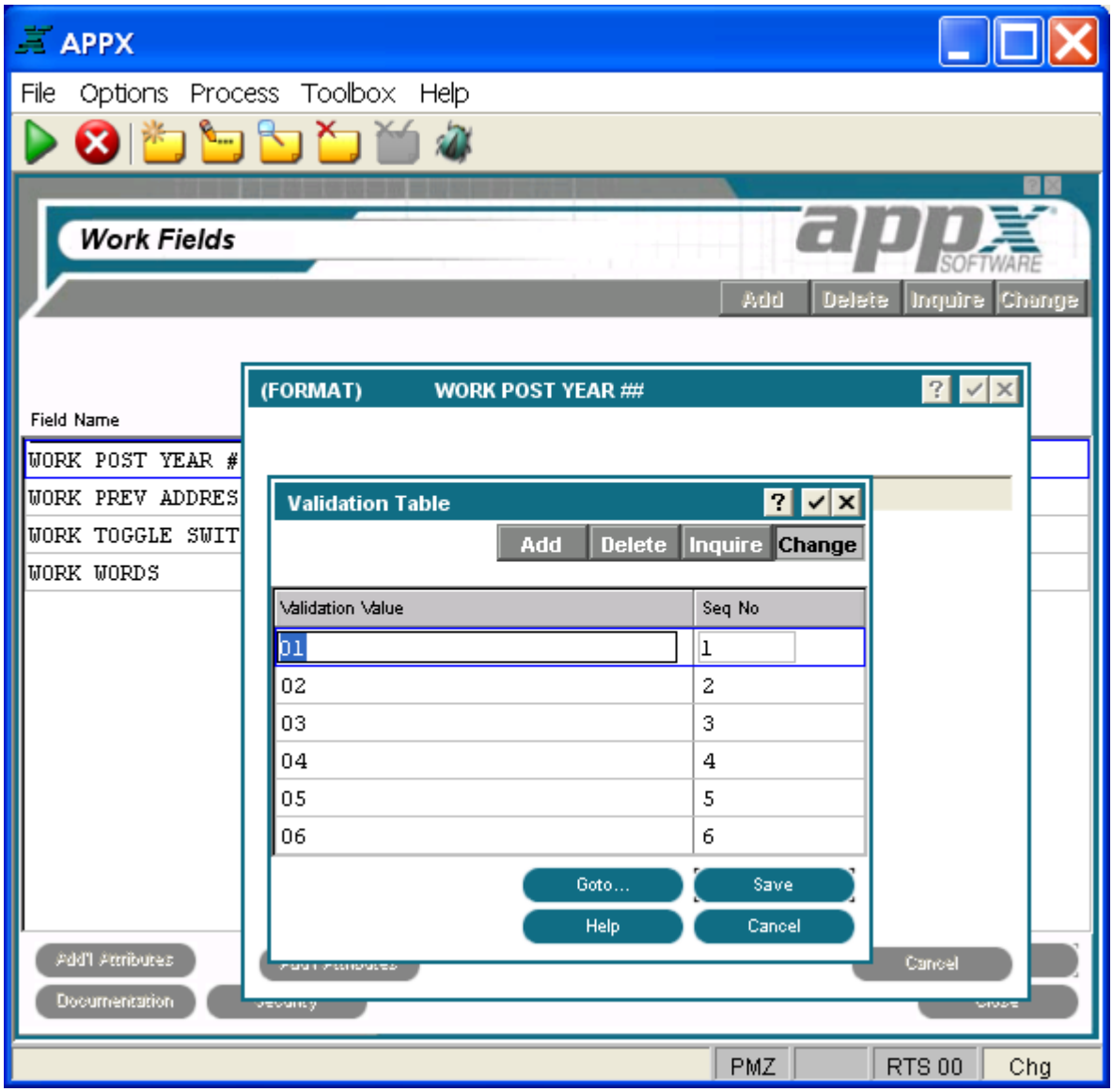


Figure 2-5-14. Format Work Fields Validation Table Overlay

You can establish a validation table for any Format work field in the data dictionary. The entries in the table are case-sensitive. If you enter a validation value in all capital letters, an application user must enter the value in all capital letters.

Validation table operations are database-independent. They access a designer-defined table in application design, not data contained in a file in a database, for the range of possible values. As a result, validation does not vary from database to database. It is automatically available to, and exactly the same for, every database that is associated with an application. All validation values are available for review and selection by the end user through the list feature.

The Validation Table overlay contains the following fields:

Validation Value

Provides a valid field value. It is case-sensitive and is used for online review, selection, and validation. Note that validation values newly added to the data dictionary are not available to a current APPX user until the user exits and re-enters APPX.

Seq No

Provides an identifying number for the associated validation value. It is defaulted by APPX and can be overridden, but it must be unique within a table.. It determines the order in which validation values are presented, both for selection by the user and for editing in application design. APPX assigns the first validation value a number of *1*, with additional values assigned a number that is equal to the highest existing sequence number plus *1*. APPX does not require that sequence numbers be contiguous within a table.

Format Work Fields Additional Attributes

From the Format work field overlay, you have an ADDITIONAL ATTRIBUTES option. The Format work field Additional Attributes overlay is shown in *Figure 2-5-15*.

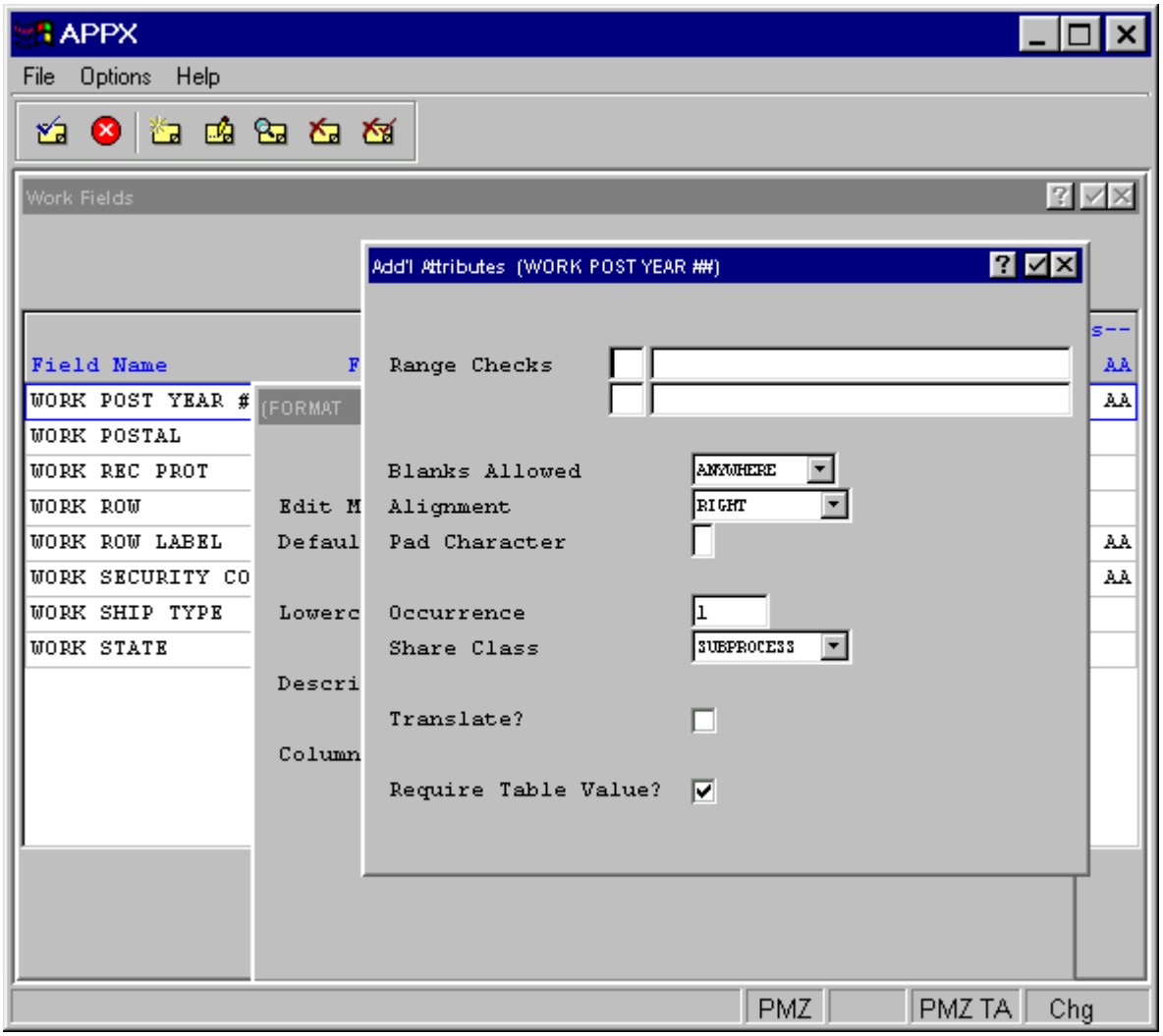


Figure 2-5-15. Format Work Field Additional Attributes Overlay

The Format Work field Additional Attributes Overlay contains the following fields:

Range Checks

Defines a series of allowable values that APPX uses for online data validation. The first component represents the relation. The options are ***EQ*** (equal to), ***NE*** (not equal to), ***GT*** (greater than), ***LT*** (less than), ***GE*** (greater than or equal to), and ***LE*** (less than or equal to). The second component represents the value that is assigned to the relation. If you enter two relations, APPX infers an AND relationship.

Blanks Allowed

Indicates whether or not blanks are permitted and, if so, where they can be placed. The *Blanks Allowed* options are ***NONE, TRAILING,*** and ***ANYWHERE***. The default is *ANYWHERE*.

Alignment

Determines whether or not data in a field is aligned for storage and/or display and, if so, the type of alignment. The *Alignment* options are ***NONE, LEFT, JUSTIFIED, RIGHT*** and ***CENTERED.*** The default is *NONE*.

Pad Character

Indicates the character to be used instead of leading and trailing spaces in a field.

Occurrence

Determines the number of times a field is present in a record. If blank, APPX assumes a value of 1. This field may contain up to five numeric characters and is restricted to a maximum of 32,767. See the Occurrence section for a more comprehensive discussion of this field.

Share Class

Defines at what level the values contained in this work field can be shared. The *Share Class* options are:

- ***SUBPROCESS*** — The values can be shared between a process and its subprocesses.
- ***RELATED*** — The values can be shared among related processes in a job.
- ***DETACHED*** — The values are available until the user logs off.
- ***NOT SHARED*** — The values are available only for the current process.
- The default is *SUBPROCESS*.

Translate?

Indicates if the value of the work field should be translated into a designated language if APPX's Language Translation feature is being used. Since this option does not require a validation table for a work field (unlike a file field), it is the designer's responsibility to ensure that a translatable work field without a validation table is read only. Click the checkbox to toggle between "yes" ☑, "no" ☐, and unknown ? values. If the translation feature is not in use, retain the default value of ?.

Require Table Value?

Indicates whether or not to require one of the validation table values to be entered. To force the user to enter one of the defined values, click the checkbox to toggle it to "yes" ☑. If a "no" ☐ value is selected, the validation table shown when the user selects the list feature, is advisory only and the user can enter a value not listed in the table. If there is no validation table defined, this option does not apply and cannot be selected. If there is a validation table and *Translate?* is ☑, this option should also be ☑.

Text Work Fields

A text work field allows a user to enter a block of text on the data entry screen.

The text work field overlay, shown in *Figure 2-5-16*, allows you to add text specific attributes to those already defined for work fields. The text work field specifications overlay displays after you complete the entries on the work fields screen if the work *Field Type* you select is *TEXT*.

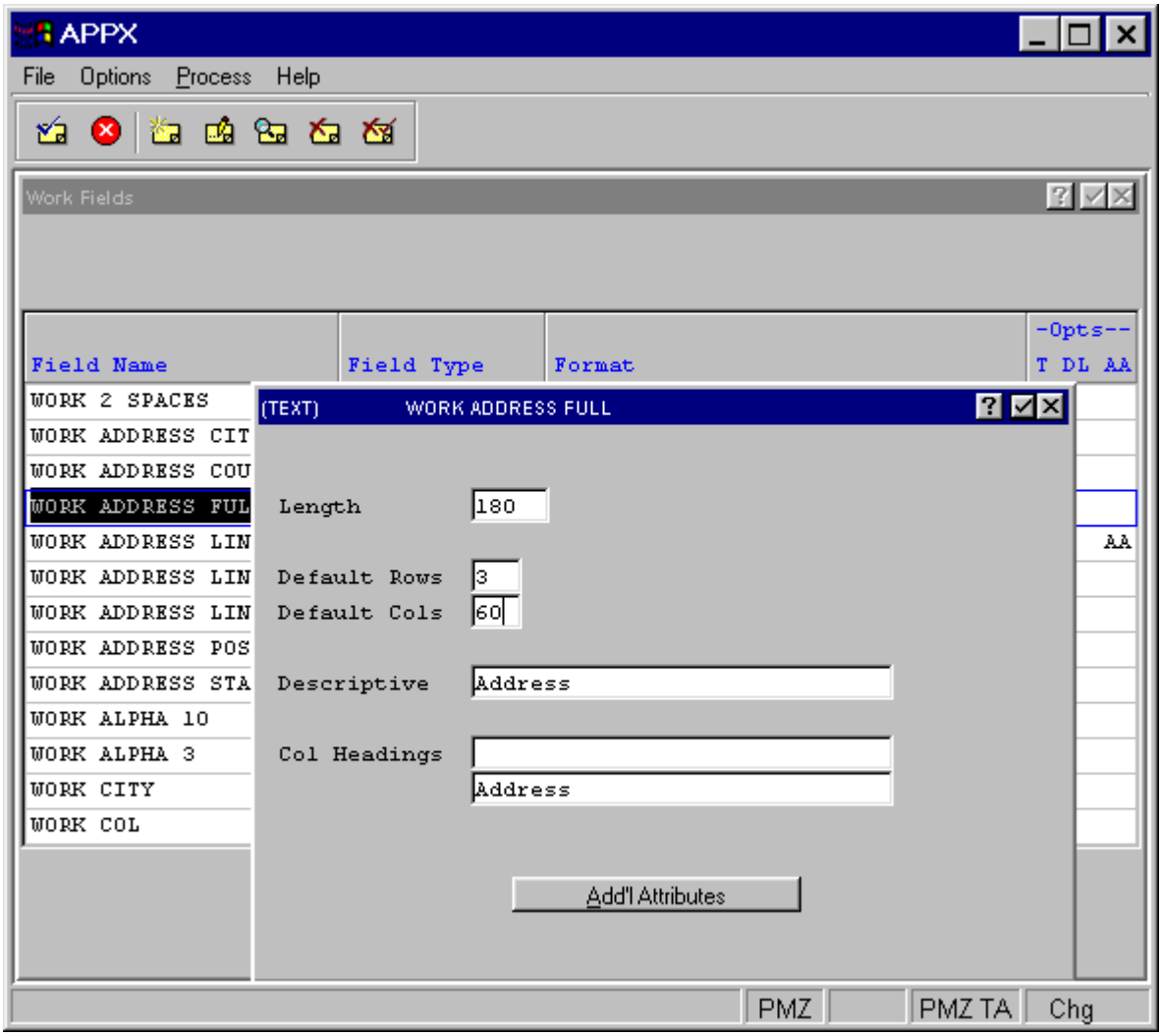


Figure 2-5-16. Text Work Field Overlay

The Text Work field Specifications overlay contains the following fields:

Length

Designates the number of character positions, or length, of this field.

Default Rows/Cols

Determines the default number of rows and columns to be used when specifying a field on an image.

Descriptive

Provides a longer, more easily understood alternative to *Domain Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Domain Name* is used.

Col Headings

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a domain itself. Refer to Standard Column Heading Image for additional information.

For Text work field type, the following option is available as described in the corresponding section:

- Additional Attributes

Text Work Fields Additional Attributes

From the Text work field overlay, you have an ADDITIONAL ATTRIBUTES option. The Text work field Additional Attributes overlay is shown in *Figure 2-5-17*.

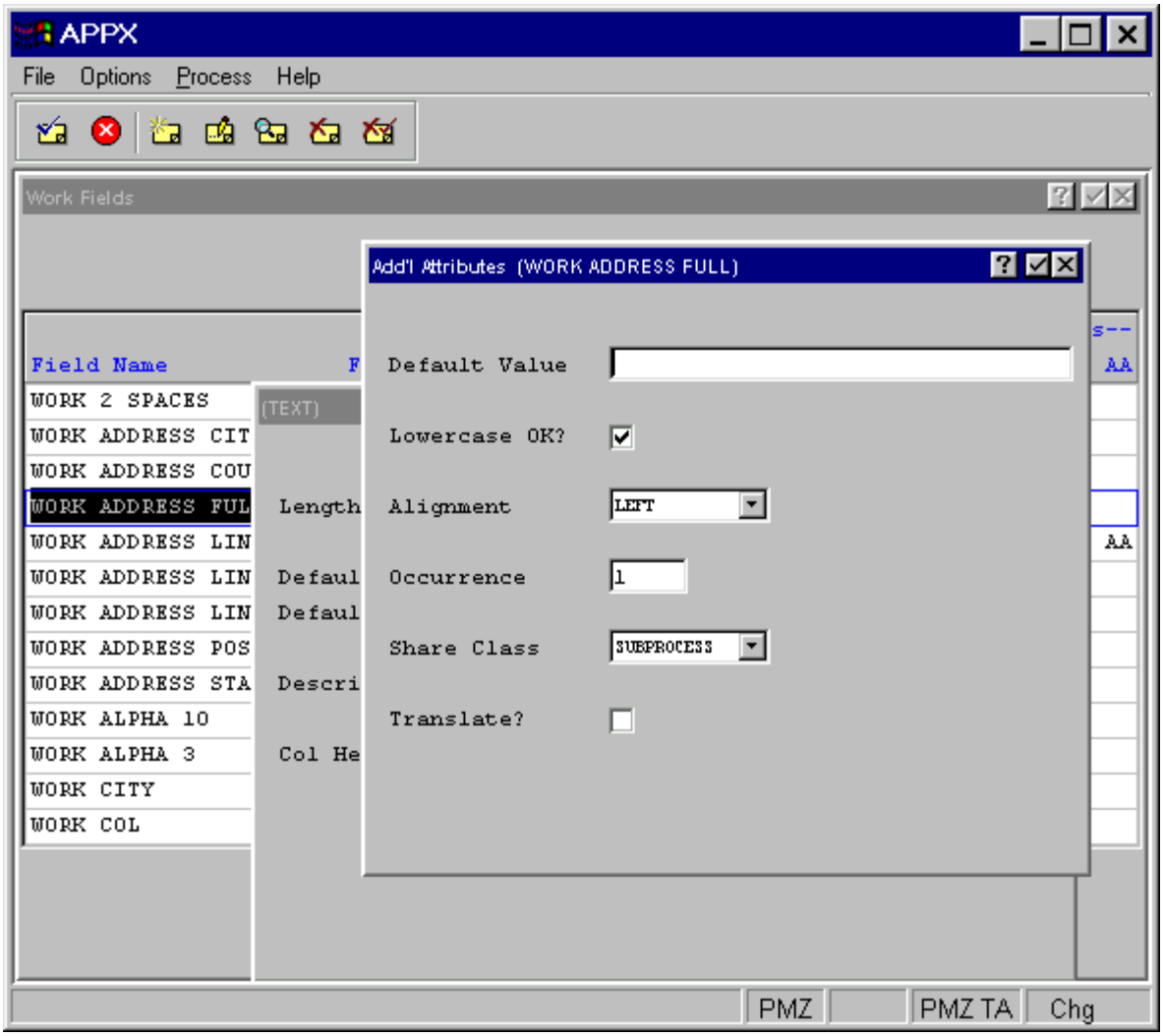


Figure 2-5-17. Text Work Field Additional Attributes Overlay

The Text Work field Additional Attributes Overlay contains the following fields:

Default Value

Establishes a modifiable starting value for all image items whenever they are first displayed in add mode in an input image, unless modified by a user. If blank, there is no starting value displayed on the image.

Lowercase OK?

Indicates whether or not a modifiable item on an interactive screen can be entered in upper- and lowercase letters. If “no” ☐, the entry is stored and displayed exclusively as uppercase, regardless of how a user enters the information. This entry does not affect statements. A lowercase value established with a statement displays as lowercase, even if this specification prohibits lowercase entries. The default is “yes” ☑.

Alignment

Determines whether or not data in a field is aligned for storage and/or display and, if so, the type of alignment. The *Alignment* options are ***NONE, LEFT, JUSTIFIED, RIGHT*** and ***CENTERED.*** The default is *NONE*.

Occurrence

Determines the number of times a field is present in a record. If blank, APPX assumes a value of 1. This field may contain up to five numeric characters and is restricted to a maximum of 32,767. See the Occurrence section for a more comprehensive discussion of this field.

Share Class

Defines the level values contained in this work field can be shared. Options are:

- ***SUBPROCESS*** — The values can be shared between a process and its subprocesses.
- ***RELATED*** — The values can be shared among related processes in a job.
- ***DETACHED*** — The values are available until the user logs off.
- ***NOT SHARED*** — The values are available only for the current process.
- The default is *SUBPROCESS*.

Translate?

Indicates if the value of the work field should be translated into a designated language if APPX’s Language Translation feature is being used. Since there is no validation table for a text field, it is the designer's responsibility to ensure that a translatable text work field is read only. Click the checkbox to toggle between “yes” ☑, “no” ☐, and unknown ☐ values. If the translation feature is not in use, retain the default value of ☐.

Token Work Fields

A token work field is like an alpha work field but it has specific values that are stored as corresponding numbers.

The token work field overlay, shown in *Figure 2-5-18*, allows you to add token specific attributes to those already defined for work fields. The token work fields specification overlay displays after you complete the entries on the work fields screen if the *Field Type* you select is *TOKEN*.

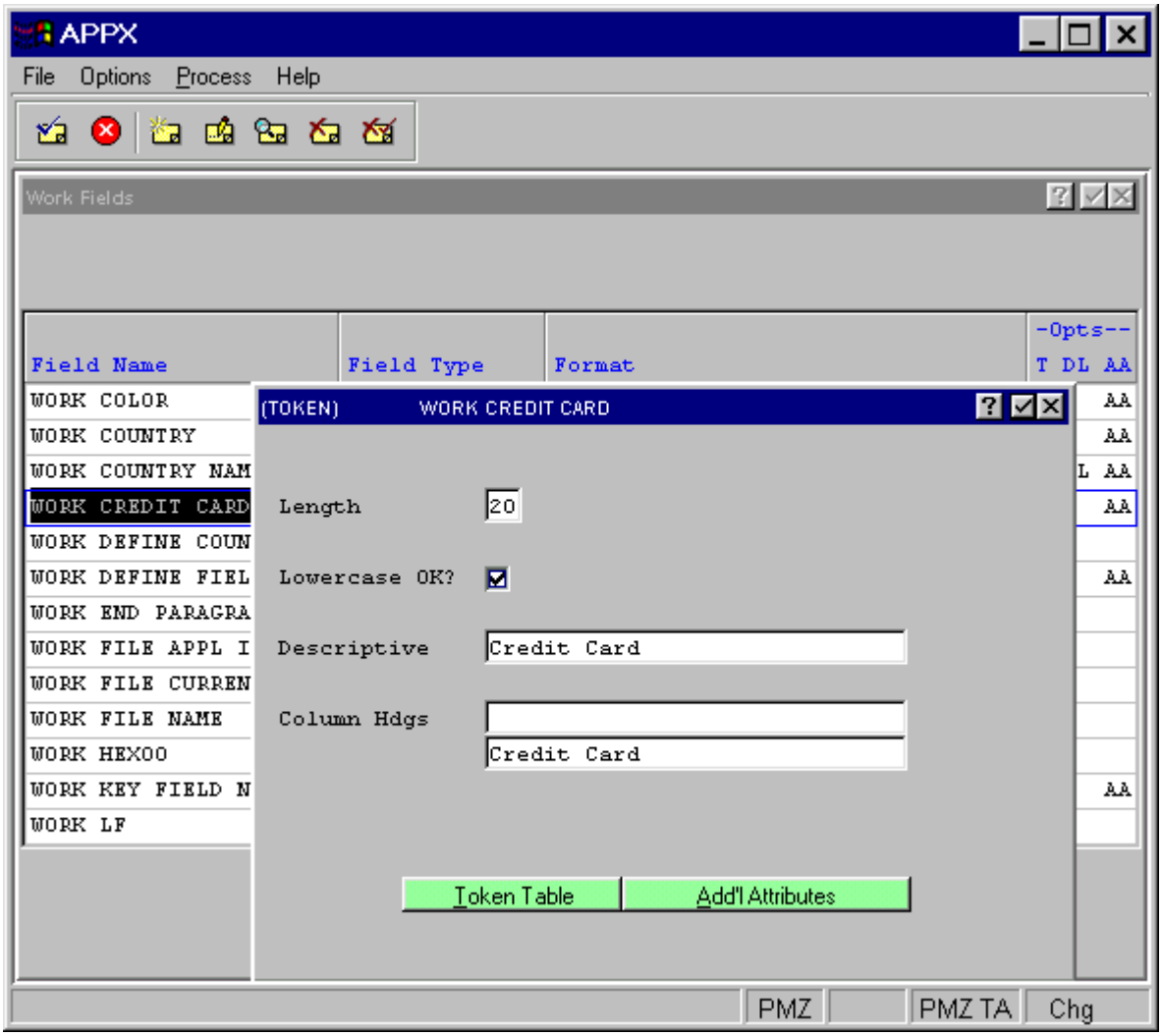


Figure 2-5-18. Token Work Field Overlay

The Token Work field Specifications overlay contains the following fields:

Length

Designates the number of character positions, or length, of this field.

Lowercase OK?

Indicates whether or not a modifiable item on an interactive screen can be entered in upper- and lowercase letters. If "no" ☐, the entry is stored and displayed exclusively as uppercase, regardless of how a user enters the information. This entry does not affect statements. A lowercase value established with a statement overrides an "no" in this field and displays as lowercase.

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

For Token work field type, the following options are available as described in the corresponding sections:

- Token Table
- Additional Attributes

Token Work Fields Token Table

TOKEN TABLE accesses the Token Table overlay, shown in *Figure 2-5-19*. A token table provides an alternative to the use of range checks for data token.

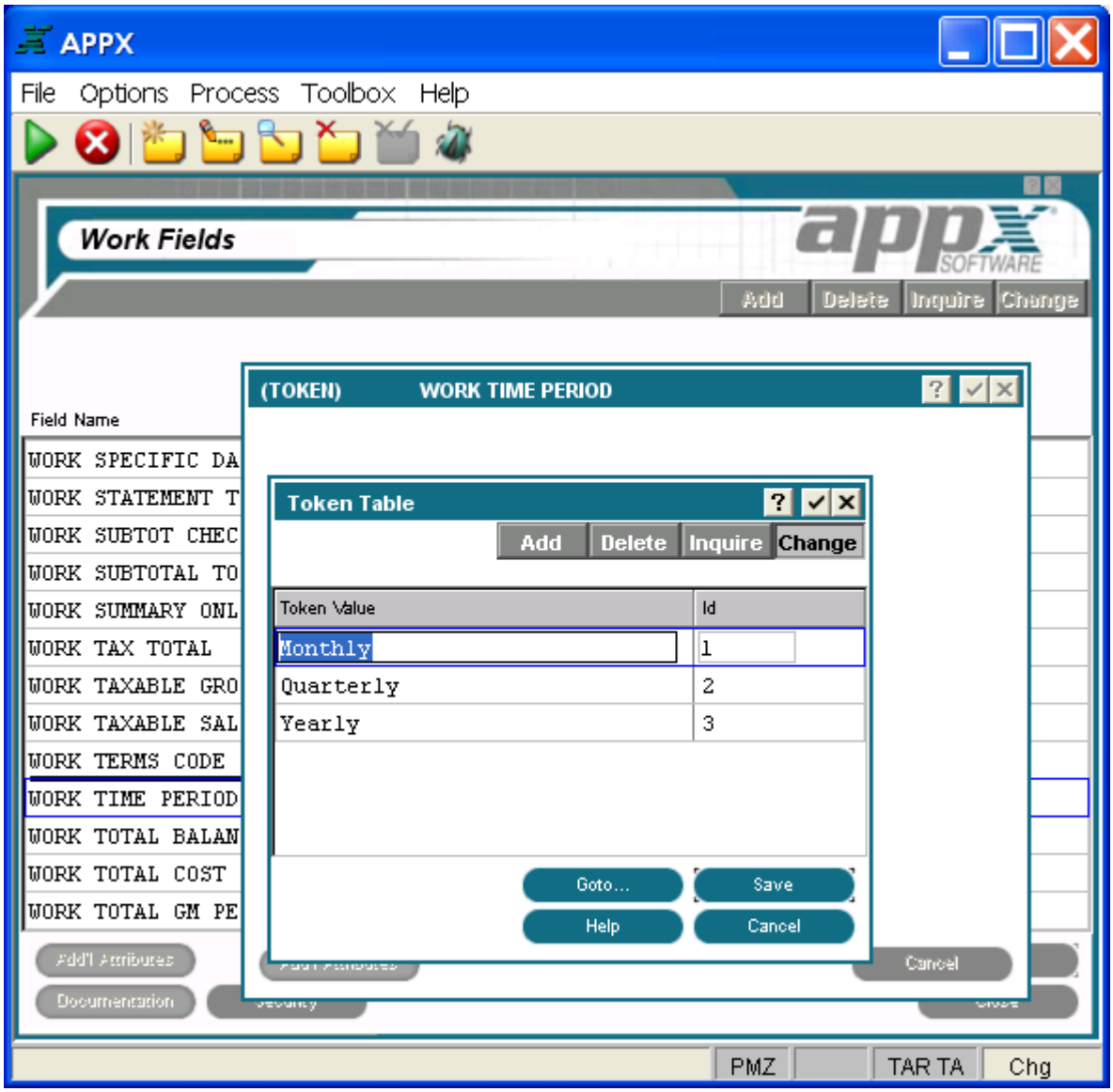


Figure 2-5-19. Token Work Fields Token Table Overlay

You can establish a token table for any Token work field in the data dictionary. The entries in the table are case-sensitive. If you enter a token value in all capital letters, an application user must enter the value in all capital letters.

Token table operations are database-independent. They access a designer-defined table in application design, not data contained in a file in a database, for the range of possible values. As a result, token does not vary from database to database. It is automatically available to, and exactly the same for, every database that is associated with an application.

The Token Table overlay contains the following fields:

Token Value

Is the alphanumeric value associated with the token ID. APPX automatically translates between token values and ID's whenever necessary. This entry is case-sensitive. If entered in capital letters, an application user must also enter the value in capital letters.

ID

Displays a unique number that is defaulted by APPX. This specification establishes the binary value that APPX stores in place of the associated token value. APPX assigns the first token value an ID of *1*, with additional values assigned an ID that is equal to the highest existing ID plus *1*. The default can be overridden, but must be unique within a table. APPX does not require that ID's be contiguous within a table.

Token Work Fields Additional Attributes

From the Token field overlay, you have an ADDITIONAL ATTRIBUTES option. The Token work field Additional Attributes overlay is shown in *Figure 2-5-20*.

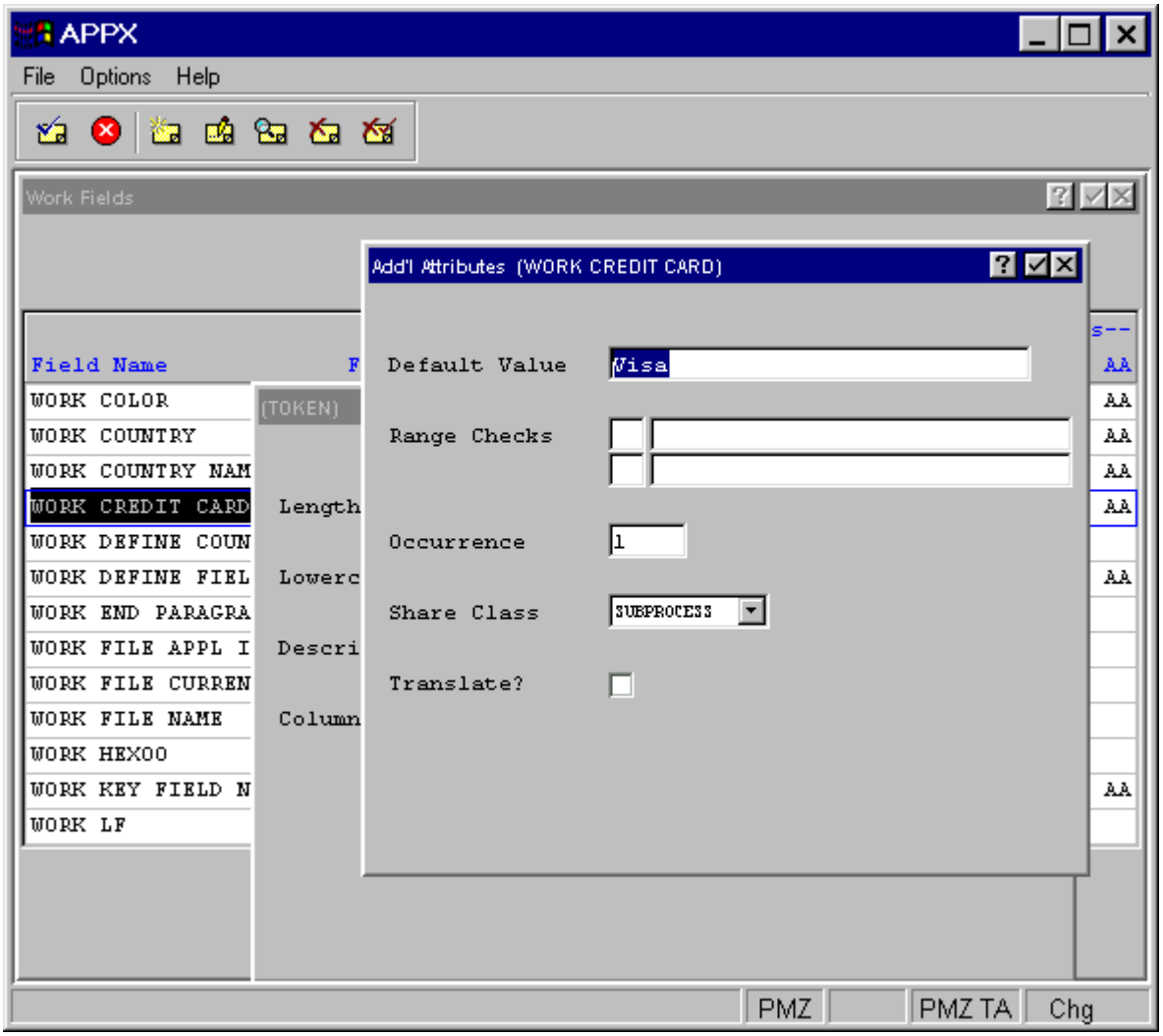


Figure 2-5-20. Token Work Field Additional Attributes Overlay

The Token Work field Additional Attributes Overlay contains the following fields:

Default Value

Establishes a modifiable starting value for all image items whenever they are first displayed in add mode in an input image, unless modified by a user. If blank, there is no starting value displayed on the image.

Range Checks

Defines a series of allowable values that APPX uses for online data validation. The first component represents the relation. The options are ***EQ*** (equal to), ***NE*** (not equal to), ***GT*** (greater than), ***LT*** (less than), ***GE*** (greater than or equal to), and ***LE*** (less than or equal to). The second component represents the value that is assigned to the relation. If you enter two relations, APPX infers an AND relationship.

Occurrence

Determines the number of times a field is present in a record. If blank, APPX assumes a value of 1. This field may contain up to five numeric characters and is restricted to a maximum of 32,767. See the Occurrence section for a more comprehensive discussion of this field.

Share Class

Defines at what level the values contained in this work field can be shared. The *Share Class* options are:

- ***SUBPROCESS*** — The values can be shared between a process and its subprocesses.
- ***RELATED*** — The values can be shared among related processes in a job.
- ***DETACHED*** — The values are available until the user logs off.
- ***NOT SHARED*** — The values are available only for the current process.
- The default is *SUBPROCESS*.

Translate?

Indicates if the value of the work field should be translated into a designated language if APPX's Language Translation feature is being used. Click the checkbox to toggle between "yes" ☑, "no" ☐, and unknown ? values. If the translation feature is not in use, retain the default value of ?.

Numeric Work Fields

A *numeric* work field is restricted to number, decimal, and sign (-) entries. Examples include amounts, prices, and quantities.

The Numeric work fields overlay, shown in *Figure 2-5-21*, permits you to add numeric-specific attributes to those already defined for the work field. It automatically displays after you complete the entries on the Work fields specifications screen if the *Field Type* you select is *NUMERIC*.

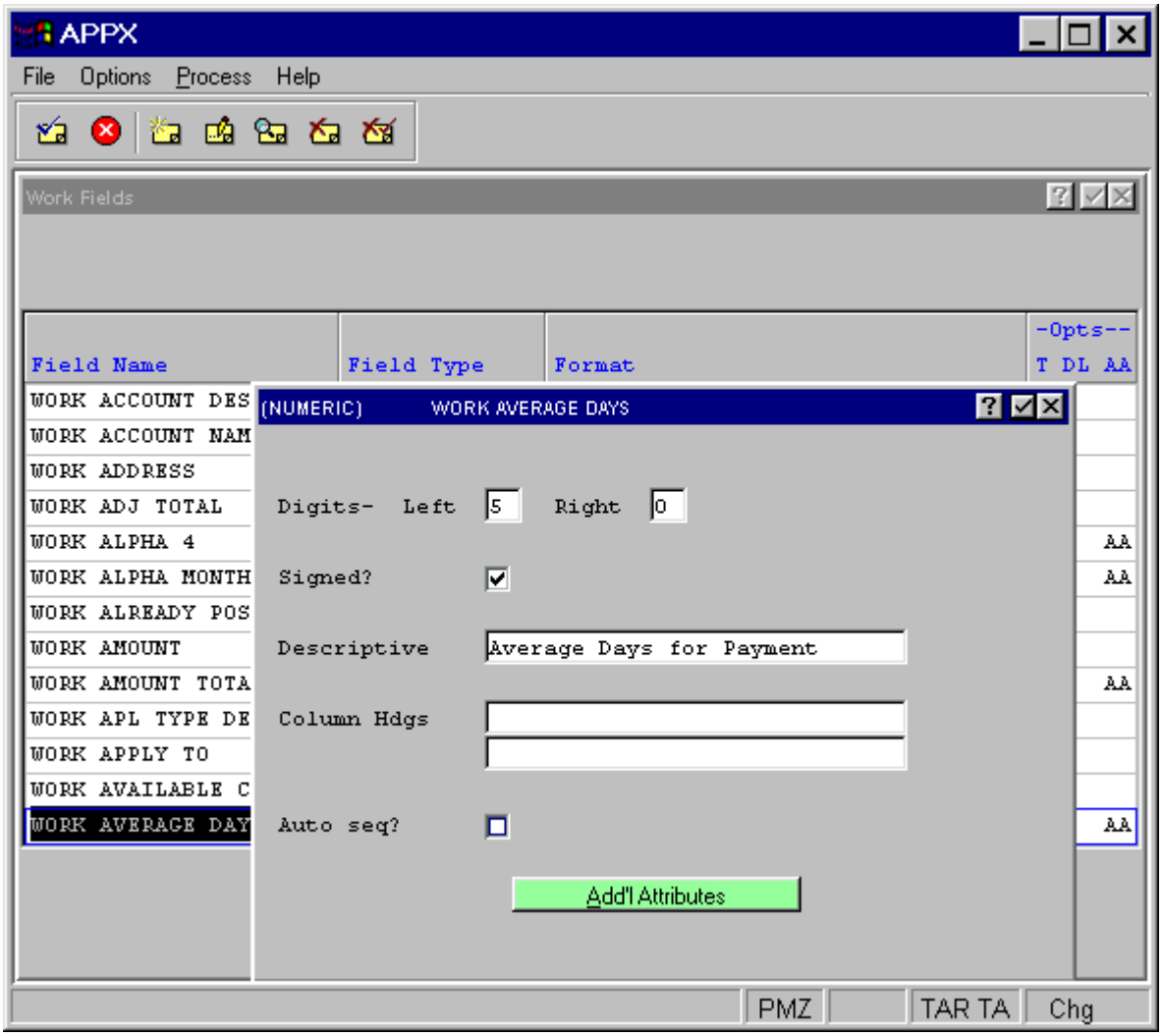


Figure 2-5-21. Numeric Work Fields Overlay

The Numeric work fields overlay contains the following fields:

- ***Digits — Left.*** Enter the number of digits to the left of the decimal.
- ***Digits — Right.*** Enter the number of digits to the right of the decimal.

Signed?

Specifies whether the numbers entered in this field are positive or negative. A "no" ☐ would ensure that only positive numbers are entered. The default is "no".

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

Auto Seq?

Select "yes" ☑ if this numeric field represents a sequence number with a value that should be maintained by APPX. Click the checkbox to toggle between "yes" ☑ and "no" ☐. Note that *Auto Seq?* only appears if the numeric field has no decimal places.

From the Numeric work fields overlay, you can select the ADDITIONAL ATTRIBUTES overlay, shown in *Figure 2-5-22.*

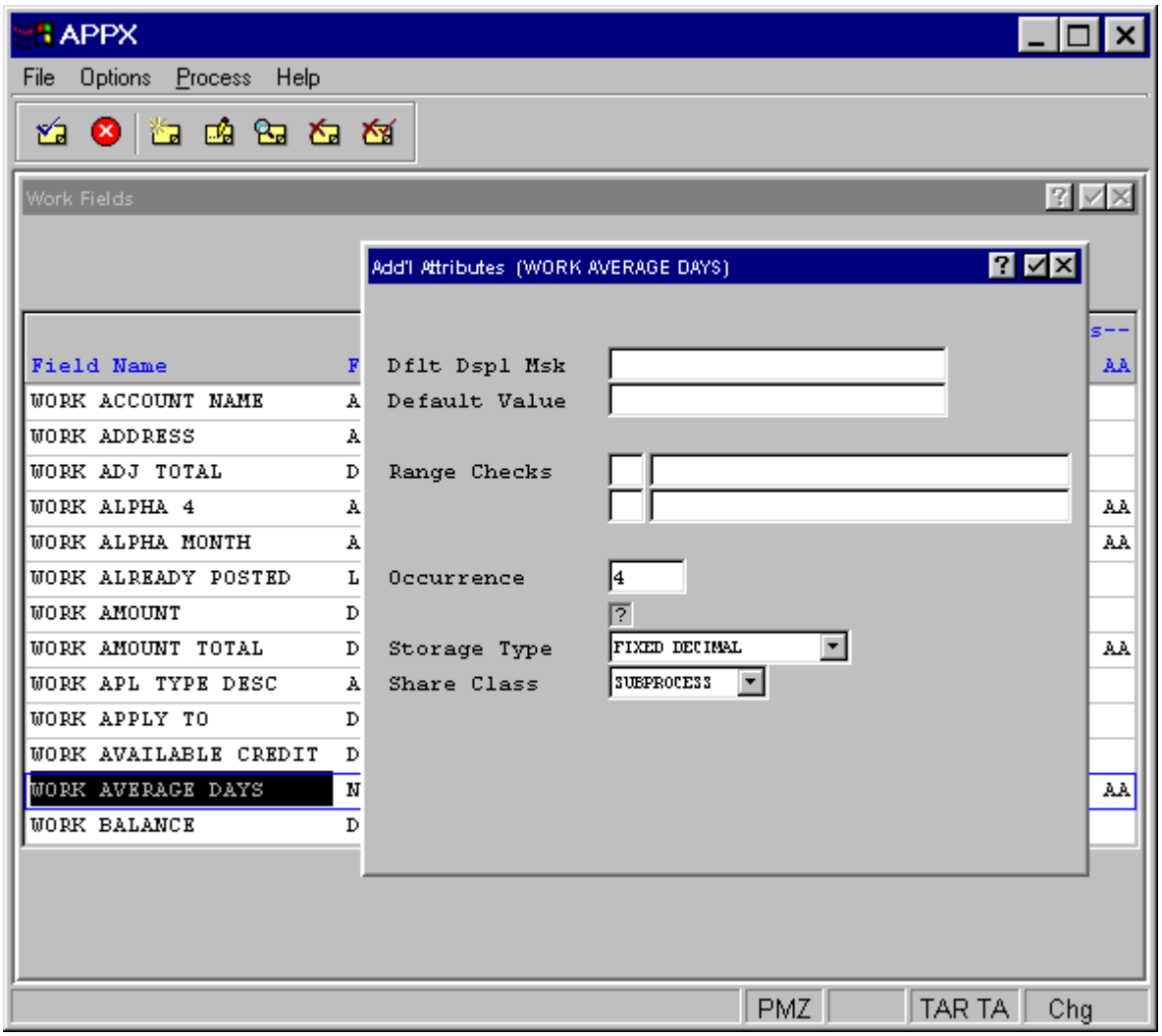


Figure 2-5-22. Numeric Work Fields Additional Attributes Overlay

The numeric work fields Additional Attributes overlay contains the following fields:

Dflt Display Msk

Designates the format that the data displays to a user in an APPX image. This entry establishes the *Display Mask* default for all image items that reference the numeric field. If blank, APPX uses a display of nonfixed leading spaces with one significant digit. Refer to the *Numeric Domains* section in *Chapter 2-3* for additional information about this entry.

Default Value

Establishes a starting value for all image items whenever they first display in add mode in an input image, unless modified by a user. If blank, the starting value is assumed to be the null value.

Range Checks

Defines a series of allowable values that APPX uses for online data validation. The first component represents the relation. The options are ***EQ*** (equal to), ***NE*** (not equal to), ***GT*** (greater than), ***LT*** (less than), ***GE*** (greater than or equal to), and ***LE*** (less than or equal to). The second component represents the value that is assigned to the relation. If you enter two relations, APPX infers an AND relationship.

Occurrence

Determines the number of times a field is present in a record. If blank, APPX assumes a value of 1. This field may contain up to five numeric characters and is restricted to a maximum of 32,767. See the Occurrence section for a more comprehensive discussion of this field.

Storage Type

Determines the internal storage format for data. The *Storage Type* options are:

- ***BINARY*** — The number is stored in binary form. The high order bit determines the sign: on or *1* for positive, and off or *0* for negative. The decimal position is established by the *Decimals — Left/Right* entries. The maximum field length is 10 digits for non-decimal numbers or 9 digits for decimal numbers.

FIXED DECIMAL

Employs a packed decimal format that stores two decimal digits for each byte with the exception of the last bit, which is the sign bit. The decimal position is established by the *Decimals — Left/Right* entries. The maximum field length is 31 digits.

FLOATING DECIMAL

Employs a combination of binary and packed decimal formats that always requires a total of nine bytes of storage. Although the field definition allows for 15 digits to the left of the decimal point plus 15 digits to the right of the decimal point, only the first 15 digits of the actual number are stored. These 15 digits are stored as packed decimal with two decimal digits for each byte (for a total storage requirement of eight bytes). The final byte (binary) determines the maximum number of digits that currently appears to the right of the decimal.

- The default is *FIXED DECIMAL*. For a comprehensive discussion of these options, see Numeric Storage Types in *Chapter 2-3*.

Share Class

Defines at what level the values contained in this work field can be shared. The *Share Class* options are:

- ***SUBPROCESS*** — The values can be shared between a process and its subprocesses.
- ***RELATED*** — The values can be shared among related processes in a job.
- ***DETACHED*** — The values are available until the user logs off.
- ***NOT SHARED*** — The values are available only for the current process.
- The default is *SUBPROCESS*.

Date Work Fields

A date work field contains date and/or time information. This information may, at your option, range from a century through a hundredth of a second. A work field definition may also include separator characters (such as */*, -, or *:*) and text, including the day of the week and the name of the month.

Examples of date/time work fields include invoice dates, order dates, elapsed time data, date/time stamps used to uniquely identify records, and audit trail work fields such as date-added and date-changed.

The Date work fields overlay, shown in *Figure 2-5-23*, permits you to add date/time-specific attributes to those already defined for the work field. It automatically displays after you complete the entries in the Work fields specifications screen if the *Field Type* you select is *DATE*.

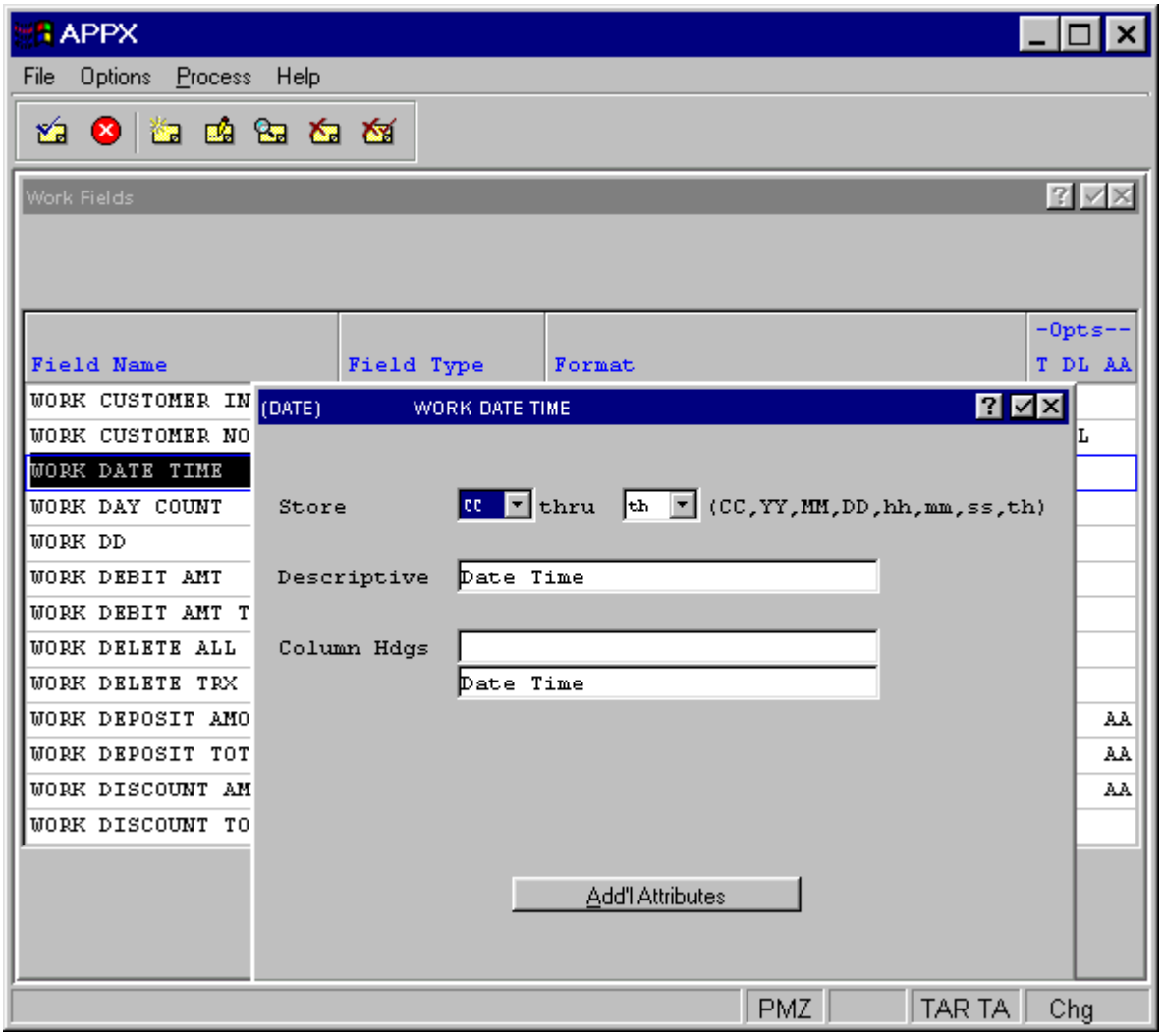


Figure 2-5-23. Date Work Fields Overlay

The Date work fields overlay contains the following fields:

Store thru

Designates the parts of a date that are physically stored. Valid entries are:

- CC (Century)
- YY (Year)
- MM (Month)
- DD (Day)
- hh (Hour)
- mm (Minute)
- ss (Second)
- th (Tick (hundredth of a second))

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

The ADDITIONAL ATTRIBUTES overlay, shown in *Figure 2-5-24*, is accessible from the Date work fields overlay.

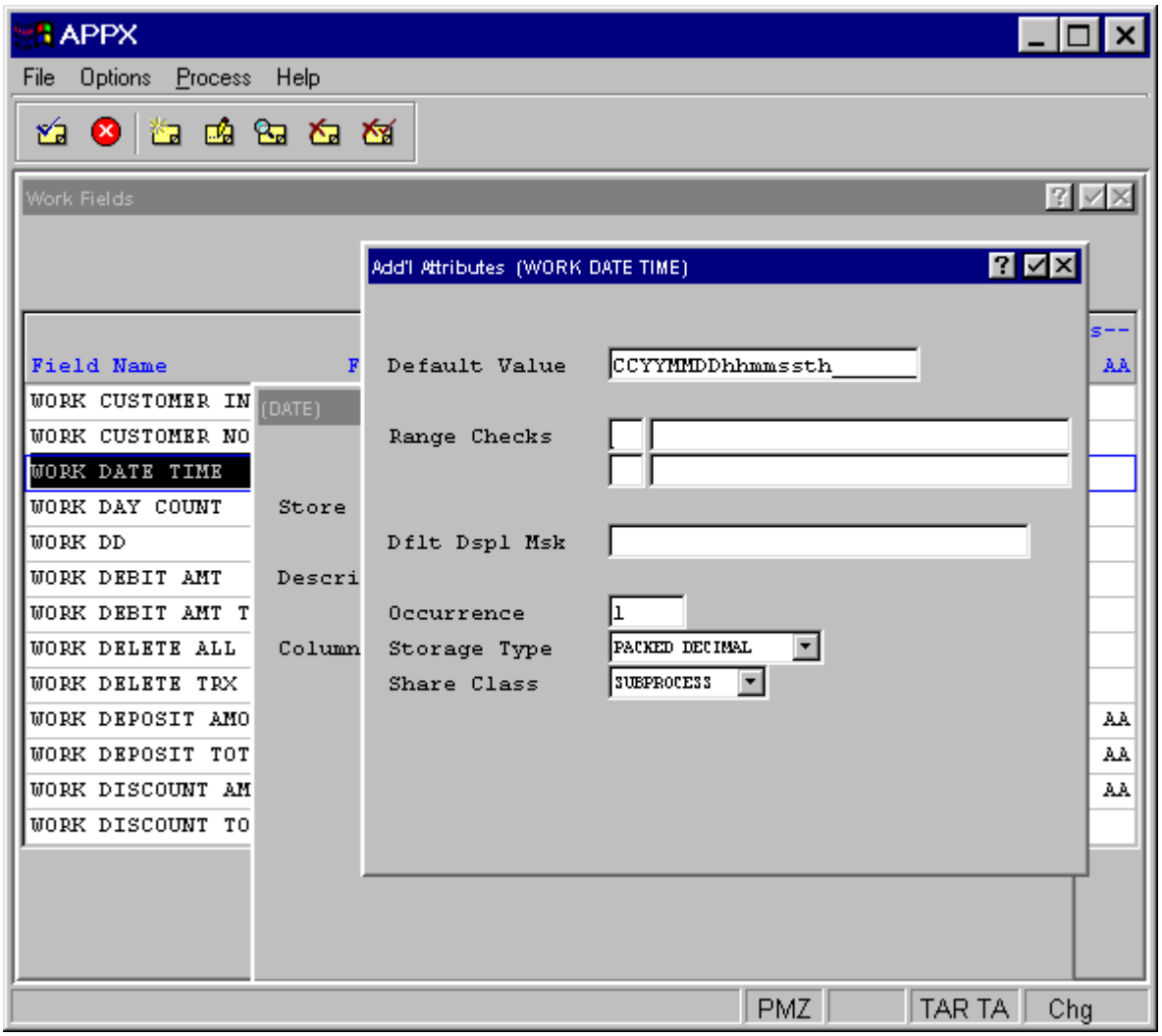


Figure 2-5-24. Date Work Fields Additional Attributes Overlay

The date work fields Additional Attributes overlay contains the following fields:

Default Value

Establishes a starting value for image items whenever they are first displayed in add mode on an input image, unless modified by a user. If blank, there is no starting value displayed.

Range Checks

Defines a series of allowable values that APPX uses for online data validation. The first component represents the relation. The options are ***EQ*** (equal to), ***NE*** (not equal to), ***GT*** (greater than), ***LT*** (less than), ***GE*** (greater than or equal to), and ***LE*** (less than or equal to). The second component represents the value that is assigned to the relation. If you enter two relations, APPX infers an AND relationship.

Dflt Dspl Msk

Controls how this field appears to a user on an APPX image, unless overridden at the image item level. If left blank, the date mask specified for the application (in Application File Maintenance in System Administration) is used or, if that is also blank, the date mask specified in System Parameters is used. Refer to the Display Mask (Date/Time) section in Chapter 3-14: Using the Character Based Image Editor for additional information about this entry.

Occurrence

Determines the number of times a field is present in a record. If blank, APPX assumes a value of 1. This field may contain up to five numeric characters and is restricted to a maximum of 32,767. See the Occurrence section for a more comprehensive discussion of this field.

Storage Type

Determines the internal storage format of the date field. The *Storage Type* options are:

- ***BINARY*** — The date is stored in binary format.

PACKED DECIMAL

Stores two decimal digits for each byte.

ALPHANUMERIC

Stores the external representation of data.

The default is *PACKED DECIMAL*.

Share Class

Defines at what level the values contained in this work field can be shared. The *Share Class* options are:

- ***SUBPROCESS*** — The values can be shared between a process and its subprocesses.
- ***RELATED*** — The values can be shared among related processes in a job.
- ***DETACHED*** — The values are available until the user logs off.
- ***NOT SHARED*** — The values are available only for the current process.
- The default is *SUBPROCESS*.

Logic Work Fields

A *logic* work field has possible values of "yes" ☑, "no" ☐, or "unknown" ?. Examples include various application parameters such as whether or not to post transactions on an accrual basis, and control work fields that govern the execution of subroutines based upon an *exists/does not exist* condition.

The Logic work fields overlay, shown in *Figure 2-5-25*, permits you to add logic-specific attributes to those already defined for the work field. It automatically displays after you complete the entries on the Work fields specifications screen if the *Field Type* you select is *LOGIC*.

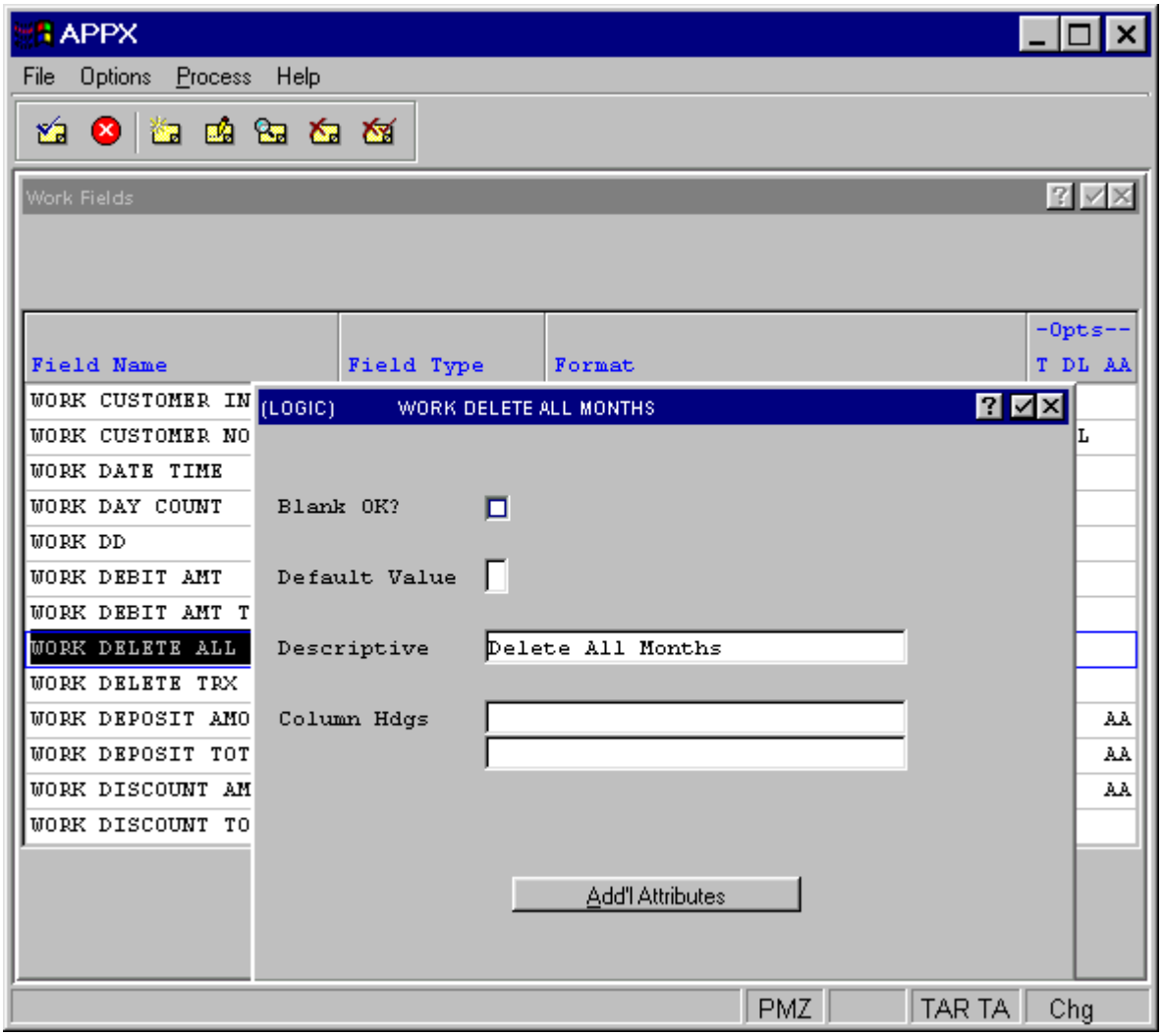


Figure 2-5-25. Logic Work Fields Overlay

The Logic work fields overlay contains the following attributes:

Blank OK?

Indicates, for information only, whether a user should be permitted to leave this field blank on an input image. (Whether or not the field can actually be left blank is controlled by the image item specification 'Required?'.) The default is “no” ☐. Click the checkbox to toggle between “yes” ☑ and “no” ☐.

Default Value

Establishes a starting value for image items whenever they first display in add mode on an input image, unless modified by a user. If blank, the starting value that displays on the image is unknown or ?. The *Default Value* options are ***blank*** (default value of *unknown* or ?), ***Y*** (default value of *yes* or ☑), and ***N*** (default value of *no* or ☐). The default is blank.

Descriptive

Provides a longer, more easily understood alternative to *Field Name,* one that is *not* required to be unique within an application. If blank at all default levels, *Field Name* is used.

Column Hdgs

Designates up to two lines of column heading text. The APPX output process facility uses this text to generate standard column heading images for *Start of Page* and *Start of Range* class frames. For cosmetic reasons, the length of headings should not exceed the length of a field itself. Refer to Standard Column Heading Image for additional information.

The ADDITIONAL ATTRIBUTES overlay, shown in *Figure 2-5-26*, is accessible from the Logic work fields overlay.

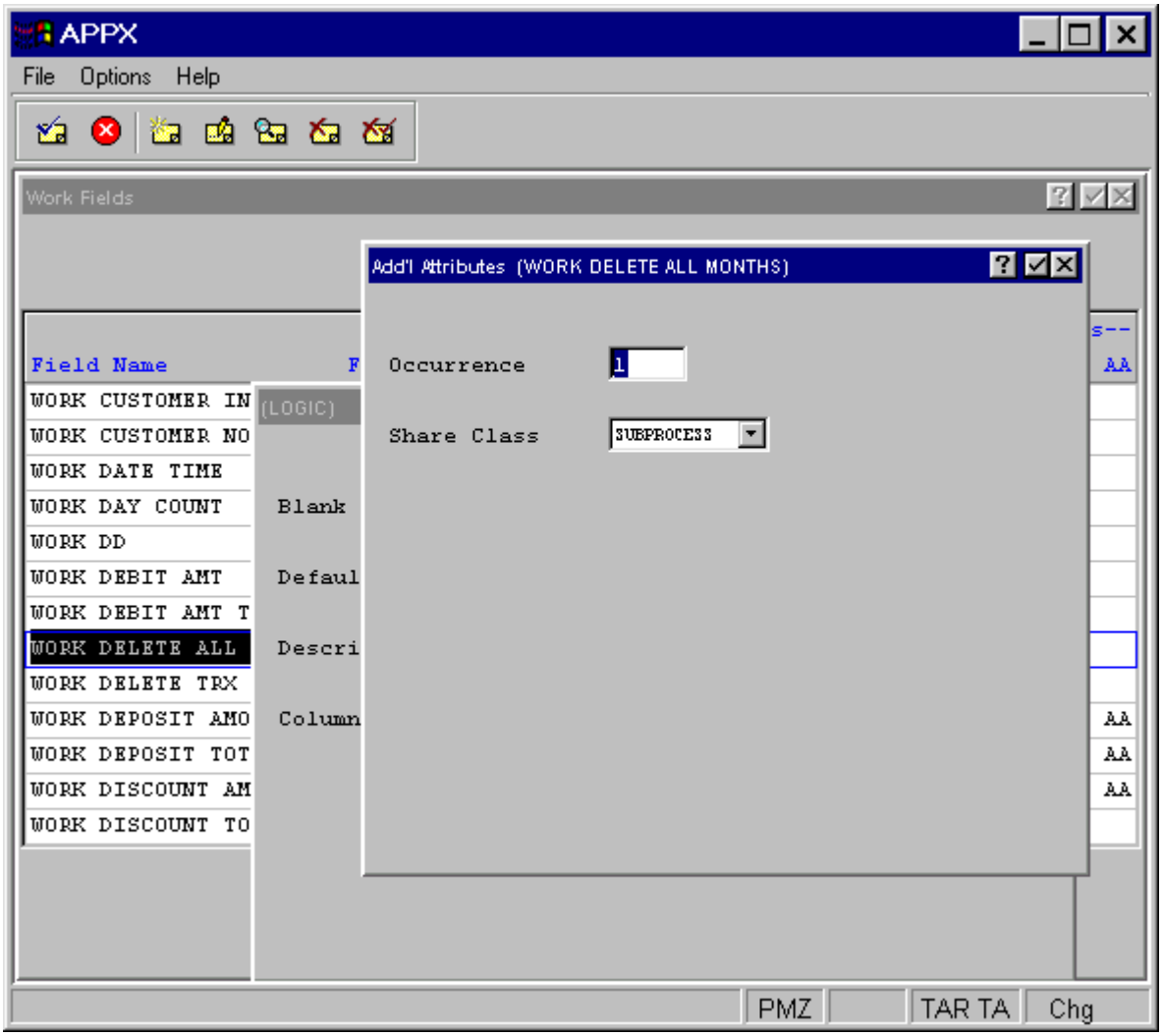


Figure 2-5-26. Logic Work Fields Additional Attributes Overlay

The logic work field Additional Attributes overlay contains the following fields:

Occurrence

Determines the number of times a field is present in a record. If blank, APPX assumes a value of 1. This field may contain up to five numeric characters and is restricted to a maximum of 32,767. See the Occurrence section for a more comprehensive discussion of this field.

Share Class

Defines at what level the values contained in this work field can be shared. The *Share Class* options are:

- ***SUBPROCESS*** — The values can be shared between a process and its subprocesses.
- ***RELATED*** — The values can be shared among related processes in a job.
- ***DETACHED*** — The values are available until the user logs off.
- ***NOT SHARED*** — The values are available only for the current process.
- The default is *SUBPROCESS*.

CHAPTER 3-1

Overview of APPX Processes

Overview of Chapters 3-1 thru 3-16

Processes are reusable software building blocks for constructing APPX applications. *Chapters 3-1* thru *3-16* of this manual provide a conceptual overview of APPX's process facility and detailed instructions for completing the specifications on process screens and overlays. With the exception of the conceptual material in *Chapters 3-1, 3-2,* and *3-16*, this part of the manual is organized according to the flow of screens and overlays in the facility.

Here is a summary of the content provided in *Chapters 3-1* thru *3-16*:

- *Chapters 3-1* and *3-2* provide a conceptual discussion of the process facility and its relationship to other facilities within APPX.
- *Chapters 3-3* through *3-12* each provide detailed information on creating processes in the process facility. These chapters follow the sequence of screens and overlays you navigate when defining a process. *Chapter 3-3* describes the specifications that are common to all types of processes in APPX. *Chapters 3-4* through *3-12* each describe the specifications that are unique to a specific type of process. Most sections are *procedural* in that they discuss the specifications you enter to define a process. Where appropriate, other sections are included to explain the importance of individual specifications in more detail, or the impact of one specification on another.
- *Chapters 3-13, 3-14* and *3-15* are also procedural. *Chapter 3-13* describes the specifications for invoking an automatic or optional child process. *Chapters 3-14* and *3-15* explain the use of the Image Editor, APPX's screen painting facility, which automatically invokes when you format images for APPX menu, input, output, and inquiry processes. *Chapter 3-14* covers the standard features of the Image Editor, applicable to both GUI and non-GUI environments. *Chapter 3-15* addresses the GUI features of the Image Editor, applicable only to GUI environments. GUI features *include* graphical attributes such as title bars, icons, pictures, and wallpapers as well as various user option presentations in the design of menu and input processes.
- *Chapter 3-16* discusses each APPX event point (the process component that permits you to insert procedural language into a process' execution sequence). The chapter includes the chronology according to which the event point executes and examples of its use.
- *Chapters 3-3* through *3-16* can be used as a reference guide by the experienced designer. Because each procedural section in the chapter corresponds to a screen or overlay, individual specifications are easy to locate. The specification definition provides complete entry information, including a specification's length and character set, its valid options, default value, and references for further information, if appropriate.

Before attempting to define a process, you should be thoroughly familiar with the information contained in both the *User Manual* and the introduction to the process facility in Chapter 1-4: Understanding Process Design of this manual.

Navigation Aids

To help you navigate the APPX process facility, refer to the schematic diagram of its logical structure in *Figure 3-1-1*. Each block represents a single screen or overlay, with stacked groups of blocks indicating mutually exclusive screen presentations. Each path indicates a menu option or an implied option, based upon the specifications you enter on a previous screen, the key you press, or the mode you select.

You will see relevant parts of this diagram repeated throughout the manual to aid you with process navigation.

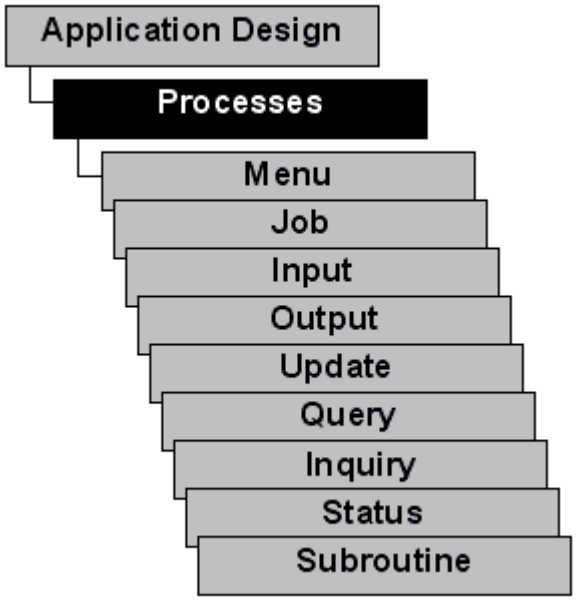


Figure 3-1-1. Process Definition Facility

The APPX Process Facility

Processes are the reusable software building blocks that, taken together and in conjunction with the data dictionary, comprise APPX applications. Each consists of the specifications, painted formats (images), and procedural routines (statements) that are required to perform a specific activity.

A process is similar to a program in traditional data processing systems in that it is:

- Executed to perform one or more related functions or activities.
- Comprised of a series of detailed instructions that are performed in a specified sequence.
- The smallest unit of an application that can execute separately.

Accessing the Process Facility

You access the APPX process facility from the Application Design menu, shown in *Figure 3-1-2*.

Application Design menu

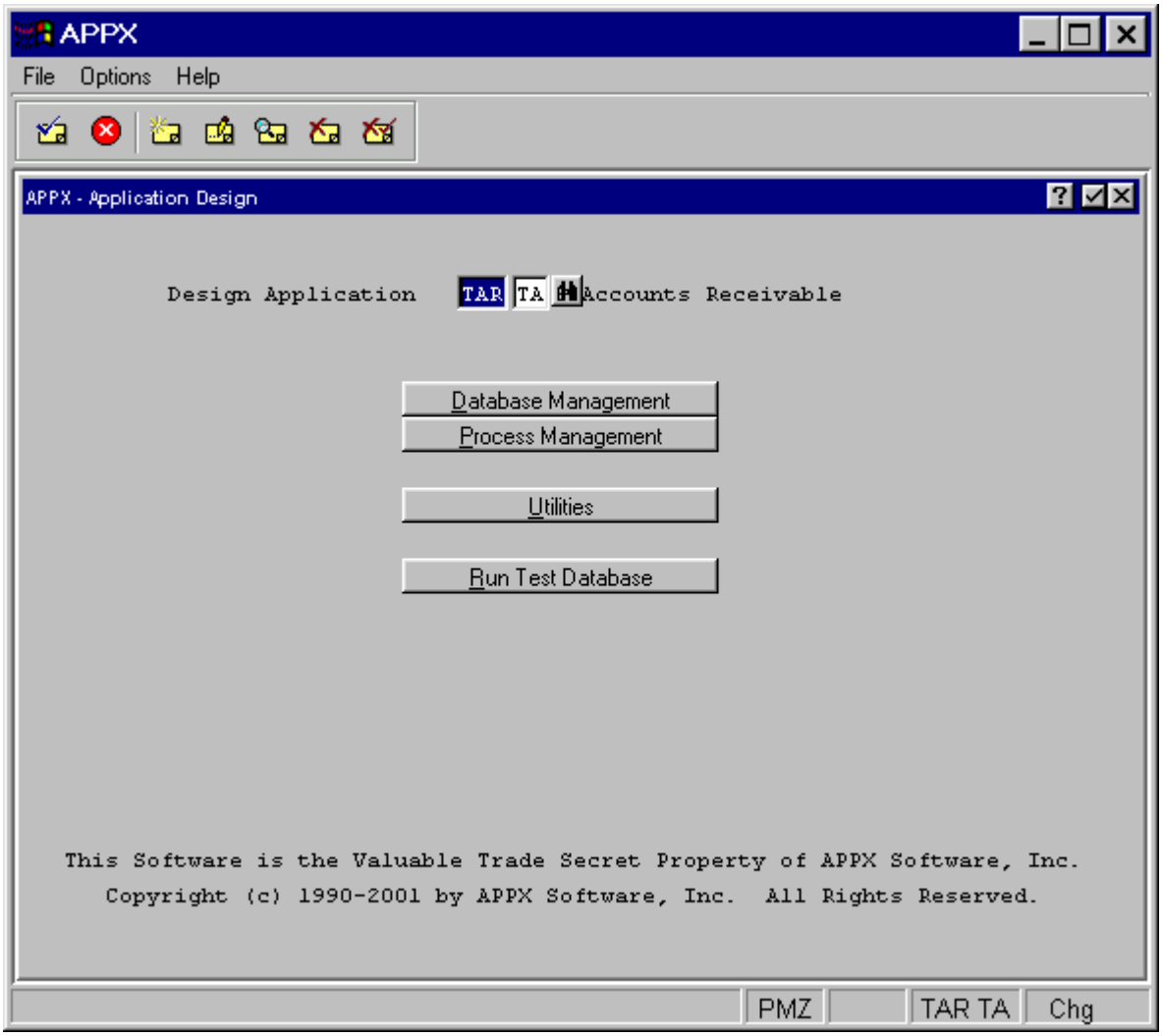


Figure 3-1-2

When you choose *Process Management* from the Application Design menu, APPX displays the menu for process definition.

Process Types

To streamline the definition of a process, APPX groups the various activities that you can reasonably expect an application to perform into nine general categories. These categories are:

- ***Menu processes,*** which provide an on-screen display from which to execute selected processes. For example, a screen that lists the reports that are available in an application and from which you select a specific report is a menu process. Refer to Chapter 3-4: Menu Processes for a discussion of menu specifications.
- ***Job processes***, which control the sequencing of other related processes. Refer to Chapter 3-5: Job Processes for a discussion of job specifications.
- ***Input processes***, which permit you to enter or maintain data in one or more files from designer-formatted screens and overlays. A sales order entry function and a customer file maintenance function are both input processes. Refer to Chapter 3-6: Input Processes for a discussion of input specifications.
- ***Output processes***, which convey information from a database, either on the screen or in hard copy. Trial balances, product purchase history reports, and customer lists are all examples of output processes. Refer to Chapter 3-7: Output Processes for a discussion of output specifications.
- ***Update processes***, which alter the database by adding records, deleting records, or changing information based on designer-established criteria. Refer to Chapter 3-8: Update Processes for a discussion of update specifications.
- ***Query processes***, which determine the records that are available to a subsequent input, update, or output process, and/or establish the data's sort hierarchy. Refer to Chapter 3-9: Query Processes for a discussion of query specifications.
- ***Inquiry processes***, which extract a record or a specific set of records from the database, and display the information on the screen. Refer to Chapter 3-10: Inquiry Processes for a discussion of inquiry specifications.
- ***Status processes***, which interrogate the database and warn the operator of possible error conditions. Refer to Chapter 3-11: Status Processes for a discussion of status specifications.
- ***Subroutine processes,*** which define reusable procedural programs that can be executed from other APPX processes. A program that converts one unit of measure to another, for example, can be written as a subroutine. Refer to Chapter 3-12: Subroutine Processes for a discussion of subroutine specifications.

Predefined Processes

In addition to creating your own processes, you can employ any of a number of *predefined processes*. A predefined process is an already-developed process, made available to you in the *runtime* application. Examples of predefined processes include APPX's standard DISPOSITION screen (shown in *Figure 3-1-3)* and SELECT DATABASE screen.

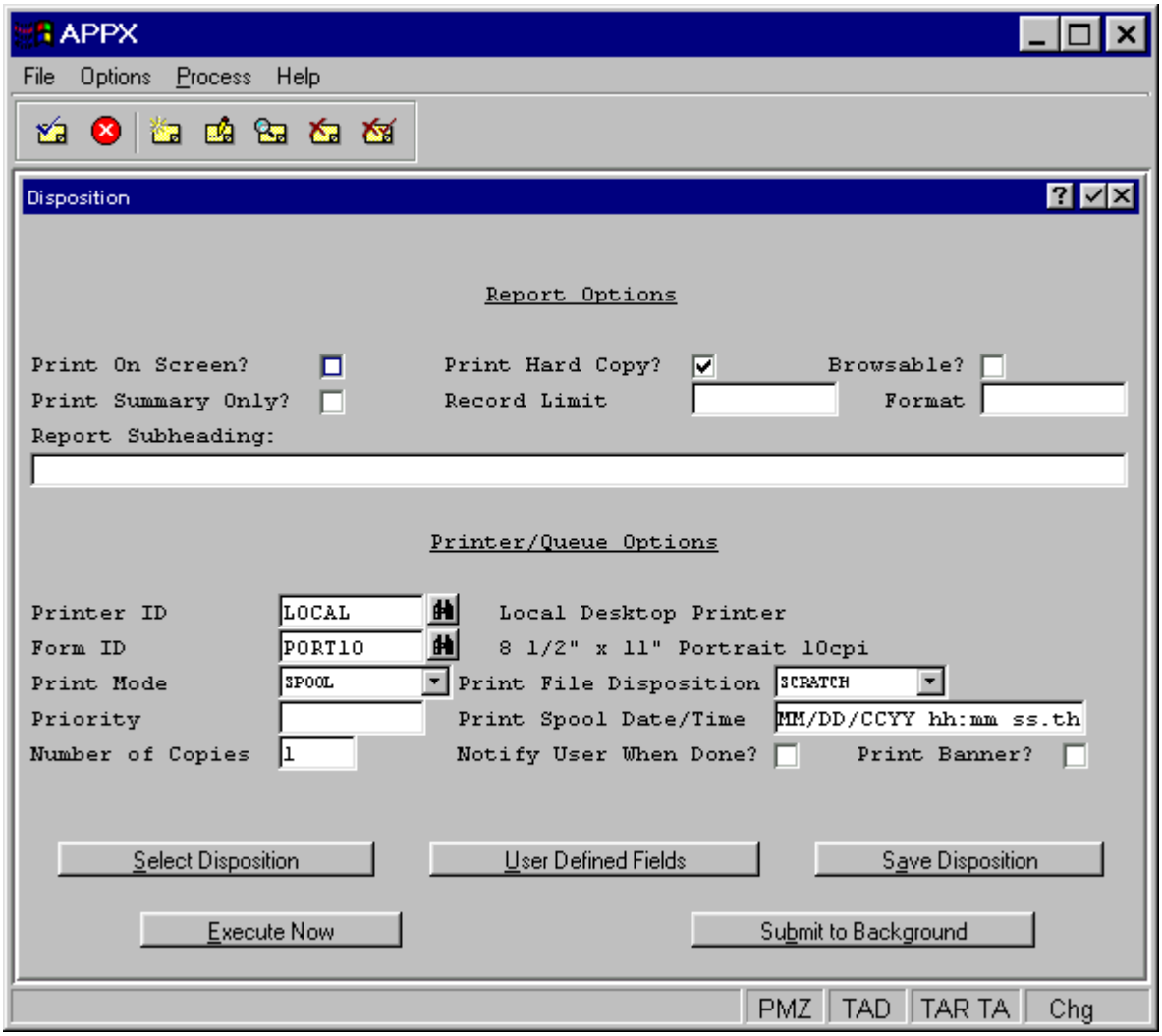


Figure 3-1-3. Disposition Predefined Process

You can invoke a predefined process at any time, and in much the same fashion as a designer-defined process. Predefined processes are located in the runtime application (application ID ---). Refer to Chapter 4-7 for a discussion of the runtime application and a complete listing of predefined processes in APPX.

Each process is comprised of one or more of the following components:

- A ***frame*** is a control structure for the sequencing of multiple events, which can occur within a single process.
- An ***image*** is a designer-painted format that displays to the user as a menu, input screen, or an output (presented either on the screen or as hard copy). This image may be *optional, repeating,* or *scrolling*.
- A ***process control file (PCF)*** is an APPX file that is automatically managed by a specific process. For example, the PCF for a process that permits you to input sales orders is the orders file. The PCF is the only file that APPX automatically reads, writes, rewrites, and deletes records for during process execution.

- A ***child process invocation*** is a set of specifications that link execution of another APPX process to a process. If the invocation of the child process is discretionary, based upon selection of a screen option, the process is an *optional child*. If the invocation is not discretionary, the process is an *automatic child*. Refer to Chapter 3-13: Automatic and Optional Children for more information.
- An ***event point*** provides an opportunity to insert designer-defined procedural code (statements) into the execution of a process. There are a variety of APPX event points, each providing access to a process at a specific point in its execution sequence.

Not all process types, however, include all of these components. Valid component and process-type combinations are outlined below in *Table 3-1-1*.

Table 3-1-1. Process Components

Component	Menu	Job	Input	Output	Update	Query	Inquiry	Status	Subr
Process Control File			Yes	Yes	Yes	Yes	Yes		
Frame		Yes	Yes	Yes	Yes	Yes	Yes		
Image			Yes	Yes			Yes		
Automatic Child Invoc		Yes	Yes	Yes	Yes		Yes		
Optional Child Invoc	Yes		Yes						
Event Points	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Frames

A *frame* is a control structure for the sequencing of events within a process. Each process consists of one or more frames, executed according to their relative sequence number.

Each frame defines a specific activity, or set of actions, that are performed as part of a process. These actions include such things as the display of information to a user, retrieval of information from a user, execution of procedural code, and the invocation of other processes. For example, a single input process may require several screens to enter all of the necessary data, each of which represents a separate frame in the process.

If a process has a related process control file (PCF), the sequence of frames is executed once for each PCF record that is processed. For example, a transaction report program, with a transaction file PCF, executes its frame sequence once for every record in the transaction file. If the process does not have a PCF, the sequences of frames executes once, and the process ends.

The frame chronology is modified slightly for the following types of frames:

- An ***optional frame,*** containing an image that conditionally displays based upon the value in the predefined field ALTERNATE IMAGE NUMBER. APPX does not display the image for an optional frame if the value is *0*.

- A ***repeating frame,*** containing an image that continues to display repeatedly until the value in the ALTERNATE IMAGE NUMBER predefined field is *0*.
- A ***scrolling frame,*** which is a type of input or output frame that displays more than a single image (one for each record, up to a maximum number of images) and formats those images into rows and columns. Refer to The Effect of Scrolling Frames on Screen and Page Positioning section of *Chapter 3-2* for a detailed discussion of scrolling frames.

The functions of a frame vary slightly from one process type to another. In general, a frame:

- Sequences multiple activities (including the invocation of automatic child processes), which occur within the confines of a single process.
- Defines a region of the screen or printed output that images can display on.
- Groups alternate images.

The frame itself incorporates both the image component of a process and a subset of the process event points. In addition, each frame may invoke one or more automatic child processes.

Images

An *image* is a designer-painted format that presents information and options to a user, or retrieves information and option requests from a user. Menu, input, inquiry, and output processes all utilize images. An example of an input image is shown in *Figure 3-1-4*.

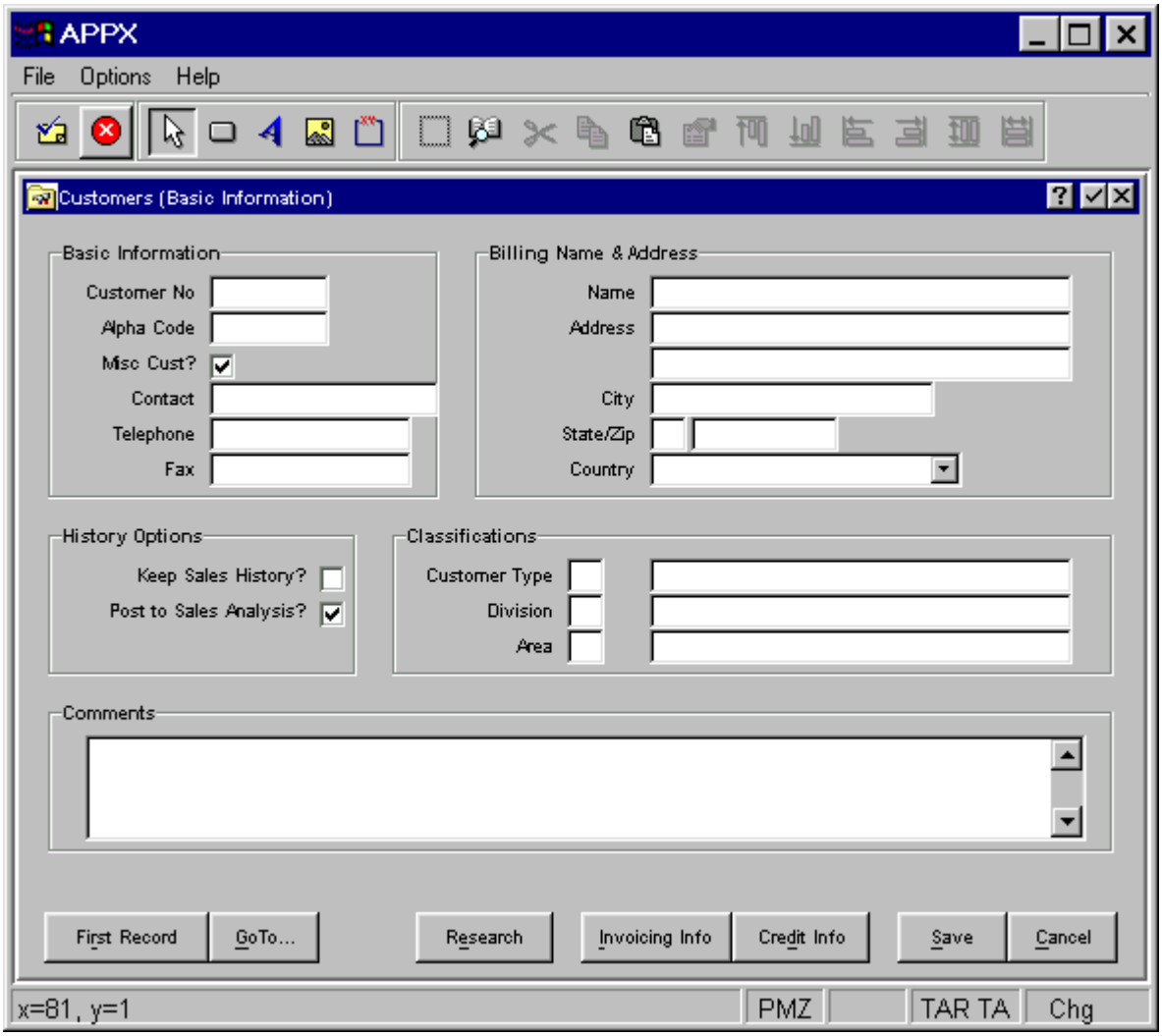


Figure 3-1-4. An APPX Input Process Image

Images consist of four primary components:

Items,

Which are individual appearances of a field from the data dictionary, as painted on an image. An item's appearance is affected by the specifications you enter in the dictionary, the display characteristics you define when the item is painted, and any graphical attributes assigned to the item.

Text

Consists of descriptions and identifying labels that are added to an image. Option names on menus, and column headings and field labels on input screens, are examples of background text.

C

Haracteristics **are** specifications that correspond to display attributes for items and text. Characteristics such as underlining apply to both screen displays and hard copy output. Other characteristics such as inverse/normal presentation, blinking, and bright/dim controls apply only to a screen image.

Graphical attributes

Are components such as wallpapers, titlebars, menu bar items, toolbar items, icons, and other features that are not part of the actual "image" you paint. These are applicable only if you are using the APPX Client for a GUI-enabled environment.

Alternate Images

An image may be the only image for a frame, or one of a number of *alternate images*. An alternate image is one of a group of mutually exclusive images, the display of which is dependent upon the number that is assigned to the image and the value in the predefined field ALTERNATE IMAGE NUMBER.

For example, you might create a customer input process that requires a user to complete specifications on two screens. The first screen contains basic customer information, including a customer type field that indicates whether a customer is domestic or international. Since different information is needed for the two types of customers, the image presented on the second screen depends upon the customer type entry. In other words, the frame definition for the first screen has only one image whereas the frame definition for the second screen has two alternate images (one domestic and one international), with image selection based upon the value in the customer type field.

Process Control Files

A process can operate on the data that is contained in any number of files through use of ILF routines, but it can *automatically manage* only one file. This automatically managed file is referred to as the *process control file (PCF) and it applies to input, output, update, query, and inquiry processes*. For example, the PCF for a customer maintenance input process is the customer file. APPX automatically reads the PCF while executing the process. In addition, for input processes only, APPX automatically writes, rewrites, and deletes records in the PCF as a function of process execution.

If you want to automatically maintain more than one file, such as entering sales order header information and line item information together, create a separate process for each file and link them together. For example, invoke the line item process as an *automatic child* from the sales order header process. APPX also provides for updating the child PCF for automatic children of input processes to ensure relational integrity. See the following sections, Automatic and Optional Child Invocation and Process Lineage, for additional information on parent and child processes.

A PCF is *optional* for input and output processes. An input process that does not include a PCF will execute once, and then automatically end (as opposed to executing once for every record). Output processes that do not specify a PCF can be used, for example, to produce standardized page headings and page footers.

Update, query, and inquiry processes must have a process control file. Menu, job, status, and subroutine processes *cannot* have a PCF.

Automatic and Optional Child Invocations

A *child* process is the process that is invoked from another process (the parent). The invocation of a child process from the parent process can be *automatic* or *optional*.

If invocation of a child is not discretionary on the part of the application user, the child process is an automatic child. Automatic children are related to a frame, and are always invoked by APPX after processing the images for that frame. Input, output, update, job, and inquiry processes are the only process types that support automatic children. Menu, query, status, and subroutine processes do not.

If invocation of a child is discretionary on the part of the application user, the child is an optional child. Optional children can only be invoked from interactive images in menu and input processes. They are related to an image, and only invoke if a user selects the options that identify them. A traditional menu image, for example, contains one option for each process you can select from the menu, as shown in *Figure 3-1-5*.

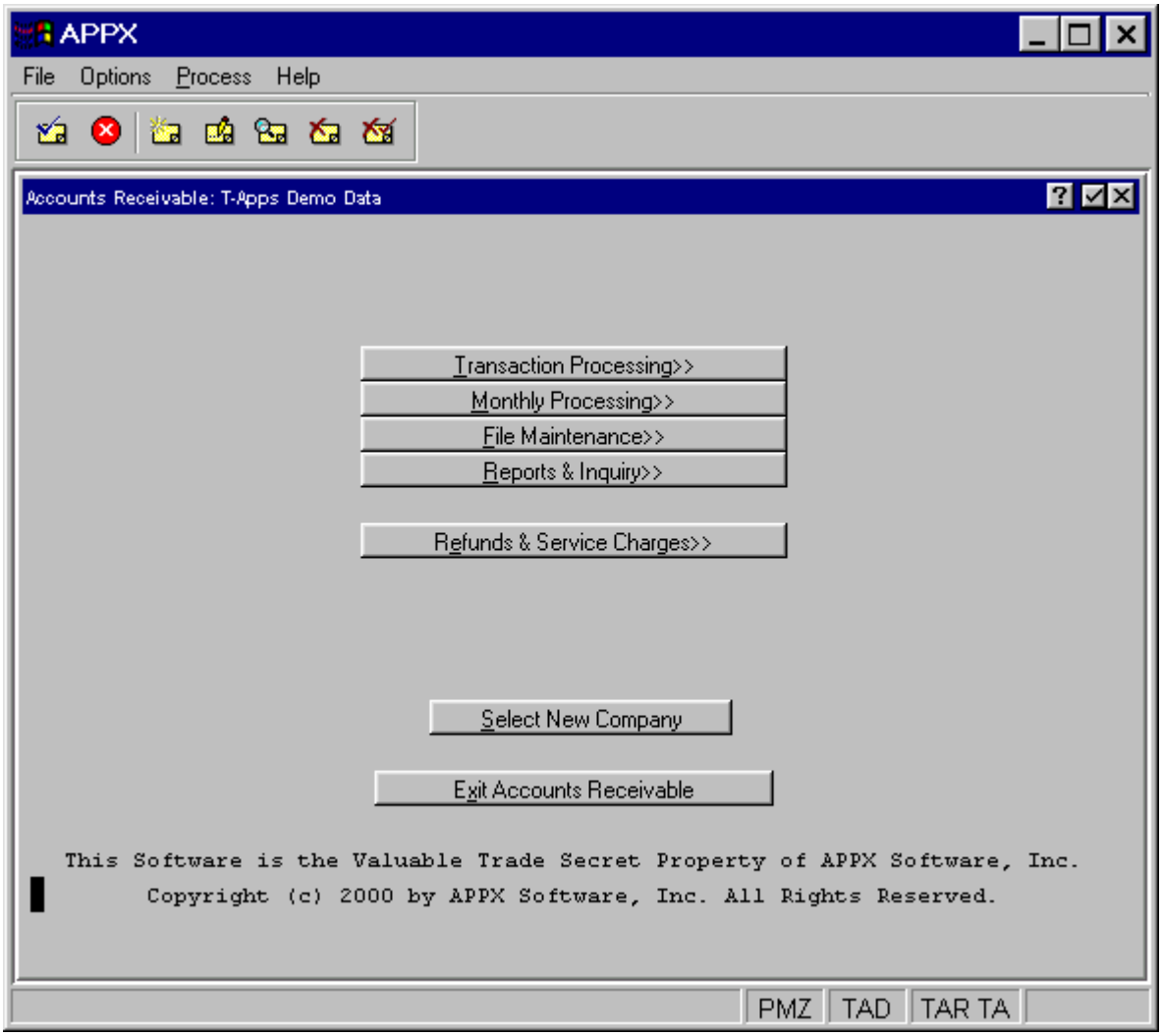


Figure 3-1-5. A Menu's Optional Children

Table 3-1-2 illustrates the types of processes that can be used to invoke children. As shown, optional children can be invoked from menu and input processes while automatic children can be invoked from job, input, output, update, and inquiry processes. Query, status, and subroutine processes cannot be used to invoke children of any kind.

Table 3-1-2. Combinations of Process Type/Child

Process Type	Optional Children?	Automatic Children?
Menu	Yes	
Job		Yes
Input	Yes	Yes
Output		Yes
Update		Yes
Query		
Inquiry		Yes
Status		
Subroutine		

You can establish selection criteria to restrict the PCF records that APPX accesses for an automatic or optional child. For example, you can use selection criteria to constrain a line item process, so only those line items that belong to the current order are accessed. In this case, the selection criteria require that the line item order number in the child process is equal to the order number in the parent process. See Chapter 3-13: Automatic and Optional Children for a detailed discussion of various options.

Invocation Types

The way a child process is invoked from its parent establishes the boundaries of shared information between the two processes. In other words, the *parent process* uses the value in an *Invocation Type* specification to define how tightly coupled the child is to the parent. It governs the information that can be passed and shared between processes. Whether or not a specific item of information *is* passed or shared depends upon pass and share specifications entered in the data dictionary, or defined explicitly with statements.

There are three distinct invocation types: *subprocess, related,* and *detached* (in order from most to least tightly coupled). If you use the data dictionary default values and do not explicitly pass information with statements, the various invocation types function as follows:

- A child invoked as a *subprocess* shares record areas, file position information (for READNEXT operations, among others), and the contents of both permanent and temporary files with the process it was invoked from.

- A child invoked as a *related* process shares the contents of permanent and temporary files with the process it was invoked from. It does not share file position information, and it maintains its own record area in memory.
- A child invoked as a *detached* process shares only the contents of any permanent files.

Refer to Chapter 1-5: Interprocess Communication for a complete discussion of detached, related, and subprocess files and processes.

Event Points

Defining an APPX application is essentially non-procedural; you complete specifications to determine *what* is to be accomplished, not *how* it is accomplished. Non-procedural specifications can be said to satisfy the data processing requirements that are imposed by routine business operations.

To respond effectively to nonstandard requirements, APPX integrates high-level procedural code, or *statements,* located in the *integrated language facility (ILF)*. APPX's statements are a set of keywords entered through a specialized editor to enforce syntax. Refer to *Chapters 4-1* thru *4-7* of this manual for a complete discussion of APPX's integrated procedural language.

You can insert this procedural code into the execution sequence of a process at various times through *event points.* The event point, an example of which is shown in *Figure 3-1-6*, provides an opportunity to insert procedural code into the execution of the process. Each event point can contain a sequence of statements that are executed at a specific, predetermined point within each APPX process to address the required specialized task. Commonly used groups of statements can also be defined once in a subroutine process, and then accessed by an event point or executed as an optional or automatic child.

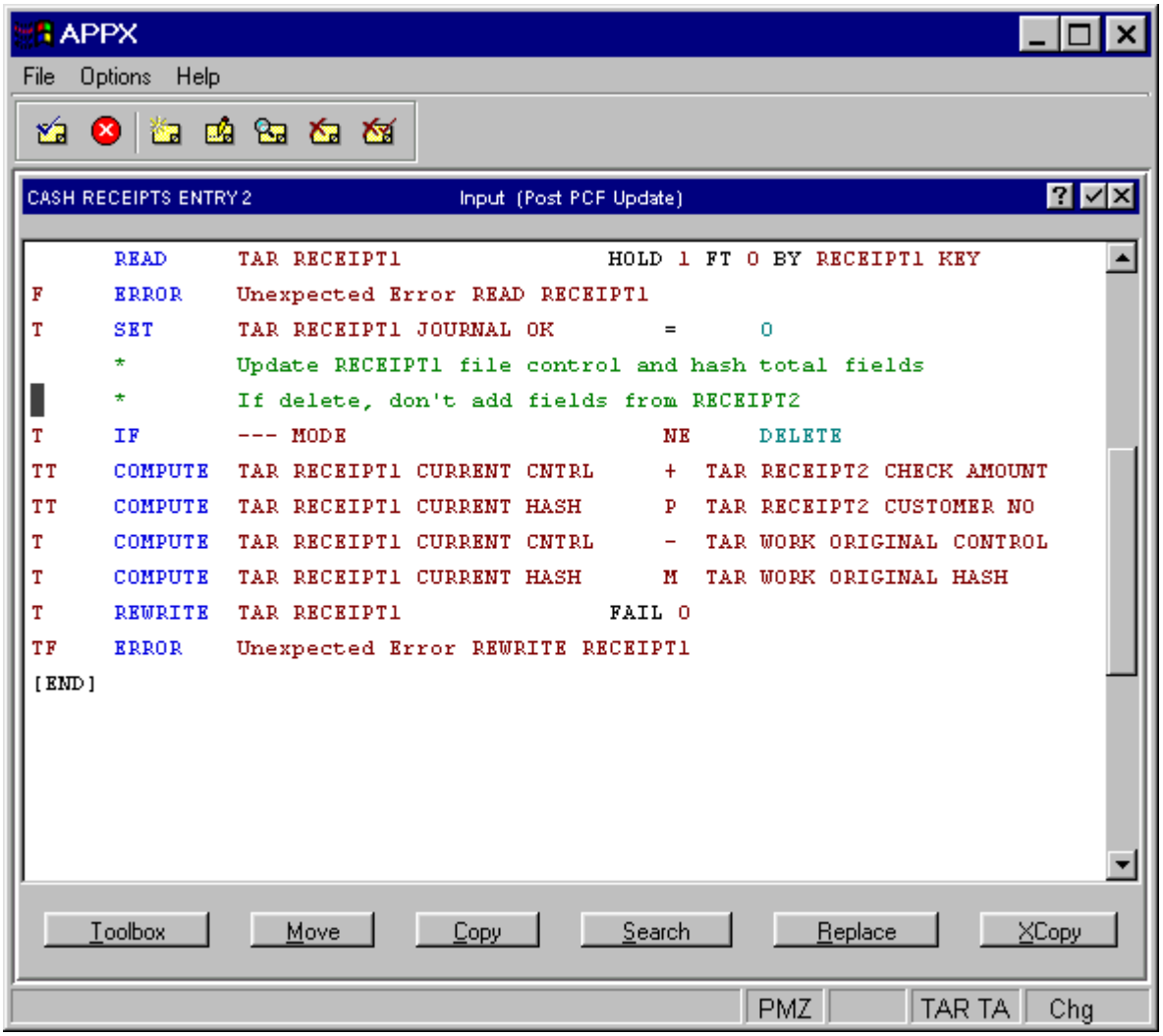


Figure 3-1-6. An Event Point

There are a number of APPX event points, each of which provides access to a process at a specific point in its execution sequence. They exist at four levels: process, frame, image, and child invocation. In addition, each type of process has a slight variation in the specific event points it includes at each level. *Table 3-1-3* summarizes the event points that execute for each process type.

Refer to Chapter 3-16: Using Event Points for general information about each event point, including a definition and the chronology according to which it executes.

Table 3-1-3. Process Event Points for Rules

Event Point	Menu	Job	Input	Output	Update	Query	Inquiry	Status	Subr
Process-Level									
Start of Process	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Set Run-Time Defaults			Yes						
Post PCF Read			Yes	Yes			Yes		
Pre-PCF Update			Yes						

Event Point	Menu	Job	Input	Output	Update	Query	Inquiry	Status	Subr
Post PCF Update			Yes						
Pre-User Selection						Yes			
Post User Selection						Yes			
Start of Query Execution						Yes			
End of Query Execution						Yes			
End of Process	Yes	Yes	Yes	Yes	Yes	Yes	Yes		
Frame-Level									
Start of Frame			Yes	Yes	Yes		Yes		
Select Image			Yes	Yes			Yes		
Global Pre-Display			Yes	Yes			Yes		
Global Option Intercept			Yes				Yes		
End of Frame			Yes	Yes			Yes		
Image-Level									
Pre-Display	Yes		Yes	Yes			Yes		
Pre-Display Verify			Yes				Yes		
Option Intercept	Yes		Yes				Yes		
Verify			Yes				Yes		
Child-Level									
Pre-Invocation	Yes	Yes	Yes	Yes	Yes		Yes		
Post Invocation	Yes	Yes	Yes	Yes	Yes		Yes		

Process Lineage

The term *process lineage* refers to the specific relationship between two or more processes in an APPX application. You can describe relationships between processes in much the same fashion as relationships between individuals in an extended family. As with family lineage, where relationship terms (such as parent or grandfather) are relative to a particular individual, process terms are relative to a specific process. The lineage concepts in this section are relevant to all APPX processes.

The Process Tree

A *process tree* is a schematic of the interconnections between processes that are in any fashion related to one another. The process facility diagram (*Figure 3-1-1*) at the beginning of this chapter is an example of a process tree.

The Parent

A *parent* process is the process another process (the child) invokes from. A parent process is commonly used in situations where you require APPX to automatically manage more than one file (PCF). The first file is the PCF for a parent process, and the other files are PCFs for child processes. For example, an invoice entry process is actually operating on the data that is maintained in two files: an invoice header file and a line items file. To automatically manage both files, APPX requires two processes. An invoice header input process is a parent to an invoice line item input process.

Children

A *child* process is the process that invokes from another process (the parent). In the previous example, the line item input process is the child of the invoice header parent.

As previously indicated, the invocation of a child process from the parent process can be automatic or optional. If invocation of the child is not discretionary on the part of the user, the child process is an automatic child. If invocation of the child is discretionary on the part of the application user, the child is an optional child.

Siblings

Two processes with the same parent are *sibling* processes. For example, an employee file maintenance process might be established as the parent for two child processes: an earnings process and a deductions process for each employee. The earnings and deductions processes are siblings because they both have the same parent process: employee file maintenance.

Ancestors and Descendents

Ancestor processes include parent processes, *grandparent* processes (the parent's parent), if any, etc.

Descendent processes include child processes, the child's children (if any), and so on.

Families

A *family* is a group of processes that include a *patriarch* process and a set of descendents linked by a common attribute (a specific invocation type, for example). The term patriarch refers to the first process that exhibits the identified attribute; it begins the process family. By definition, the patriarch will not share the attribute with its parent process.

The use of the term *family* is generally further qualified by including the name of the specification that is shared (or the specification value). These terms include:

- ***Subprocess family***, which is a group of processes that are invoked with an *Invocation Type* (on the Child specification screen) of *SUBPROCESS*. The first process in a process tree that is invoked as a subprocess begins this family, and the first process that is invoked as a related or a detached process marks the end of this family.

- ***Related process family***, which is a group of processes that are invoked with an *Invocation Type* (on the Child specification screen) of *SUBPROCESS* or *RELATED*. The first process in a process tree that is invoked as a subprocess or a related process begins this family, and the first process that is invoked as a detached process marks the end of this family.
- ***Report family***, which is a group of output processes in a subprocess family that share a common print file (the *Report File Name* on the Output Processes specification screen).
- ***Log family***, which is a group of update processes in a subprocess family that share a common log file (the *Log File Name* on the Update Processes' Additional Attributes specification screen).

The Process Cycle

A single execution of an entire sequence of frames, images, event points, and children for one record in the process control file is referred to as a *process cycle*. For example, assume you created a process with a single frame, a single image, and one automatic child. In addition, you defined statements in *Start of Frame* and *End of Frame*, *Pre-Display* and *Option Intercept*, and *Pre-Invocation* and *Post Invocation* event points. For each record, the process cycle consists of the following APPX actions:

- Read a record in the process control file.
- Invoke the *Start of Frame* event point.
- Perform image processing.
- Invoke the *Pre-Display* event point.
- Display the image to the screen.
- (User completes the entry.)
- Invoke the *Option Intercept* event point.
- Invoke the *End of Frame* event point.
- Perform automatic child processing.
- Invoke the *Pre-Invocation* event point.
- Perform processing as specified for the child process.
- Invoke the *Post Invocation* event point.

CHAPTER 3-2

Getting Started

Introduction to Process Concepts

The APPX process facility recognizes nine process types: *menu, input, output, query, subroutine, update, job, status* and *inquiry*. Each represents a specific kind of activity, and each is comprised of a subset of the structural components (PCFs, frames, and event points, for example) that were described in the previous chapter. However, while the structural components are essentially the same from type to type, the way these components are used to perform a required activity varies. For example, menu and input processes use images to format the screens that display to a user. Output processes, on the other hand, generally use images to format hard copy, such as a list or report. And subroutine processes do not display information to a user, so no images are required.

Because of such differences, there are a number of concepts that apply to one or more, but not all, process types in APPX. To continue the previous example, the way you position images on a frame has obvious significance to a menu, input, or output process, but no relevance at all to a subroutine. To carry the example even further, there are subtle differences between image positioning for input and output images, because the former display on the screen, while the latter generally print on hard copy. The material that follows discusses a number of these process-specific concepts in detail.

The Image Editor

Image editing (screen painting) refers to the APPX facility that allows you to format images for menus, input screens, and output. With image editor options, you design the visual presentation of each image and define specific operational parameters. As you design the image, you are seeing it exactly as it will appear to your users.

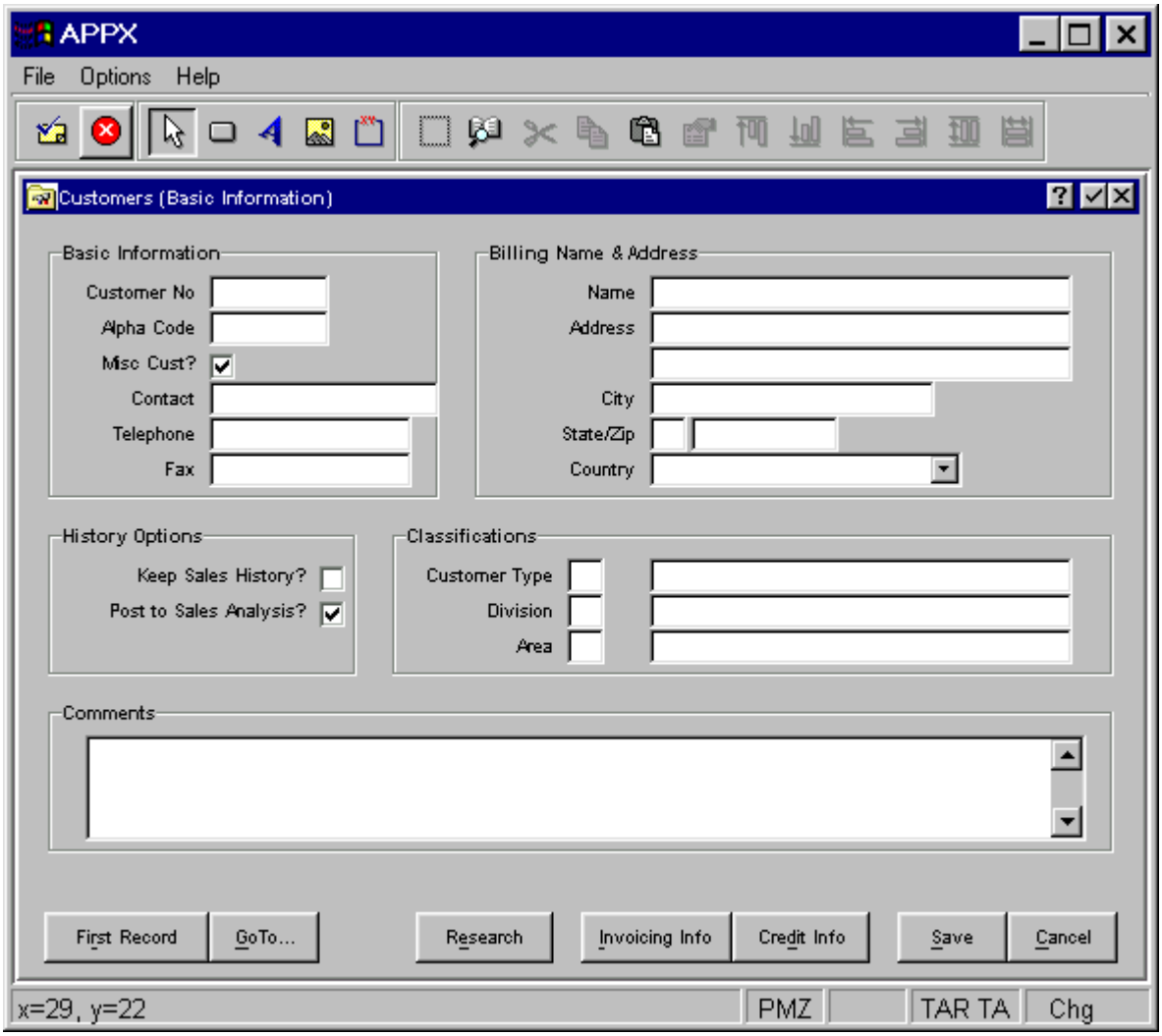


Figure 3-2-1. Image Editor

The Image Editor includes options to:

- Format items (data dictionary fields) and objects on images.
- Define background text and characteristics on images.
- Establish data lookups to read data in related files.
- Assign graphical attributes to images or image items or objects.

Refer to *Chapter 3-14: Using the Image Editor* or *Chapter 3-15: Using GUI Features of the Image Editor* for more information on the Image Editor.

Automatic Data Lookup and Display

Data lookups link a field in one file to additional data in a related file by means of a key field in the related file. In other words, they tell APPX how to access another file during process execution. You define a lookup with the image editor as an integral part of defining an item on an image.

For example, you may enter a customer number on an order image. If you want to automatically include additional descriptive information (such as the customer's name) on the image, you can do so with a data lookup to the customer file. See *Chapter 3-14* or *3-15* on using the Image Editor, for additional information.

Frame/Image Positioning

When you create frames and images for menu, input, and output processes, you are asked to define a size and display position for each.

- Sizes are stated in terms of the number of rows and columns a frame or image occupies. It is important to note that, while images consume physical space on the screen or printed page, frames do not. Frames simply designate regions for image placement.
- The position a frame or image occupies (where it displays on the screen or page) is determined by a number of factors, including the type of process, the way the process was invoked, and whether or not it includes a scrolling display.

You can surround frames and/or images in menu or input processes with a box. Boxes do not consume rows or columns within the space that is defined for a frame. They do, however, consume more physical space than overlines and underlines. The physical space on a screen that is consumed by a box is system-specific.

Various factors affect the actual positioning of a frame or image, as discussed in the following sections:

- Screen Positioning
- The Effect of Child Invocation Screen Display
- Page Positioning
- The Effect of Scrolling Frames on Screen and Page Positioning

Screen Positioning

For menus and input processes, you must designate where the frames and images of a process are placed on a screen:

- Frames are positioned relative to coordinates of a screen. The amount of screen surface that is available for frame display is equal to the size of the surface less three rows (reserved for the APPX status display at the bottom of the screen).
- Images are positioned relative to the coordinates of a frame.

You designate position by entering two specifications, one representing a coordinate on a row axis and the other a coordinate on a column axis. Together, these coordinates determine the position at which the frame displays on the screen or the image displays on a frame. (The concepts for positioning a frame on a screen or an image on a frame are identical. Unless otherwise specified, the following information applies equally to both.)

You can enter coordinates as positive numbers, indicating image positioning relative to one of the borders of a frame:

- A row coordinate indicates the placement of the upper border of the image relative to the upper border of a frame. For example, an entry of *1* aligns the upper border of an image with that of a frame. An entry of *2* leaves a single blank row between the border of a frame and the border of an image. A negative entry aligns the upper border of an image with that of the bottom border of a frame. For example, a entry of *-3* places the first row of the image 3 lines from the bottom of the frame.
- A column coordinate indicates the placement of the left border of an image relative to the left border of a frame. An entry of *1* aligns the left border of an image with that of a frame. An entry of *2* leaves a single blank column between the border of a frame and the border of an image.
- Use entries of *0* in either the column or row coordinates (or both) to center the image within the frame's total columns and/or rows.

The Effect of Child Invocation on Screen Display

Your screen display is affected by the way you invoke a child process:

- If invoked as a subprocess, a child's images can overlay the images of the parent process.
- If invoked as a related or detached process, a child's images cannot overlay the images of the parent process. The screen is automatically cleared immediately before the child displays, and is refreshed immediately after the child completes.

Page Positioning

You define sizes and positions for frames and images in output processes just as you do in input and menu processes. The primary difference between screen positioning for a menu or input process and *page positioning* for an output process, however, relates to the timing of the display. This timing issue is a direct result of the difference in display media for an input process (a screen) versus an output process (a printer).

When displaying an input process, for example, it is a relatively simple matter to add an image that displays before, beside, or perhaps between existing images by refreshing the screen. It is an entirely different matter when those images are committed to paper. You cannot request a printer to back up to print an image beside a previously printed image, or perhaps insert a new image between two printed images.

To overcome this printer limitation, APPX uses a preliminary assembly area for output frames and images called the *Print Beside buffer*. Various components of the output can be pieced together in the buffer before they are committed to paper. A buffer permits you to insert images before or beside existing images, much as you do when displaying to a screen. See the Print Beside section for details.

If you do not use a Print Beside buffer in your output process, each image is immediately committed to the print file during execution. As a result, each image must print *below* the previous image (unless, of course, a new page is started).

The Effect of Scrolling Frames on Screen and Page Positioning

A *scrolling frame* is a type of input or output frame that displays more than a single image (one image for each record, up to a maximum number of images), and formats those images in multiple rows and/or columns. The maximum number of images that can display in a single scrolling frame is limited by the relative sizes of the frame and the image. The images that display in the frame can all be the same alternate image definition, or consist of any combination of alternate images.

Images for scrolling frames do not have predefined positions. APPX determines the (multiple) positions for displaying an image within a scrolling frame. These positions are influenced both by the scrolling method, and the relative sizes of the frame and its alternate images.

APPX supports two general methods of scrolling:

- Matrix Scrolling Mode
- Columnar Scrolling Mode

Matrix Scrolling Mode

In ***matrix scrolling***, APPX divides a frame into a number of equally sized regions based upon a composite maximum image size. This composite image is large enough to accommodate any of the possible alternate images for the frame, regardless of which one actually displays. APPX calculates this composite size in two ways by: combining the width of the alternate image that has the largest *Columns* specification, and the height of the image that has the largest *Rows* specification; and adding any extra rows or columns you defined to border the image. APPX then calculates the maximum number of regions that will fit down the frame and the maximum number of regions across the frame, based upon this composite image. Any extra space is evenly distributed between the regions.

For example, assume your frame size is 16 rows by 33 columns, as shown in *Figure 3-2-2* below. You defined two alternate images for the frame: one is three rows by 10 columns and the other is five rows by seven columns. You did not specify extra rows or columns. The composite image is five rows by 10 columns. This composite image can be displayed three times down the frame (calculated by dividing 5 into 16) and three times across the frame (10 into 33), leaving one extra row and three extra columns. To evenly distribute the remaining space, APPX allocates five rows and 11 columns to each region in the matrix, and leaves one extra row at the bottom of the frame.

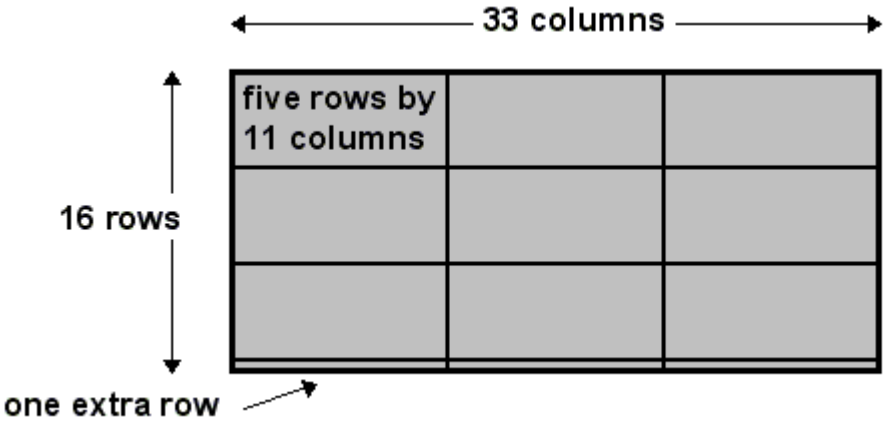


Figure 3-2-2. Region Allocation within a Frame

Because the regions are based upon a composite image size, rather than the actual size of each alternate image within a frame, the images in matrix scrolling are always perfectly aligned. Regardless of the alternate images selected, all images in the same row have the same row axis coordinate, and all images in the same column have the same column axis coordinate, as shown in *Figure 3-2-3*.

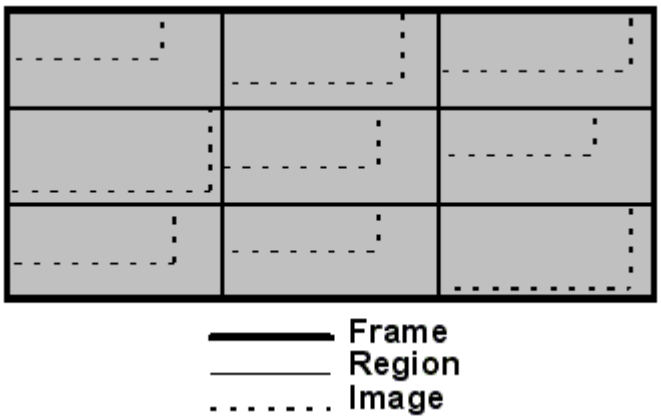


Figure 3-2-3. The Positioning of Alternate Images in a Frame

For both input and output processes, you determine the sequential placement of images within the regions by requesting one of the following two matrix scroll types. The upper-left corner of each region is the starting coordinate for a selected image.

Down, APPX places the first image in the upper left corner of the frame. Succeeding images are placed immediately below the previous image until the column is complete. The next image is placed at the top of the column immediately to the right of the completed column and the first image. As each column completes, the next image begins a new column until all columns complete (see *Figure 3-2-4*).

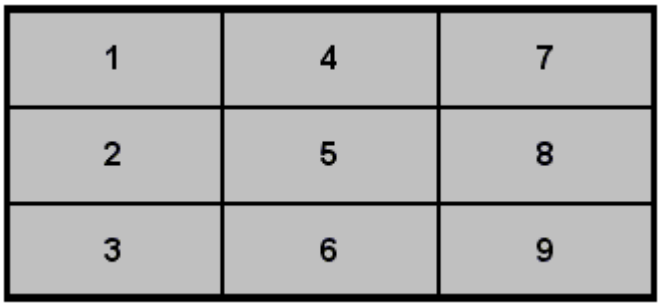


Figure 3-2-4. Down Scrolling

Across. APPX places the first image in the upper left corner of the frame. Succeeding images are placed immediately to the right of the previous image until the row is complete. The next image is placed in the far left-hand position of the row immediately below the completed row and the first image. As each row complete, the next image begins a new row until all rows complete (see *Figure 3-2-5*).

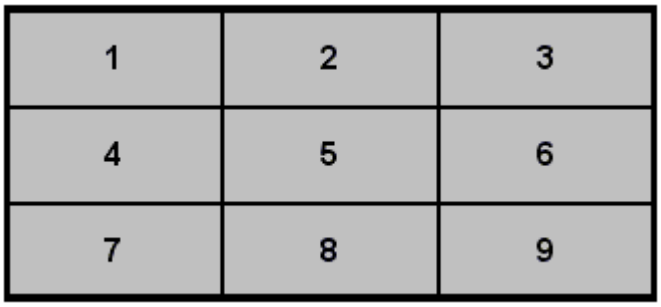


Figure 3-2-5. Across Scrolling

Columnar Scrolling Mode

In ***columnar scrolling,*** shown in *Figure 3-2-6* below, the display size of an image is based upon the actual number of rows in the alternate image selected, not upon the region of a matrix scroll. APPX determines column width based upon the number of columns in the widest alternate image and an even distribution of the remainder, if any.

The sequential placement of images is similar to that of matrix-down in that each image is placed below the previous image until the column completes, and the next image begins a new column immediately to the right of the previous column. Unlike matrix scrolling, however, the images in a column are not aligned horizontally unless all the alternate images are the same length.

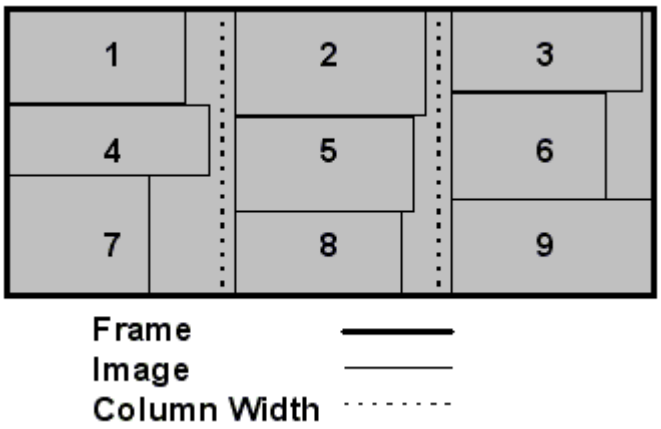


Figure 3-2-6. Columnar Scrolling

Columnar scrolling never displays a partial image in a column, either leading or trailing. All images must be totally visible within the column. If an image does not fit on the bottom of the current column, the entire image displays from the top of the next column (or on the next screen if the current column is the far right-hand column on the screen).

Input Process Concepts

Input processes are interactive; users navigate from screen to screen, select options, and enter and manipulate data. You control the way users interact with APPX by establishing *entry levels*, *keymaps, modes,* and *interactive phases*.

Input Entry Levels

When you paint an item on an input image, you designate the item's *entry level* (0 through 9). This entry level establishes the point in time when the item becomes editable.

- If the entry level is *0*, the item is always editable.
- If the entry level is nonzero, a user first sees the item in the data entry level represented by the nonzero value. For example, an entry level of *1* indicates that the item is initially active. An entry of *2* indicates that the item becomes active after the user selects RETURN for the first time. An entry level of *3* indicates that the item becomes active after the user selects RETURN for the second time, and so on through the highest entry level used.

The modifiability of an item based on its entry level can be overridden with OK INPUT and NO INPUT statements.

Input Keymaps

The specific keys users press to perform each input-related activity is installation-specific. The key on the keyboard (or other input device) that controls each operation and option is mapped according to the devices at your installation. The resulting keymap tells APPX which keys control which operations. Refer to your system administrator for information about establishing customized keymaps in APPX.

The Data Entry Keymap

The data entry keymap provides capabilities relevant to entering, displaying, and manipulating information on the screen. For input processes, the user is always in the data entry keymap.

In the data entry keymap, only items that have an entry level less than or equal to the current entry level are visible and modifiable on the screen. For example, when APPX first displays the screen, only items with an entry level of *1* are visible. After the user selects `RETURN`, the items for entry level *2* become visible. This process continues until the entry levels for all items on the image are complete.

You can also establish that an item associated with a specific entry level is non-modifiable either by completing specifications in the image editor, or by explicitly defining a NO INPUT statement.

Input Modes

APPX input processes execute in any of four modes: add, delete, inquire, or change. The mode establishes the type of operation a user performs, relative to records in an APPX file.

Add mode
Permits users to save new records in a file.

Delete mode
Permits users to remove records from a file.

Inquire mode
Permits users to view records in a file.

Change mode
Permits users to modify records in a file.

A user can select the corresponding icon from the tool bar to switch modes at any time. You can, however, restrict a user's ability to access any mode. You can also interrogate the mode by using statements. Refer to the *User Manual* for additional information on the use of modes.

Input Interactive Phases

Each mode in an APPX input process can be further divided into one of six interactive phases: *data addition, data modification, data inquiry, data deletion, data scroll,* and *key entry*. The relationship between mode and interactive phase is shown in *Table 3-2-1* below.

Table 3-2-1. The Relationship between Modes and Phases for the Data Entry Keymap

Mode	Interactive Phase	Description
Add	Data Addition	Screen displayed for data entry.
	Data Modification	Record on scrolling display selected for editing.
	Data Inquiry	Record on scrolling display selected for Inquiry. Record already written to disk and may beheld by another user.
	Data Scroll	Scrolling display being populated.
Change	Data Modification	Record displayed for editing.
	Key Entry	Key field(s) displayed for selection of record.
	Data Scroll	Scrolling display being populated.
Delete	Data Deletion	Record displayed for deletion.
	Key Entry	Key field(s) displayed for selection of record.
	Data Scroll	Scrolling display being populated.
Inquire	Data Inquiry	Record displayed for Inquiry.
	Key Entry	Key field(s) displayed for selection of record.
	Data Scroll	Scrolling display being populated.

Automatically Saved Records

Input processes are the only type of process that *automatically* save records. APPX automatically saves records in input processes:

- After a user completes an input process cycle (a single execution of all the frames in a process) that affects the contents of a record.
- Before invoking any automatic child process from an input process.
- Before invoking an optional child process from an input process.

Refer to Chapter 3-13: Automatic and Optional Children for more information on child processes and the *Invocation Type* specification.

Output Process Concepts

Output processes permit you to format complex reports based on information contained in, or derived from, an APPX database. To provide this flexibility, the output process includes two additional design components: *frame class* and *Print Beside* specifications.

Output Frame Classes

Specifications for output frames are classified according to whether they apply to a report in total, pages, or specific sort ranges in a report, or to individual records in the body of a report. Those specifications that relate to each of these categories are grouped together on a single definition screen and referenced by a *frame class*. Because of this classification, there are seven frame classes that can be entered when defining a report (presented in their order of appearance in the output):

Start of Report frames

Print once at the beginning of a report. APPX executes a page break immediately after the last *Start of Report* frame prints.

Start of Page frames

Print once at the beginning of each page of a report. They are primarily used for page headings and column headings. Standard images for both of these are optionally available. Refer to Standard Column Heading Image for additional information.

Start of Range frames

Print immediately before the first record for which the value in a specific sort field (the range field) changes. They are primarily used for subheadings and lower-level column headings. Standard images for both of these are optionally available. Refer to Standard Column Heading Image for additional information.

Record frames

Manipulate and print information read from the process control file. They comprise the body of a report. *Record* frames can be defined to scroll.

End of Range frames

Print immediately after the last record in a series of records that have the same value in a specific sort field (the range field). They are primarily used for subtotals, and a standard subtotal image is optionally available.

End of Page frames

Print once at the conclusion of each page of a report.

End of Report frames

Print once at the conclusion of a report. APPX executes a page break immediately before the first *End of Report* frame executes.

See the next section for a discussion of output frame class execution.

Output Frame Class Execution

For each frame class, you can define multiple frames, each of which executes in *Sequence Number* order.

Report and page class frames never execute unless APPX executes at least one range or *Record* class frame. This guarantees that you do not generate empty reports, comprised exclusively of report headings and footers, and page headings and footers.

With the exception of range class frames, all frames within a class execute when the frame class executes. Individual range class frames execute conditionally; if the range field is not used as a sort field when the report executes, the range class frame does not execute.

Figure 3-2-7 depicts the placement of frames on a printed page.

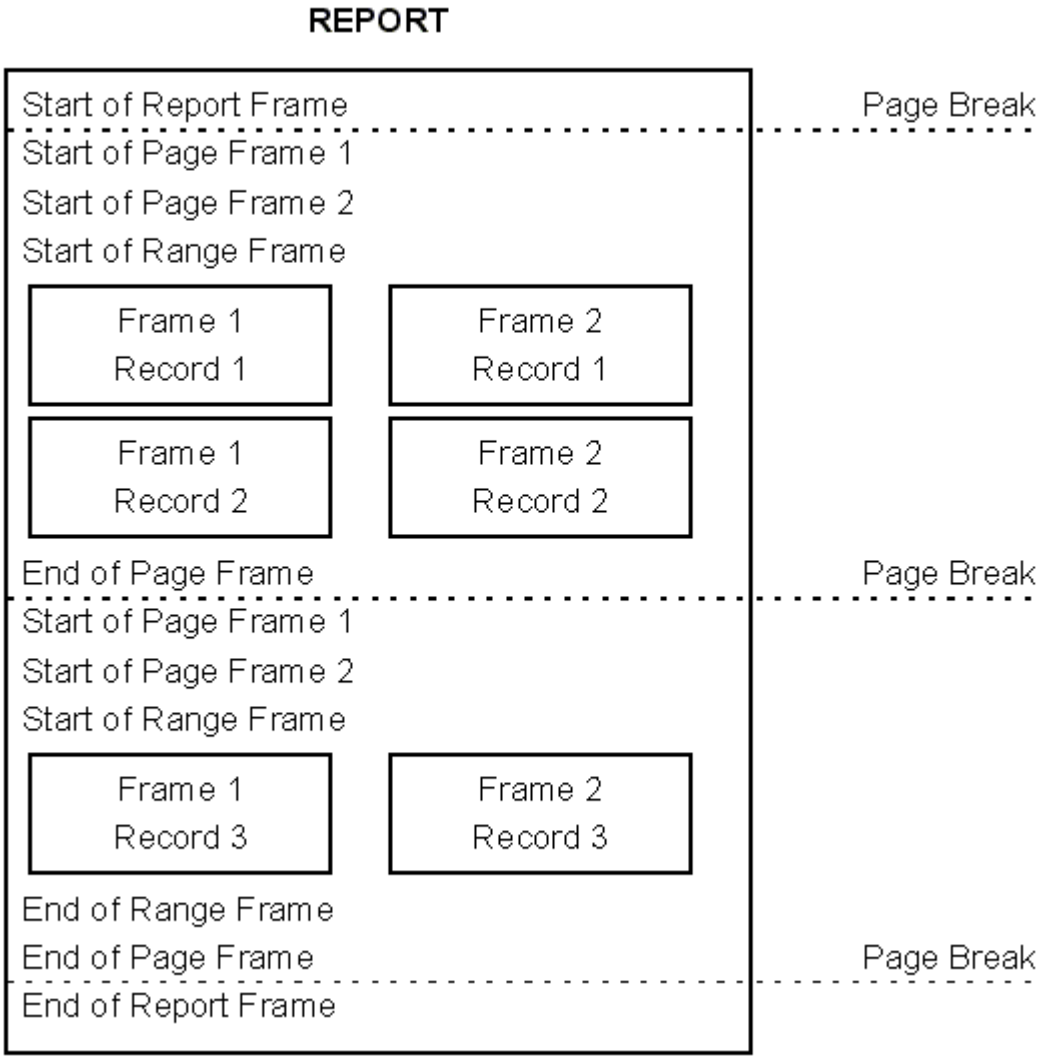


Figure 3-2-7. Placement of Frames on a Printed Page

Print Beside

In APPX it is possible to print a report that contains items that print across the page by using the Print Beside feature available via the Output Frame Additional Attributes Option. As you read the following paragraphs, refer to the Print Beside output processes in the 1EX application included with your copy of APPX. In this example, *Start Print Beside?* is set to ☑ in the record frame of the parent process in a 2-level output. A diagram of how such an example behaves on a printed page is shown in *Figure 3-2-8* below.

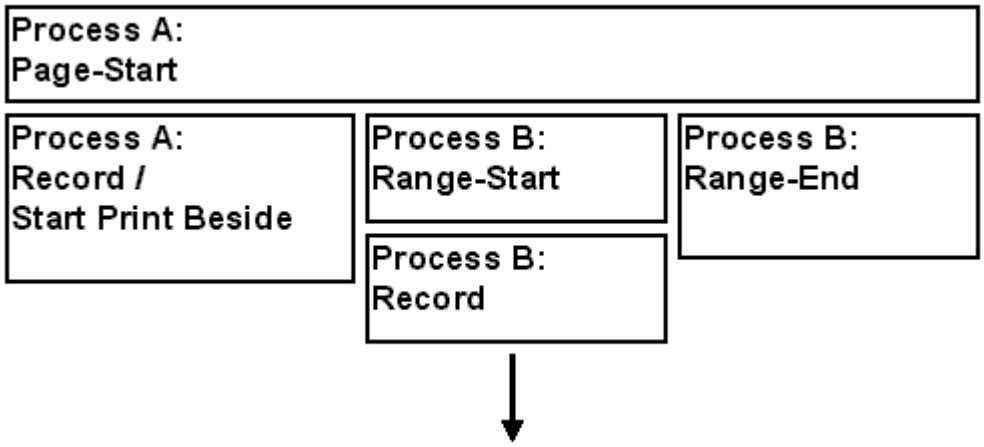


Figure 3-2-8. Placement of Frames on a Printed Page Using Print Beside

Any frame with *Start Print Beside?* set to ☑ will allow subsequent frames to print beside or to the right of this frame as long as the column position of the subsequent frame does not overlap the preceding one. If the column position overlaps, the record is printed below the preceding one. In the example above, the column position of *Process B: Record* overlaps *Process B: Range-Start*, so instead of printing beside, it prints below. Note, however, that the next frame, *Process B: Range End*, prints beside *Process B: Range Start* since its column position did not overlap. In other words, once *Start Print Beside?* is set to ☑, all subsequent frames will print beside or to the right of the initiating frame as long as it is physically possible. When APPX gets back to the frame that started Print Beside, it prints the information and starts a new Print Beside buffer in memory. A sample report using the Print Beside example diagrammed above is shown in *Figure 3-2-9* below.

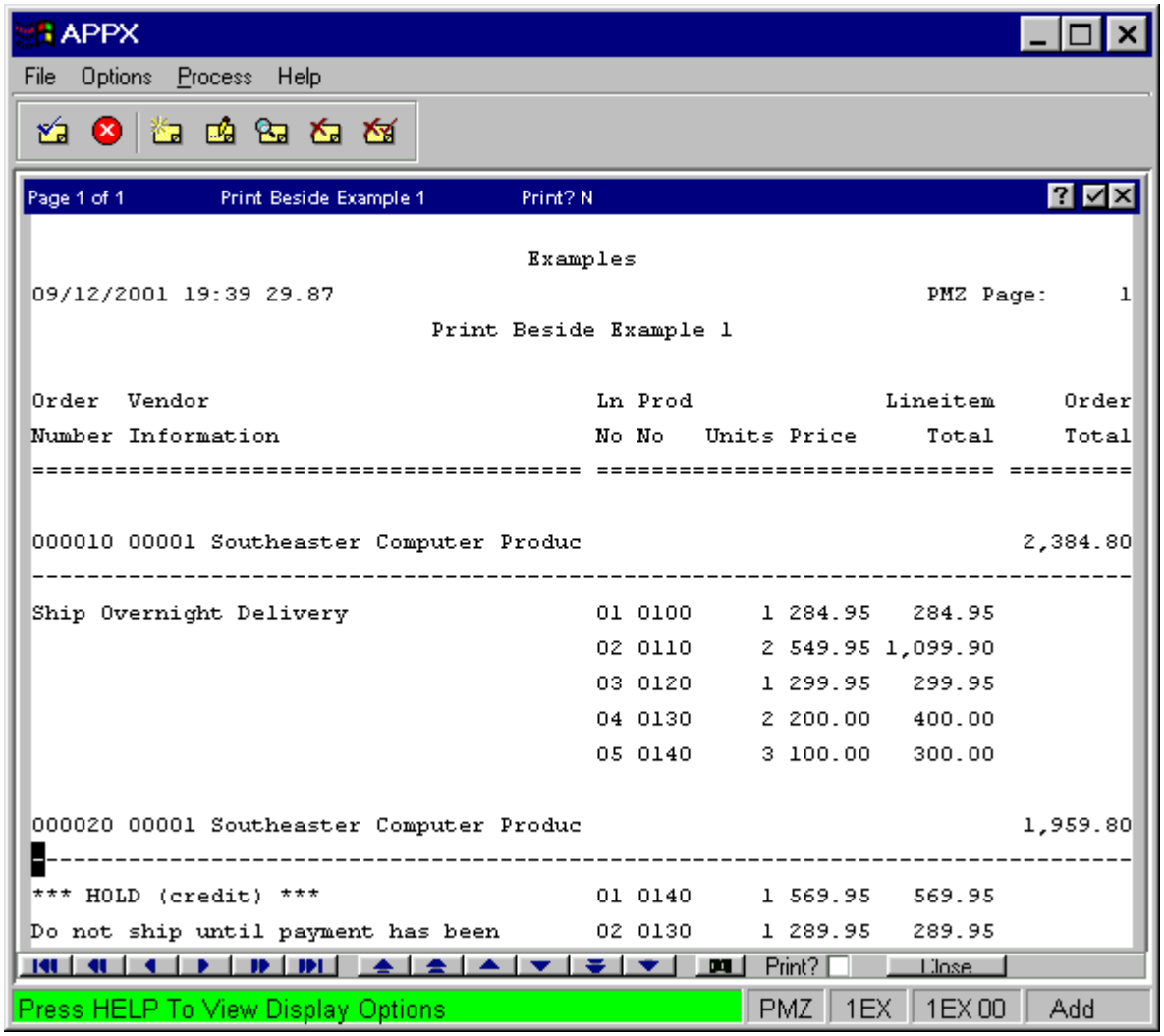


Figure 3-2-9. Sample Report Using Print Beside

In this report, *Process A: Record* includes the Order Number and Vendor Information as well as a text field for shipping information. Since Print Beside is set to ☑ here, the next frame, *Process B: Range-Start*, prints to the right of *Process A: Record*. *Process B: Record* containing the line items you wish to print, is defined with a column position overlapping that of *Process B: Range-Start*, so instead of printing beside *Process B: Range-Start*, APPX prints it below. Note that the purpose of *Process B: Range Start* is to drop you down one line so that the line items print on the line below the subsequent Order Total which is contained in the next frame, *Process B: Range-End*. The Order Total prints beside since its column position does NOT overlap the far right edge on any preceding frame in the record. When APPX gets to the next Order Number/Vendor Information record, where Print Beside was set to ☑, it starts the printing cycle over.

Keep in mind that the Print Beside feature takes up significantly more memory than regular printing.

Update Process Concepts

Update processes update and transfer data between files. The update process permits the execution of statements that are not associated with either an input or an output process.

The structure of the update process is similar to the structure of the output process. Both include process control files, frames, event points, and automatic children. As with output processes, frames for update processes are divided into classes, in this case *Record, Start of Range*, and *End of Range* class frames.

Unlike output processes, however, update processes do not format data for presentation to a user. As a result, they do not use images, and do not include report or page class frames.

Query Process Concepts

Query processes determine the records that are available to a subsequent input, update, or output process, and/or establish the order the records are processed in (the sort hierarchy).

At your option, queries permit an application user to dynamically define:

- The records that are available to a process.
- The order in which those records are presented.

The query affects only those subsequent processes that you elect to *use* the query for. You do so by completing a *Use Query* specification for the process' child invocation. Refer to *Chapter 3-13, Child Specifications,* for instructions on completing the *Use Query* specification.

Query Setup versus Execution

A query process is comprised of the following two parts:

Setup,
Which populates three temporary memory files that contain a query's sort and record selection constraints. It optionally displays Record Selection and Sort Order screens from which the application user can dynamically modify sort and selection criteria.

Execution,
Which consists of the actual selecting and sorting of records in conformance with the criteria established during query setup.

Query setup executes when a query process is invoked as an automatic child from another APPX process. Query execution is performed in conjunction with the process that uses the query. *Figure 3-2-10* is an example of a process tree, with query and output processes executing as children of a

controlling job process. Query setup executes first, permitting interactive editing of the query specifications (record selection criteria and sort order). Query execution is performed in conjunction with the subsequent output process that uses the query.

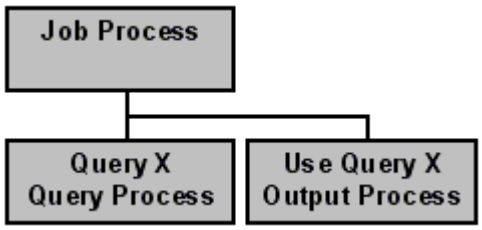


Figure 3-2-10. An Example Process Tree

If a query process is not explicitly invoked as a child, query setup executes implicitly with query execution and in conjunction with the child process that uses the query. In such a case, query setup's Record Selection and Sort Order screens do not display to a user.

It is important to note that, while queries execute at the same time as the first process that uses the query, *Query Execution* event points do not share record areas or record pointers with the other process. The two processes, the query, and the process using the queried records, can be considered to execute in parallel, communicating with one another only through the records that are retrieved. (The process requests a record from the query; the query returns a record.) Consequently, APPX cannot guarantee that the query, particularly the *End of Query Execution* event point, will complete before *Start of Process* for the process that is using the queried records. Therefore, you should not use a query to compute summary information for use by a subsequent process.

Query Files

Query files consist of a *process control file (PCF),* three *temporary files,* a *pointer file,* and an optional *alternate control file.*

- Process control files are mandatory for queries; they govern the execution of a query by identifying the file that contains the records to be constrained and/or sorted.
- Queries populate three temporary files at the end of setup. *QSLCT* maintains selection constraints, *QPARM* maintains selection expressions, and *QSORT* stores sort fields. These temporary files are populated based on the sort and selection criteria established for the query in application design, and adjusted by the application user during query setup. The contents of these files determine the records from the PCF that are available to any process using the query.
- During execution, the query process creates a pointer file, which includes the keys to all records that satisfy the selection criteria, in the order that is designated by the sort.

Once the temporary files and the pointer file are populated, the query is referred to as *active*. In general, a query remains active throughout the execution of a subprocess family or a related process family. When a process that was invoked as detached ends, any queries within its scope automatically end.

This permits you to have more than one query active at any time, and to use the same query with more than one process in a family, as shown in the *Figure 3-2-11*.

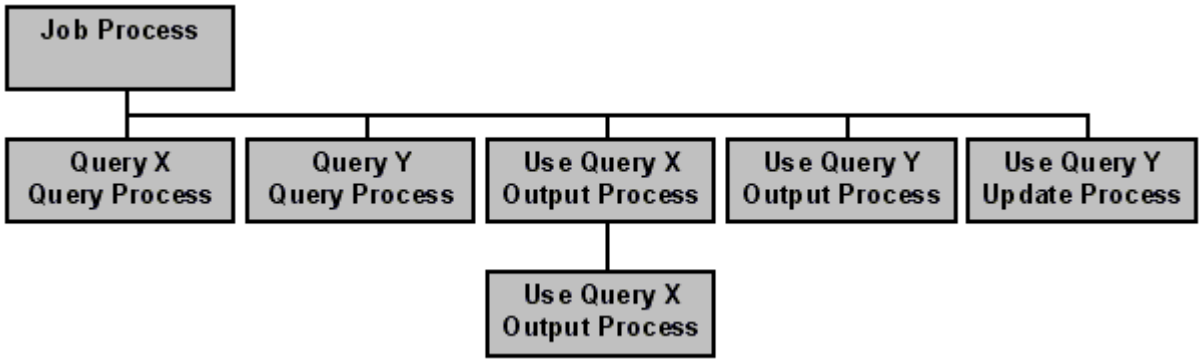


Figure 3-2-11. A Process Tree with Multiple Potentially Active Queries

Job Process Concepts

Job processes are used to control the sequencing of related processes. The job process executes a series of automatic children. It serves as a control structure to group the related child processes.

Inquiry Process Concepts

Inquiry processes have most of the characteristics of output processes, including all of the same kinds of frame classes. They differ from output processes in only three ways: 1) the assumption is made that the image is going to the screen so the default width is 80 columns and the *Print On Screen* value is set to *Y*; 2) the assumption is made that the user wants a specific record, or set of records, so the process definition function creates an input frame that executes before the output and requests the record's key; and 3) there is no Print Beside feature.

Status Process Concepts

Status processes provide various error handling options. The status process maintains a log file of error and warning messages that generate during the execution of the process. This log is displayed to the screen. If error conditions are detected, you can stop the execution of the process.

Subroutine Process Concepts

Subroutine processes define *reusable procedural programs* that execute from other APPX processes. The subroutine process is essentially a means of grouping a number of statements that can execute as a unit.

You can execute a subroutine:

- As a child of another APPX process.
- From an event point in another process with a COPY, SUBR or GOSUB statement.

Performance Tuning

Several options in APPX permit you to affect the performance of your application. Because there are tradeoffs associated with performance tuning, such as consumption of additional disk space, all performance-enhancing options are not automatically employed. The best combination of options is determined by the specific circumstances of your installation.

Saving Executable Modules

An *executable module (EM)* is the end result of compiling a process. It is the APPX-executable format that is generated by the process compiler, and accessed by the process loader and process interpreter to execute the process. A process is automatically verified and compiled by APPX when the process is loaded.

To avoid verifying and compiling a process each time it is loaded, and thereby increase performance, you can elect to save an EM on disk. Once an EM is compiled and saved, APPX refers to the saved EM during execution. The process loader, however, verifies that the saved EM is current. If changes to application specifications make the EM obsolete, APPX automatically generates a new EM.

Each EM is automatically assigned an eight-character alphanumeric file name. This unique string is never changed. If APPX finds it necessary to recreate an existing EM, the new EM retains the same file name.

You can designate whether or not you want to save EMs at the installation, application, and process levels. You establish installation and application values in system administration. You can optionally override these designations for each individual process in application design. If you leave a process specification blank, the application value is assumed. If both are blank, the installation value is assumed.

There is a cost associated with saving an EM; it consumes disk space. This trade-off must be evaluated based upon the specifics of each installation. As a general rule, however, EMs should be saved for larger processes, because they require significant compile time. For smaller processes that are relatively quick to compile, the consumption of additional disk space may not be warranted.

Refer to *Chapter 3-3, Process Definition*, for *Executable Module — Save on Disk?* specifications.

EM Caching

Without explicit designer intervention, a *related* or *detached* process' executable module (EM) is automatically loaded into memory when the process begins, and automatically removed from memory when the process completes. A *subprocess'* EM remains in memory only as long as the parent process' EM remains in memory. You can override this automatic removal of the EM from memory with APPX's *EM Caching* capabilities.

EM caching refers to the technique whereby the EM for a non-active process is retained in memory. Thus, the use of EM caching enhances perceived performance by reducing the time that is required to begin a process. When entering sales orders, for example, a user accesses both an order header input process and an order line items input process. If the EM for the order header process is not cached, it has to be reloaded each time the line items for one order are completed and the user begins a new order. If the order header process is cached, its EM is already in memory when the user completes line item entry. Note that subprocesses are always cached for at least the duration of the subprocess family. Thus, in this example, the line item process is always cached during sales order entry.

EM caching is controlled through a *Cache Priority* specification for each process, and through *Cache Level* and *Maximum Number of Cached EMs* specifications that are defined in system administration for each user:

- The cache priority (a number from 0 through 99) determines the likelihood that the EM for a specific process is retained in memory, relative to the EMs for other processes. A process with a cache priority of *0* is always discarded from memory when the process completes. A process with a cache priority of *99* has the highest priority; it is the most likely to be retained in memory after the process completes. (A subprocess assumes the cache priority of its parent process.)
- The cache level is the user's "automatic discard level." In other words, any process that has a priority equal to or less than this number is never cached. The cache level ensures that low-priority EMs (determined relative to a specific user) do not consume space in memory.
- The cache limit determines the maximum number of EMs for non-active processes that can be retained in memory at the same time. When this limit is reached, APPX does not prohibit a new EM from entering the cache, as long as its priority is greater than that of at least one EM already in the cache. The EM with the lowest priority (whether this EM is already in the cache or is attempting to enter the cache) is discarded.

As a result of this combination of process-level and user-level specifications, caching definitions are extremely flexible. They are both process-specific and user-specific. The same processes may be retained in memory for one user and not another. The same process run by the same user may be retained in memory during one session, and discarded from memory during another session, depending on the relative priorities of other processes that are executed during the session.

Refer to the *System Management Manual* for additional information on caching specifications.

Documentation

APPX provides screens to document each process as an integral part of its definition. When you create a process, you define:

- ***User documentation.*** The text you enter on these screens is the basis for APPX's online help facility. An example is shown in *Figure 3-2-12.* (User documentation can also be directly maintained using the Documentation Management option from the APPX Main Menu, as described in *Chapter 4* of the *System Management Manual*.)
- ***Technical documentation.*** The text you enter on these screens provides additional technical documentation.

Each User and Technical Documentation screen includes specifications that identify the process being documented, and provides 18 lines of up to 78 characters to explain the process. The *External Copy* feature lets you access other documentation records and copy them into User and Technical Documentation screens.

When you enter text in any of the documentation screens, APPX provides a number of editing functions, including word-wrap, text insertion, text redisplay, and end paragraph control. Refer to the *User Manual* for a discussion of text editing capabilities.

User Documentation screen

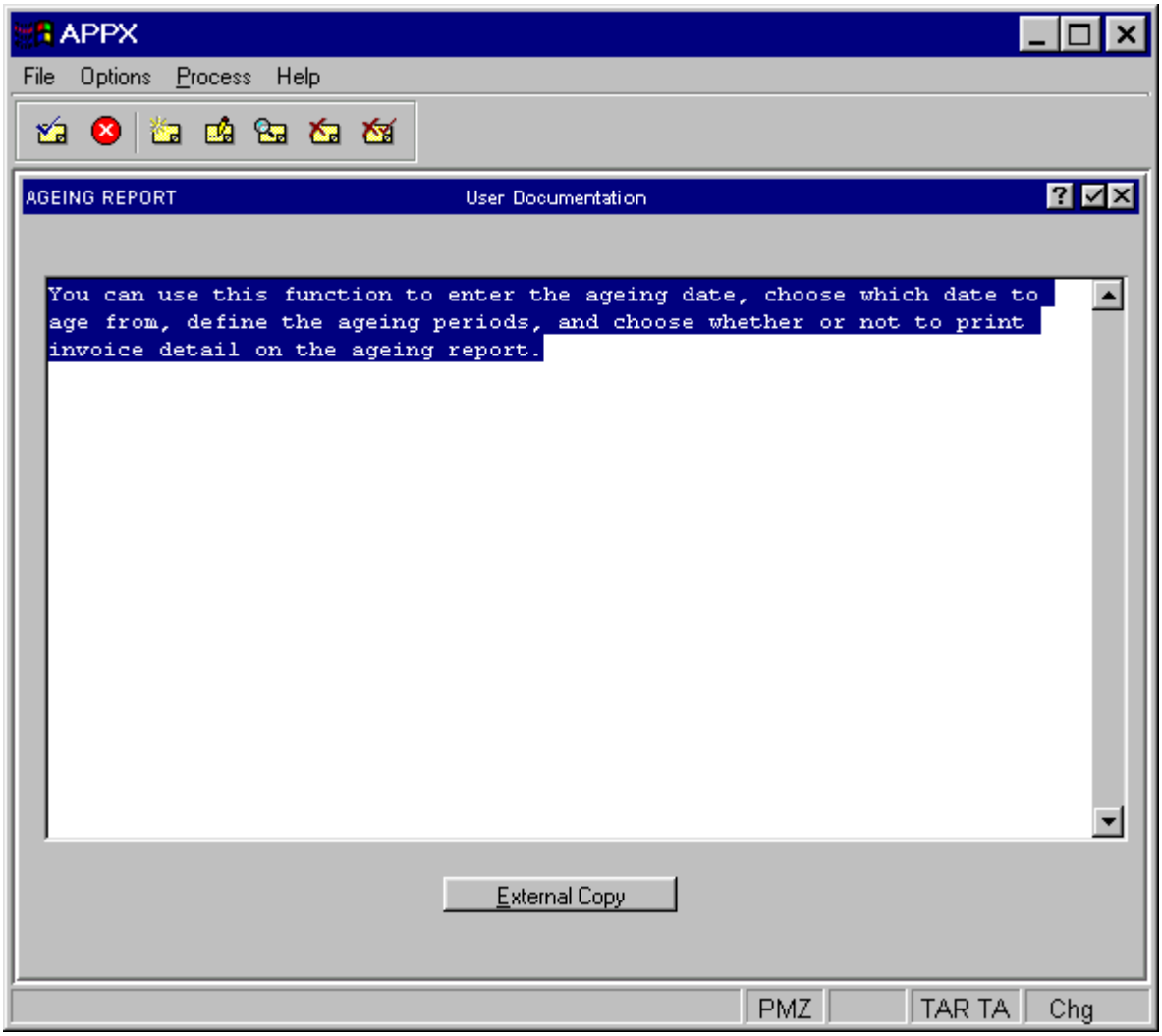


Figure 3-2-12

Accessing Process Documentation Screens

Documentation is an integral part of the processes facility. You access:

- User or technical documentation for any process in the process facility by selecting the appropriate tab on the process Documentation overlay.
- The external copy of a documentation record by selecting an option on the process User or Technical Documentation screens.

Security

Security refers to the facility through which programs and data are protected from unauthorized access and/or modification. Within the APPX environment, security is optional, and can be implemented or modified any time during the life of an application.

There are two phases of APPX security: design and implementation. Each of these are the responsibility of different functions within your organization.

- A designer establishes what to protect.
- A security administrator determines who needs access to secured components of applications, and in what combinations.

Each process permits the specification of an *Execute Security Code*. Only those users who have rights to the security code are permitted to execute the process. Refer to the Database Security section of this manual for an overview of APPX security and a discussion of data security. Refer to the *System Management Manual* for a discussion of security implementation.

CHAPTER 3-3

Process Definition

Introduction to Process Definition

Processes are reusable software building blocks for constructing APPX applications. You can access the nine types of processes from the Application Design Specifications Menu shown in *Figure 3-3-1* below. Processes are grouped on this menu according to whether or not they support image design. **Form Based Objects** include processes that invoke the Image Editor to define or edit images. **Processing Objects** include processes that do not involve the design of images.

Application Design Specifications menu

Figure 3-3-1

Form Based Objects include the following four types of processes:

- ***Menus*** provide an on-screen display from which to execute selected processes. For example, a screen that lists the reports available in an application and from which you select a specific report is a menu process. Refer to Chapter 3-4: Menu Processes for a discussion of menu specifications.
- ***Inputs*** permit you to enter or maintain data in one or more files from designer-formatted screens and overlays. A sales order entry function and a customer file maintenance function are both input processes. Refer to Chapter 3-6: Input Processes for additional information.
- ***Inquiries*** are exactly like output processes, except that they automatically print to the screen, and allow a user to select a specific record for inquiry. Refer to Chapter 3-10: Inquiry Processes for more information.
- ***Outputs*** convey information from a database, either on the screen or in hard copy, according to a designer-defined format. Trial balances, product purchase history reports, and customer lists are all examples of output processes. See Chapter 3-7: Output Processes for more information.

Processing Objects include the following five types of processes:

- ***Queries*** determine the records that are available to a subsequent input, update, or output process, and/or establish the sort hierarchy. See Chapter 3-9: Query Processes for more information.
- ***Updates*** alter a database by adding, deleting, and changing records in the database's files. Refer to Chapter 3-8: Update Processes for additional information.
- ***Subroutines*** define reusable procedural programs that can be executed from other APPX processes. A program that converts one unit of measure to another, for example, can be written as a subroutine. Refer to Chapter 3-12: Subroutine Processes for more information.
- ***Job streams*** control the sequencing of related processes. Refer to Chapter 3-5: Job Processes for more information.
- ***Status checks*** provide various error reporting options. Refer to Chapter 3-11: Status Processes for additional information.

Common Process Specifications

Common process specifications define attributes that are applicable to all processes regardless of type (input, output, update, etc.). A Subroutine Processes screen, shown in *Figure 3-3-2, illustrates these common elements.*

A Subroutine Processes screen

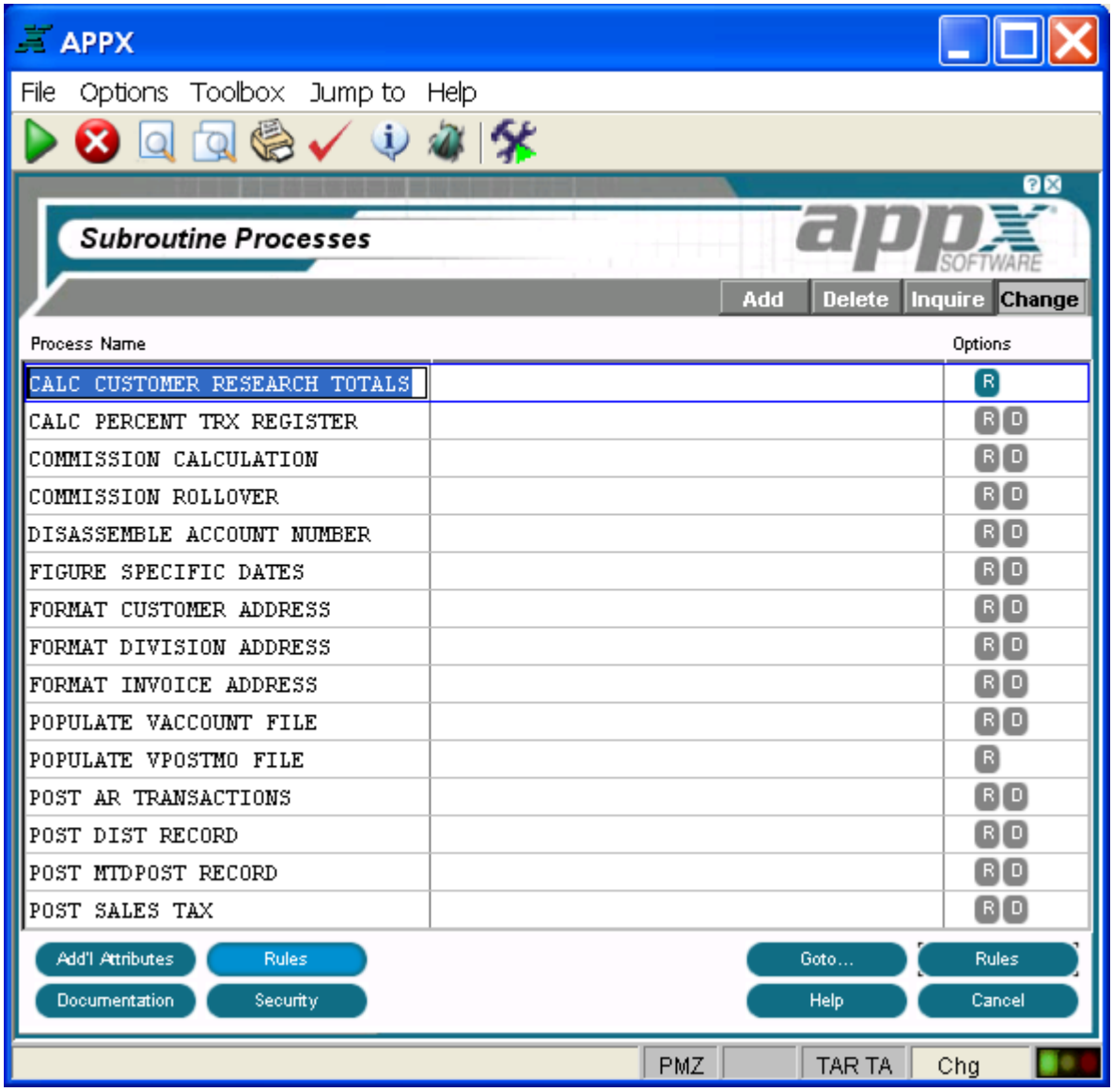


Figure 3-3-2

Every Processes screen regardless of type contains the following items:

- *Process Name* identifies a specific process of the type named in the screen heading. Process names may contain up to 30 uppercase alphanumeric characters and must be unique within type and application.
- *Options* are indicators that are present whenever the options they denote have been defined. Options applicable to this screen are AA (Additional Attributes), R (Rules), D (Documentation), and S (Security).

Buttons are provided at the bottom of each Processes screen for accessing options. Process options vary depending on the type of process you are entering, but every process offers the following options:

- ADDITIONAL ATTRIBUTES - See the Process Additional Attributes Option for details.
- RULES - See the Process Rules Option for details.
- DOCUMENTATION - See the Process Documentation Option for details.
- SECURITY - See the Process Security Option for details.
- GOTO... - Allows you to identify and retrieve another record by optionally entering a full or partial key.
- HELP - Allows you to view online help text for a selected item.
- CLOSE or CANCEL - In Inquire mode, CLOSE allows you to end the current process. Otherwise, CANCEL allows you to end the current process without saving changes, if any, to the current record.

In addition, the menu bar on every Processes screen provides the following pulldown menus:

- **Toolbox** - See the Process Toolbox Pulldown Menu for details.
- **Jump to** - See the Process Jump to Pulldown Menu for details.

Process Additional Attributes Option

When you select the ADDITIONAL ATTRIBUTES option from a Processes screen, a process Additional Attributes overlay is displayed similar to the sample shown in *Figure 3-3-3 for a subroutine process.* It illustrates the fields applicable for all processes, although some processes have still more attributes.

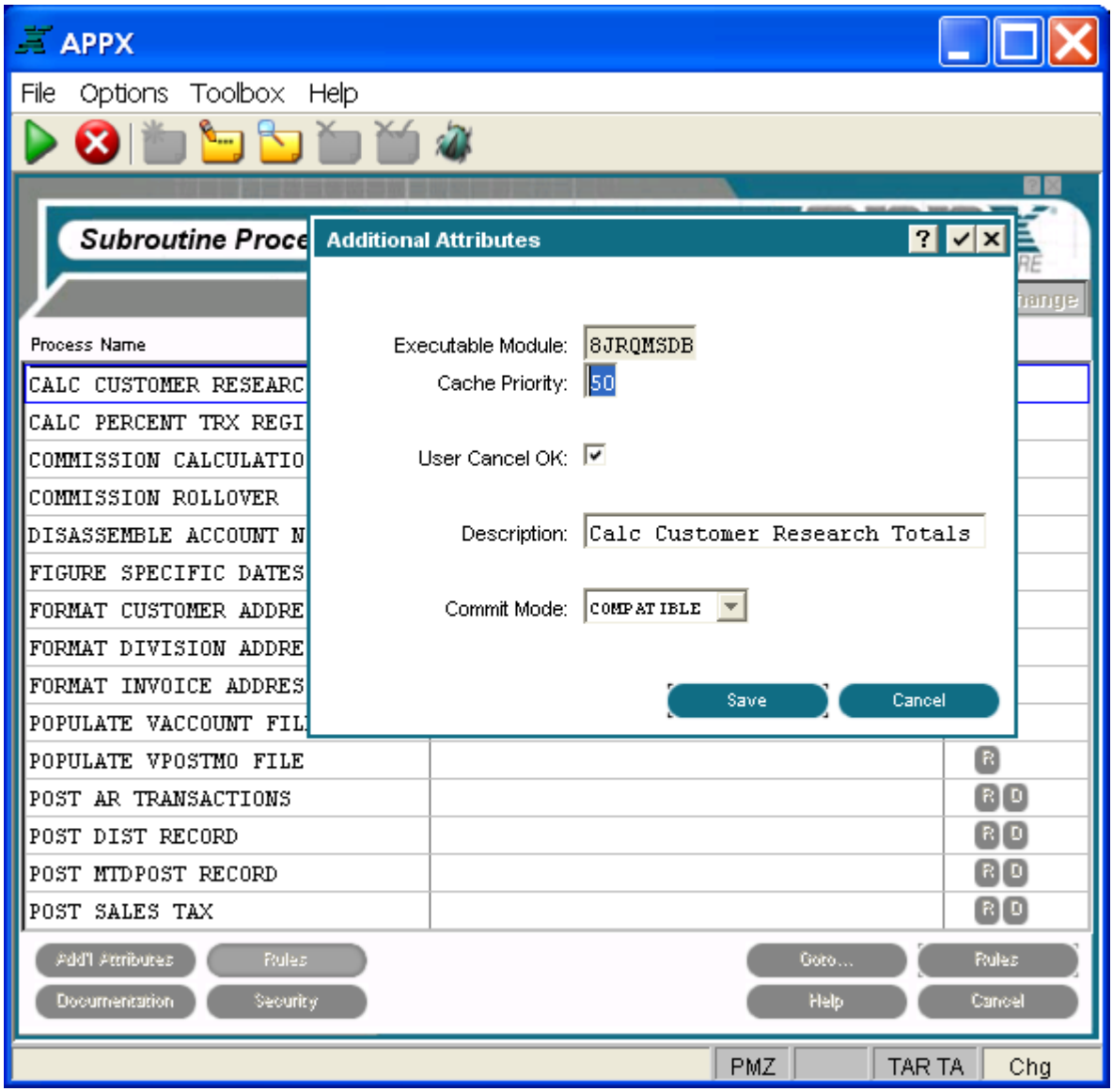


Figure 3-3-3. Sample Process Additional Attributes Overlay

The sample process Additional Attributes overlay contains the following fields common to all processes:

- ***Executable Module*** displays the name of the executable module (EM) that is automatically generated by APPX. APPX derives the file name from the date and time the currently selected process was added. The name consists of a unique string of eight alphanumeric characters and cannot be changed. If APPX finds it necessary to recreate the EM, the new EM retains the same file name. See Saving Executable Modules in Chapter 3-2 for more information.
- ***Cache Priority***, represented by two numeric characters, establishes the likelihood that the process will remain in memory after execution, relative to other processes that the user has also completed executing. A process with a cache priority of *99* is most likely to remain in memory. A process with a cache priority of *1* is least likely to remain in memory. A cache priority of *0* indicates that the process is never cached. The default is 50.

- ***User Cancel OK?*** contains either “yes” ☑ or “no” ☐ and determines whether or not a user can press the CANCEL option key to stop the execution of this process. The default is ‘yes”. You can conditionally control the user’s ability to select the CANCEL option with statements and the predefined field CANCEL OK. You can toggle between “yes” and “no” by clicking on the checkbox.
- ***Description*** is incorporated in the message that displays on the screen when the process is compiling. It is used to communicate with users since it provides a more descriptive and easily understood alternative to *Process Name*, which is the default if this field is left blank.
- ***Commit Mode*** controls the commit mode for a process and can be selected from a drop-down list of available options. *Commit Mode* determines those points in time when APPX will commit changes made to the tables stored within an RDBMS. It is only applicable in conjunction with an RDBMS; it has no effect with APPXIO. Also, not all File Managers support this feature. Although initially set here for the process, it can be added or overridden by use of predefined field COMMIT MODE. The following options are available:

COMPATIBLE .. Changes are committed as soon as all locks are freed. This provides maximum compatibility between APPXIO files and RDBMS tables.

AUTOCOMMIT . APPX will commit each modification as it is made.
***This value will be implemented in a future release.**

PCF CYCLE ..... APPX will commit modifications as each PCF record is processed.

END ..... APPX will only commit modifications at the end of the process family.

MANUAL ..... APPX will not automatically commit changes. You must use the COMMIT statement to do so.

Process Rules Option

When you select the RULES option from a Processes screen, an Event Points For Rules overlay is displayed. *Figure 3-3-4* shows the overlay applicable for a subroutine process. It contains the only event point that is common to such overlays for all processes.

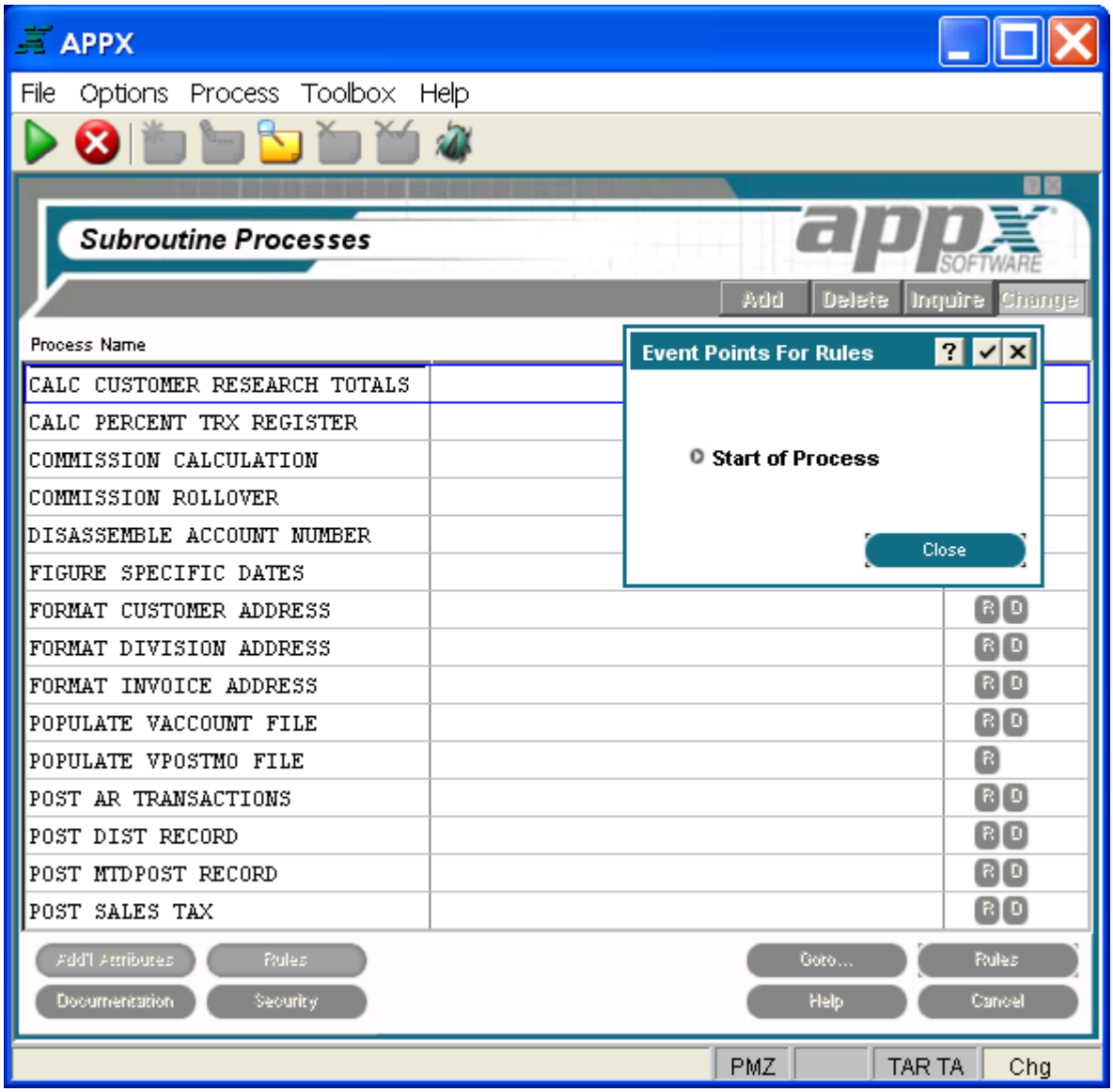


Figure 3-3-4. Sample Event Points for Rules Overlay

All types of processes have the Start of Process event point. Other event points are available depending on the type of the currently selected process. See the chapters later in this manual about specific types of processes for information about event points available from each type.

Start of Process

START OF PROCESS accesses the ILF editor for the *Start of Process* event point. Upon completion, you are returned back to the Event Points for Rules overlay.

Refer to *Chapter 3-16: Using Event Points* for more information about event points.

Process Documentation Option

APPX provides for two types of documentation: user documentation and technical documentation. User documentation is the basis for the online help facility. Technical documentation is the basis for printing technical documentation from the Application Design Documentation Menu.

When you select the DOCUMENTATION option from a Processes screen, a Documentation overlay is displayed similar to the sample shown in *Figure 3-3-5 below, with the User tab selected by default. The Technical tab has an identical format.* This presentation is the same for all processes.

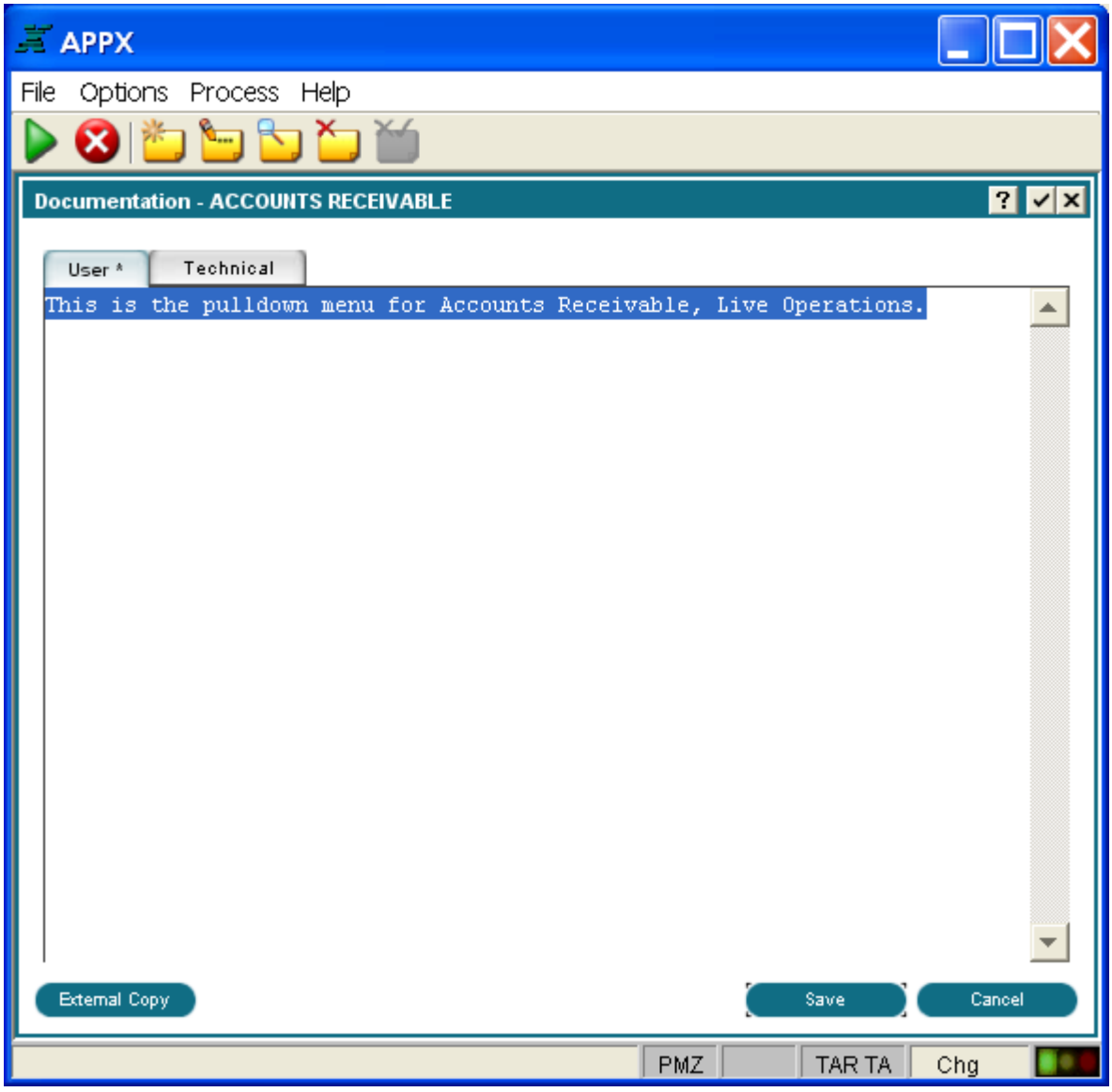


Figure 3-3-5. Process Documentation Overlay

Each Documentation screen contains the following field:

- *Text* provides up to 17, 77-character lines to describe a process, or a total of 1,264 alphanumeric characters. APPX's text editing functions, including word-wrap, text insertions, text display, and end

paragraph control can be used when entering this field. Refer to the *User Manual* for additional information on text editing capabilities.

A button is provided at the bottom of each Documentation screen for accessing the following option:

- EXTERNAL COPY - Provides for accessing and copying another documentation record into the *Text* field.

Process Security Option

When you select the SECURITY option from a Processes screen, a Security overlay is displayed. *Figure 3-3-6* shows the overlay applicable for all processes except inputs. Input processes provide for mode security codes as well as execute security.

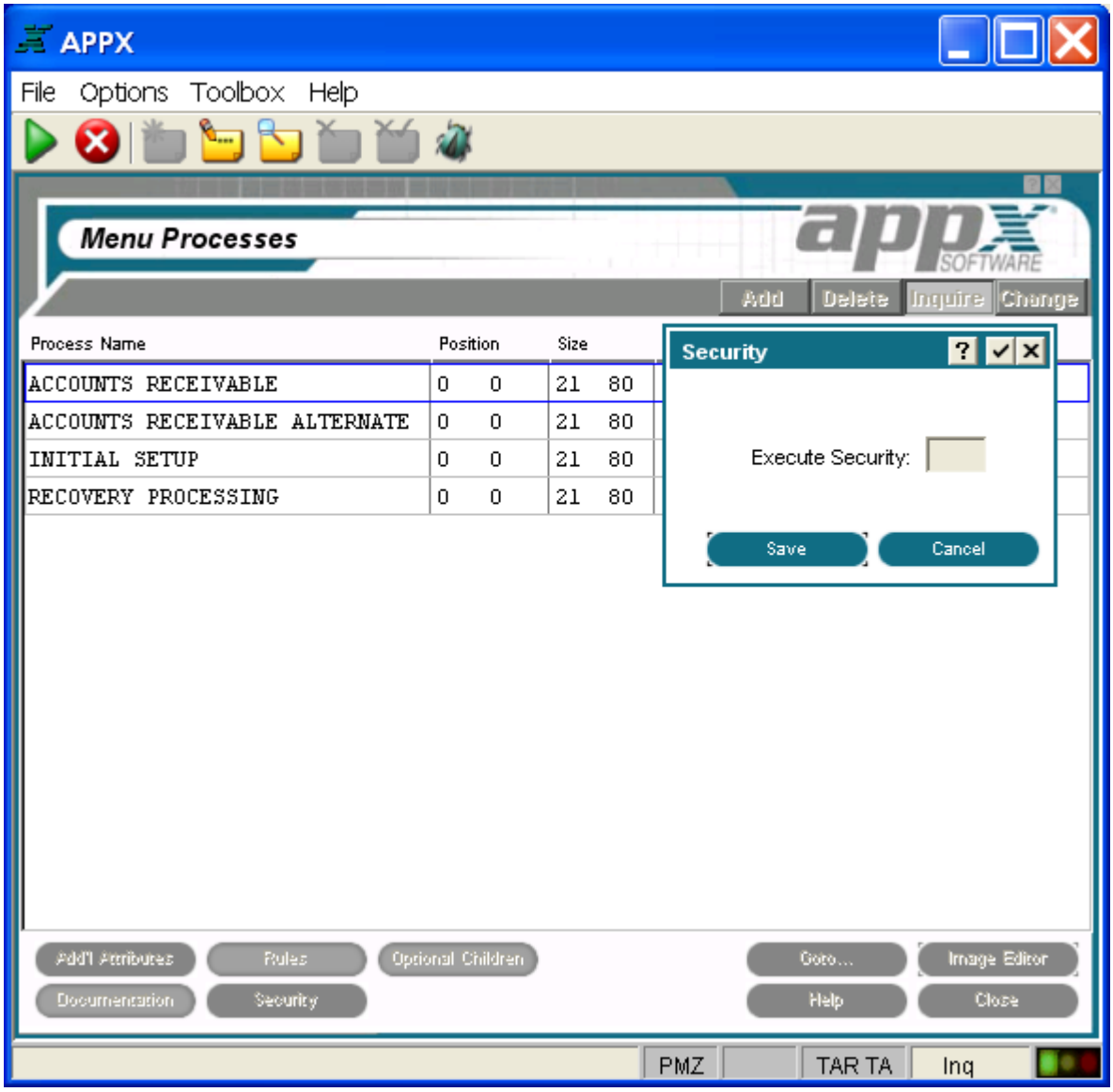


Figure 3-3-6. Sample Process Security Overlay

The sample process Security overlay contains the following field common to all processes:

- ***Execute Security*** identifies the APPX security code to which a user must have rights in order to run the process (menu, job, input, etc.). The security code must be four numeric digits. If blank, there are no restrictions.

Refer to Security in Chapter 3-2 for an overview of APPX security.

Process Toolbox Pulldown Menu

When you select **Toolbox** from the menu bar on a Processes screen, the process Toolbox Pulldown Menu appears as shown in *Figure 3-3-7* below. The same toolbox applies to all processes. Note that each tool can also be accessed by selecting the corresponding icon on the Processes screen toolbar.

Process Toolbox Pulldown menu

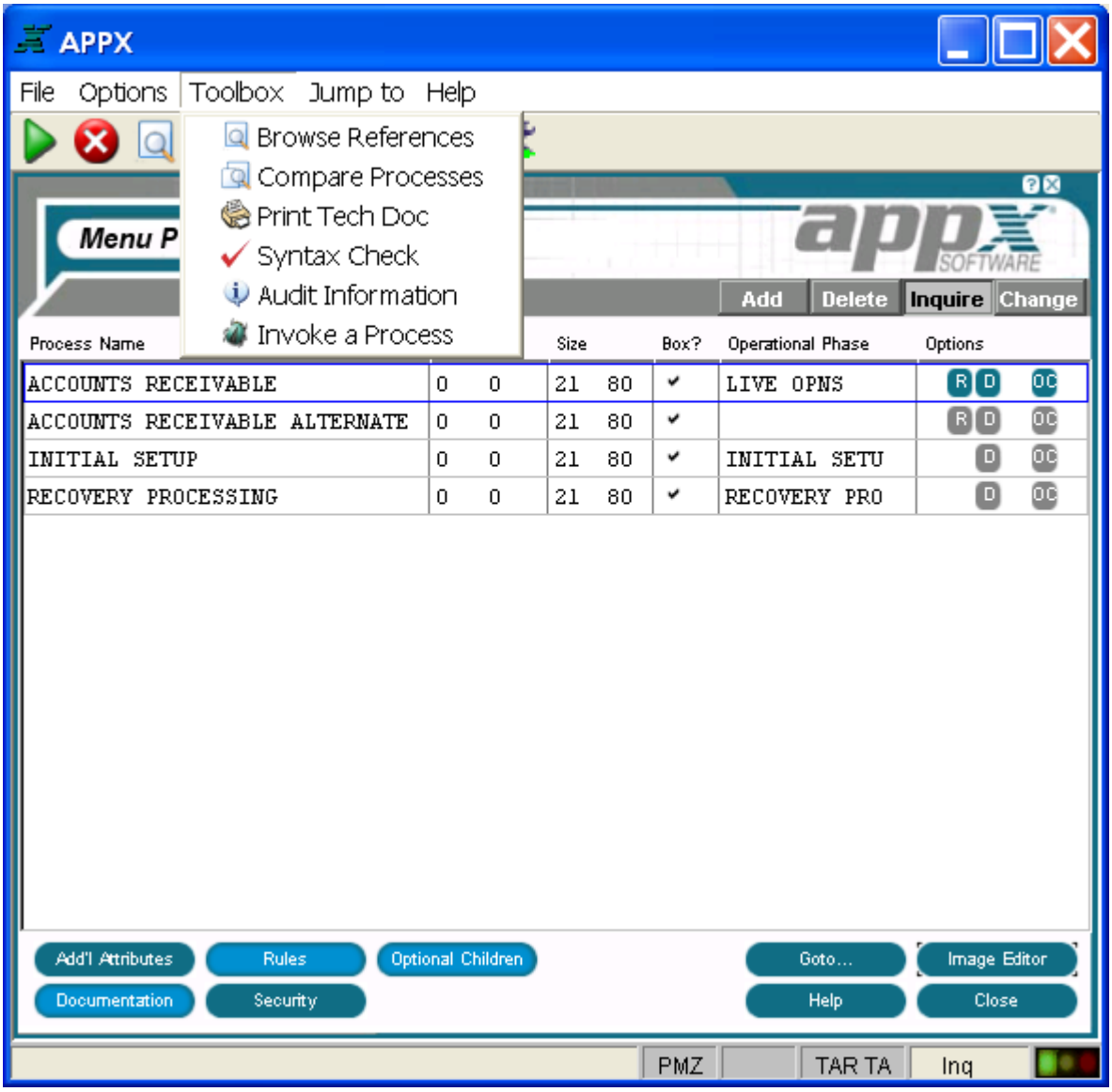


Figure 3-3-7

The process Toolbox Pulldown Menu offers the following options for all processes:

Browse References

The BROWSE REFERENCES option runs a modified version of "Cross Reference – All" and produces a list of design components utilized by the currently selected process. The results can be displayed interactively where links allow you to see the context of the references, edit the referenced component, and drill down to the child process. This option can be used to identify, and optionally edit, the file level event points utilized by this process.

Compare Processes

The COMPARE PROCESSES option compares the currently selected process to another process that you identify and produces a report listing any differences. You can also opt to browse the results interactively.

Print Technical Documentation

The PRINT TECHNICAL DOCUMENTATION option allows you to print the technical documentation for the currently selected process to the screen or to hard copy.

Syntax Check

The SYNTAX CHECK option allows you to create the Executable Module for the currently selected process without running the process. If the Executable Module already exists and is still valid, this option is ignored if invoked. Note that this option (`OPTION 97`), like any Toolbox option with a value greater than 90, is available for use directly from any screen in Application Design without going to the Toolbox.

Audit Information

The AUDIT INFORMATION option allows you to see who created the currently selected process and who changed it most recently. It also provides you with the date that the process was created and the date that it was last changed. Note that this option (`OPTION 98`), like any Toolbox option with a value greater than 90, is available for use directly from any screen in Application Design without going to the Toolbox. See Audit Information Overlay for an example of the overlay.

Invoke a Process

The INVOKE A PROCESS option is available to facilitate testing. When you select this option, APPX defaults to the currently selected process but you may override the defaults to invoke any process. Note that this option, like any Toolbox option with a value greater than 90, is available for use directly from any screen in Application Design without going to the Toolbox. See Invoke a Process Overlay for details on using this option.

Audit Information Overlay

The Audit Information Overlay is the same when you select the AUDIT INFORMATION option for an object anywhere in Application Design. *Figure 3-3-8* below shows the overlay called from a menu process. Audit information shows who created the currently selected object and who changed it most recently. It also shows the date that the object was created and the date that it was last changed.

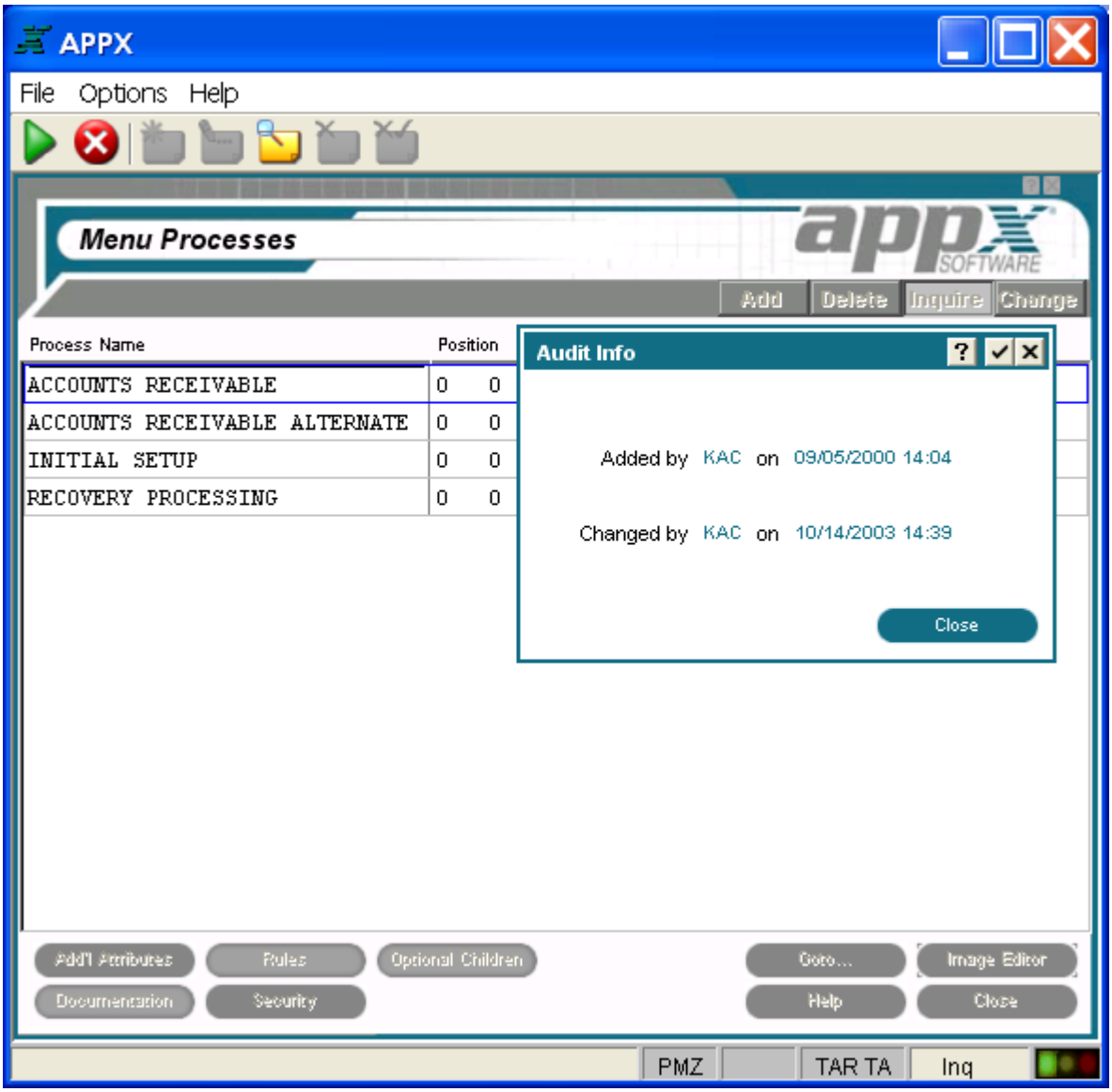


Figure 3-3-8. Audit Information Overlay

Invoke a Process Overlay

The Invoke a Process overlay is the same when you select the INVOKE A PROCESS option anywhere in Application Design, although the level of defaults may vary. *Figure 3-3-9* below shows the overlay called from a menu process which defaults the currently selected process.

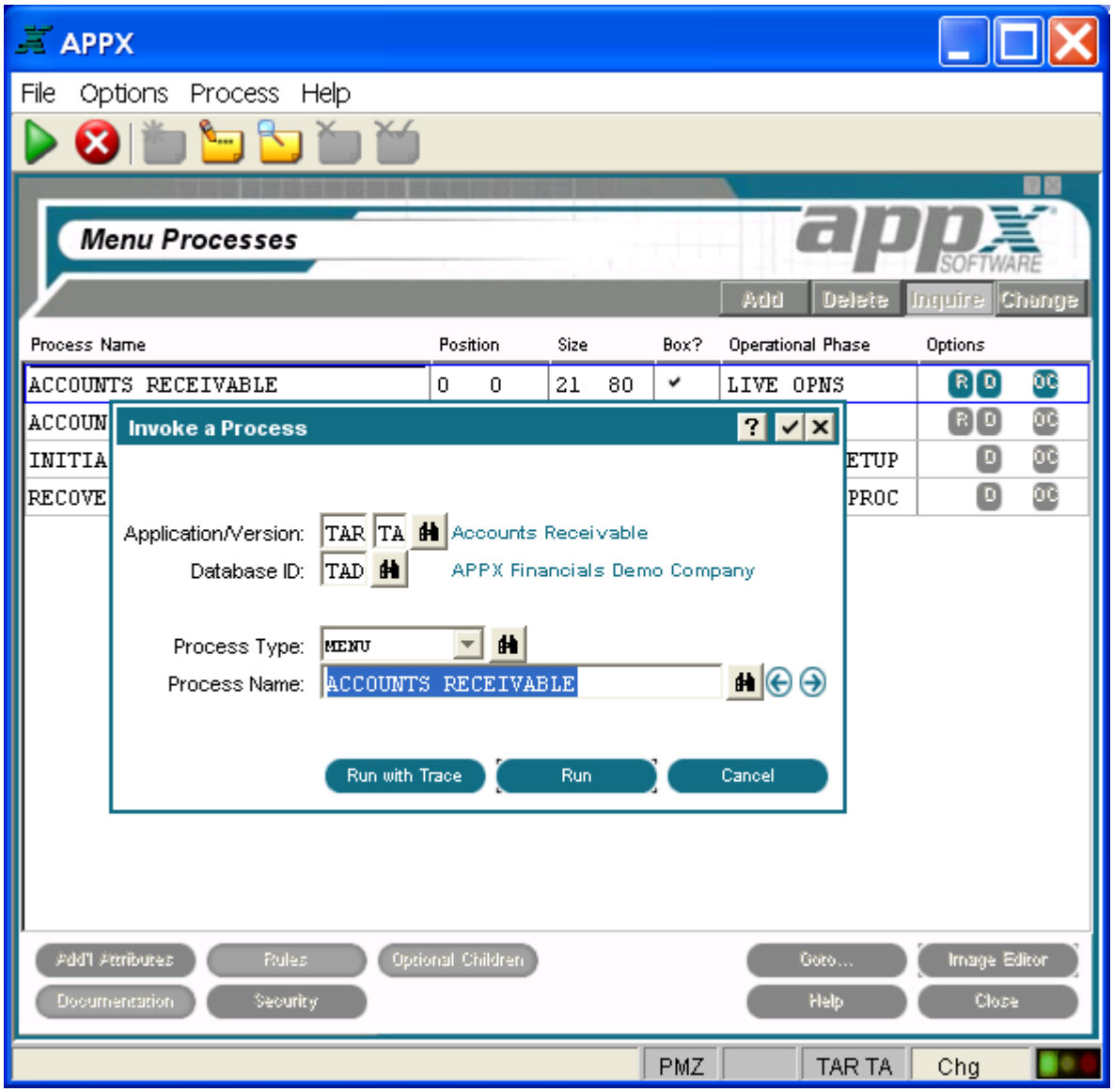


Figure 3-3-9. Invoke a Process Overlay

You can change the default values on the Invoke a Process overlay to invoke any process name or type that you choose. This overlay also contains the following option:

Run with Trace

The RUN WITH TRACE option supplies you with a trace log that itemizes everything APPX has done during the execution of the selected process when the selected process is finished. Each line of the trace log shows a sequence number and a description of the logged event. The display of the trace log permits you to browse the trace in a hierarchical fashion. Events that have additional detail are identified by a *greater than* sign (>) next to the sequence number. To view the additional detail, place the cursor on the record and select enter. To return to a higher level, select end. Selecting the CANCEL icon from the toolbar exits the trace display from any level.

Use the INQUIRE option to switch to key entry at any time; this allows you to reposition yourself to a particular sequence number. While viewing a trace log, you can also use the hidden *Option 1* to view the entire trace in chronological (sequence number) order.

Process Jump to Pulldown Menu

When you select **Jump to** from the menu bar on a Processes screen, the process Jump to Pulldown Menu appears as shown in *Figure 3-3-10* below. The same toolbox applies to all processes. Note that each tool can also be accessed by selecting the corresponding icon on the Processes screen toolbar.

Process Jump to Pulldown menu

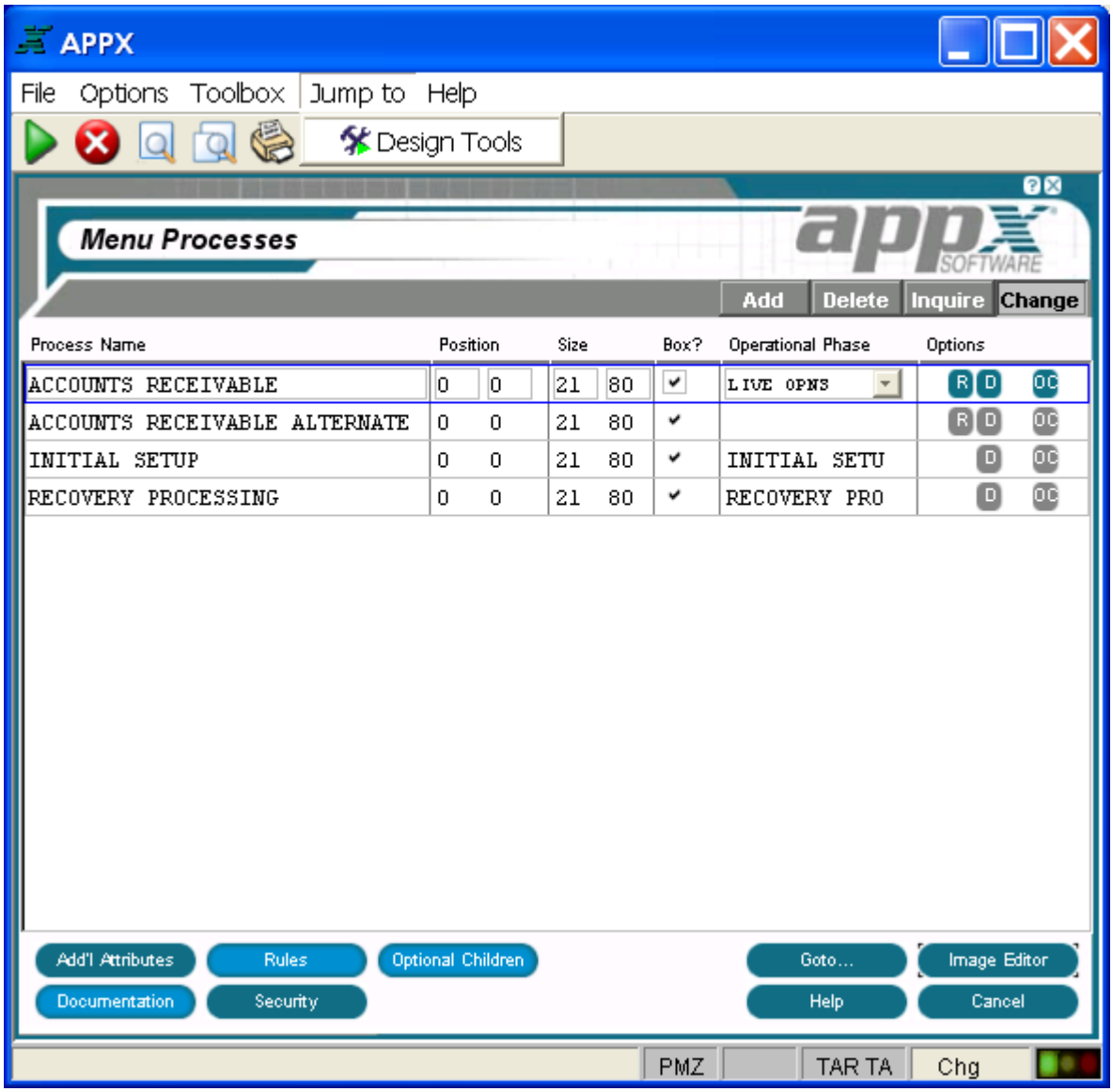


Figure 3-3-10

The process Jump to Pulldown Menu offers the following option for all processes:

Design Tools

The DESIGN TOOLS option calls the Application Design Tools Menu shown in *Figure 3-3-11* below. This is the same menu that is displayed when you select the Tools tab from the Application Design screen. When you exit from this screen, you will return to where you were when you selected the DESIGN TOOLS option.

Application Design Tools menu

Figure 3-3-11

CHAPTER 3-4

Menu Processes

Introduction to Menu Processes

Menu processes provide an on-screen display from which to execute selected processes. For example, a screen that lists the reports available in an application and from which you select a specific report is a menu process.

In a typical application, you begin process definition by designing and specifying the main menu that is presented when a user first enters the application. You then proceed to identify what happens when each of the options on the menu is selected. APPX allows you to define up to 249 options (choices) on each menu. An option may invoke another menu, and the number of "nested" menu levels that you can specify is unlimited.

Menu Process Specifications

Menu processes specifications establish the general attributes of a menu process. The Menu Processes screen, shown in *Figure 3-4-1*, is displayed when you select MENUS from the Application Design Specifications Menu.

Menu Processes screen

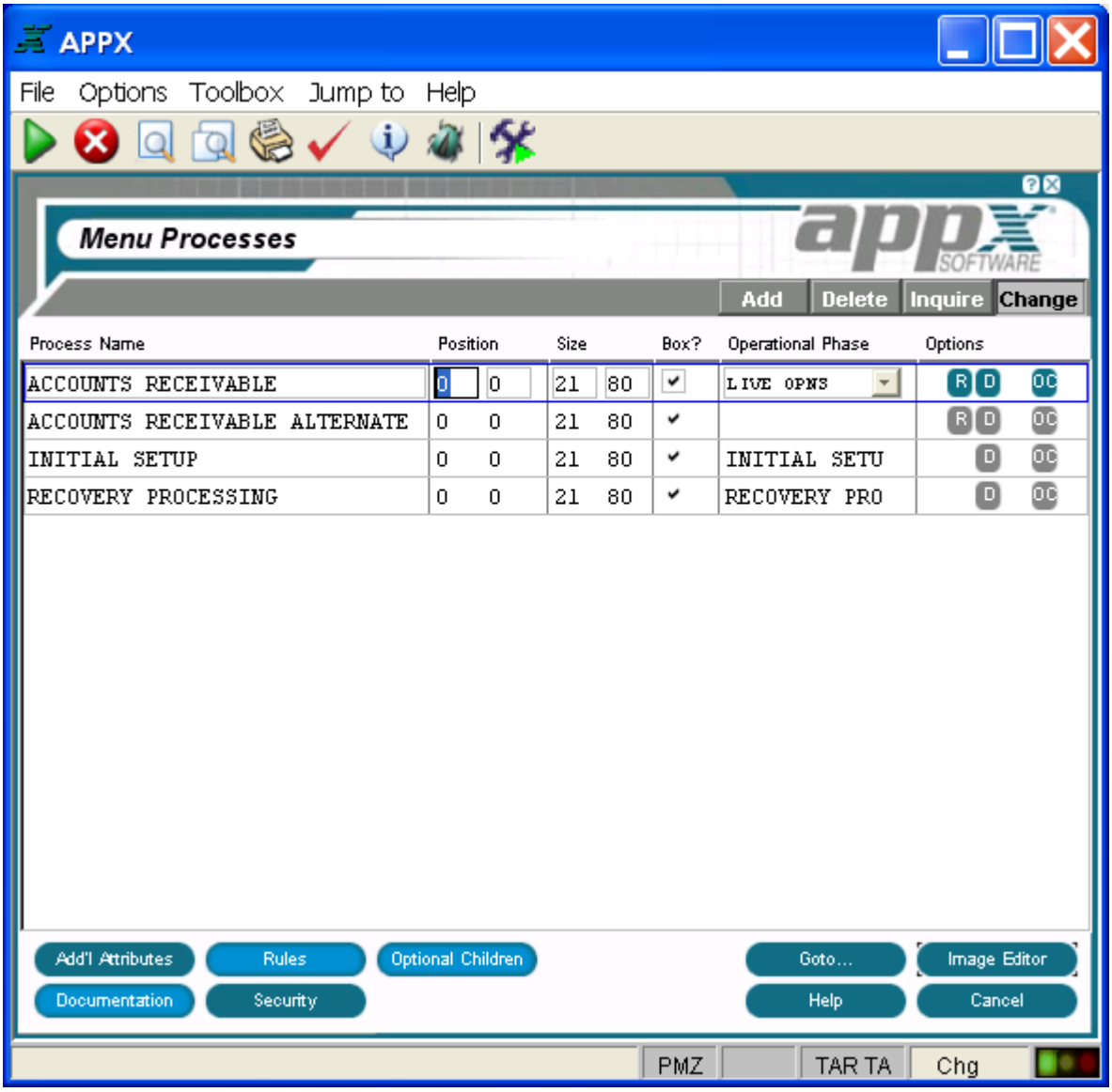


Figure 3-4-1

The Menu Processes screen contains the following fields:

- ***Process Name*** identifies a specific process and consists of up to 30 uppercase alphanumeric characters that must be unique within type and application.
- ***Position*** fields contain up to three numeric characters each and determine the placement of the rows and columns of a frame relative to the rows and columns on the screen display. Row entries designate positioning relative to the top of the screen. Column entries designate positioning relative to the left of the screen. The default is *1, 1*. See Frame/Image Positioning in *Chapter 3-2* for more information.

- ***Size*** fields, which consist of three numeric characters each, designate the number of rows and columns that comprise the displayed frame. This size may be greater than the size of the screen if panning capabilities are desired. The default is *21, 80*.
- ***Box?***, represented by "yes" ☑ or "no" ☐, indicates whether or not APPX draws a box around an image before it displays to a user. The border does not consume rows or columns within the space that is defined for the frame/image, but does consume space on the screen. The default is "no" and the value can be toggled by clicking the checkbox.
- ***Operational Phase*** allows you to enter the appropriate operational phase, if the currently selected menu is the main menu for one of the three operational phases. Valid entries are:

INITIAL SETUP

LIVE OPNS

RECOVERY PROC

- ***Options*** are indicators that are present whenever the options they denote have been defined. Options applicable to this screen are AA (Additional Attributes), R (Rules), D (Documentation), S (Security), and OC (Optional Children).

Buttons are provided at the bottom of each Menu Processes screen for accessing the following options:

- ADDITIONAL ATTRIBUTES - Same as the Process Additional Attributes Option described in Chapter 3-3.
- RULES - See the Menu Process Rules Option for details.
- DOCUMENTATION - Same as the Process Documentation Option described in Chapter 3-3.
- SECURITY - Same as the Process Security Option described in Chapter 3-3.
- OPTIONAL CHILDREN - See the Menu Process Optional Children Option for details.
- GOTO... - Allows you to identify and retrieve another record by optionally entering a full or partial key.
- IMAGE EDITOR - See Menu Image in the Image Editor for details. This is the default option.
- HELP - Allows you to view online help text for a selected item.
- CLOSE or CANCEL - In Inquire mode, CLOSE allows you to end the current process. Otherwise, CANCEL allows you to end the current process without saving changes, if any, to the current record.

In addition, the menu bar provides the following pulldown menus:

- **Toolbox** - Same as the Process Toolbox Pulldown Menu described in Chapter 3-3.
- **Jump to** - Same as the Process Jump to Pulldown Menu described in Chapter 3-3.

Menu Process Rules Option

When you select the RULES option from the Menu Processes screen, the menu processes Event Points for Rules overlay is displayed as shown in *Figure 3-4-2 below*. This overlay provides options to access the ILF Editor for APPX event points. If statements are defined in an event point, the respective text on the screen is highlighted.

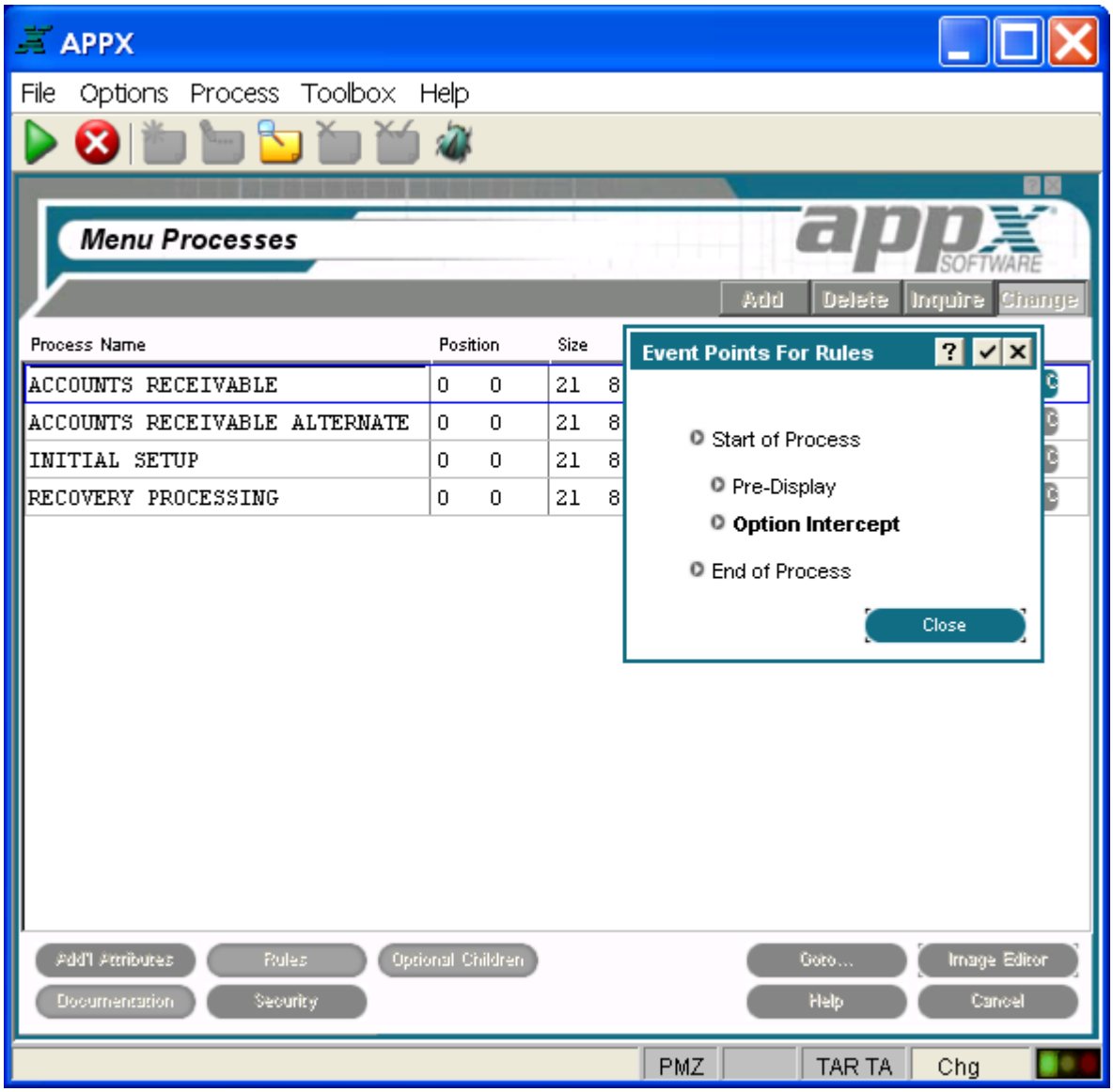


Figure 3-4-2. Menu Processes Event Points for Rules Overlay

The Menu Processes Event Points overlay contains the following options:

Start of Process

START OF PROCESS accesses the ILF editor for the *Start of Process* event point. Upon completion, you are returned to the process Event Points overlay.

Pre-Display

PRE-DISPLAY accesses the ILF editor for the *Pre-Display* event point. Upon completion, you are returned to the process Event Points overlay.

Option Intercept

OPTION INTERCEPT accesses the ILF editor for the *Option Intercept* event point. Upon completion, you are returned to the process Event Points overlay.

End of Process

END OF PROCESS accesses the ILF editor for the *End of Process* event point. Upon completion, you are returned to the process Event Points overlay.

Refer to Chapter 3-16: Using Event Points for more information about event points.

Menu Process Optional Children Option

When you select the OPTIONAL CHILDREN option from the Menu Processes screen, the menu processes Optional Children screen is displayed as shown in *Figure 3-4-3 below.* This screen defines attributes to govern the discretionary invocation of another APPX process from the menu process image. These specifications include the name of the child process, the invocation type, and whether or not various activities at the child level affect the parent process.

Menu Processes Optional Children screen

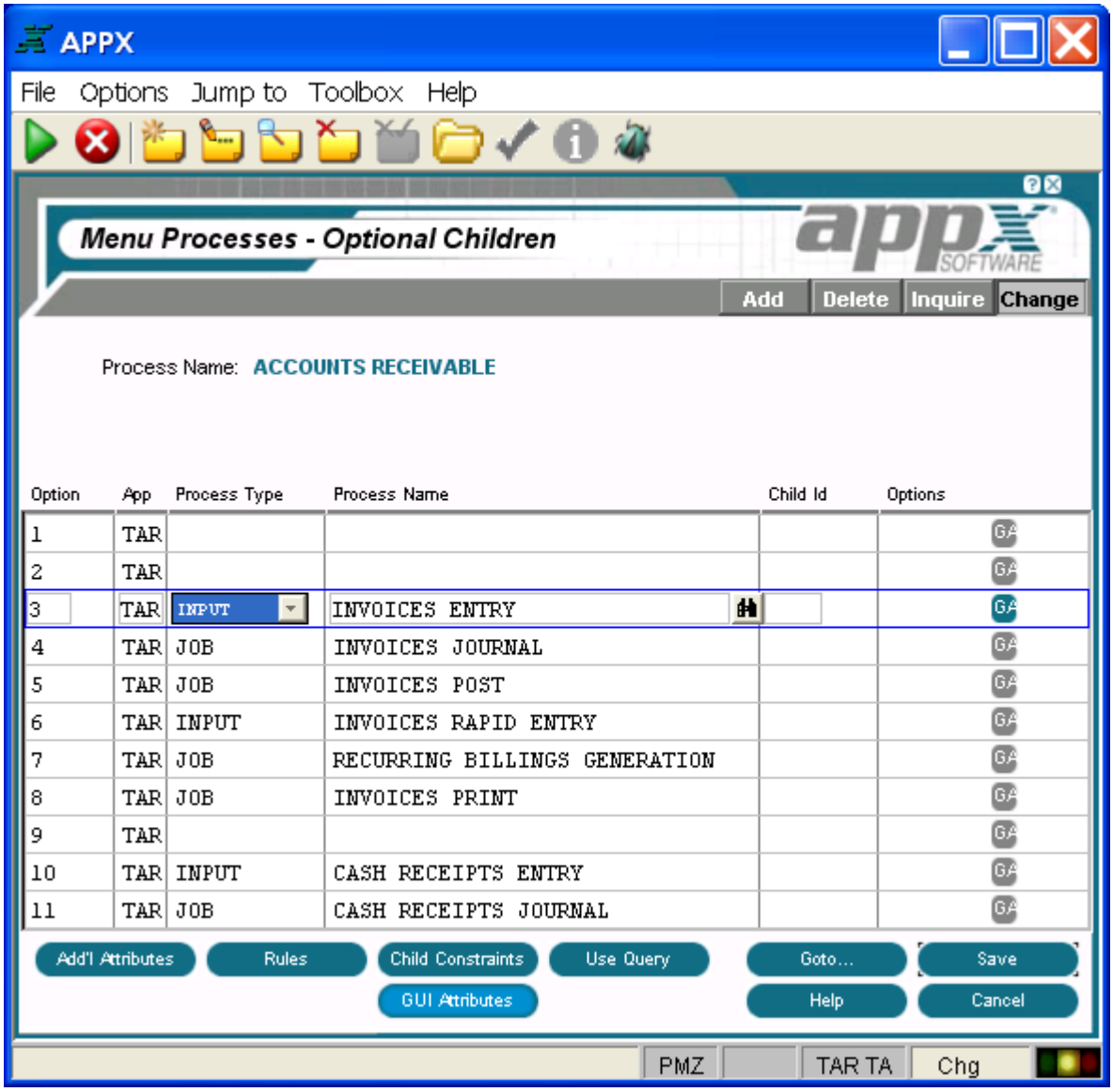


Figure 3-4-3

Defining optional children is described in depth in Chapter 3-13: Automatic and Optional Children. Upon completion, you are returned to the Menu Processes screen.

Menu Image in the Image Editor

After you complete specifications for a menu process on the Menu Processes screen, you automatically enter the Image Editor when you press RETURN or select the IMAGE EDITOR option. The Image Editor, APPX's screen painting facility, is the facility used to format menus, inputs, outputs, and inquiry screens. As you design the menu screen, you are seeing the image exactly as it will appear to your users. *Figure 3-4-4 below shows a menu image being edited in the Image Editor.*

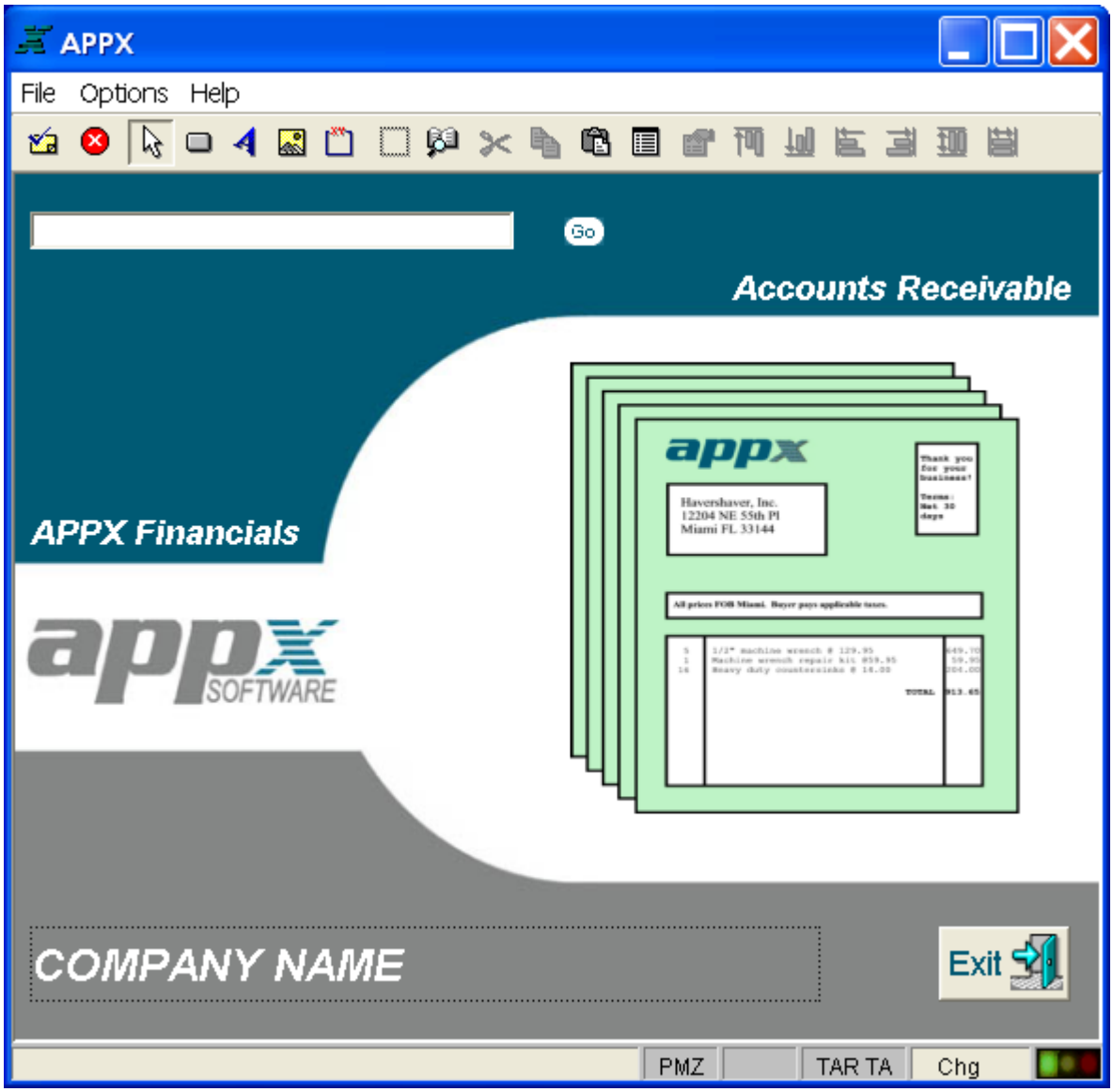


Figure 3-4-4. Menu Image in the Image Editor

The standard features of the Image Editor can be employed in character mode using only a keyboard. However, you need a mouse to take advantage of the GUI features within the Image Editor. GUI tools provide designers with the ability to create screen images complete with pictures, buttons, icons, wallpapers, and other graphical features. Another GUI tool is a data palette offering drag and drop selection and placement of data fields.

Basic functionality of the Image Editor is fully explored in Chapter 3-14: Using the Image Editor and a complete discussion of its GUI features is presented in Chapter 3-15: Using GUI Features of the Image Editor.

CHAPTER 3-5

Job Processes

Introduction to Job Processes

Job processes are used to control the sequencing of other related processes.

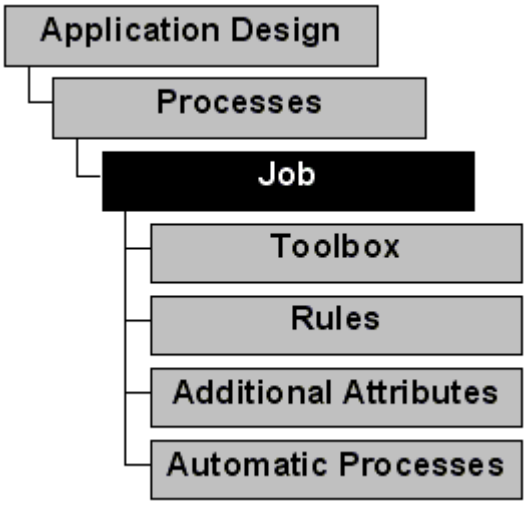


Figure 3-5-1. Job Process Definition

Job Process Specifications

Job process specifications establish the general attributes of an job process, including the “In Progress” message. The Job Processes screen, displayed when you select JOB from the Process Management overlay, is shown in *Figure 3-5-2*.

Job Processes screen

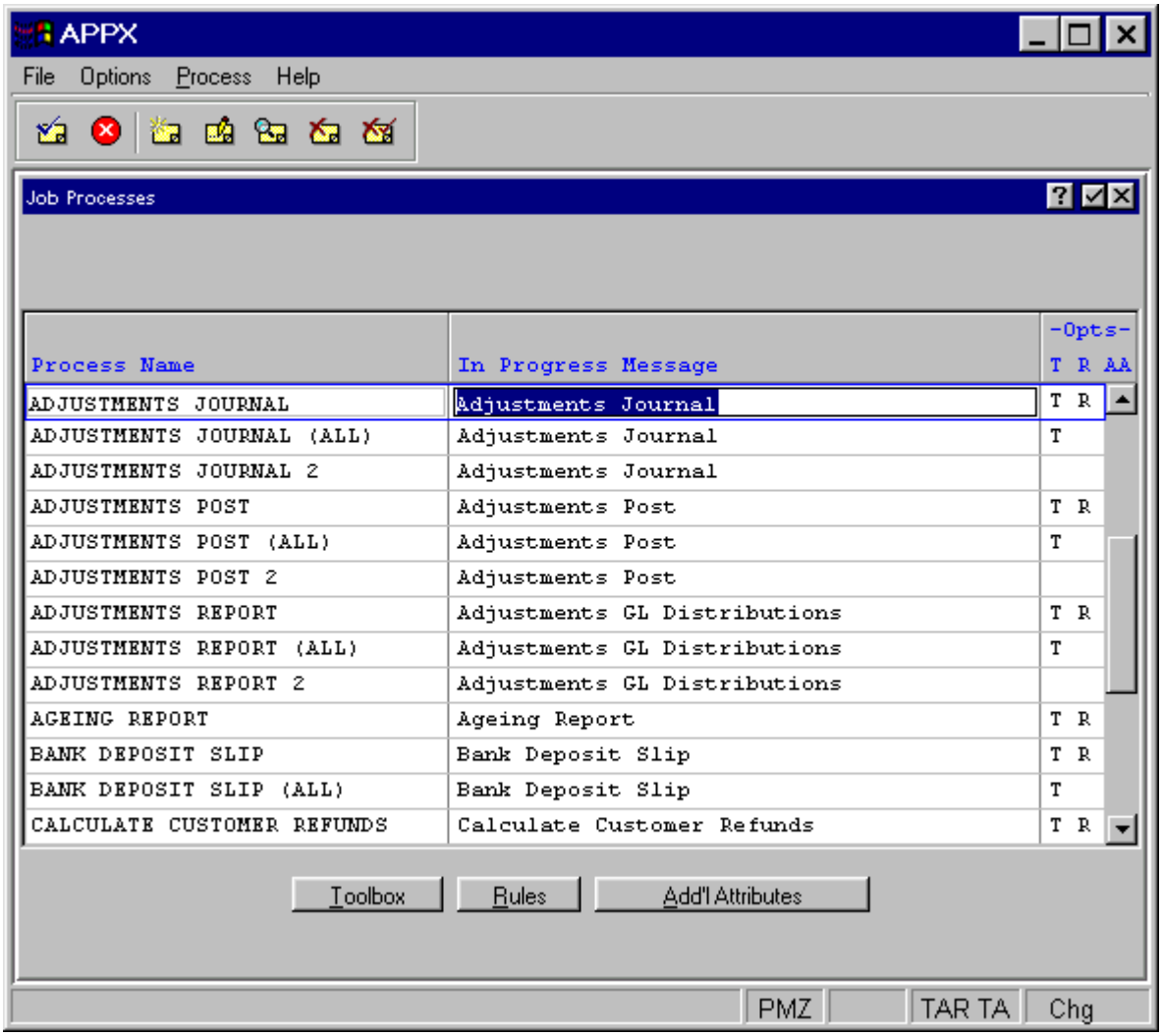


Figure 3-5-2

The Job Processes screen contains the following fields:

Process Name

Identifies a specific process and must be unique within type and application.

In Progress Message

Field provides text for the message that displays on the screen while the job process executes in the foreground.

Opts

Are indicators that are present whenever documentation (T), rules (R), or additional attributes (AA) have been defined for this process.

The Job Processes screen contains the following options as described in the corresponding sections:

- Toolbox
- Rules
- Additional Attributes

Job Process Toolbox Option

The Toolbox overlay, which appears on the screen when you invoke the TOOLBOX option from the Job Processes screen contains a number of features such as Documentation Management and Security Management that will help you design a job process. The options listed in the Toolbox overlay, shown in *Figure 3-5-3,* are described briefly below. See *Chapter 3-3, Process Definition* for a more information of these options.

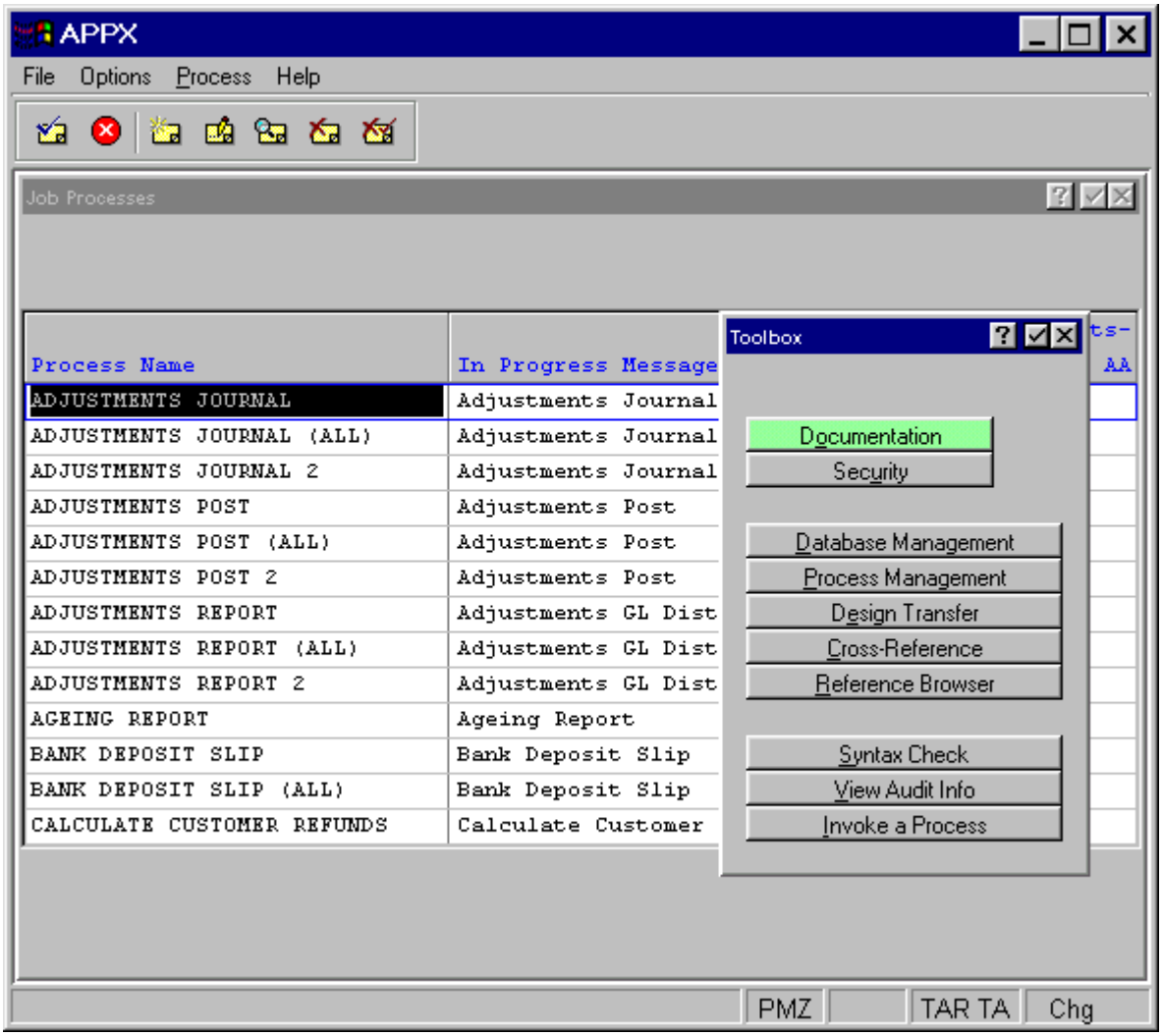


Figure 3-5-3. Job Processes Toolbox Overlay

Documentation Option

When you select the DOCUMENTATION option from the Toolbox overlay, you will first see the Documentation overlay and must select either user or technical documentation. User documentation is the basis for the online help facility. Technical documentation is the basis for hard-copy technical documentation.

Security Option

The Job Processes SECURITY overlay allows you to identify the APPX security code to which a user must have rights in order to run the job process. The security code must be four numeric characters. If blank, there are no restrictions.

Database Management Option

The DATABASE MANAGEMENT option allows you to add new files or maintain existing ones without leaving job process design.

Process Management Option

The PROCESS MANAGEMENT option allows you to add other types of processes or maintain existing ones without leaving process design for job processes.

Design Transfer Option

The DESIGN TRANSFER option allows you to copy the attributes from a process that you have already designed and apply it to a new process.

Cross Reference

The Cross Reference option enables you to determine the location and frequency of a selected domain, file, field, application ID, process or ILF statement within a specific database.

Reference Browser Option

The REFERENCE BROWSER option runs a modified version of "Cross Reference – All" and produces a list of design components utilized by the currently selected job process. The results can be displayed interactively where links allow you to see the context of the references, edit the referenced component and drill down to the child process. This option can be used to identify, and optionally edit, the file level event points utilized by this job process.

Syntax Check Option

This feature allows you to create the Executable Module for the current job process without actually running it. If the Executable Module for the job process is valid this option is ignored when invoked. Note that this option (`OPTION 97`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

View Audit Info

This feature allows you to see who created the currently selected job process and who changed it most recently. It also gives the date of creation and the date it was last changed. Note that this option (`OPTION 98`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Invoke a Process

This feature is available to facilitate testing. It allows you to invoke any process; the currently selected job process using the Test Database defined in system administration is the default. Note that this option (`OPTION 99`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Job Process Rules Option

Selecting the RULES option accesses the job processes Event Points For Rules overlay which allows you to select from two options that access the ILF Editor for APPX event points. The overlay is shown in *Figure 3-5-4.* If statements were defined in the event point, the text on the screen is highlighted. Refer to Chapter 3-16: Using Event Points for more information. The Job Processes Event Points overlay contains the following options:

Start of Process

START OF PROCESS accesses the ILF editor for the *Start of Process* event point. Upon completion, you are returned to the process Event Points overlay.

End of Process

END OF PROCESS accesses the ILF editor for the *End of Process* event point. Upon completion, you are returned to the process Event Points overlay.

Refer to Chapter 3-16: Using Event Points for more information.

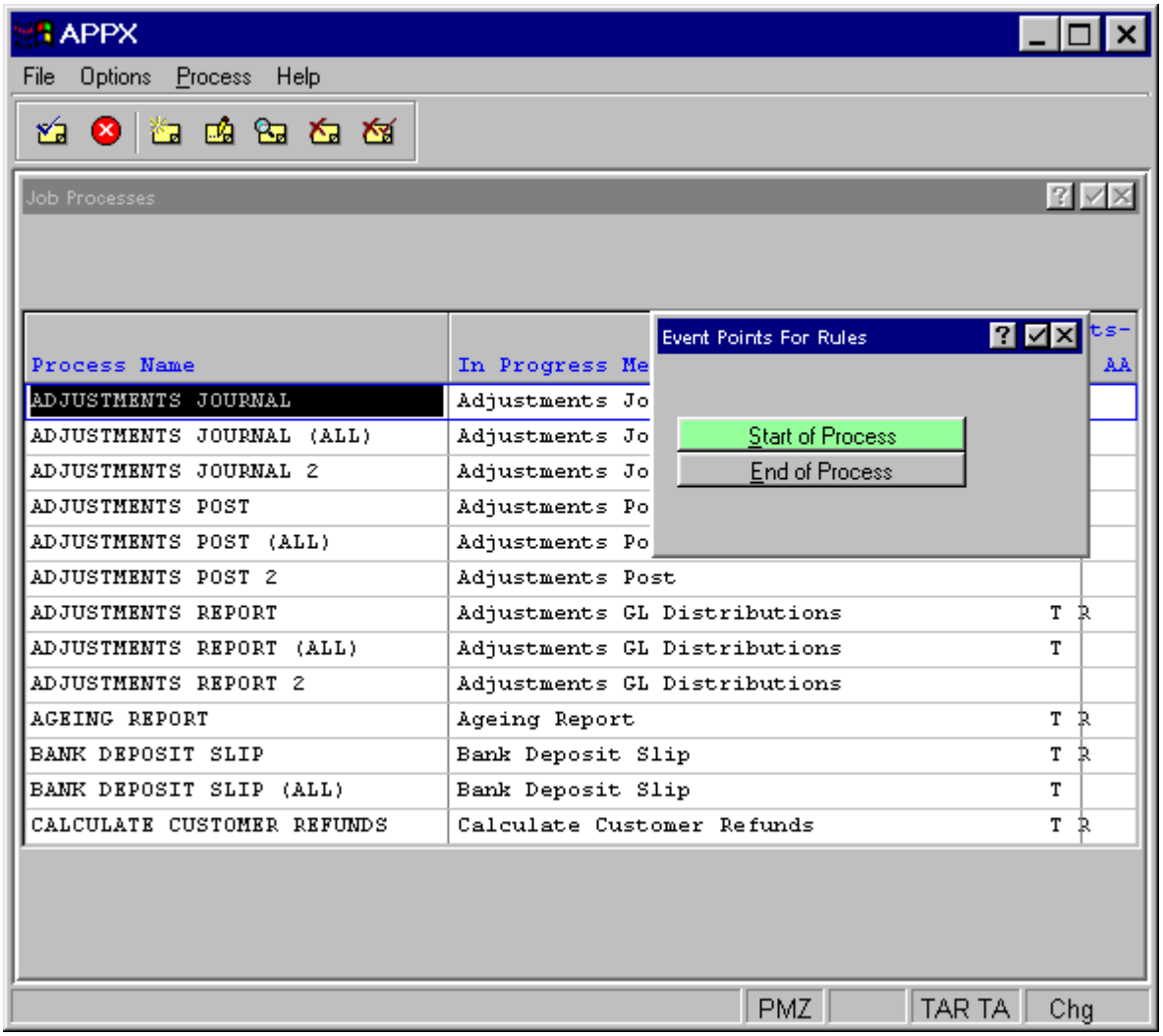


Figure 3-5-4. Job Processes Event Points For Rules Overlay

Job Process Additional Attributes Option

ADDITIONAL ATTRIBUTES accesses the job processes Additional Attributes overlay where you will find additional parameters. The job processes Additional Attributes overlay, shown in *Figure 3-5-5,* contains the following fields.

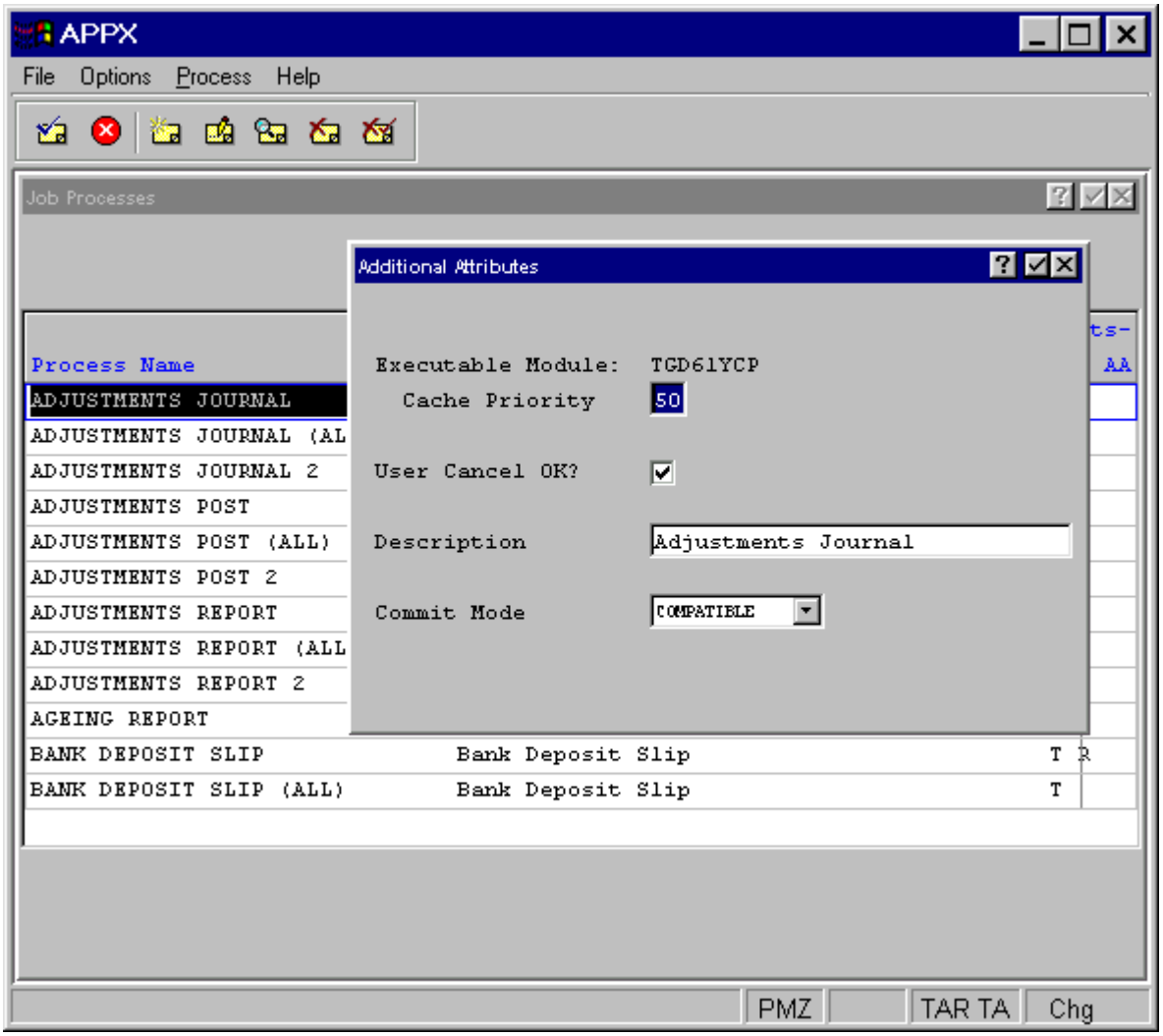


Figure 3-5-5. Job Processes Additional Attributes Overlay

Executable Module

Displays the name of the executable module (EM) that is automatically generated by APPX. APPX derives the file name from the date and time the currently selected job process was added. The name consists of a unique string of eight alphanumeric characters and cannot be changed. If APPX finds it necessary to recreate the EM, the new EM retains the same file name. See *Chapter 3-3, Process Definition*, for more information.

Cache Priority

Establishes the likelihood that the currently selected job process will remain in memory after execution, relative to other processes that the user has also completed executing. A process with a cache priority of *99* is most likely to remain in memory. A process with a cache priority of *1* is least likely to remain in memory. A cache priority of *0* indicates that the process is never cached. The default is *50*. See *Chapter 3-3, Process Definition*, for more information.

User Cancel OK?

Contains either “yes” ☑ or “no” ☐ and determines whether or not a user can press the Cancel option key to stop the execution of the currently selected job process. The default is “yes” but can be toggled to “no” by clicking the checkbox. You can conditionally control the user’s ability to select the CANCEL option with statements and the predefined field CANCEL OK.

Description

Is incorporated in the message that displays on the screen when the currently selected job process is compiling. It is used to communicate with users since it provides a more descriptive and easily understood alternative to *Process Name*, which is the default if this field is left blank. See *Chapter 3-3, Process Definition*, for more information.

Commit Mode

Controls the commit mode for a process and can be selected from a drop-down list of available options. *Commit Mode* determines those points in time when APPX will commit changes made to the tables stored within an RDBMS. It is only applicable in conjunction with an RDBMS; it has no effect with APPXIO. Also, not all File Managers support this feature. Although initially set here for the process, it can be added or overridden by use of predefined field COMMIT MODE. The following options are available:

COMPATIBLE. Changes are committed as soon as all locks are freed. This provides maximum compatibility between APPXIO files and RDBMS tables.

AUTOCOMMIT*. APPX will commit each modification as it is made.
***This value will be implemented in a future release.**

PCF CYCLE. APPX will commit modifications as each PCF record is processed.

END. APPX will only commit modifications at the end of the process family.

MANUAL. APPX will not automatically commit changes. You must use the COMMIT statement to do so.

Job Process Automatic Child Processes

A job consists of a series of *Automatic Child Processes* that are executed sequentially. These processes are defined using the overlay shown in *Figure 3-5-6.* For more information regarding Automatic Child Processes and the options available on this overlay, see *Chapter 3-13 Automatic and Optional Child Processes.*

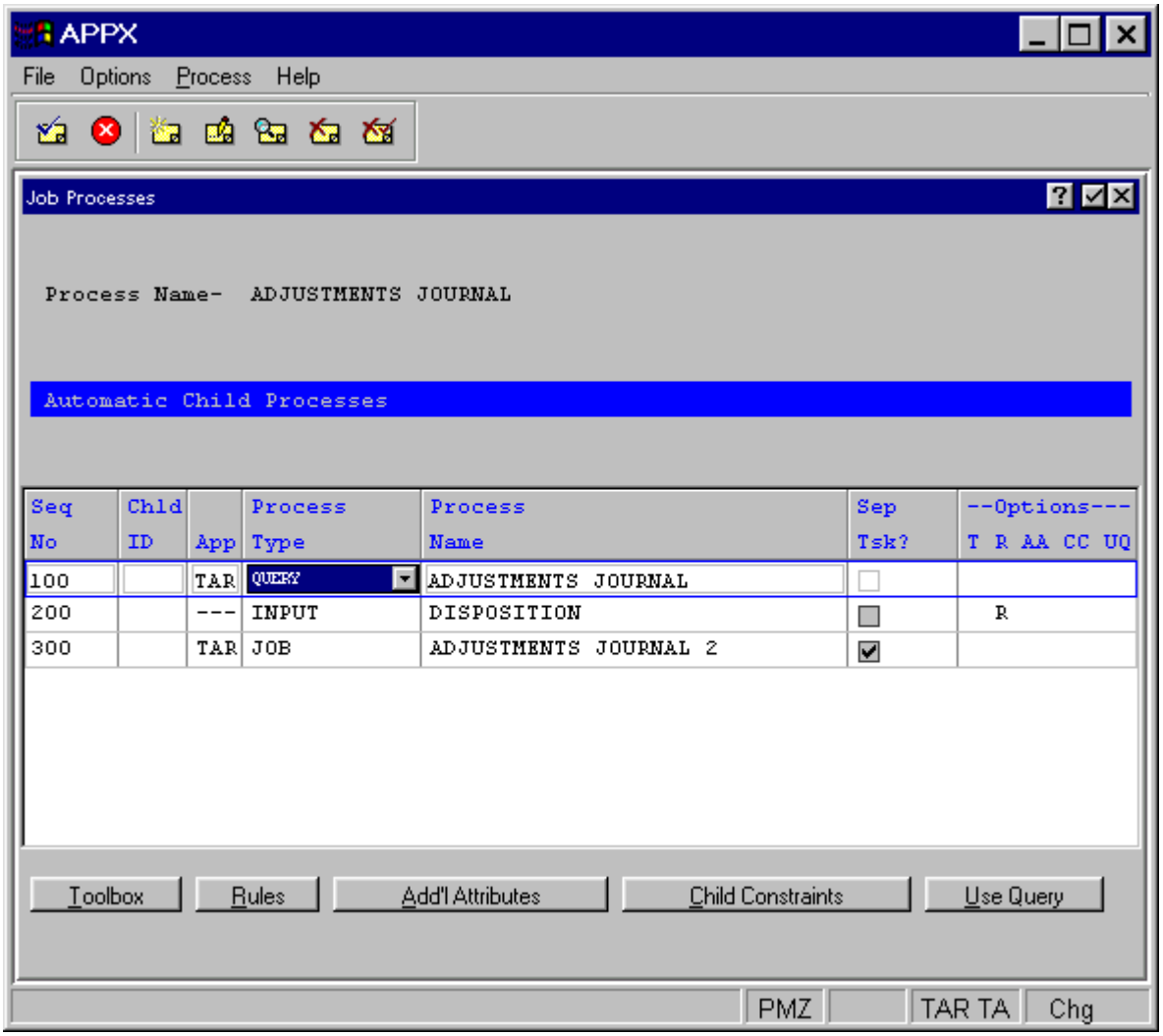


Figure 3-5-6. Automatic Child Processes Overlay For Jobs

CHAPTER 3-6

Input Processes

Introduction to Input Processes

Input processes permit you to enter or maintain data in one or more files from designer-formatted screens and overlays. A sales order entry function and a customer file maintenance function are both examples of input processes.

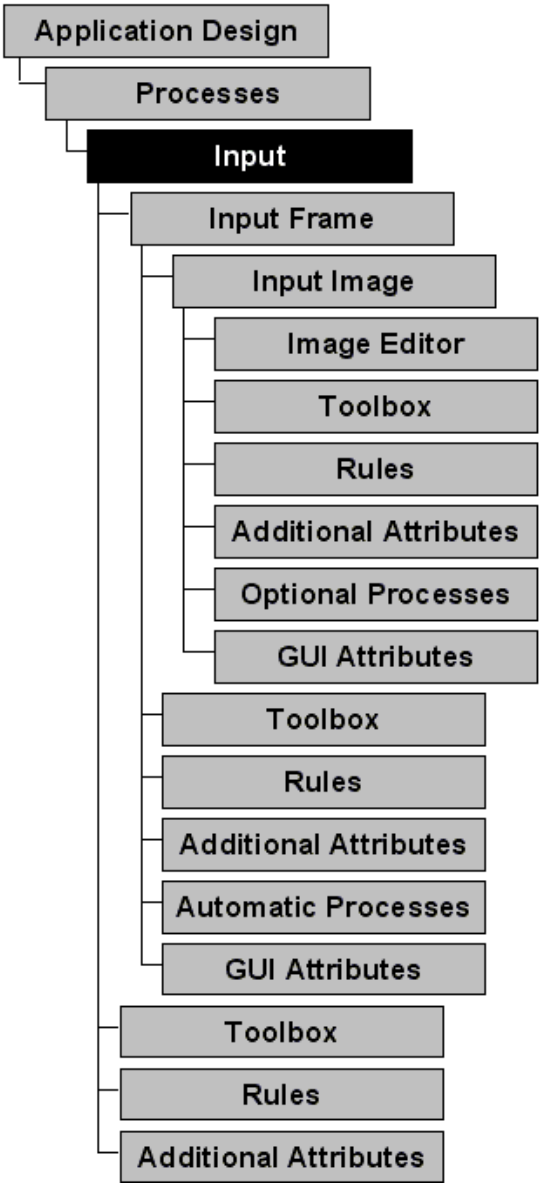


Figure 3-6-1. Input Process Definition

Input Process Specifications

Input process specifications establish the general attributes of an input process, including a process control file and operational characteristics. The Input Processes screen, shown in *Figure 3-6-2 below,* displays when you select INPUTS from the Application Design Specifications Menu.

Input Processes screen

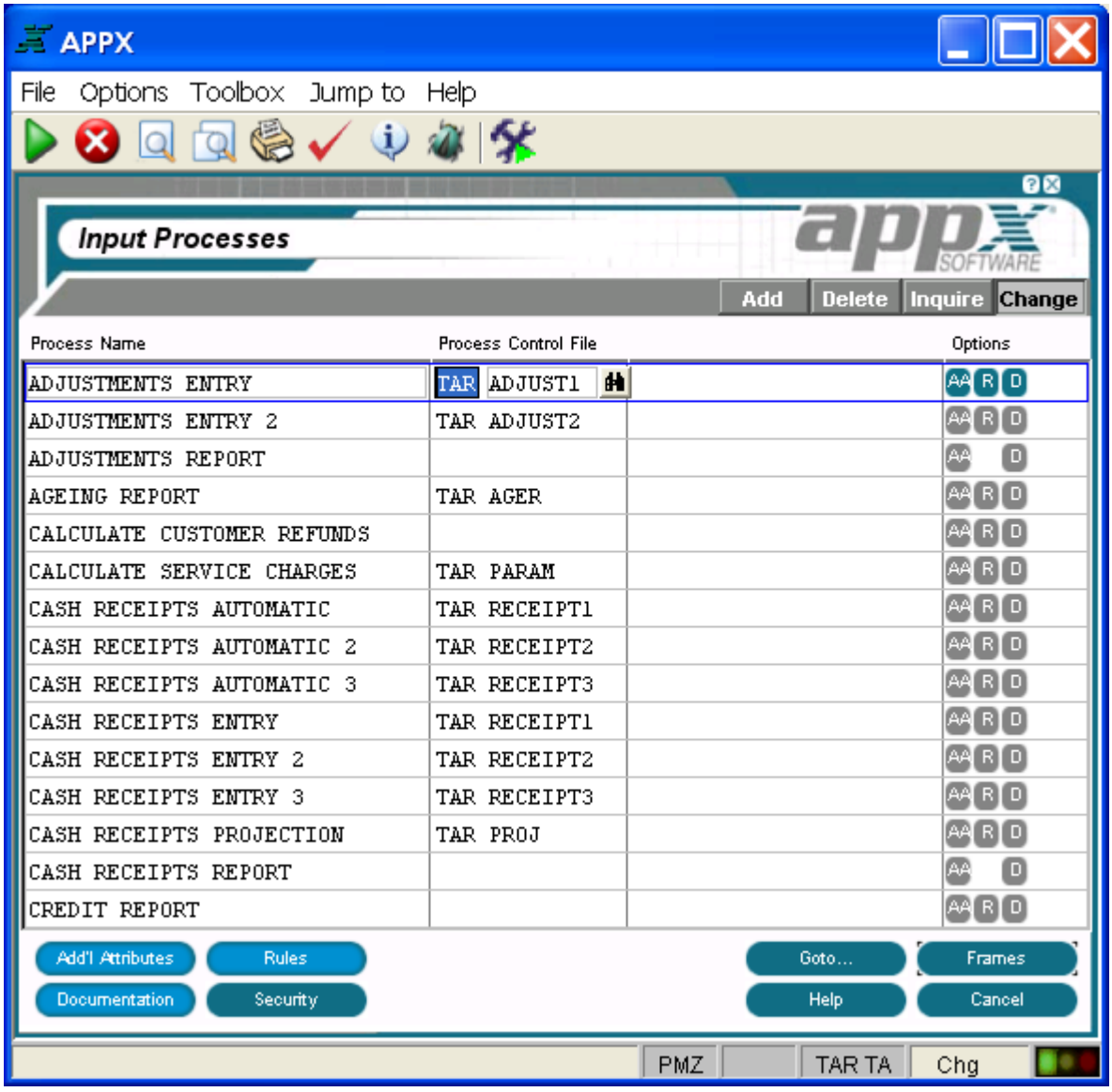


Figure 3-6-2

The Input Processes screen contains the following fields:

- *Process Name* identifies a specific process and consists of up to 30 uppercase alphanumeric characters that must be unique within type and application.

- ***Process Control File*** is *optional* for input processes. If present, it identifies a file by application ID and file name that is to be automatically managed by APPX during the execution of this process. APPX automatically reads, writes, rewrites, and deletes records in the process control file (PCF) during input process execution (and provides for updating the child PCF for any automatic children). If the PCF is left blank, the input process will execute once and then automatically end (as opposed to executing once for every record).
- ***Options*** are indicators that are present whenever the options they denote have been defined. Options applicable to this screen are AA (Additional Attributes), R (Rules), D (Documentation), and S (Security).

Buttons are provided at the bottom of each Input Processes screen for accessing the following options:

- ADDITIONAL ATTRIBUTES - See the Input Process Additional Attributes Option for details.
- RULES - See the Input Process Rules Option for details.
- DOCUMENTATION - Same as the Process Documentation Option described in Chapter 3-3.
- SECURITY - Same as the Process Security Option described in Chapter 3-3.
- GOTO... - Allows you to identify and retrieve another record by optionally entering a full or partial key.
- FRAMES - See Input Frame Specifications for details. This is the default option.
- HELP - Allows you to view online help text for a selected item.
- CLOSE or CANCEL - In Inquire mode, CLOSE allows you to end the current process. Otherwise, CANCEL allows you to end the current process without saving changes, if any, to the current record.

In addition, the menu bar provides the following pulldown menus:

- **Toolbox** - Same as the Process Toolbox Pulldown Menu described in Chapter 3-3.
- **Jump to** - Same as the Process Jump to Pulldown Menu described in Chapter 3-3.

Input Process Additional Attributes Option

When you select the ADDITIONAL ATTRIBUTES *option* from the Input Processes screen, the Input Processes Additional Attributes overlay is displayed as shown in *Figure 3-6-3 below. This overlay permits the definition of more properties affecting the operation of the input process.*

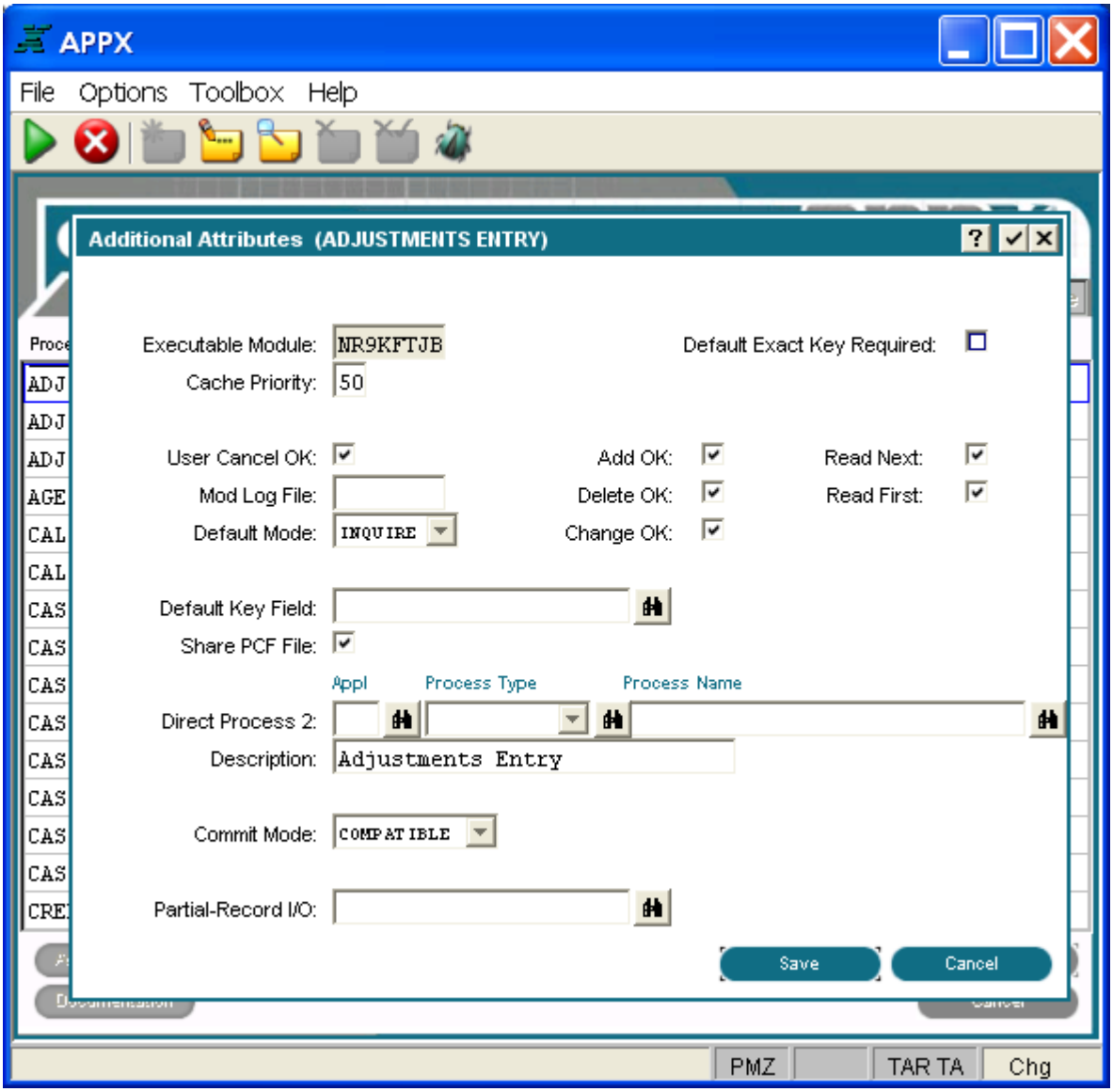


Figure 3-6-3. Input Processes Additional Attributes Overlay

The Additional Attributes overlay for Input processes contains the following fields:

- ***Executable Module*** displays the name of the executable module (EM) that is automatically generated by APPX. APPX derives the file name from the date and time the currently selected process was added. The name consists of a unique string of eight alphanumeric characters and cannot be changed. If APPX finds it necessary to recreate the EM, the new EM retains the same file name. See Saving Executable Modules in Chapter 3-2 for more information.
- ***Default Exact Key Required?*** contains either “yes” ☑ or “no” ☐ and establishes the default value for the *Exact Key Required?* option. This option controls whether or not a user is required to enter a complete and exact key value to retrieve a record. The designer may override this default value with statements. If the field is set to “no”, users are allowed to enter partial key values. APPX retrieves the record with the next highest key value if the entered value is not on file. The default is "no".

- ***Cache Priority***, represented by two numeric characters, establishes the likelihood that the process will remain in memory after execution, relative to other processes that the user has also completed executing. A process with a cache priority of *99* is most likely to remain in memory. A process with a cache priority of *1* is least likely to remain in memory. A cache priority of *0* indicates that the process is never cached. The default is 50.
- ***User Cancel OK?*** contains either "yes" ☑ or "no" ☐ and determines whether or not a user can press the CANCEL option key to stop the execution of this process. The default is 'yes". You can conditionally control the user's ability to select the CANCEL option with statements and the predefined field CANCEL OK. You can toggle between "yes" and "no" by clicking on the checkbox.
- ***Add OK?*** and ***Delete OK?*** and ***Change OK?*** control whether or not a user can access this process in add, delete and change modes. Each of these fields requires "yes" ☑ or "no" ☐ and the values can be toggled by clicking the respective checkbox. The default is "yes".
- ***Read Next?*** controls whether the user can retrieve the next record in the file by simply selecting ENTER. If "no" ☐, the user must identify the next record to be retrieved. The default is "yes".
- ***Read First?*** controls whether the first record of the PCF file is presented to the user upon entry into this process. "Yes" ☑ indicates it will and "no" ☐ indicates it will not.
- ***Mod Log File*** generates a modification log when this input process is run when you enter up to eight alphanumeric characters for the print file name. If ☐, no modification log is printed. A modification log lists all additions, deletions, and changes to the process control file. During initial setup, a modification log is never printed. During recovery processing, a modification log is always printed, regardless of whether a *Mod Log Print File* name is specified or not. If no name is specified, *MODLOG* is used. The default is ☐.
- ***Default Mode*** determines the initial mode in which the process invokes. The *Default Mode* options are add, delete, inquire, change. The default is *Change.* If you want to override this specification, there are two methods. See Start of Process in *Chapter 3-16: Using Event Points* to explain how to override the default mode using statements. The other method would be to select "yes" in the *Override Default Mode* field on the Additional Attributes overlay when defining this process as an optional child from menu, input, or job processes.
- ***Default Key Field*** designates the specific keypath that is initially presented to a user invoking this process. This specification is overridden by the default keypath that is defined for the child process invocation of the process, if applicable. The keypath may also be dynamically specified via ILF code. APPX determines the appropriate keypath at runtime. If no explicit specification of keypath is found, the primary key is used.
- ***Share PCF File*?** contains either "yes" ☑ or "no" ☐ and determines whether or not the file can be shared. If "no", the file is opened for exclusive use. Note that care should be exercised in opening a file for exclusive use to avoid unintended consequences. For instance, if another process attempts to read this file while it is in exclusive access, the read will not wait for the lock to be released. Instead, the read will fail and act in accordance with the fail type specified in the read statement. The default is "yes" but can be toggled to "no" by clicking the checkbox.

- ***Direct Process 2*** designates the Application ID, Process Type, and Process Name of the process to be executed when the application-specific Direct Process option is selected from this input process. The Application ID contains three alphanumeric characters; the Process Type contains ten alphanumeric characters, and the Process Name contains up to 30 alphanumeric characters. Refer to the *User Manual* for additional information on the use of Direct Process options.
- ***Description*** is incorporated in the message that displays on the screen when the process is compiling. It is used to communicate with users since it provides a more descriptive and easily understood alternative to *Process Name*, which is the default if this field is left blank.
- ***Commit Mode*** controls the commit mode for a process and can be selected from a drop-down list of available options. *Commit Mode* determines those points in time when APPX will commit changes made to the tables stored within an RDBMS. It is only applicable in conjunction with an RDBMS; it has no effect with APPXIO. Also, not all File Managers support this feature. Although initially set here for the process, it can be added or overridden by use of predefined field COMMIT MODE. The following options are available:

COMPATIBLE .. Changes are committed as soon as all locks are freed. This provides maximum compatibility between APPXIO files and RDBMS tables.

AUTOCOMMIT . APPX will commit each modification as it is made.
***This value will be implemented in a future release.**

PCF CYCLE ..... APPX will commit modifications as each PCF record is processed.

END ..... APPX will only commit modifications at the end of the process family.

MANUAL ..... APPX will not automatically commit changes. You must use the COMMIT statement to do so.

- ***Partial-Record I/O*** is used to specify a field from the PCF file so that instead of operating on the entire PCF record, APPX operates on the designated field only. If the field is a group header, then I/O will be performed on all members of the group. Utilizing partial-record I/O can greatly improve performance when the PCF is a table in an RDBMS. This is especially true when the table contains many columns, but relatively few are being utilized by the process. The SCAN option is available for field selection.

Input Process Rules Option

When you select the RULES *option* from the Input Processes screen, the Input Processes Event Points for Rules overlay is displayed as shown in *Figure 3-6-4 below*. This overlay provides options to access the ILF Editor for APPX event points. If statements are defined in an event point, the respective text on the screen is highlighted.

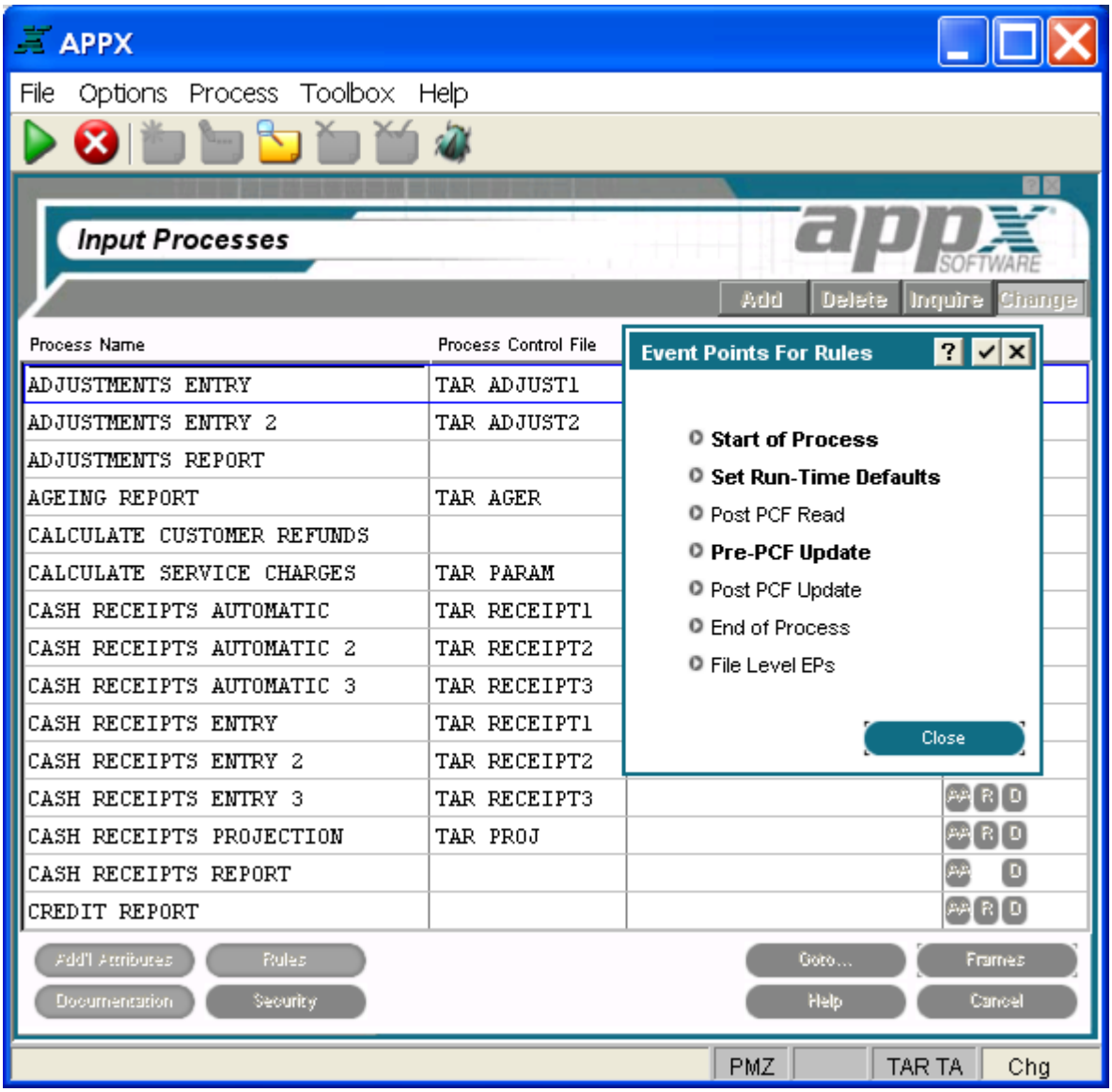


Figure 3-6-4. Input Processes Event Points for Rules Overlay

The Input Processes Event Points for Rules overlay contains the following options:

Start of Process

START OF PROCESS accesses the ILF editor for the *Start of Process* event point. Upon completion, you are returned to the Event Points overlay.

Set Run-Time Defaults

SET RUN-TIME DEFAULTS accesses the ILF editor for the *Set Run-Time Defaults* event point. Upon completion, you are returned to the Event Points overlay.

Post PCF Read

POST PCF READ accesses the ILF editor for the *Post PCF Read* event point. Upon completion, you are returned to the Event Points overlay.

Pre-PCF Update

PRE-PCF UPDATE accesses the ILF editor for the *Pre-PCF Update* event point. Upon completion, you are returned to the Event Points overlay.

Post PCF Update

POST PCF UPDATE accesses the ILF editor for the *Post PCF Update* event point. Upon completion, you are returned to the Event Points overlay.

End of Process

END OF PROCESS accesses the ILF editor for the *End of Process* event point. Upon completion, you are returned to the Event Points overlay.

File Level EPs

FILE LEVEL EPs provides access to the File Level Event Points for the PCF file, as shown in Figure 3-6-5 below. These event points are not directly associated with the process but with the PCF file. See the Files Rules Option *in Chapter 2-4* for more information on using these event points.

Refer to Chapter 3-16: Using Event Points for more information about event points.

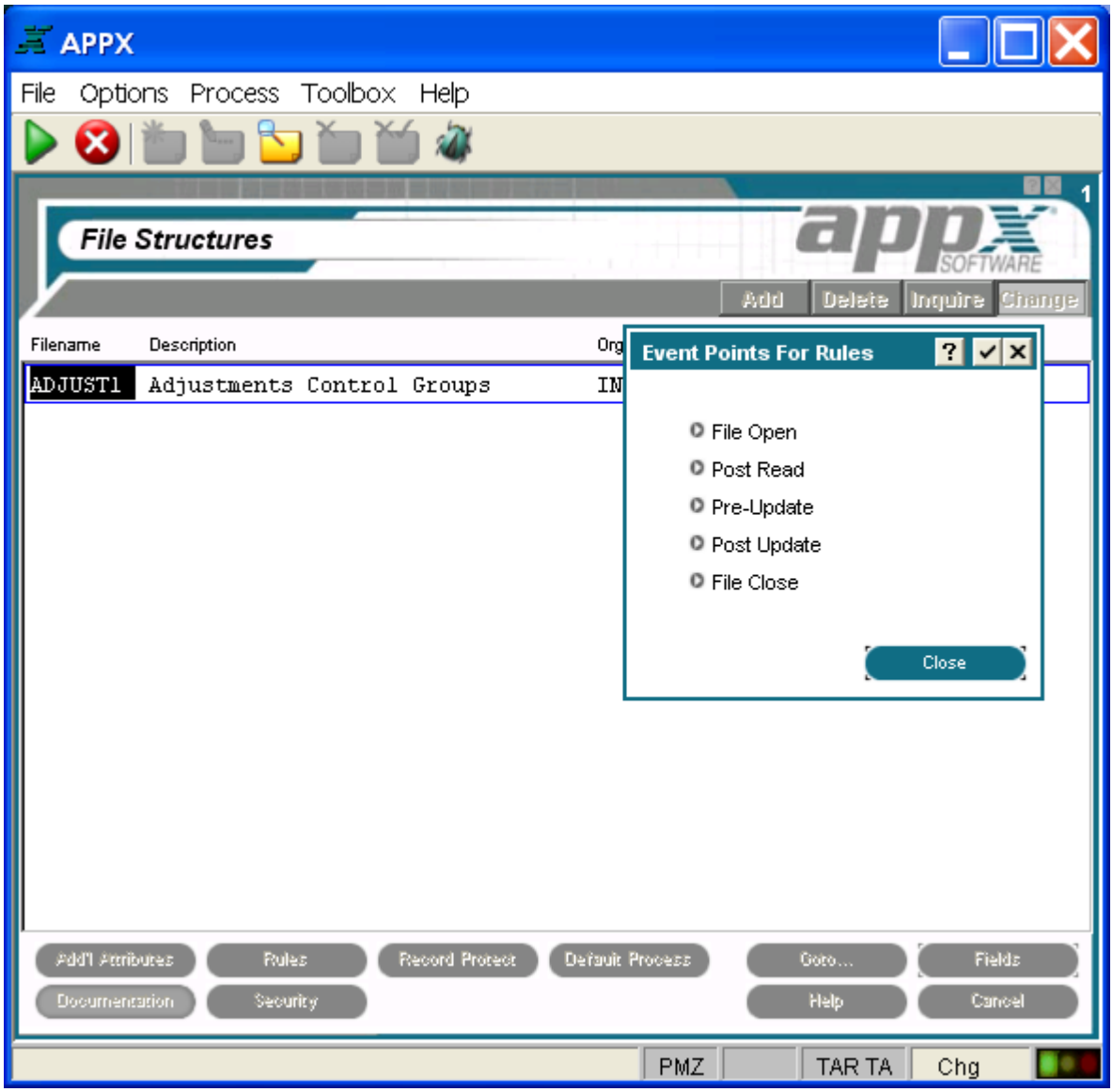


Figure 3-6-5. File Level Event Points for Rules Overlay

Input Frame Specifications

When you select the FRAMES option (or press RETURN) on the Input Processes screen, after completing specifications for an input process, the Input Process Frames screen is displayed as shown in *Figure 3-6-6 below.* Input frame specifications define attributes that govern both the visual presentation of a frame and the way it executes.

Input Process Frames screen

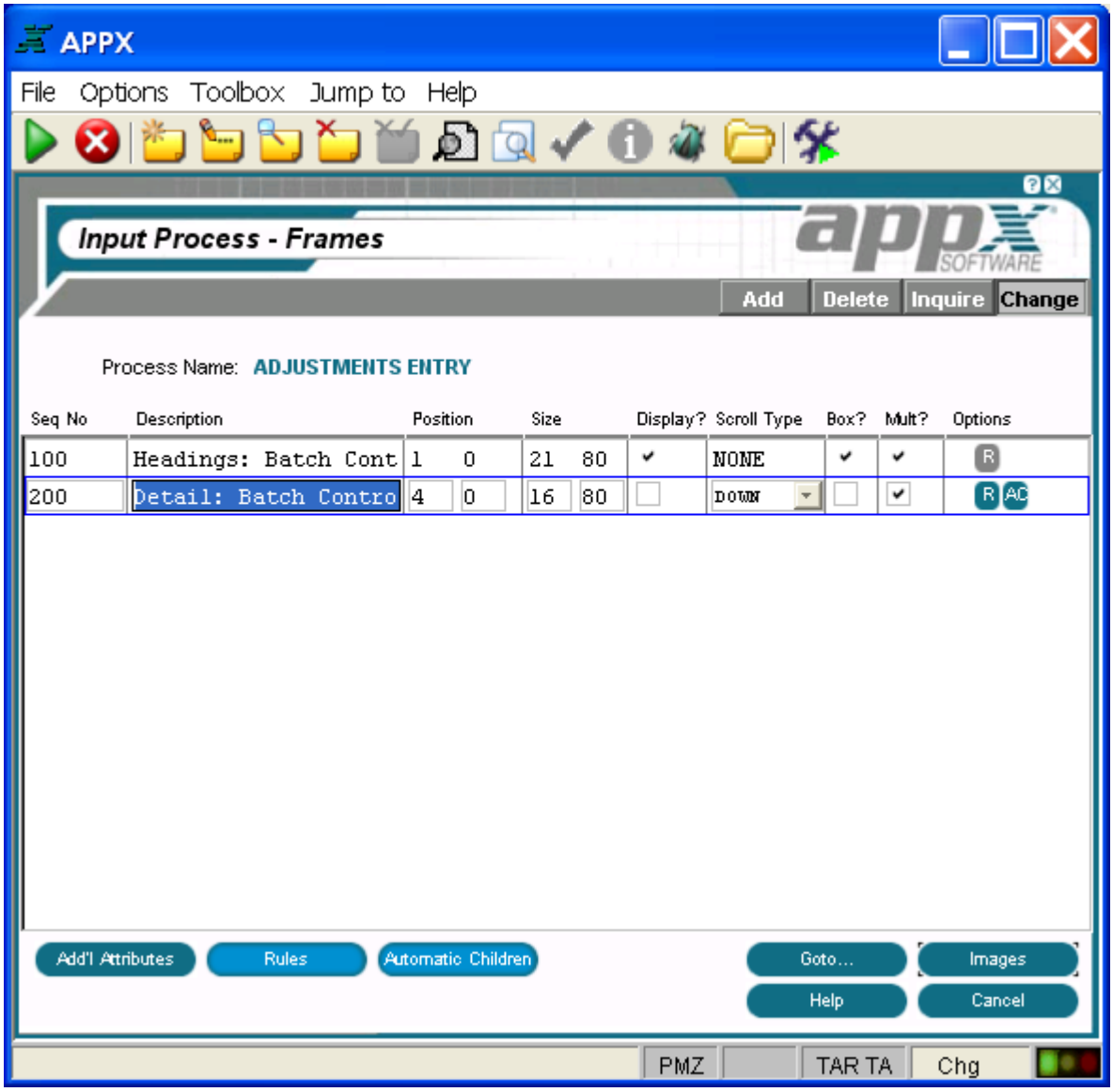


Figure 3-6-6

The Input Process Frames screen contains the following fields:

- ***Seq No*** displays a sequence number that is defaulted by APPX and used to identify this input frame. The default sequence number can be overridden, but must be unique within the process. Sequence numbers determine the order of the frames in the process structure. The first frame that is defined in add mode is assigned sequence number *100*, and additional frames are assigned a number equal to the highest existing sequence number, plus *100*.
- ***Description*** identifies this input frame. The description is for documentation purposes only.
- ***Position*** fields determine the placement of the rows and columns of a frame relative to the rows and columns on the screen display. Row entries designate positioning relative to the top of the screen

(or bottom if negative). Column entries designate positioning relative to the left of the screen. The default is *1, 1*. See Frame/Image Positioning in *Chapter 3-2* for more information.

- ***Size*** fields, which consist of three numeric characters, designate the number of rows and columns that comprise the displayed frame. This size may be greater than the size of the screen if panning capabilities are desired. The default is *21, 80*.
- ***Display?,*** represented by "yes" or "no" , indicates whether or not APPX stops and waits for the user to respond to the image. If "yes" is selected, APPX does not stop, but goes on to the next image. If "no" is selected, APPX stops on the image, even if no modifiable fields have been defined.
- ***Scroll Type*** indicates whether or not the images on the frame scroll and, if so, the type of scrolling to be performed as described below. The default is *NONE. See* The Effect of Scrolling Frames on Screen and Page Positioning section of *Chapter 3-2* for a detailed discussion of scrolling frames.

NONE ..... The image on the frame does not scroll.

DOWN ..... APPX divides the frame into a grid of equal-sized regions based upon a calculated composite image size. The first image is placed into the upper left region in the frame. Succeeding images are placed in the region below the previous image until the column is complete. As each column is completed, the next image is placed at the top of the column immediately to the right of the completed column.

ACROSS ..... APPX divides the frame into a grid of equal-sized regions based upon a calculated composite image size. The first image is placed into the upper left region in the frame. Succeeding images are placed in the region to the right of the previous image until the row is complete. As each row is completed, the next image is placed in the far left-hand position of the row immediately below the completed row.

COLUMNAR ... APPX divides the frame into columns based on the widest alternate image. The first image is placed into the upper left region in the frame. Succeeding images are placed immediately below the previous image until the column is complete. The column is considered complete if the next image cannot display in its entirety within the space available in the column. As each column is completed, the next image is placed at the top of the column immediately to the right of the completed column.

- ***Box?***, represented by "yes" or "no" , indicates whether or not APPX draws a box around a frame/image before it displays to a user. The border does not consume rows or columns within the space that is defined for the frame/image, but does consume space on the screen. The default is "no" but can be toggled to "yes" by clicking the checkbox.
- ***Mult?*** indicates whether multiple images can be defined and edited for this frame. Multiple images are used to present data entry images with different fields to users. For example, in an Employee

input process, you may want to present one data entry image if the employee is a salaried employee and a different image if the employee is a paid an hourly wage.

- ***Options*** are indicators that are present whenever the options they denote have been defined. Options applicable to this screen are AA (Additional Attributes), R (Rules), and AC (Automatic Children).

Buttons are provided at the bottom of each Input Process Frames screen for accessing the following options:

- ADDITIONAL ATTRIBUTES - See the Input Frame Additional Attributes Option for details.
- RULES - See the Input Frame Rules Option for details.
- AUTOMATIC CHILDREN - See the Input Frame Automatic Children Option for details.
- GOTO... - Allows you to identify and retrieve another record by optionally entering a full or partial key.
- IMAGES - See Input Image Specifications for details. This is the default option.
- HELP - Allows you to view online help text for a selected item.
- CLOSE or CANCEL - In Inquire mode, CLOSE allows you to end the current process. Otherwise, CANCEL allows you to end the current process without saving changes, if any, to the current record.

In addition, the menu bar provides the following pulldown menus:

- **Toolbox** - See the Input Frame Toolbox Pulldown Menu for details.
- **Jump to** - Same as the Process Jump to Pulldown Menu described in Chapter 3-3.

Input Frame Additional Attributes Option

When you select the ADDITIONAL ATTRIBUTES option on the Input Process Frames screen, the input process frame Additional Attributes overlay is displayed as shown in *Figure 3-6-7 below,* where you can access additional parameters.

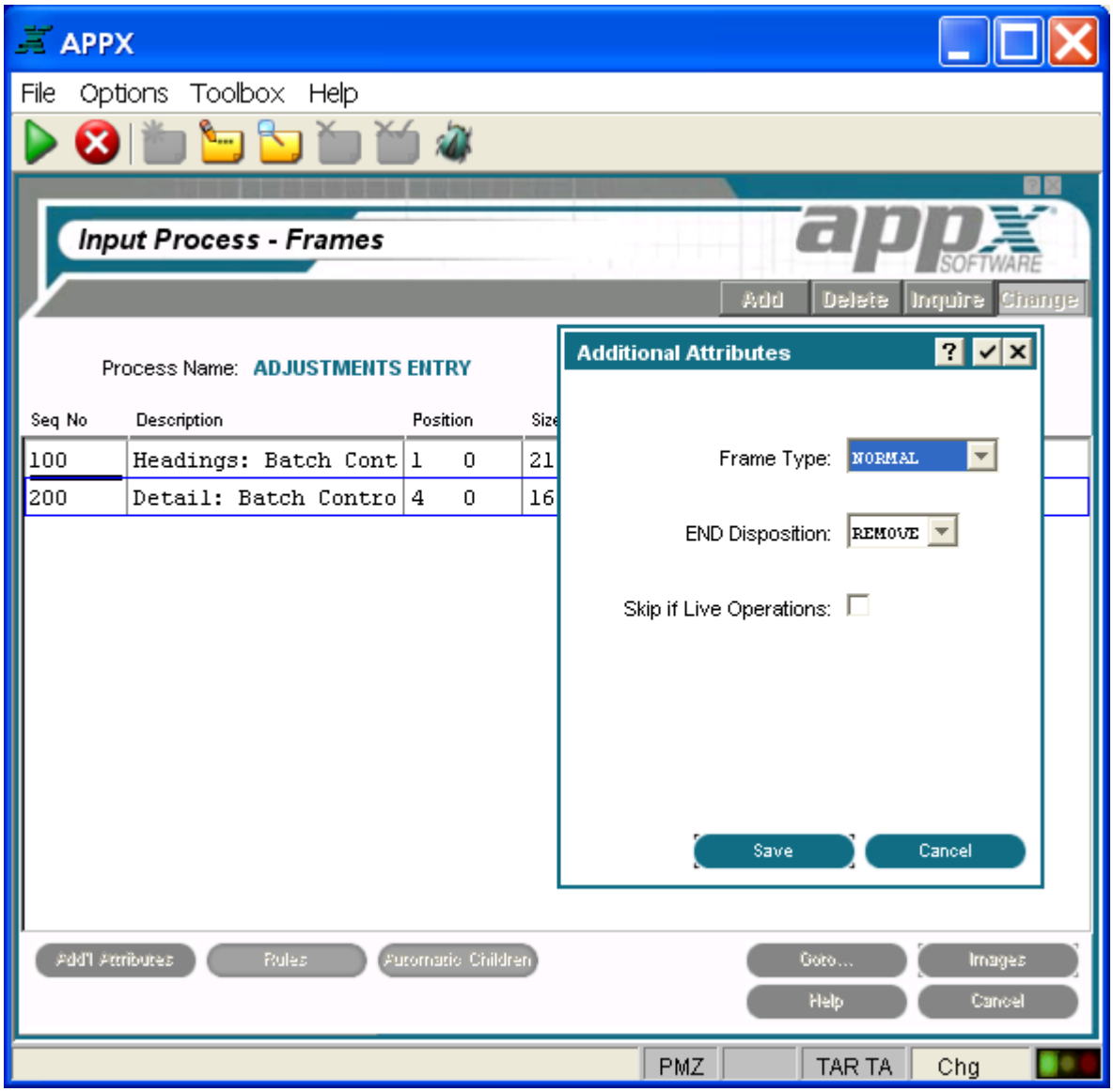


Figure 3-6-7. Input Frame Additional Attributes Overlay

The input frame Additional Attributes overlay contains the following fields.

- ***Frame Type*** governs the execution of an image or images for the frame. There are three frame types.

NORMAL ..... Executes the first (lowest numbered) alternate image unless another image is explicitly selected with statements and the predefined field ALTERNATE IMAGE NUMBER.

OPTIONAL ..... Executes the image that is identified by the *Normal Frame Type* above, with the following exception: If the value in the predefined field ALTERNATE IMAGE NUMBER is *0*, do not execute an image.

REPEATING ... Executes the image repeatedly until the value in the ALTERNATE IMAGE NUMBER is set to *0* by statements.

The default is *Normal.* See *Frames* in Chapter 3-1 for more information about optional and repeating frames.

- ***END Disposition*** designates the effect on the image displayed for this frame of a user selecting the END option. There are three *END Disposition* options:

HOLD ..... Causes APPX to automatically control the disposition of the frame. The image for the frame displays until the process cycle completes (until all frames execute for one PCF record), at which time the image is removed.

CLEAR ..... Causes the contents of the frame to clear, but the image for the frame still exists and still obscures the contents of any prior images that it overlays on the screen.

REMOVE ..... Causes the frame itself to be removed, and will no longer obscure the contents of any prior frames that it overlays on the screen.

The default is *REMOVE.*

- ***Skip If Live Operations?*** determines whether or not the frame is exclusive to the initial setup and recovery processing phases of operation. If "yes" ☑, the frame is not executed during the live operations phase. The default is "no" ☐ but can be toggled to "yes" by clicking the checkbox.

Input Frame Rules Option

When you select the RULES option on the Input Process Frames screen, the input process frame Event Points for Rules overlay is displayed as shown in *Figure 3-6-8 below*. This overlay provides options to access the ILF Editor for APPX event points. If statements are defined in an event point, the respective text on the screen is highlighted.

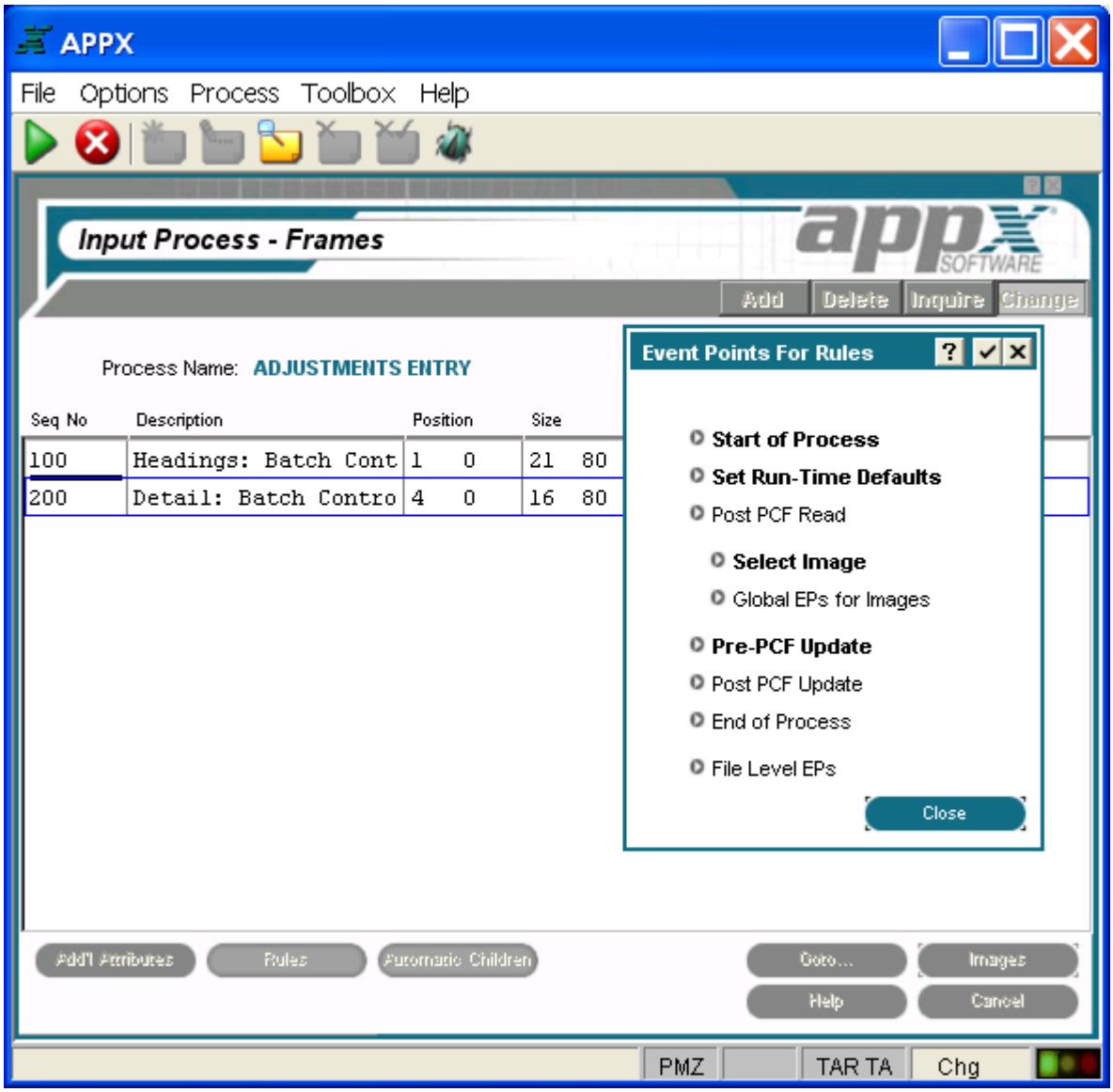


Figure 3-6-8. Input Frame Event Points for Rules Overlay

The input frame Event Points for Rules overlay contains the following options:

Start of Process

START OF PROCESS accesses the ILF editor for the *Start of Process* event point. Upon completion, you are returned to the Event Points overlay.

Set Run-Time Defaults

SET RUN-TIME DEFAULTS accesses the ILF editor for the *Set Run-Time Defaults* event point. Upon completion, you are returned to the Event Points overlay.

Post PCF Read

POST PCF READ accesses the ILF editor for the *Post PCF Read* event point. Upon completion, you are returned to the Event Points overlay.

Select Image

SELECT IMAGE accesses the ILF editor for the *Select Image* event point. Upon completion, you are returned to the Event Points overlay.

Global EP's For Images

GLOBAL EP'S FOR IMAGES calls the Global Event Points overlay shown in *Figure 3-6-9 below.*

Figure 3-6-9. Global Event Points Overlay

This overlay provides options for four additional event points:

- START OF FRAME accesses the ILF editor for the *Start of Frame* event point. Upon completion, you are returned to the Global Event Points overlay.
- GLOBAL PRE-DISPLAY accesses the ILF editor for the *Global Pre-Display* event point. Upon completion, you are returned to the Global Event Points overlay.
- GLOBAL OPTION INTERCEPT accesses the ILF editor for the *Global Option Intercept* event point. Upon completion, you are returned to the Global Event Points overlay.
- END OF FRAME accesses the ILF editor for the *End of Frame* event point. Upon completion, you are returned to the Global Event Points overlay.

Pre-PCF Update

PRE-PCF UPDATE accesses the ILF editor for the *Pre-PCF Update* event point. Upon completion, you are returned to the Event Points overlay.

Post PCF Update

POST PCF UPDATE accesses the ILF editor for the *Post PCF Update* event point. Upon completion, you are returned to the Event Points overlay.

End of Process

END OF PROCESS accesses the ILF editor for the *End of Process* event point. Upon completion, you are returned to the Event Points overlay.

File Level EPs

FILE LEVEL EPS provides access to the File Level Event Points for the PCF file. These event points are not directly associated with the process but with the PCF file. See the Files Rules Option *in Chapter 2-4* for more information on using these event points.

Refer to Chapter 3-16: Using Event Points for more information about event points.

Input Frame Automatic Children Option

When you select the AUTOMATIC CHILDREN option from the Input Process Frames screen, the Input Process Automatic Children screen is displayed as shown in *Figure 3-6-10*. This allows you to attach a specific process to the input process frame currently selected so that it invokes automatically following execution of the frame. See Automatic and Optional Child Invocations in Chapter 3-1 as well as Chapter 3-13: Automatic and Optional Children for more information.

Input Process Automatic Children screen

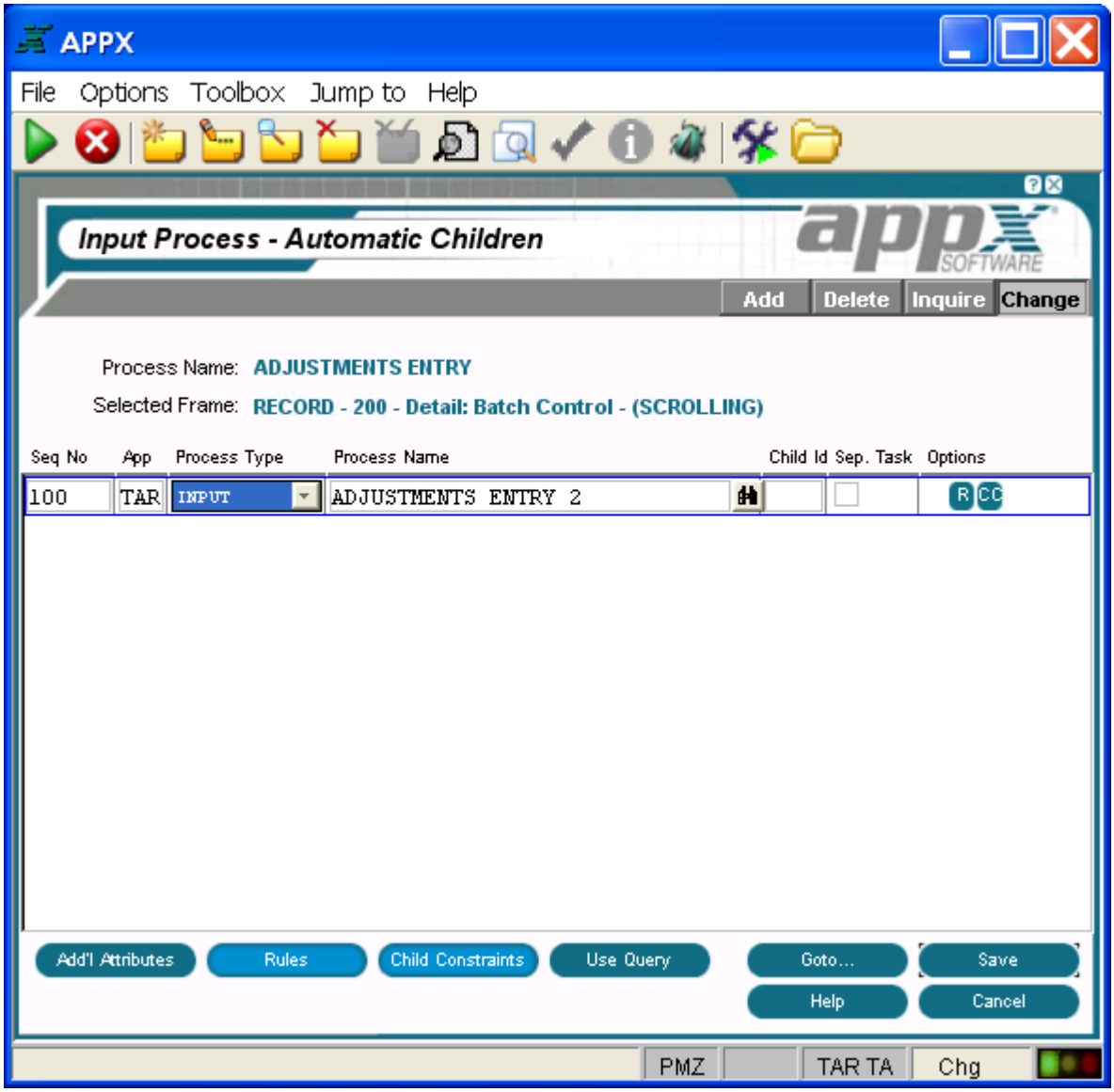


Figure 3-6-10

Input Frame Toolbox Pulldown Menu

When you select **Toolbox** from the menu bar on the Input Process Frames screen, the Toolbox Pulldown Menu appears as shown in *Figure 3-6-11* below. Note that each tool can also be accessed by selecting the corresponding icon on the screen toolbar.

Input Frames Toolbox Pulldown menu

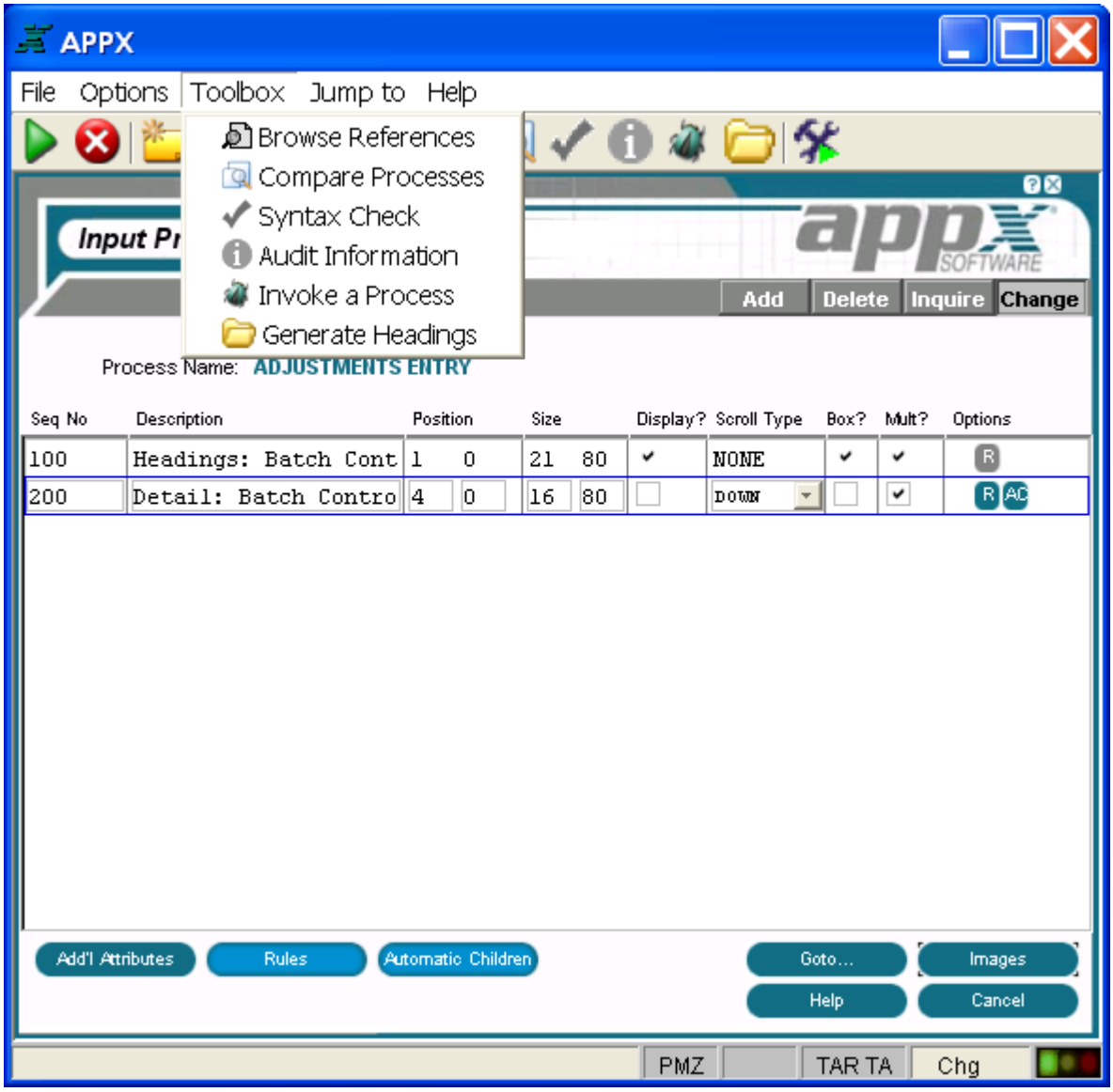


Figure 3-6-11

The input frames Toolbox Pulldown Menu offers the following options:

Browse References

The BROWSE REFERENCES option runs a modified version of "Cross Reference – All" and produces a list of design components utilized by the current process. The results can be displayed interactively where links allow you to see the context of the references, edit the referenced component, and drill down to the child process. This option can be used to identify, and optionally edit, the file level event points utilized by this process.

Compare Processes

The COMPARE PROCESSES option compares the current process to another process that you identify and produces a report listing any differences. You can also opt to browse the results interactively.

Syntax Check

The SYNTAX CHECK option allows you to create the Executable Module for the current process without running the process. If the Executable Module already exists and is still valid, this option is ignored if invoked. Note that this option (`OPTION 97`), like any Toolbox option with a value greater than 90, is available for use directly from any screen in Application Design without going to the Toolbox.

Audit Information

The AUDIT INFORMATION option allows you to see who created the current process and who changed it most recently. It also provides you with the date that the process was created and the date that it was last changed. Note that this option (`OPTION 98`), like any Toolbox option with a value greater than 90, is available for use directly from any screen in Application Design without going to the Toolbox. See Audit Information Overlay for an example of the overlay.

Invoke a Process

The INVOKE A PROCESS option is available to facilitate testing. When you select this option, APPX defaults to the current process but you may override the defaults to invoke any process. Note that this option, like any Toolbox option with a value greater than 90, is available for use directly from any screen in Application Design without going to the Toolbox. See Invoke a Process Overlay for details on using this option.

Generate Headings

The GENERATE HEADINGS option generates a column headings frame for the input frame you are currently positioned to. However, the input frame must be positioned at row 5 or higher.

Input Image Specifications

Input image specifications define the visual and operational attributes of each alternate image for an input frame. These specifications are determined, in part, by whether or not the currently selected frame is defined with a scroll type on the Input Process Frames screen. The scrolling input Images screen and the non-scrolling input Images screen are described separately in the following sections:

- Scrolling Input Image Specifications
- Non-Scrolling Input Image Specifications

Scrolling Input Image Specifications

When you select the IMAGES option (or press RETURN) on the Input Process Frames screen for a frame with scrolling defined, the Scrolling Input Images screen is displayed as shown in *Figure 3-6-12 below*.

Scrolling Input Images screen

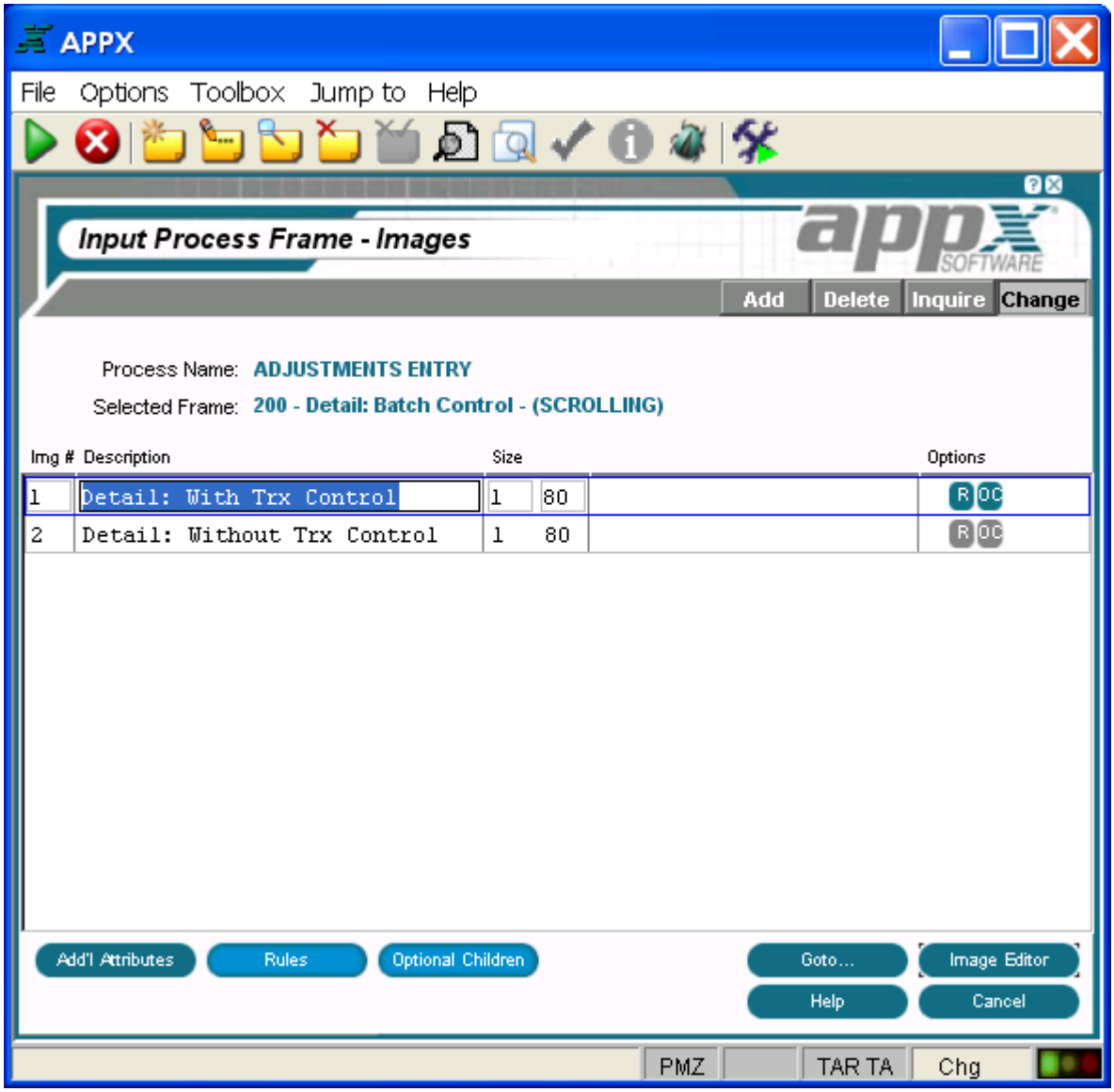


Figure 3-6-12

The scrolling input Images screen contains the following fields:

- *Img #* displays a sequence number that is defaulted by APPX. The image number uniquely identifies each alternate image for the frame and is referenced when determining which image, if any, is to display. The first image that is defined in add mode is assigned alternate image number *1*, and additional images are assigned a number equal to the highest existing image number, plus *1*. The

default number can be overridden, but must be unique within the frame. Refer to the Images section in *Chapter 3-1* for a discussion of alternate images.

- ***Description*** identifies this alternate image. The description is for documentation purposes only.
- ***Size*** designates the number of rows and columns that comprise the displayed image. The image cannot be larger than the frame to which it belongs. If blank, the size of the image defaults to the size of the frame. The default is blank. See Frame/Image Positioning in *Chapter 3-2* for more information.
- ***Options*** are indicators that are present whenever the options they denote have been defined. Options applicable to this screen are AA (Additional Attributes), R (Rules), and OC (Optional Children).

Buttons are provided at the bottom of each Images screen for accessing the following options:

- ADDITIONAL ATTRIBUTES - See the Scrolling Input Image Additional Attributes Option for details.
- RULES - See the Input Image Rules Option for details. This overlay is the same for both scrolling and non-scrolling frames.
- OPTIONAL CHILDREN - See the Input Image Optional Children Option for details. This screen is the same for both scrolling and non-scrolling frames.
- GOTO... - Allows you to identify and retrieve another record by optionally entering a full or partial key.
- IMAGE EDITOR - See the Scrolling Input Image in the Image Editor for details. This is the default option.
- HELP - Allows you to view online help text for a selected item.
- CLOSE or CANCEL - In Inquire mode, CLOSE allows you to end the current process. Otherwise, CANCEL allows you to end the current process without saving changes, if any, to the current record.

In addition, the menu bar provides the following pulldown menus:

- **Toolbox** - Same as the Input Frame Toolbox Pulldown Menu except that there is no GENERATE HEADINGS option at the image level.
- **Jump to** - Same as the Process Jump to Pulldown Menu described in Chapter 3-3.

Scrolling Input Image Additional Attributes Option

When you select the ADDITIONAL ATTRIBUTES option on the Scrolling Input Images screen, the scrolling input Additional Attributes overlay is displayed as shown in *Figure 3-6-13 below,* where you can access additional parameters.

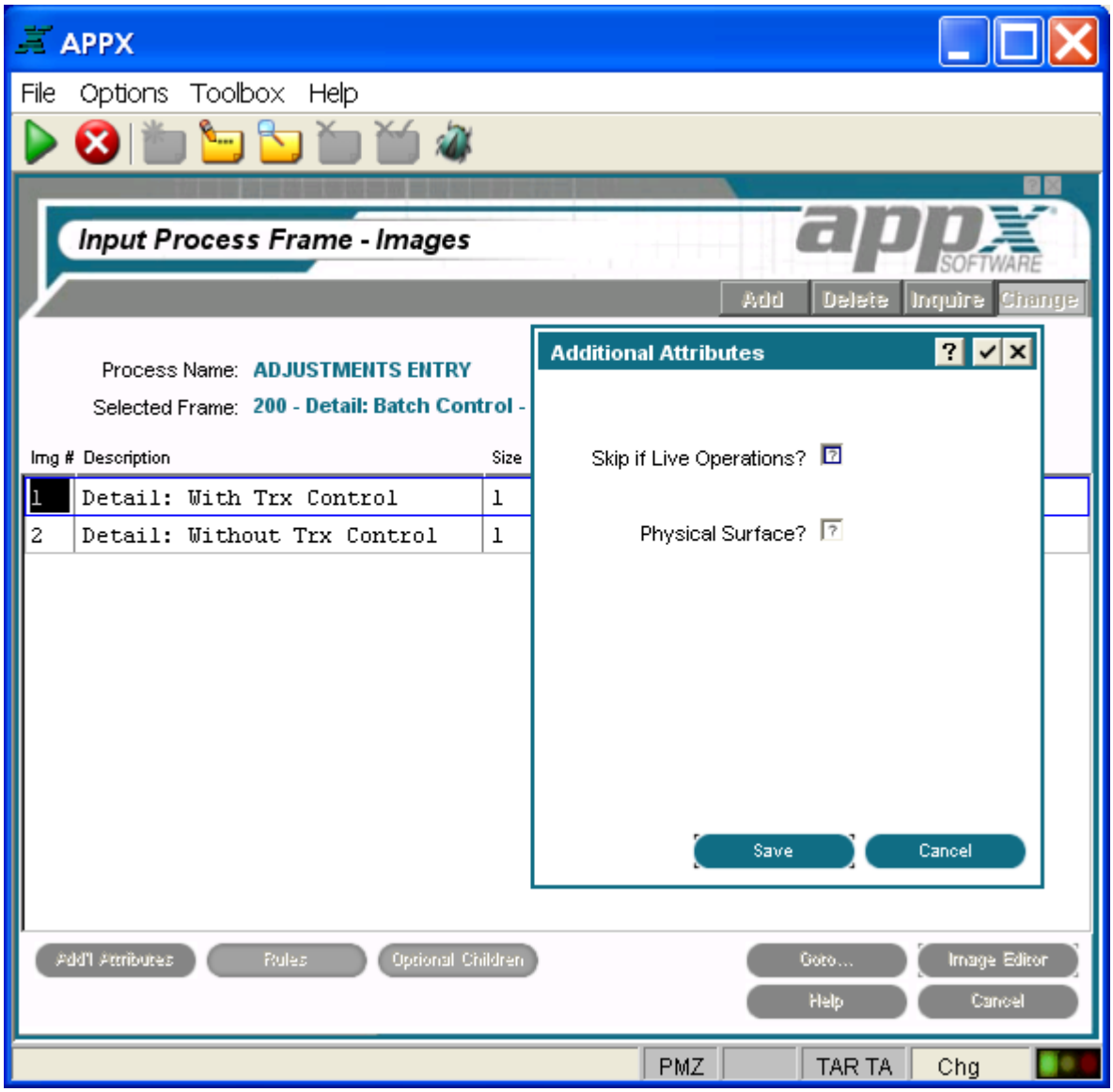


Figure 3-6-13. Scrolling Input Additional Attributes Overlay

The scrolling input Additional Attributes overlay contains the following fields.

- *Skip If Live Operations?* determines whether or not this image is exclusive to the initial setup and recovery processing phases of operation. If "yes" or ☑, the image is not executed during the live operations phase.
- *Physical Surface?* controls whether or not this image is "attached" to the physical surface. Click the checkbox to toggle between values of "yes" ☑, "no" ☐, or "unknown" ?. "Yes" indicates that the image will always display in front of images that are not attached to the physical surface and that panning options do not apply. The image will always be "seen" by the user even if the logical surface is panned far to the right or other images not attached to the physical surface are executed.

Input Image Rules Option

When you select the RULES option on either the Scrolling Input Images screen or the Non-Scrolling Input Images screen, the input image Event Points for Rules overlay is displayed as shown in *Figure 3-6-14 below*. This overlay provides options to access the ILF Editor for APPX event points. If statements are defined in an event point, the respective text on the screen is highlighted.

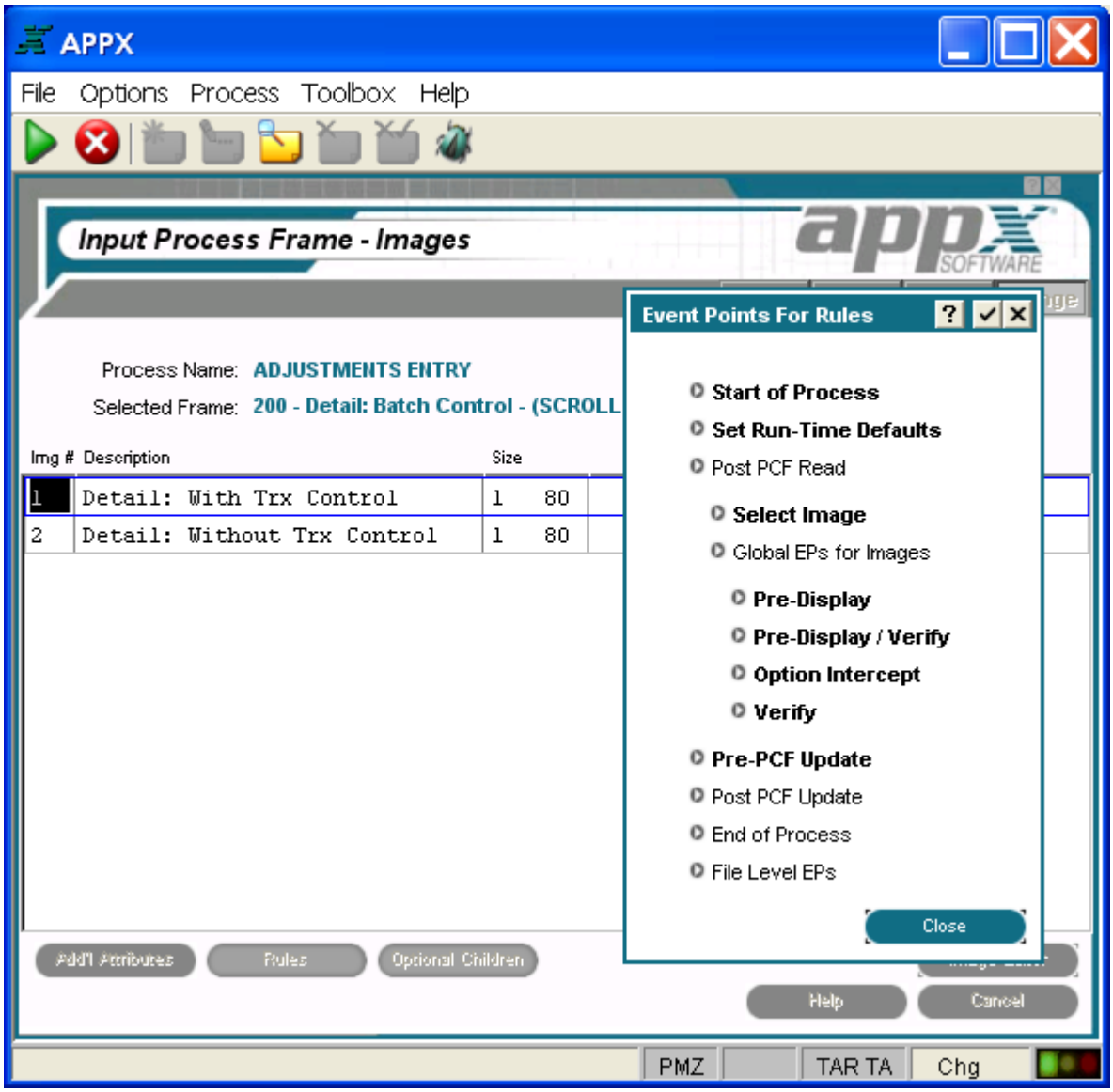


Figure 3-6-14. Input Image Event Points for Rules Overlay

The input image Event Points for Rules overlay contains the following options:

Start of Process

START OF PROCESS accesses the ILF editor for the *Start of Process* event point. Upon completion, you are returned to the Event Points overlay.

Set Run-Time Defaults

SET RUN-TIME DEFAULTS accesses the ILF editor for the *Set Run-Time Defaults* event point. Upon completion, you are returned to the Event Points overlay.

Post PCF Read

POST PCF READ accesses the ILF editor for the *Post PCF Read* event point. Upon completion, you are returned to the Event Points overlay.

Select Image

SELECT IMAGE accesses the ILF editor for the *Select Image* event point. Upon completion, you are returned to the Event Points overlay.

Global EP's For Images

GLOBAL EP'S FOR IMAGES calls the Global Event Points overlay shown in *Figure 3-6-15 below.*

Figure 3-6-15. Global Event Points Overlay

This overlay provides options for four additional event points:

- START OF FRAME accesses the ILF editor for the *Start of Frame* event point. Upon completion, you are returned to the Global Event Points overlay.
- GLOBAL PRE-DISPLAY accesses the ILF editor for the *Global Pre-Display* event point. Upon completion, you are returned to the Global Event Points overlay.
- GLOBAL OPTION INTERCEPT accesses the ILF editor for the *Global Option Intercept* event point. Upon completion, you are returned to the Global Event Points overlay.
- END OF FRAME accesses the ILF editor for the *End of Frame* event point. Upon completion, you are returned to the Global Event Points overlay.

Pre-Display

PRE-DISPLAY accesses the ILF editor for the *Pre-Display* event point. Upon completion, you are returned to the Event Points overlay.

Pre-Display/Verify

PRE-DISPLAY/VERIFY accesses the ILF editor for the *Pre-Display/Verify* event point. Upon completion, you are returned to the Event Points overlay.

Option Intercept

OPTION INTERCEPT accesses the ILF editor for the *Option Intercept* event point. Upon completion, you are returned to the Event Points overlay.

Verify

VERIFY accesses the ILF editor for the *Verify* event point. Upon completion, you are returned to the Event Points overlay.

Pre-PCF Update

PRE-PCF UPDATE accesses the ILF editor for the *Pre-PCF Update* event point. Upon completion, you are returned to the Event Points overlay.

Post PCF Update

POST PCF UPDATE accesses the ILF editor for the *Post PCF Update* event point. Upon completion, you are returned to the Event Points overlay.

End of Process

END OF PROCESS accesses the ILF editor for the *End of Process* event point. Upon completion, you are returned to the Event Points overlay.

File Level EPs

FILE LEVEL EPS provides access to the File Level Event Points for the PCF file. These event points are not directly associated with the process but with the PCF file. See the Files Rules Option *in Chapter 2-4* for more information on using these event points.

Refer to Chapter 3-16: Using Event Points for more information about event points.

Input Image Optional Children Option

When you select the OPTIONAL CHILDREN option on either the Scrolling Input Images screen or the Non-Scrolling Input Images screen, the Input Process Optional Children screen is displayed as shown in *Figure 3-6-16*. This allows you to define attributes to govern the discretionary invocation of another APPX process from an image in an input process. See Automatic and Optional Child Invocations in Chapter 3-1 as well as Chapter 3-13: Automatic and Optional Children for more information.

Input Process Optional Children screen

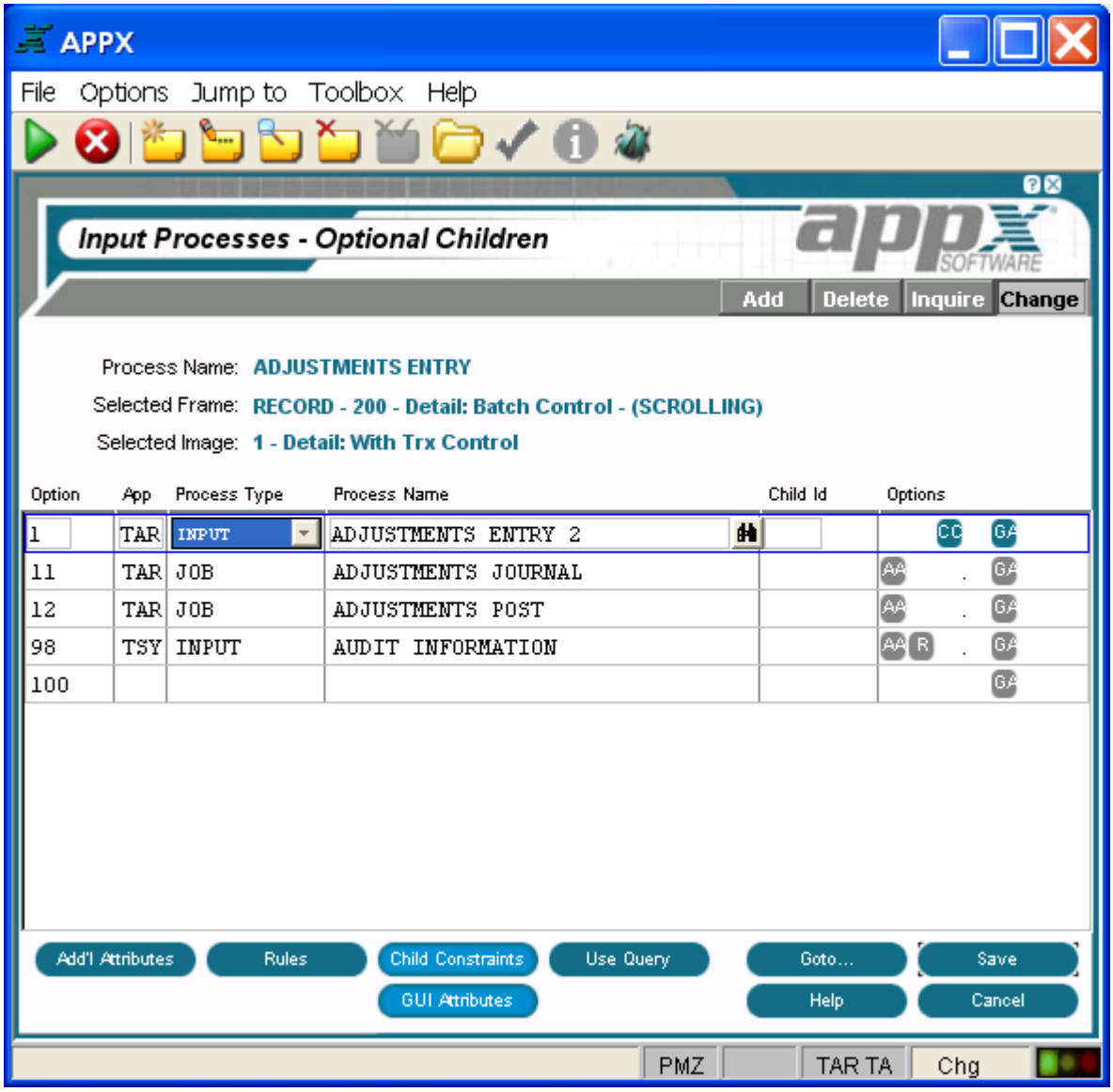


Figure 3-6-16

Scrolling Input Image in the Image Editor

After you complete specifications for an image on the Scrolling Input Images screen, you automatically enter the Image Editor when you press RETURN or select the IMAGE EDITOR option. The Image Editor, APPX's screen painting facility, is the facility used to format menus, inputs, outputs, and inquiry screens. As you design the input screen, you are seeing the image exactly as it will appear to your users. *Figure 3-6-17 below shows a scrolling input image being edited in the Image Editor.*

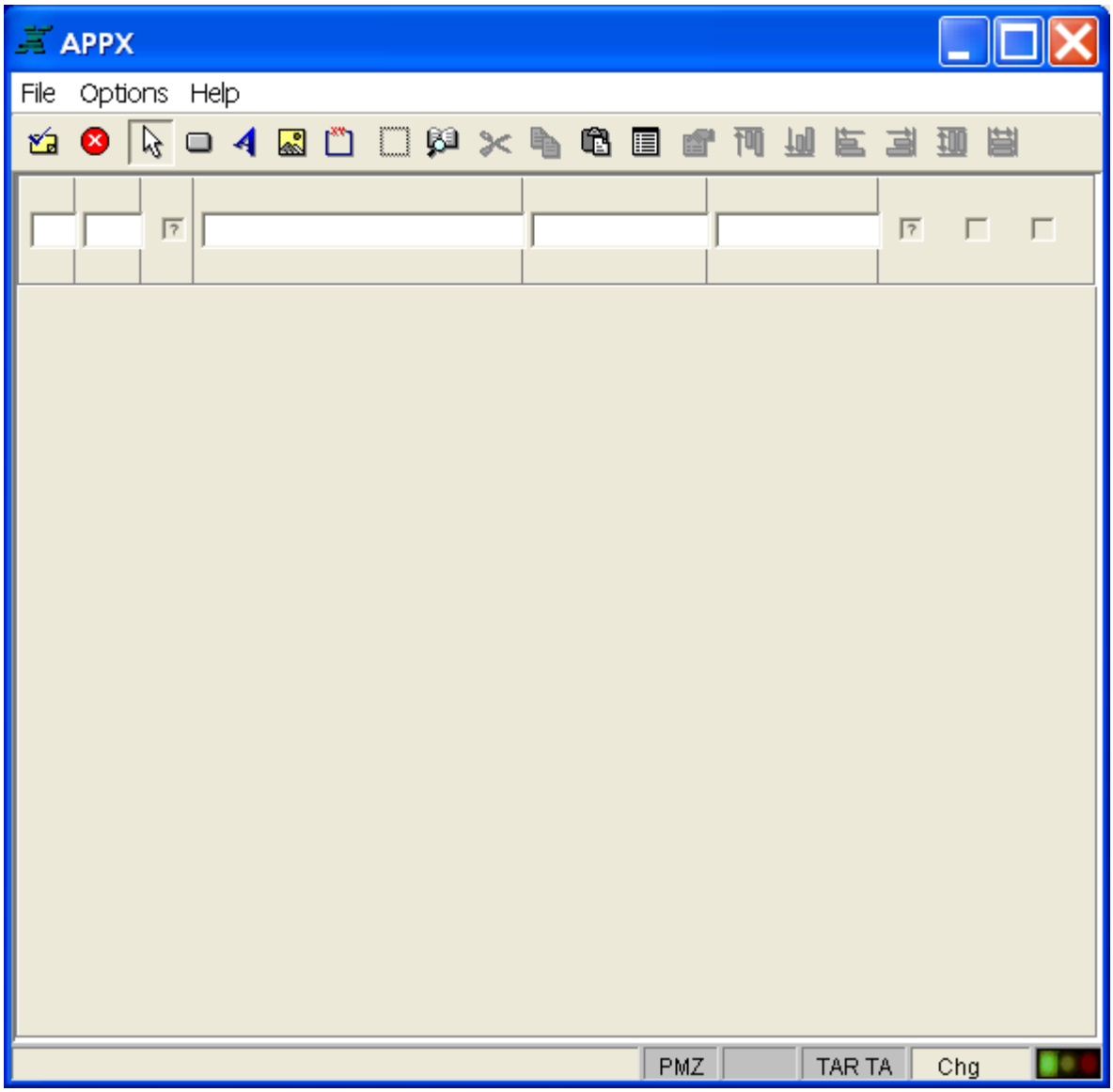


Figure 3-6-17. Scrolling Input Image in the Image Editor

The standard features of the Image Editor can be employed in character mode using only a keyboard. However, you need a mouse to take advantage of the GUI features within the Image Editor. GUI tools provide designers with the ability to create screen images complete with pictures, buttons, icons, wallpapers, and other graphical features. Another GUI tool is a data palette offering drag and drop selection and placement of data fields.

Basic functionality of the Image Editor is fully explored in Chapter 3-14: Using the Image Editor and a complete discussion of its GUI features is presented in Chapter 3-15: Using GUI Features of the Image Editor.

> Note that there is normally no need to identify the data fields in a scrolling image. Column headings are usually defined for the data fields in an image in an earlier display-only frame, such as that shown in *Figure 3-6-18* below. This particular image also includes buttons for the optional children attached to the scrolling image.

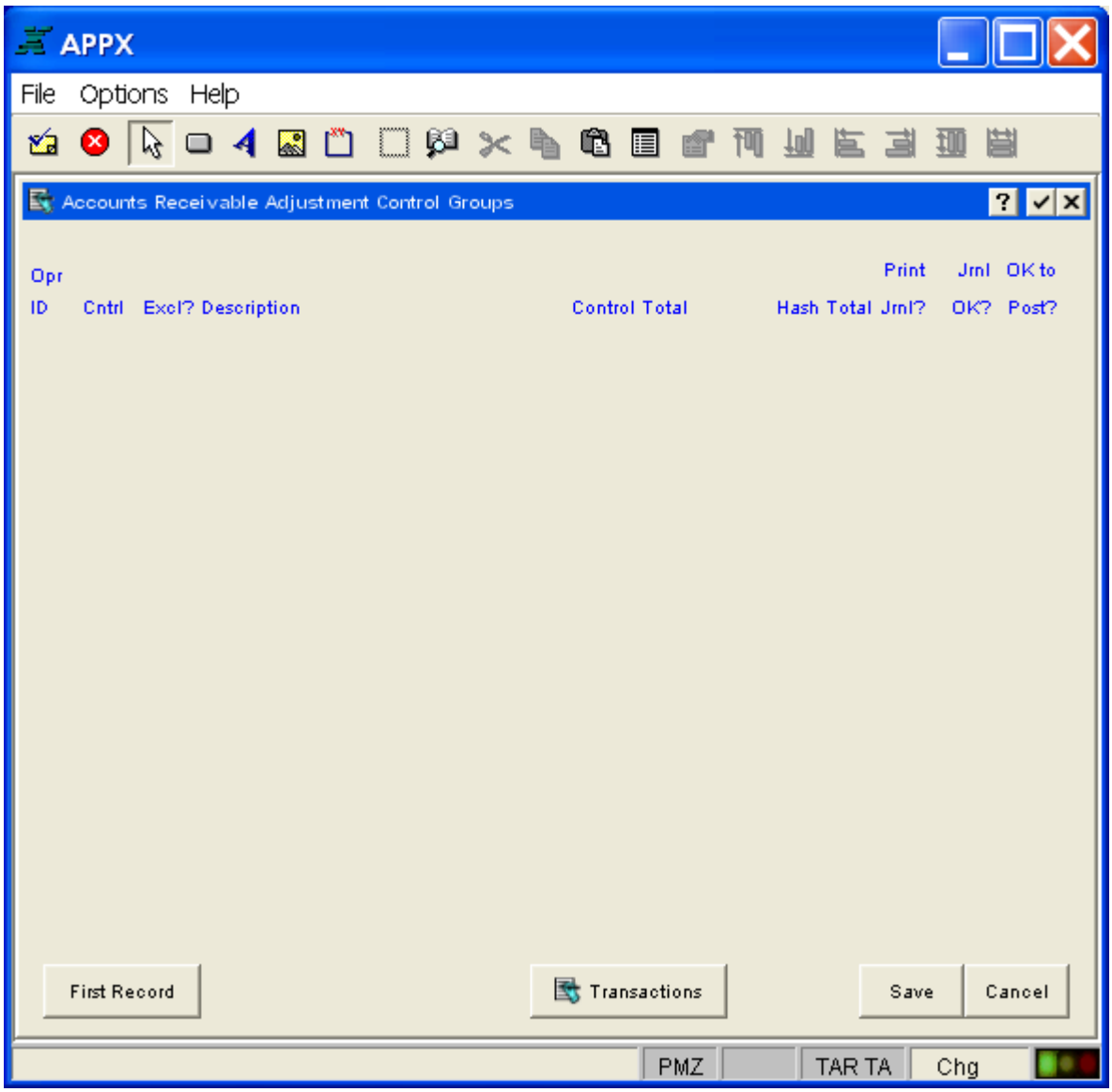


Figure 3-6-18. Input Image with Column Headings in the Image Editor

Non-Scrolling Input Image Specifications

When you select the IMAGES option (or press RETURN) on the Input Process Frames screen for a frame with no scrolling defined, the Non-Scrolling Input Images screen is displayed as shown in *Figure 3-6-19 below.*

Non-Scrolling Input Images screen

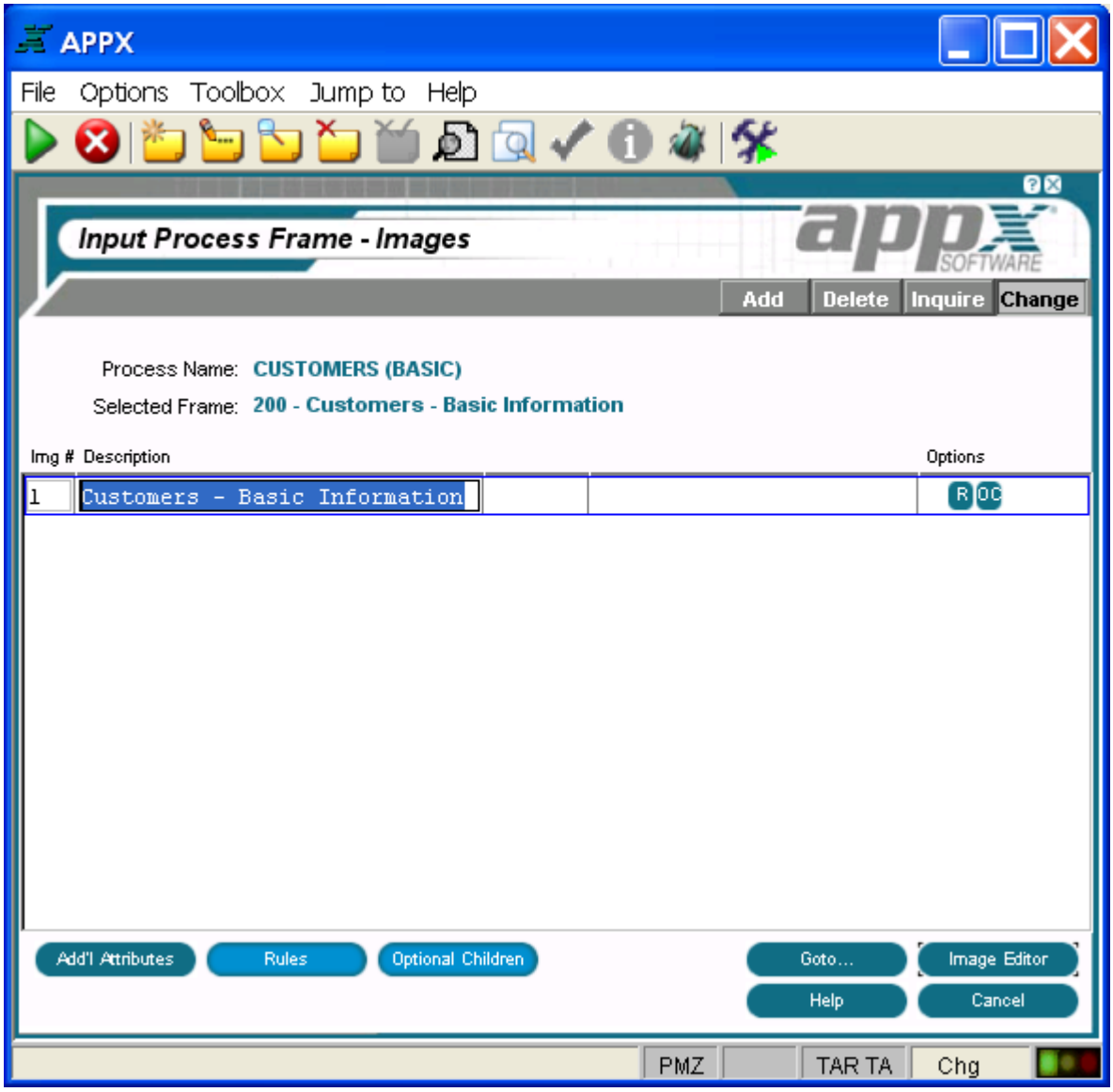


Figure 3-6-19

The non-scrolling input Images screen contains the following fields:

- ***Img #*** displays a sequence number that is defaulted by APPX. The image number uniquely identifies each alternate image for the frame and is referenced when determining which image, if any, is to display. The first image that is defined in add mode is assigned alternate image number *1*, and additional images are assigned a number equal to the highest existing image number, plus *1*. The default number can be overridden, but must be unique within the frame. Refer to the Images section in *Chapter 3-1* for a discussion of alternate images.

- ***Description*** identifies this alternate image. The description is for documentation purposes only.
- ***Options*** are indicators that are present whenever the options they denote have been defined. Options applicable to this screen are AA (Additional Attributes), R (Rules), and OC (Optional Children).

Buttons are provided at the bottom of each Images screen for accessing the following options:

- ADDITIONAL ATTRIBUTES - See the Non-Scrolling Input Image Additional Attributes Option for details.
- RULES - See the Input Image Rules Option for details. This overlay is the same for both scrolling and non-scrolling frames.
- OPTIONAL CHILDREN - See the Input Image Optional Children Option for details. This screen is the same for both scrolling and non-scrolling frames.
- GOTO... - Allows you to identify and retrieve another record by optionally entering a full or partial key.
- IMAGE EDITOR - See the Non-Scrolling Input Image in the Image Editor for details. This is the default option.
- HELP - Allows you to view online help text for a selected item.
- CLOSE or CANCEL - In Inquire mode, CLOSE allows you to end the current process. Otherwise, CANCEL allows you to end the current process without saving changes, if any, to the current record.

In addition, the menu bar provides the following pulldown menus:

- **Toolbox** - Same as the Input Frame Toolbox Pulldown Menu except that there is no GENERATE HEADINGS option at the image level.
- **Jump to** - Same as the Process Jump to Pulldown Menu described in Chapter 3-3.

Non-Scrolling Input Image Additional Attributes Option

When you select the ADDITIONAL ATTRIBUTES option on the Non-Scrolling Input Images screen, the non-scrolling input Additional Attributes overlay is displayed as shown in *Figure 3-6-20 below,* where you can access additional parameters.

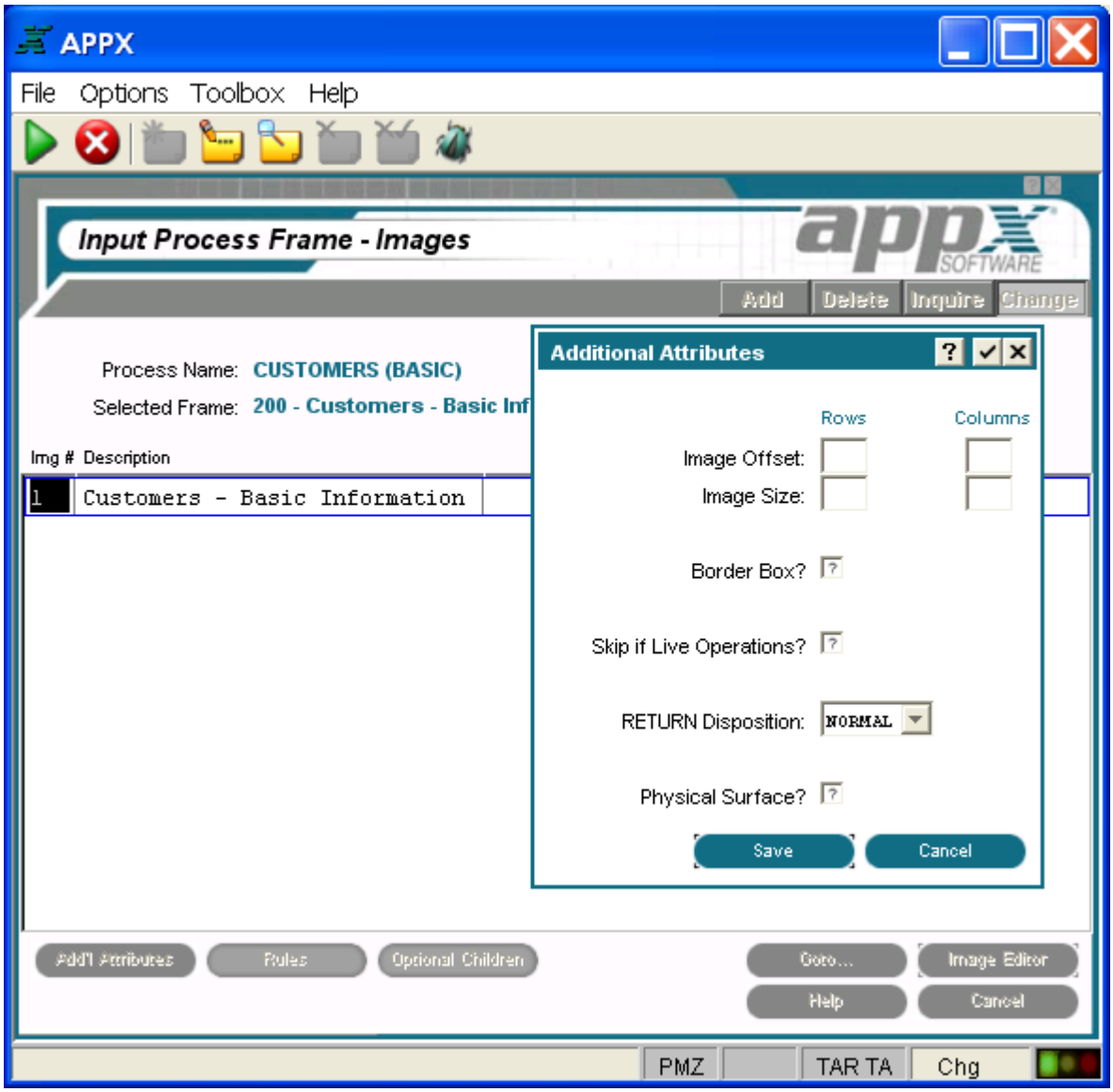


Figure 3-6-20. Non-Scrolling Input Additional Attributes Overlay

The non-scrolling input Additional Attributes overlay contains the following fields.

- ***Image Offset*** determines the placement of the rows and columns of this image, relative to the frame. Row entries designate positioning, relative to the top of the frame. Column entries designate positioning relative to the left of the frame. If blank, the APPX defaults to same row/column position as the frame. The default is blank. See Frame/Image Positioning in *Chapter 3-2* for more information.
- ***Image Size*** designates the number of rows and columns that comprise the displayed image. The image cannot be larger than the frame it belongs to. If blank, the size of the image defaults to the size of the frame. The default is blank.
- ***Border Box?*** indicates whether or not APPX draws a box around this image before it displays to a user. The border does not consume rows or columns within the space that is defined for the image,

but does consume space on the screen. If the box specification is combined with an overline, an underline, or both, APPX replaces the top and/or bottom of the box with the overline and/or underline. This minimizes the space that is consumed by the border. The physical space on the screen that is consumed by the box is system-specific.

- ***Skip If Live Operations?*** determines whether or not this image is exclusive to the initial setup and recovery processing phases of operation. If "yes" or ☑, the image is not executed during the live operations phase.
- ***RETURN Disposition*** option designates the effect on the image displayed for this frame of a user pressing RETURN or selecting ENTER ☑. There are three *RETURN Disposition* options.

NORMAL ..... Causes APPX to automatically control the disposition of the frame. The image for the frame displays until the process cycle completes (until all frames execute for one PCF record), at which time the image is removed.

CLEAR ..... Causes the contents of the frame to clear, but the image for the frame still exists and still obscures the contents of any prior images that it overlays on the screen.

REMOVE ..... Causes the frame itself to be removed, and will no longer obscure the contents of any prior frames that it overlays on the screen.

- ***Physical Surface?*** controls whether or not this image is "attached" to the physical surface. Click the checkbox to toggle between values of "yes" ☑, "no" ☐, or "unknown" ?. "Yes" indicates that the image will always display in front of images that are not attached to the physical surface and that panning options do not apply. The image will always be "seen" by the user even if the logical surface is panned far to the right or other images not attached to the physical surface are executed.

Non-Scrolling Input Image in the Image Editor

After you complete specifications for an image on the Non-Scrolling Input Images screen, you automatically enter the Image Editor when you press RETURN or select the IMAGE EDITOR option. The Image Editor, APPX's screen painting facility, is the facility used to format menus, inputs, outputs, and inquiry screens. As you design the input screen, you are seeing the image exactly as it will appear to your users. *Figure 3-6-21 below shows a non-scrolling input image being edited in the Image Editor.*

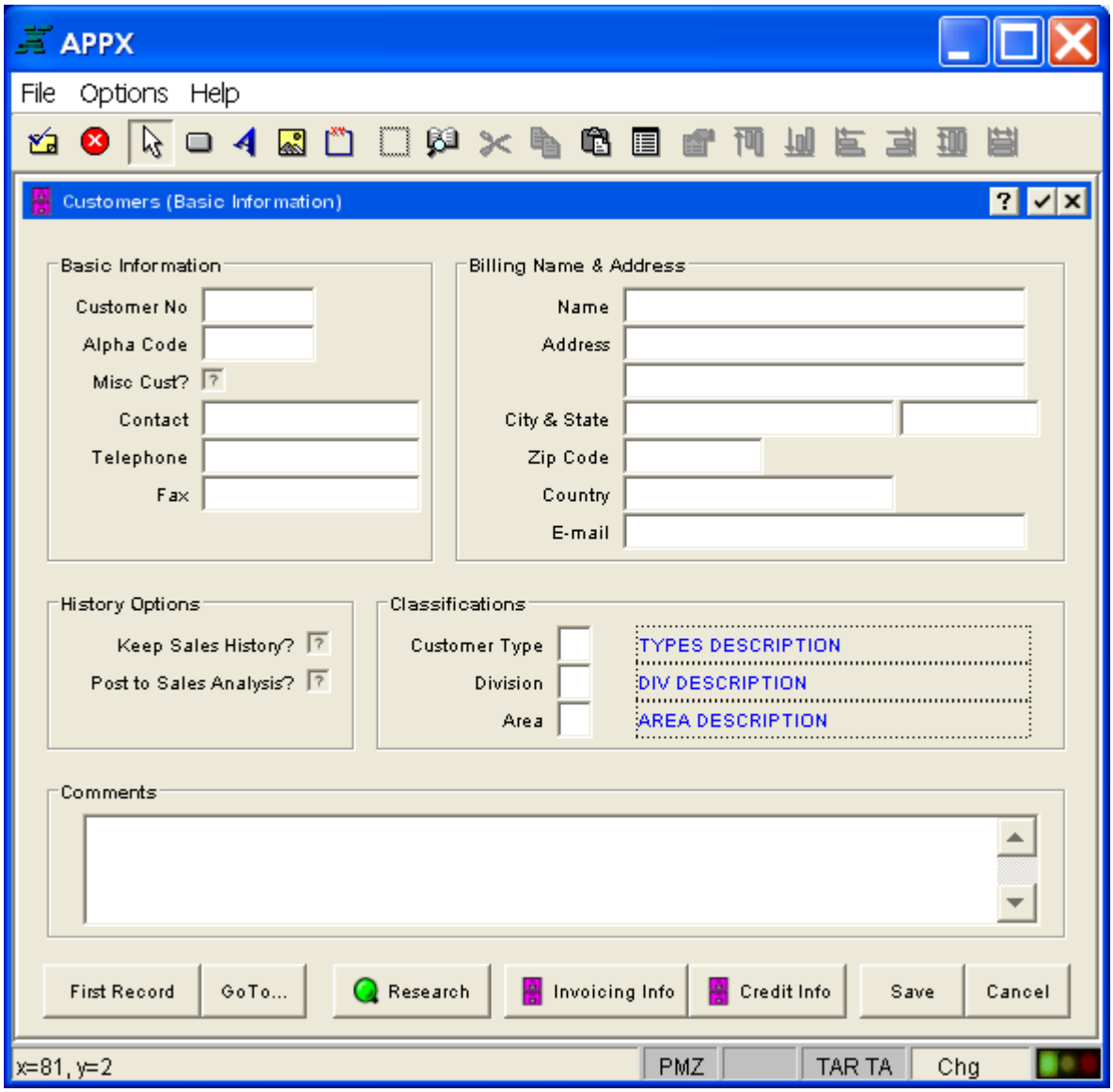


Figure 3-6-21. Non-Scrolling Input Image in the Image Editor

The standard features of the Image Editor can be employed in character mode using only a keyboard. However, you need a mouse to take advantage of the GUI features within the Image Editor. GUI tools provide designers with the ability to create screen images complete with pictures, buttons, icons, wallpapers, and other graphical features. Another GUI tool is a data palette offering drag and drop selection and placement of data fields.

Basic functionality of the Image Editor is fully explored in Chapter 3-14: Using the Image Editor and a complete discussion of its GUI features is presented in Chapter 3-15: Using GUI Features of the Image Editor.

> Note that data fields in a non-scrolling image should normally be identified for the user by a label or other mechanism. This particular image also includes buttons for the optional children attached to the non-scrolling image.

CHAPTER 3-7

Output Processes

Introduction to Output Processes

Output processes convey information from the database, either on the screen or in hard copy, according to a designer-defined format. Trial balances, product purchase history reports, and customer lists are all examples of output processes.

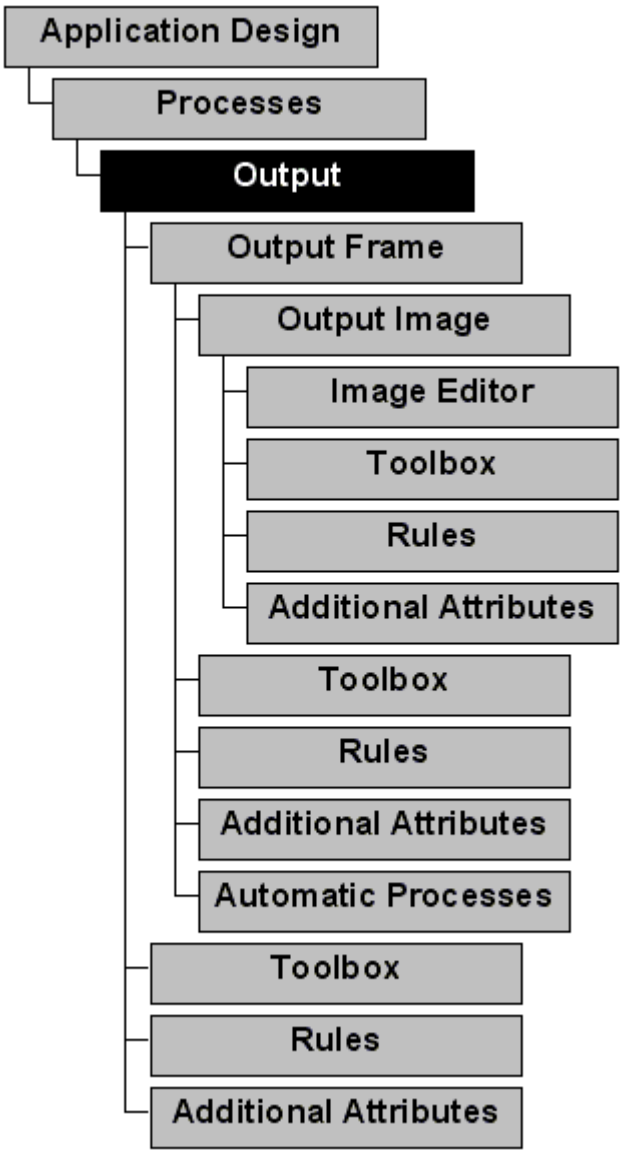


Figure 3-7-1. Output Process Definition

Output Process Specifications

Output process specifications establish the general attributes of an output process such as a process print file and process control file, the name of the report, and various format characteristics. The Output Processes screen, shown in *Figure 3-7-2 below,* displays when you select OUTPUTS from the Application Design Specifications Menu.

Output Processes screen

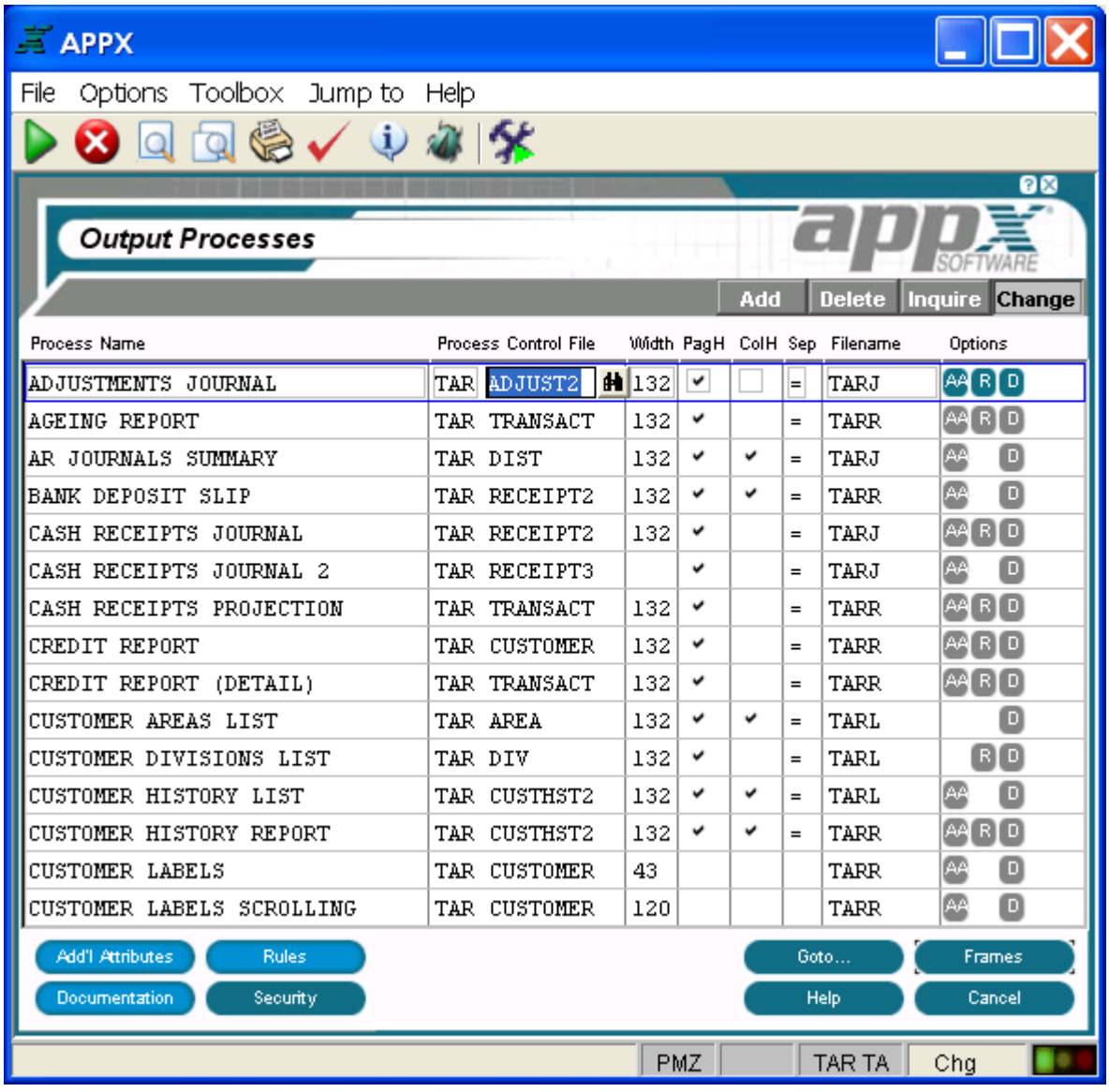


Figure 3-7-2

The Output Processes screen contains the following fields:

- ***Process Name*** identifies a specific process and consists of up to 30 uppercase alphanumeric characters that must be unique within type and application.

- ***Process Control File*** is *optional* for output processes. If present, it identifies a file by application ID and file name that is to be automatically managed by APPX during the execution of this process. APPX automatically reads, writes, rewrites, and deletes records in the process control file (PCF) during output process execution (and provides for updating the child PCF for any automatic children). If the PCF is left blank, the output process will execute once and then automatically end (as opposed to executing once for every record).
- ***Width*** indicates the maximum number of characters across, or in other words the number of columns, to allocate for each row in the report format. APPX uses the value you enter here to produce standard images of the proper width. If blank, the default width of the form that the report prints on defaults to the form width which is defined in system administration. If the form is narrower than the width of the formatted report, APPX truncates the output. If the form is wider than the width of the report, however, APPX uses the larger width for standard images, but does not expand the format of the report itself. The default is *132*.
- ***PagH*** determines whether or not APPX automatically prints a standard page heading image. If "yes" ☑, this image prints before any *Start of Page* frames for the report. Be careful not to also explicitly define a *Start of Page* frame with a standard page heading image, which could cause page headings to print twice at the top of each page in the report. If "no" ☐, you can explicitly define a *Start of Page* frame with a standard page heading image or, alternatively, define your own page heading using a *Start of Page* frame. The default is "yes".
- ***ColH*** determines whether or not APPX automatically prints a *Start of Page* frame with a standard column heading image. APPX generates column headings based upon specifications in the data dictionary for each item included in the output image. If "yes" ☑, this image prints after any *Start of Page* frames for the report. Be careful not to also explicitly define a *Start of Page* frame with a standard column heading image, which would cause column headings to print twice for each page in the report. If "no" ☐, you can explicitly define a *Start of Page* frame with a standard column heading image or, alternatively, define your own column headings using a *Start of Page* frame. The default is "yes". Refer to Standard Column Heading Image Generation for additional information.
- ***Sep*** identifies one alphanumeric character that prints continuously across the report to separate standard page headings, page frames, and standard column headings from the body of the report. If blank, APPX does not print a heading separator. The default is =.
- ***Filename*** establishes the name of the report file that is used to store the contents of the printed report. If necessary to guarantee uniqueness, APPX may append one or more characters to the value you enter here, or replace one or more of the ending characters. If two or more output processes *in the same subprocess family* are defined with identical *Print File* values, the processes share a common print file. They print as a single report, with continuous page numbers. If this specification is blank, APPX uses the last *Print File* that is active within the subprocess family. Refer to Process Lineage in Chapter 3-1 for a definition of subprocess family.
- ***Options*** are indicators that are present whenever the options they denote have been defined. Options applicable to this screen are AA (Additional Attributes), R (Rules), D (Documentation), and S (Security).

Buttons are provided at the bottom of each Output Processes screen for accessing the following options:

- ADDITIONAL ATTRIBUTES - See the Output Process Additional Attributes Option for details.
- RULES - See the Output Process Rules Option for details.
- DOCUMENTATION - Same as the Process Documentation Option described in Chapter 3-3.
- SECURITY - Same as the Process Security Option described in Chapter 3-3.
- GOTO... - Allows you to identify and retrieve another record by optionally entering a full or partial key.
- FRAMES - See Output Frame Specifications for details. This is the default option.
- HELP - Allows you to view online help text for a selected item.
- CLOSE or CANCEL - In Inquire mode, CLOSE allows you to end the current process. Otherwise, CANCEL allows you to end the current process without saving changes, if any, to the current record.

In addition, the menu bar provides the following pulldown menus:

- **Toolbox** - Same as the Process Toolbox Pulldown Menu described in Chapter 3-3.
- **Jump to** - Same as the Process Jump to Pulldown Menu described in Chapter 3-3.

Standard Column Heading Image Generation

If the *ColH* specification on the Output Processes screen is set to "yes" for a process, APPX will generate a standard column heading image at runtime for the process.

APPX's standard column heading image consists of one double-line row of column heading text. This row is populated based upon the items painted on the first row of a related record class frame. If there is additional space available in the column heading row, APPX examines each subsequent row in the record class frame, in order. If a row contains an item that can be placed in an available space in a column heading, the item's text is added to the column heading image. As a result, the final column heading image is actually a composite of heading text for fields from any combination of rows on a record class image.

APPX applies the following guidelines when formatting text on a standard column heading image:

- ***Single-line entries.*** If either line of the two-line *Column Hdgs* specification is blank, text occupies the line two (bottom) position in a column heading image.
- ***Justification.*** An item's justification depends on the type of item (numeric or logic, for example) and the value in the *Alignment* specification. Column headings for ***Logic*** items are always left-justified. Column headings for ***Date/time*** items are always left-justified. Column headings for ***Alphanumeric*** items are justified according to the *Alignment* specification in the data dictionary. Column headings for ***Numeric*** items are right-justified unless explicitly left-justified when painted on a screen in the APPX process facility.

- ***Truncation.*** APPX truncates column heading text if the text is longer than the item it relates to, always maintaining at least one blank position between column headings. APPX never leaves a leading blank for a left-justified item, or a trailing blank for a right-justified item.
- ***Underlines.*** For *Start of Range* class frames only, APPX places a single (dashed) line below column heading text for each item. The width of the underline is equal to the larger of the two lines of heading text for an item.

Output Process Additional Attributes Option

When you select the ADDITIONAL ATTRIBUTES *option* from the Output Processes screen, the Output Processes Additional Attributes overlay is displayed as shown in *Figure 3-7-3 below. This overlay permits the definition of more properties affecting the operation of the output process.*

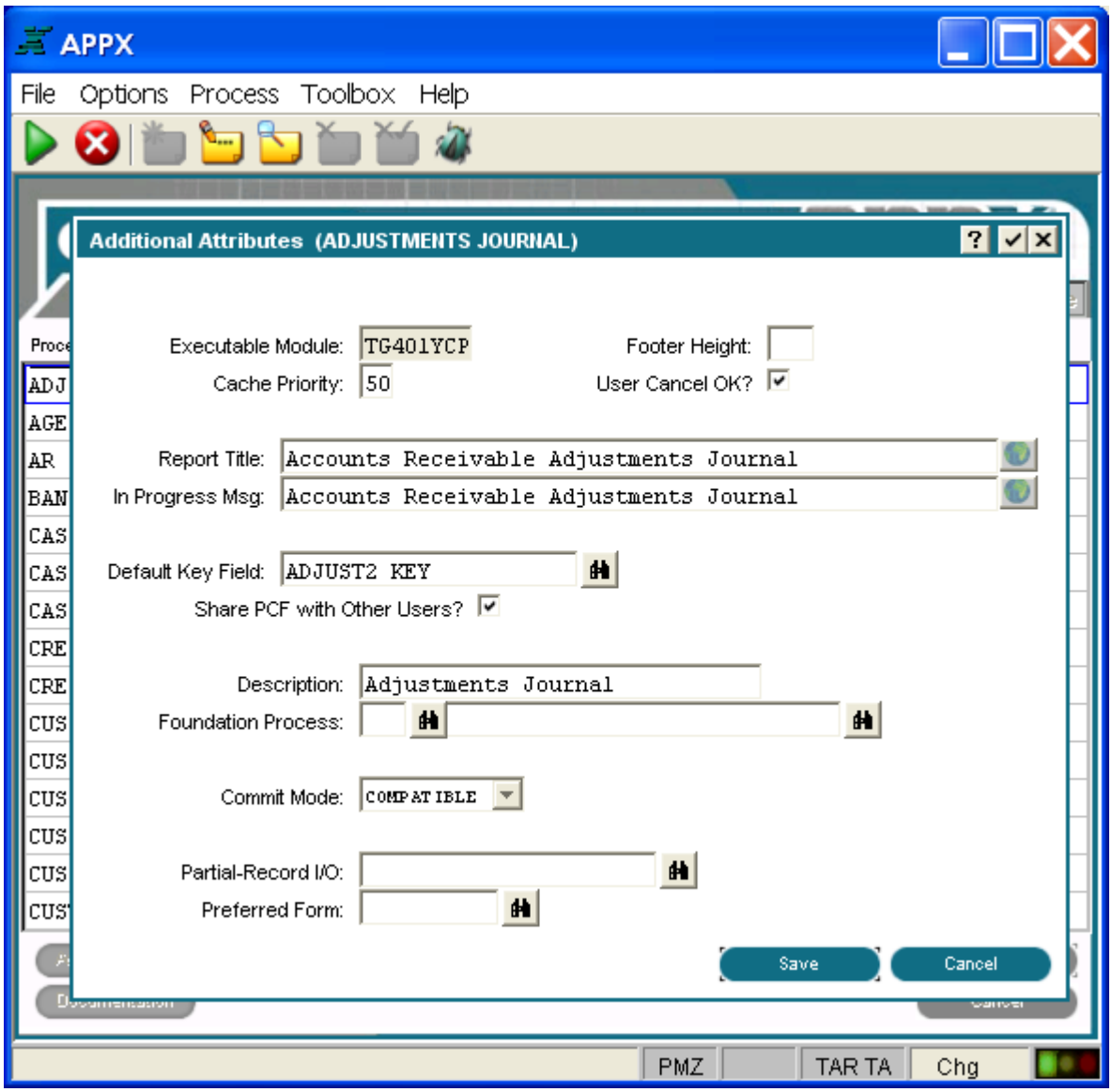


Figure 3-7-3. Output Processes Additional Attributes Overlay

The Additional Attributes overlay for Output processes contains the following fields:

- ***Executable Module*** displays the name of the executable module (EM) that is automatically generated by APPX. APPX derives the file name from the date and time the currently selected process was added. The name consists of a unique string of eight alphanumeric characters and cannot be changed. If APPX finds it necessary to recreate the EM, the new EM retains the same file name. See Saving Executable Modules in Chapter 3-2 for more information.
- ***Footer Height*** indicates the amount of space, in number of rows, to reserve on each page for printing the page footer. APPX uses this value to determine placement of *End of Page* frames. If blank, APPX does not reserve space for a page footer.
- ***Cache Priority***, represented by two numeric characters, establishes the likelihood that the process will remain in memory after execution, relative to other processes that the user has also completed executing. A process with a cache priority of *99* is most likely to remain in memory. A process with a cache priority of *1* is least likely to remain in memory. A cache priority of *0* indicates that the process is never cached. The default is 50.
- ***User Cancel OK?*** contains either "yes" ☑ or "no" ☐ and determines whether or not a user can press the CANCEL option key to stop the execution of this process. The default is 'yes". You can conditionally control the user's ability to select the CANCEL option with statements and the predefined field CANCEL OK. You can toggle between "yes" and "no" by clicking on the checkbox.
- ***Report Title*** defines the text that is used to name the report. This text automatically displays in the standard image for a *Start of Page* frame which is printed at the top of each page of the output.
- ***In Progress Msg*** provides text for the message that displays to the screen while the report is executing in the foreground.
- ***Default Key Field*** designates the specific keypath that is initially presented to a user invoking this process. If blank, the primary key is used. This specification is overridden by the default keypath that is defined for the child invocation of the process.
- ***Share PCF with Other Users?*** determines whether other users can update the file concurrently. The default is "yes" ☑ but can be toggled to "no" ☐ by clicking the checkbox.
- ***Description*** is incorporated in the message that displays on the screen when the output process is compiling. It is used to communicate with users since it provides a more descriptive and easily understood alternative to *Process Name*, which is the default if this field is left blank. The Additional Attributes overlay for all types of processes contain this field. See *Chapter 3-3, Process Definitio*n, for more information.
- ***Foundation Process*** designates another output process (application name and process name) to provide *Start of Report, End of Report, Start of Page* and *End of Page* frames for the current output process. If this field is left blank, the report and page frames of the current process will be used. Note that if the current output process executes as a subprocess child of another output process, the process assumes the report and page frames of the parent process.
- ***Commit Mode*** controls the commit mode for a process and can be selected from a drop-down list of available options. *Commit Mode* determines those points in time when APPX will commit changes

made to the tables stored within an RDBMS. It is only applicable in conjunction with an RDBMS; it has no effect with APPXIO. Also, not all File Managers support this feature. Although initially set here for the process, it can be added or overridden by use of predefined field COMMIT MODE. The following options are available:

COMPATIBLE .. Changes are committed as soon as all locks are freed. This provides maximum compatibility between APPXIO files and RDBMS tables.

AUTOCOMMIT . APPX will commit each modification as it is made.
***This value will be implemented in a future release.**

PCF CYCLE ..... APPX will commit modifications as each PCF record is processed.

END ..... APPX will only commit modifications at the end of the process family.

MANUAL ..... APPX will not automatically commit changes. You must use the COMMIT statement to do so.

- ***Partial-Record I/O*** is used to specify a field from the PCF file so that instead of operating on the entire PCF record, APPX operates on the designated field only. If the field is a group header, then I/O will be performed on all members of the group. Utilizing partial-record I/O can greatly improve performance when the PCF is a table in an RDBMS. This is especially true when the table contains many columns, but relatively few are being utilized by the process. The SCAN option is available for field selection.
- ***Preferred Form*** is used to help tune the Image Editor session to match form specifications when designing for PDF output.

Output Process Rules Option

When you select the RULES *option* from the Output Processes screen, the Output Processes Event Points for Rules overlay is displayed as shown in *Figure 3-7-4 below.* This overlay provides options to access the ILF Editor for APPX event points. If statements are defined in an event point, the respective text on the screen is highlighted.

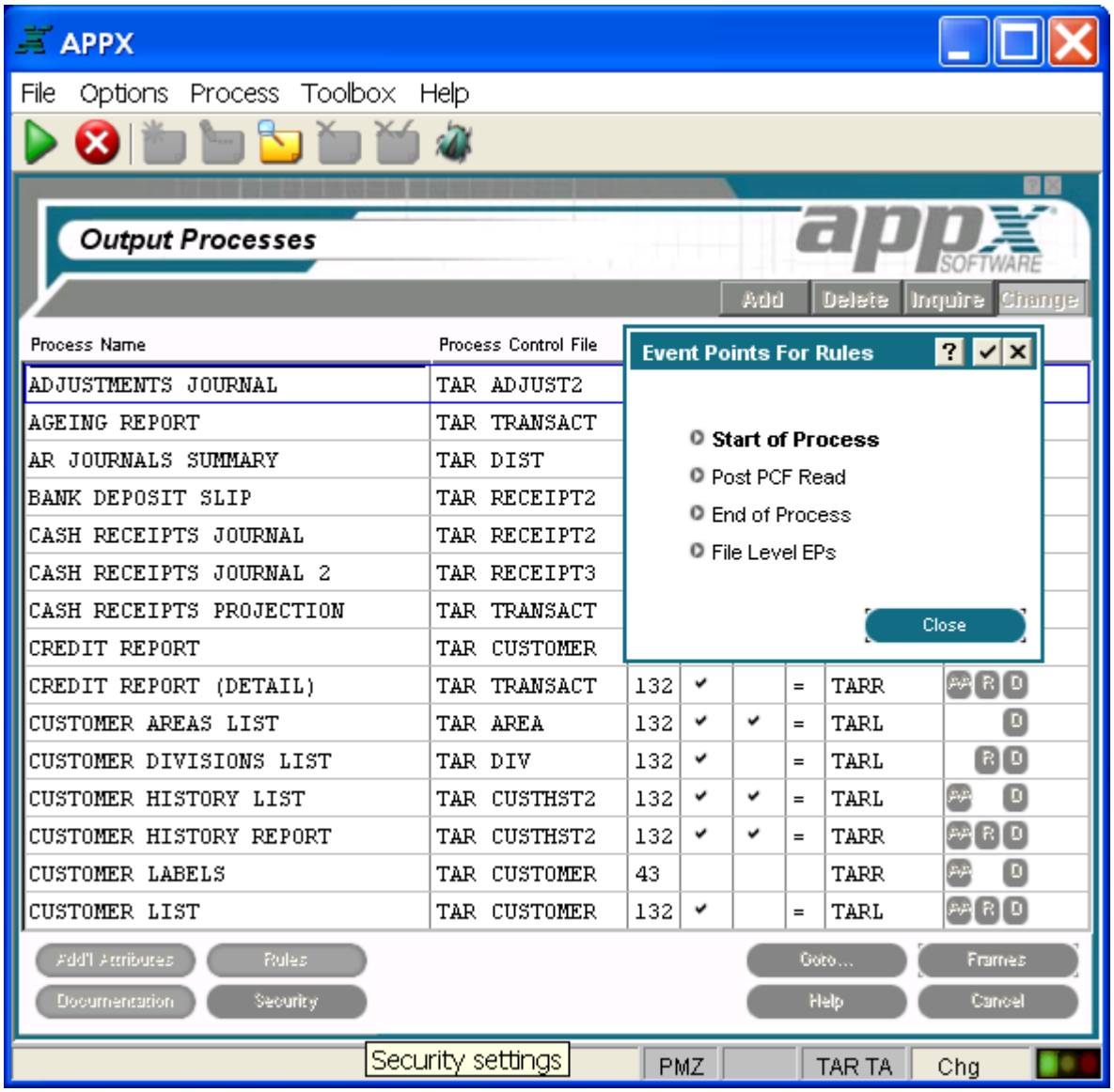


Figure 3-7-4. Output Processes Event Points for Rules Overlay

The Output Processes Event Points for Rules overlay contains the following options:

Start of Process

START OF PROCESS accesses the ILF editor for the *Start of Process* event point. Upon completion, you are returned to the Event Points overlay.

Post PCF Read

POST PCF READ accesses the ILF editor for the *Post PCF Read* event point. Upon completion, you are returned to the Event Points overlay.

End of Process

END OF PROCESS accesses the ILF editor for the *End of Process* event point. Upon completion, you are returned to the Event Points overlay.

File Level EPs

FILE LEVEL EPs provides access to the File Level Event Points for the PCF file. These event points are not directly associated with the process but with the PCF file. See the Files Rules Option *in Chapter 2-4* for more information on using these event points.

Refer to Chapter 3-16: Using Event Points for more information about event points.

Output Frame Specifications

When you select the FRAMES option (or press RETURN) on the Output Processes screen, after completing specifications for an output process, the Output Process Frames screen is displayed as shown in *Figure 3-7-5 below. Out*put frame specifications define attributes that govern both the visual presentation of a frame and the way it executes.

Output Process Frames screen

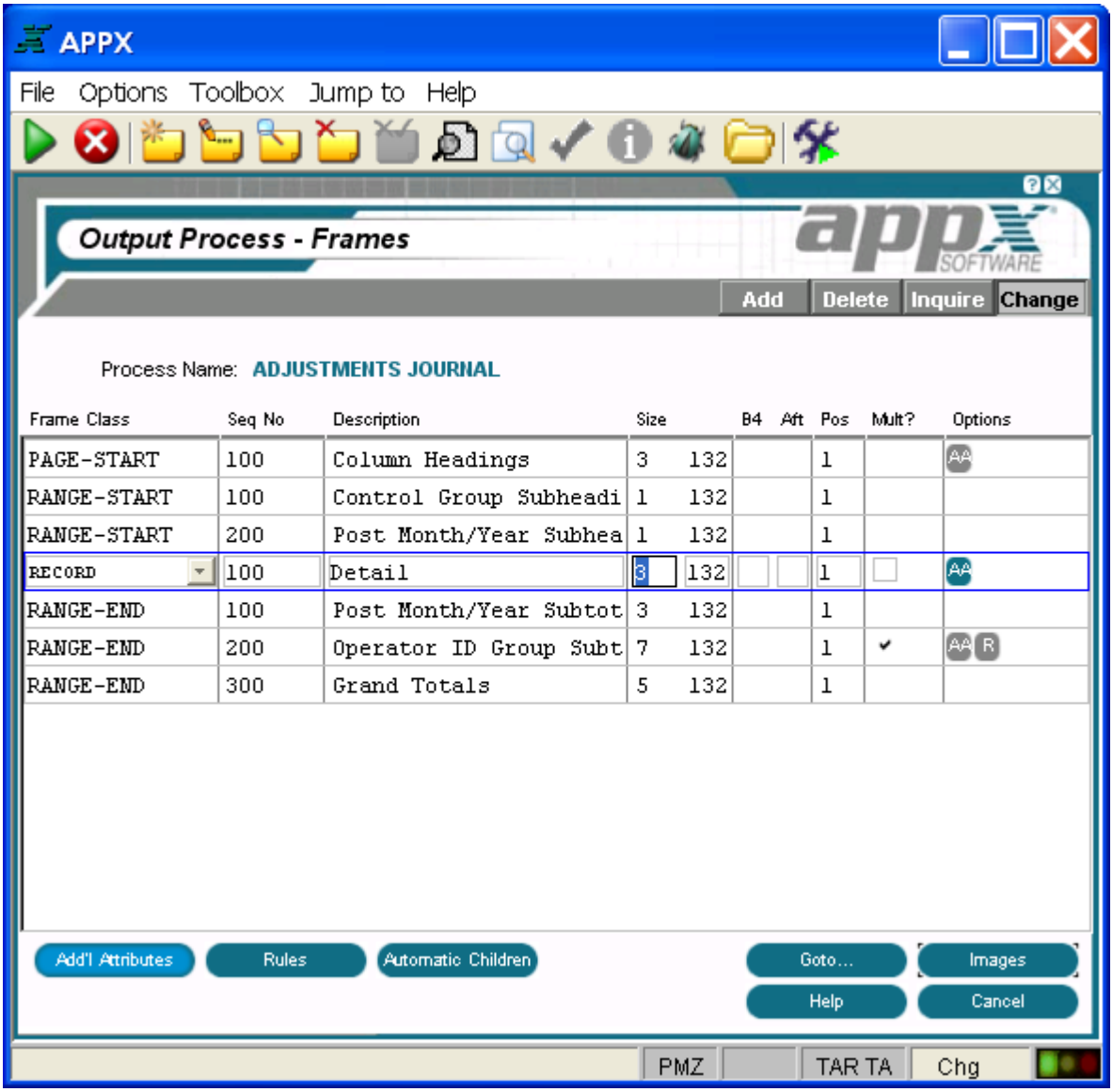


Figure 3-7-5

The Output Process Frames screen contains the following fields:

- ***Frame Class*** defines the type of frame with which you are working. Valid options are:

Report-Start

Frames print once at the beginning of a report. To print on a separate page, the designer should set *New Page After?* in Output Frame Additional Attributes for the last *Report-Start* frame.

Page-Start

Frames print once at the beginning of each page of a report. They are primarily used for page headings and column headings. Standard images for both of these are optionally available. See Standard Column Heading Image Generation for more information on column headings.

Range-Start

Frames print immediately before the first record for which the value in a specific sort field (the range field) changes. They are primarily used for subheadings and lower-level column headings. Standard images for both of these are optionally available. See Standard Column Heading Image Generation for more information on column headings.

Record

Frames manipulate and print information read from the process control file. They comprise the body of a report. *Record* frames can be defined to scroll.

Range-End

Frames print immediately after the last record in a series of records that have the same value in a specific sort field (the range field). They are primarily used for subtotals, and a standard subtotal image is optionally available.

Page-End

Frames print once at the conclusion of each page of a report.

Report-End

Frames print once at the conclusion of a report. To print on a separate page, the designer should set *New Page Before?* in Output Frame Additional Attributes for the first *Report-End* frame.

- ***Seq No*** displays a sequence number that is defaulted by APPX and used to identify this output frame. The default sequence number can be overridden, but must be unique within each frame class for the process. Sequence numbers determine the order of the frames in the process structure. The first frame that is defined in add mode is assigned sequence number *100*, and additional frames are assigned a number equal to the highest existing sequence number, plus *100*.
- ***Description*** identifies this output frame. The description is for documentation purposes only.
- ***Size*** fields designate the number of rows and columns that comprise the frame. The row default is 1, and the column default corresponds to the report width entered for the output process, if any.
- ***B4*** indicates the number of blank lines to print before executing this frame. The default is *0*. In some cases, such as optional or repeating frames, it may be more appropriate to specify lines to skip at the image level. See the Scrolling or Non-Scrolling Image Additional Attributes options as applicable.
- ***Aft*** indicates the number of blank lines to print subsequent to executing this frame. The default is *0*. In some cases, such as optional or repeating frames, it may be more appropriate to specify lines to

skip at the image level. See the Scrolling or Non-Scrolling Image Additional Attributes options as applicable.

- ***Pos*** designates the placement of the frame relative to the far left-side position of the report. See Frame/Image Positioning in *Chapter 3-2* for more information.
- ***Mult?*** A "yes" ☑ in this field indicates that this output frame contains more than one image. If "no" ☐ is selected, you will be unable to add more than one entry at the image level. The default is "no" but can be toggled to "yes" by clicking the checkbox.
- ***Options*** are indicators that are present whenever the options they denote have been defined. Options applicable to this screen are AA (Additional Attributes), R (Rules), and AC (Automatic Children).

Buttons are provided at the bottom of each Output Process Frames screen for accessing the following options:

- ADDITIONAL ATTRIBUTES - See the Output Frame Additional Attributes Option for details.
- RULES - See the Output Frame Rules Option for details.
- AUTOMATIC CHILDREN - See the Output Frame Automatic Children Option for details.
- GOTO... - Allows you to identify and retrieve another record by optionally entering a full or partial key.
- IMAGES - See Output Image Specifications for details. This is the default option.
- HELP - Allows you to view online help text for a selected item.
- CLOSE or CANCEL - In Inquire mode, CLOSE allows you to end the current process. Otherwise, CANCEL allows you to end the current process without saving changes, if any, to the current record.

In addition, the menu bar provides the following pulldown menus:

- **Toolbox** - See the Output Frame Toolbox Pulldown Menu for details.
- **Jump to** - Same as the Process Jump to Pulldown Menu described in Chapter 3-3.

Output Frame Additional Attributes Option

When you select the ADDITIONAL ATTRIBUTES option on the Output Process Frames screen for a frame with frame class ***Record***, the output process *Record* frame Additional Attributes overlay is displayed as shown in *Figure 3-7-6 below,* where you can access additional parameters.

The *Record* overlay also applies for frames with frame class ***Report-Start***, ***Page-Start***, ***Page-End***, or ***Report-End***, except that these frame classes do not have a *Scroll Type* parameter.

See *Figure 3-7-7* at the end of this section for the overlay that applies for frames with frame class ***Range-Start*** or ***Range-End***.

Figure 3-7-6. Output Record Frame Additional Attributes Overlay

The output *Record* frame Additional Attributes overlay contains the following fields.

- ***Frame Type*** governs the execution of an image or images for the frame. There are three frame types.

NORMAL ..... Executes the first (lowest numbered) alternate image unless another image is explicitly selected with statements and the predefined field ALTERNATE IMAGE NUMBER.

OPTIONAL ..... Executes the image that is identified by the *Normal Frame Type* above with the following exception: If the value in the predefined field ALTERNATE IMAGE NUMBER is *0*, do not execute an image.

REPEATING ... Executes the image repeatedly until the value in the ALTERNATE IMAGE NUMBER is set to *0* by statements.

The default is *Normal.* See *Frames* in Chapter 3-1 for more information about optional and repeating frames.

- ***Scroll Type*** indicates whether or not the images on the frame scroll and, if so, the type of scrolling to be performed. The field only appears for a ***Record*** frame. The *Scroll Type* options are:

NONE ..... The image on the frame does not scroll.

DOWN ..... APPX divides the frame into a grid of equal-sized regions, based upon a calculated composite image size. The first image is placed into the upper left region in the frame. Succeeding images are placed in the region below the previous image until the column is complete. As each column is completed, the next image is placed at the top of the column immediately to the right of the completed column.

ACROSS ..... APPX divides the frame into a grid of equal-sized regions, based upon a calculated composite image size. The first image is placed into the upper left region in the frame. Succeeding images are placed in the region to the right of the previous image until the row is complete. As each row is completed, the next image is placed in the far left-hand position of the row immediately below the completed row.

The default is *None. It is uncommon to use scrolling images in output processes since the desired effect can usually be accomplished by reducing the size of the frame relative to the report width. See* The Effect of Scrolling Frames on Screen and Page Positioning section of *Chapter 3-2* for a detailed discussion of scrolling frames.

- ***New Page Before?*** determines whether or not to begin a new page before executing this frame. If the report is between pages before this frame executes, a "yes" ☑ entry does not cause a blank page to print. The default is "no" ☐.
- ***New Page After?*** determines whether or not to begin a new page subsequent to executing this frame. If the report is between pages after this frame executes, a "yes" ☑ entry does not cause a blank page to print. The default is "no" ☐.
- ***Skip If Live Operations?*** determines whether or not the frame is exclusive to the initial setup and recovery processing phases of operation. If "yes" ☑, the frame does not execute during the live operations phase. If "no" ☐, the frame executes during all phases of operation. The default is "no".
- ***Form Position (Row)*** designates the placement of the frame on the printed format. When implemented, the row is defined relative to the rows on the printed page (unless Print Beside is in effect, in which case the row is relative to the Print Beside buffer). See Frame/Image Positioning in *Chapter 3-2* for more information.
- ***Start Print Beside?*** A "yes" ☑ in this checkbox tells APPX to print subsequent frames to the right of this frame if physically possible. See the Print Beside section in *Chapter 3-2* for more information about this feature.

When you select the ADDITIONAL ATTRIBUTES option on the Output Process Frames screen for a frame with frame class ***Range-Start*** or ***Range-End***, a slightly different Additional Attributes overlay is displayed as shown in *Figure 3-7-7 below*.

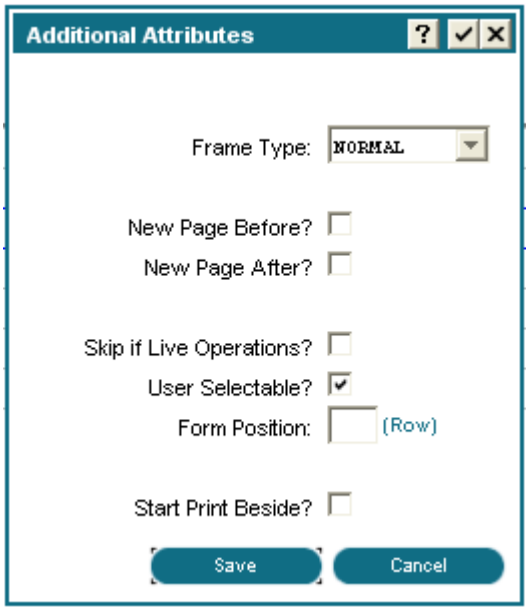


Figure 3-7-7. Output Range-Start and Range-End Frame Additional Attributes Overlay

The output *Range-Start* and *Range-End* frame Additional Attributes overlay contains the same fields described above for a *Record* frame, except that there is no *Scroll Type* field. Also, it contains the following field that is unique to *Range-Start* and *Range-End* frames:

- ***User Selectable?*** indicates whether or not a user can control printing of the frame through a query. If "no" ☐, the user cannot turn off this frame in the query. If “yes” ☑, the user can optionally cause the frame to print or not. The default is “yes”.

Output Frame Rules Option

When you select the RULES option on the Output Process Frames screen for any frame regardless of frame class, the output process frame Event Points for Rules overlay is displayed as shown in *Figure 3-7-8 below*. This overlay provides options to access the ILF Editor for APPX event points. If statements are defined in an event point, the respective text on the screen is highlighted.

Figure 3-7-8. Output Frame Event Points for Rules Overlay

The output frame Event Points for Rules overlay contains the following options:

Start of Process

START OF PROCESS accesses the ILF editor for the *Start of Process* event point. Upon completion, you are returned to the Event Points overlay.

Post PCF Read

POST PCF READ accesses the ILF editor for the *Post PCF Read* event point. Upon completion, you are returned to the Event Points overlay.

Select Image

SELECT IMAGE accesses the ILF editor for the *Select Image* event point. Upon completion, you are returned to the Event Points overlay.

Global EP's For Images

GLOBAL EP'S FOR IMAGES calls the Global Event Points overlay shown in *Figure 3-7-9 below*.

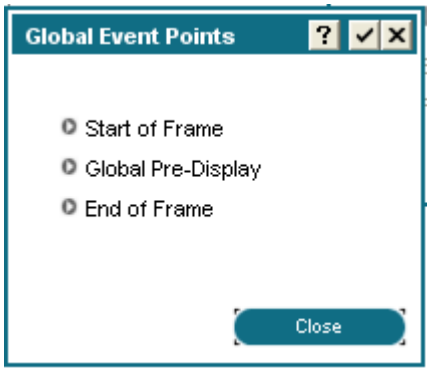


Figure 3-7-9. Global Event Points Overlay

This overlay provides options for three additional event points:

- START OF FRAME accesses the ILF editor for the *Start of Frame* event point. Upon completion, you are returned to the Global Event Points overlay.
- GLOBAL PRE-DISPLAY accesses the ILF editor for the *Global Pre-Display* event point. Upon completion, you are returned to the Global Event Points overlay.
- END OF FRAME accesses the ILF editor for the *End of Frame* event point. Upon completion, you are returned to the Global Event Points overlay.

End of Process

END OF PROCESS accesses the ILF editor for the *End of Process* event point. Upon completion, you are returned to the Event Points overlay.

File Level EPs

FILE LEVEL EPS provides access to the File Level Event Points for the PCF file. These event points are not directly associated with the process but with the PCF file. See the Files Rules Option *in Chapter 2-4* for more information on using these event points.

Refer to Chapter 3-16: Using Event Points for more information about event points.

Output Frame Automatic Children Option

When you select the AUTOMATIC CHILDREN option on the Output Process Frames screen for any frame regardless of frame class, the Output Process Automatic Children screen is displayed as shown in *Figure 3-7-10*. This allows you to attach a specific process to the output process frame currently selected so that it invokes automatically following execution of the frame. See Automatic and Optional Child Invocations in Chapter 3-1 as well as Chapter 3-13: Automatic and Optional Children for more information.

Output Process Automatic Children screen

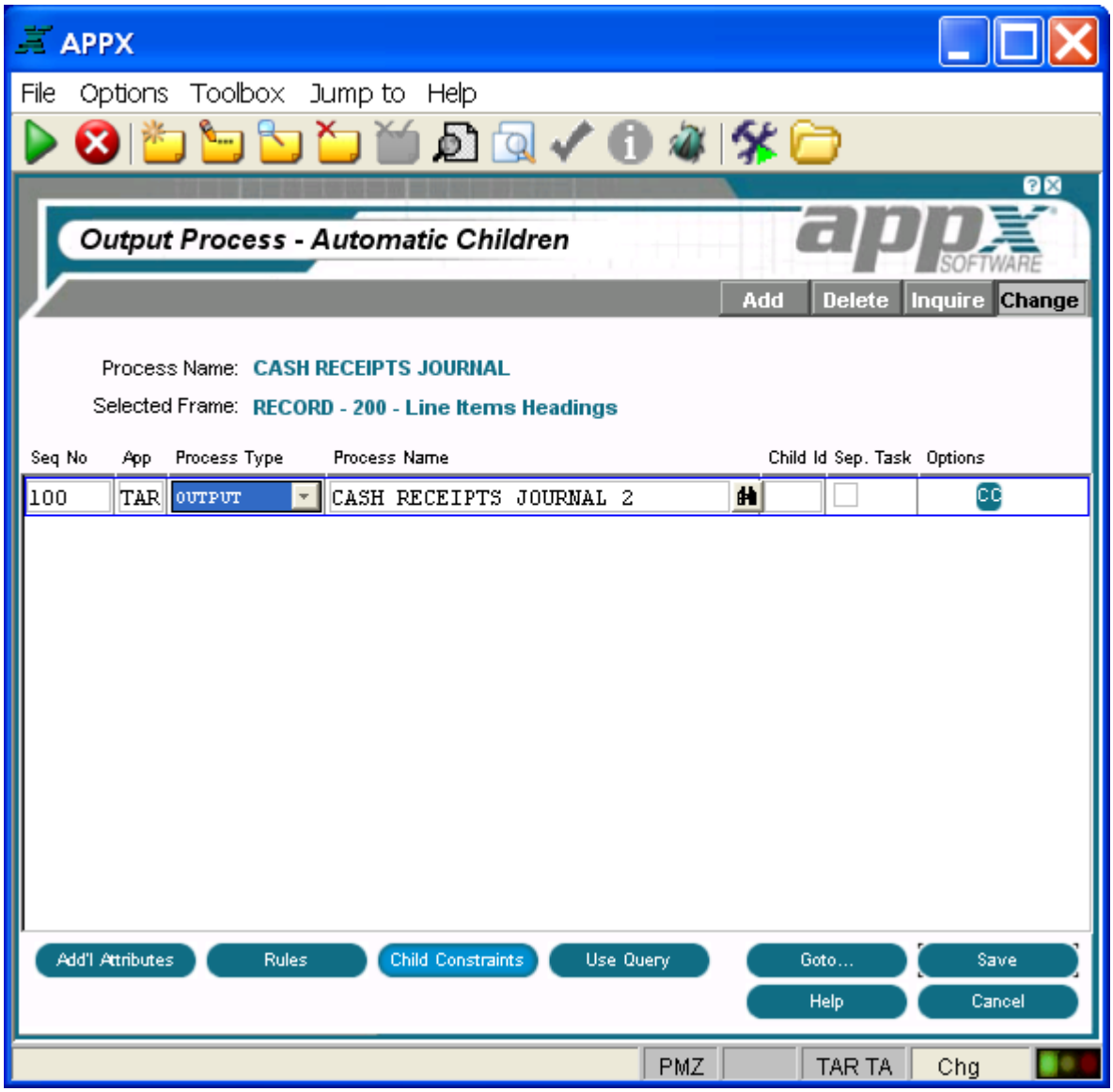


Figure 3-7-10

Output Frame Toolbox Pulldown Menu

When you select **Toolbox** from the menu bar on the Output Process Frames screen, the Toolbox Pulldown Menu appears as shown in *Figure 3-7-11* below. Note that each tool can also be accessed by selecting the corresponding icon on the screen toolbar.

Output Frames Toolbox Pulldown menu

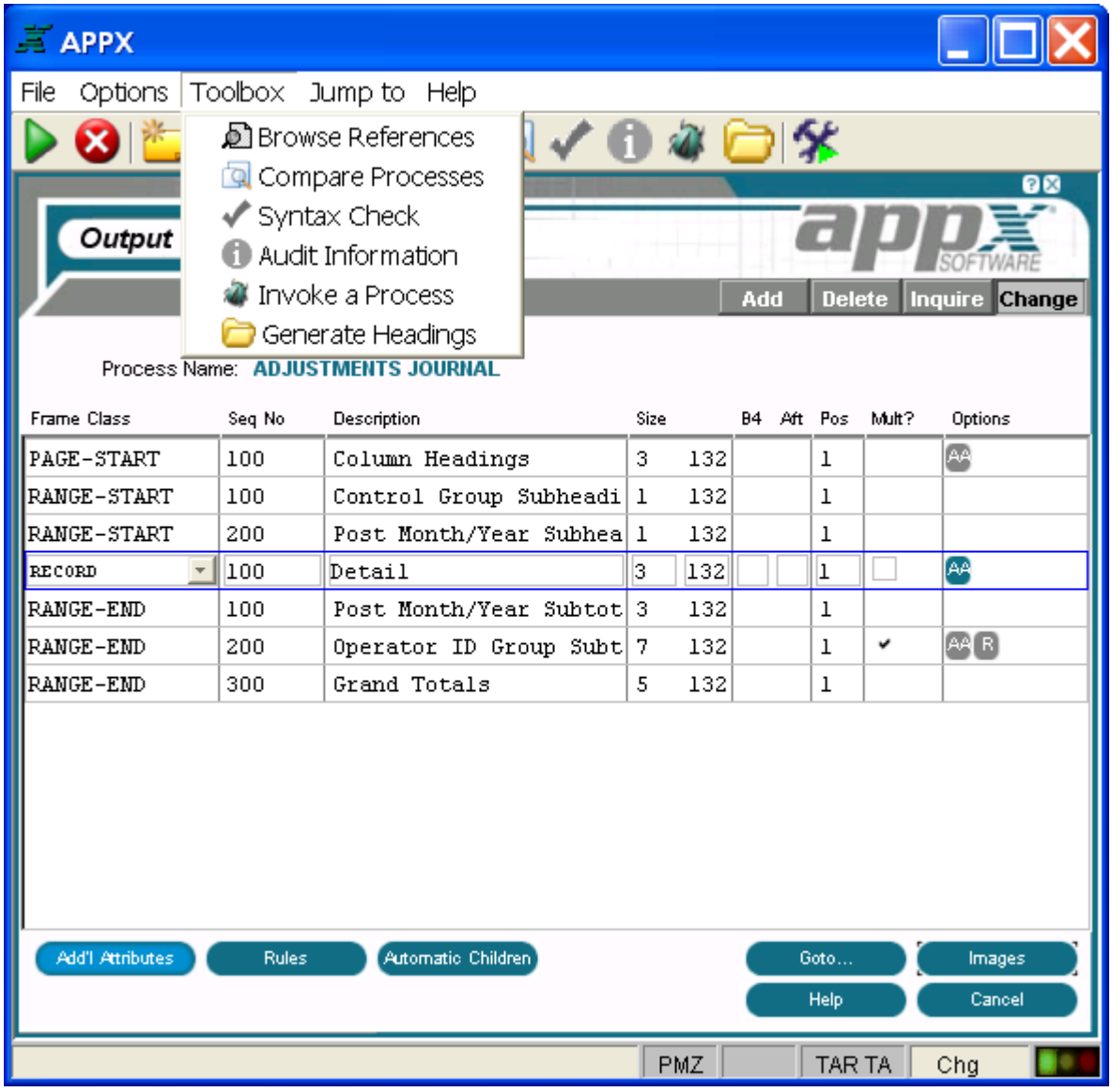


Figure 3-7-11

The output frames Toolbox Pulldown Menu offers the following options:

Browse References

The BROWSE REFERENCES option runs a modified version of "Cross Reference – All" and produces a list of design components utilized by the current process. The results can be displayed interactively where links allow you to see the context of the references, edit the referenced component, and drill down to the child process. This option can be used to identify, and optionally edit, the file level event points utilized by this process.

Compare Processes

The COMPARE PROCESSES option compares the current process to another process that you identify and produces a report listing any differences. You can also opt to browse the results interactively.

Syntax Check

The SYNTAX CHECK option allows you to create the Executable Module for the current process without running the process. If the Executable Module already exists and is still valid, this option is ignored if invoked. Note that this option (`OPTION 97`), like any Toolbox option with a value greater than 90, is available for use directly from any screen in Application Design without going to the Toolbox.

Audit Information

The AUDIT INFORMATION option allows you to see who created the current process and who changed it most recently. It also provides you with the date that the process was created and the date that it was last changed. Note that this option (`OPTION 98`), like any Toolbox option with a value greater than 90, is available for use directly from any screen in Application Design without going to the Toolbox. See Audit Information Overlay for an example of the overlay.

Invoke a Process

The INVOKE A PROCESS option is available to facilitate testing. When you select this option, APPX defaults to the current process but you may override the defaults to invoke any process. Note that this option, like any Toolbox option with a value greater than 90, is available for use directly from any screen in Application Design without going to the Toolbox. See Invoke a Process Overlay for details on using this option.

Generate Headings

The GENERATE HEADINGS option generates a column headings frame for the output frame you are currently positioned to.

Output Image Specifications

Output image specifications define the visual and operational attributes of each alternate image for an output frame. These specifications are determined, in part, by whether or not the currently selected frame is defined with a scroll type in the Output Frame Additional Attributes accessed from the Output Process Frames screen. However, it is unusual to employ scrolling in an output process.

The scrolling output Images screen and the non-scrolling output Images screen are described separately in the following sections:

- Scrolling Output Image Specifications
- Non-Scrolling Output Image Specifications

Scrolling Output Image Specifications

When you select the IMAGES option (or press RETURN) on the Output Process Frames screen for a frame with scrolling defined in Additional Attributes, the Scrolling Output Images screen is displayed as shown in *Figure 3-7-12 below. Note that it is uncommon to use scrolling images in output processes since the desired effect can usually be accomplished by reducing the size of the frame relative to the report width.*

Scrolling Output Images screen

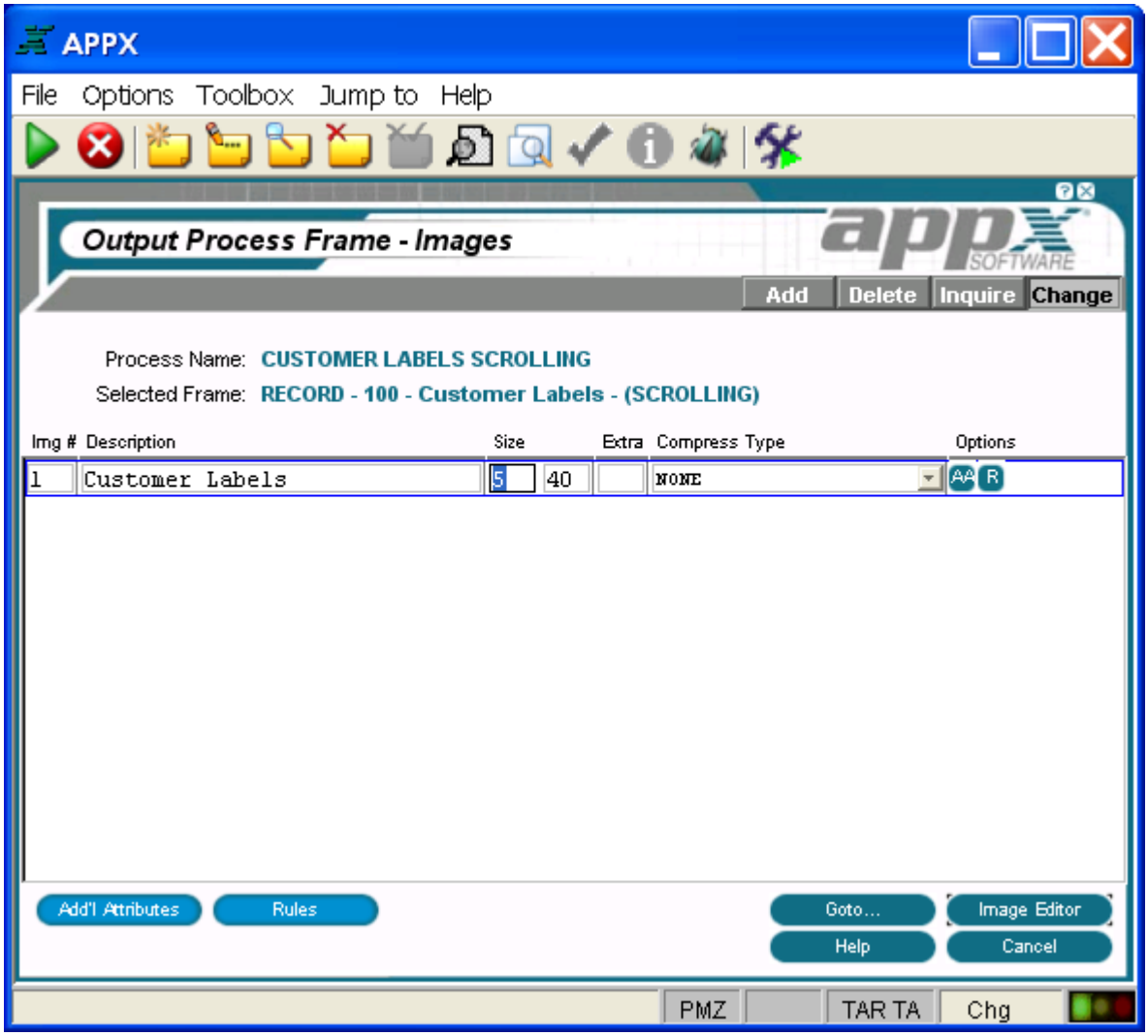


Figure 3-7-12

The scrolling output Images screen contains the following fields:

- ***Img #*** displays a sequence number that is defaulted by APPX. The image number uniquely identifies each alternate image for the frame and is referenced when determining which image, if any, is to display. The first image that is defined in add mode is assigned alternate image number *1*, and additional images are assigned a number equal to the highest existing image number, plus *1*. The default number can be overridden, but must be unique within the frame. Refer to the Images section in *Chapter 3-1* for a discussion of alternate images.
- ***Description*** identifies this alternate image. The description is for documentation purposes only.
- ***Size*** designates the number of rows and columns that comprise the displayed image. The image cannot be larger than the frame to which it belongs. If blank, the size of the image defaults to the size of the frame. The default is blank. See Frame/Image Positioning in *Chapter 3-2* for more information.
- ***Extra*** indicates the number of extra lines or rows to add to the size of the image. For scrolling frames only, APPX computes the composite image size based upon the *Size* entry and this entry.
- ***Compress Type*** indicates whether or not to suppress blank lines in an image and, if so, the type of lines to compress. The *Compress Type* options are:

NONE ..... Do not compress.

EMBEDDED BLANK LINES ... Compress embedded blank lines.
(However, the image still occupies the full number of rows.)

TRAILING BLANK LINES ..... Compress trailing blank lines.

EMBEDDED/TRAILING ..... Compress embedded and trailing blank lines.

- ***Options*** are indicators that are present whenever the options they denote have been defined. Options applicable to this screen are AA (Additional Attributes) and R (Rules).

Buttons are provided at the bottom of each Images screen for accessing the following options:

- ADDITIONAL ATTRIBUTES - See the Scrolling Output Image Additional Attributes Option for details.
- RULES - See the Output Image Rules Option for details. This overlay is the same for both scrolling and non-scrolling frames.
- GOTO... - Allows you to identify and retrieve another record by optionally entering a full or partial key.
- IMAGE EDITOR - See the Output Image in the Image Editor for details. This is the default option.
- HELP - Allows you to view online help text for a selected item.
- CLOSE or CANCEL - In Inquire mode, CLOSE allows you to end the current process. Otherwise, CANCEL allows you to end the current process without saving changes, if any, to the current record.

In addition, the menu bar provides the following pulldown menus:

- **Toolbox** - Same as the Output Frame Toolbox Pulldown Menu except that there is no GENERATE HEADINGS option at the image level.
- **Jump to** - Same as the Process Jump to Pulldown Menu described in Chapter 3-3.

Scrolling Output Image Additional Attributes Option

When you select the ADDITIONAL ATTRIBUTES option on the Scrolling Output Images screen, the scrolling output image Additional Attributes overlay is displayed as shown in *Figure 3-7-13 below,* where you can access additional parameters.

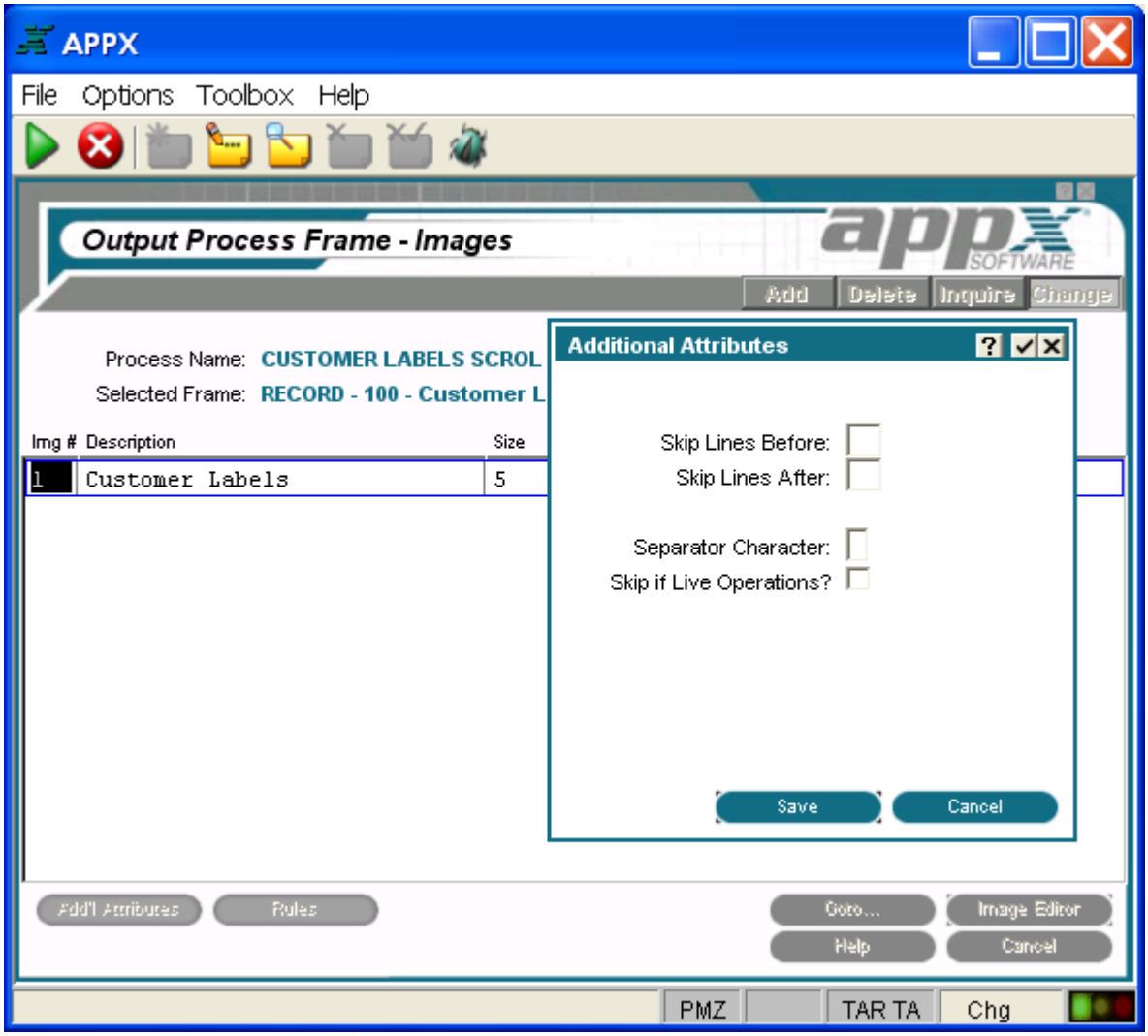


Figure 3-7-13. Scrolling Output Image Additional Attributes Overlay

The scrolling output image Additional Attributes overlay contains the following options:

- ***Skip Lines Before*** represents the number of lines to be skipped prior to the image being printed.

- ***Skip Lines After*** defines the number of lines to be skipped after the image is printed.
- ***Separator Character*** indicates whether a continuous line of a designated character (usually -, =, or *) should be following this image to separate it from the rest of the report. This field is optional and leaving it blank indicates that no separation line is to be printed.
- ***Skip If Live Operations?*** determines whether the frame is exclusive to the initial setup and recovery processing phases of operation. If "yes" ☑, the frame is not executed during live operations. If "no" ☐, the frame executes during all phases of operation. The default is "no".

Non-Scrolling Output Image Specifications

When you select the IMAGES option (or press RETURN) on the Output Process Frames screen for a frame with no scrolling defined in Additional Attributes, the Non-Scrolling Output Images screen is displayed as shown in *Figure 3-7-14 below. Most output images are non-scrolling.*

Non-Scrolling Output Images screen

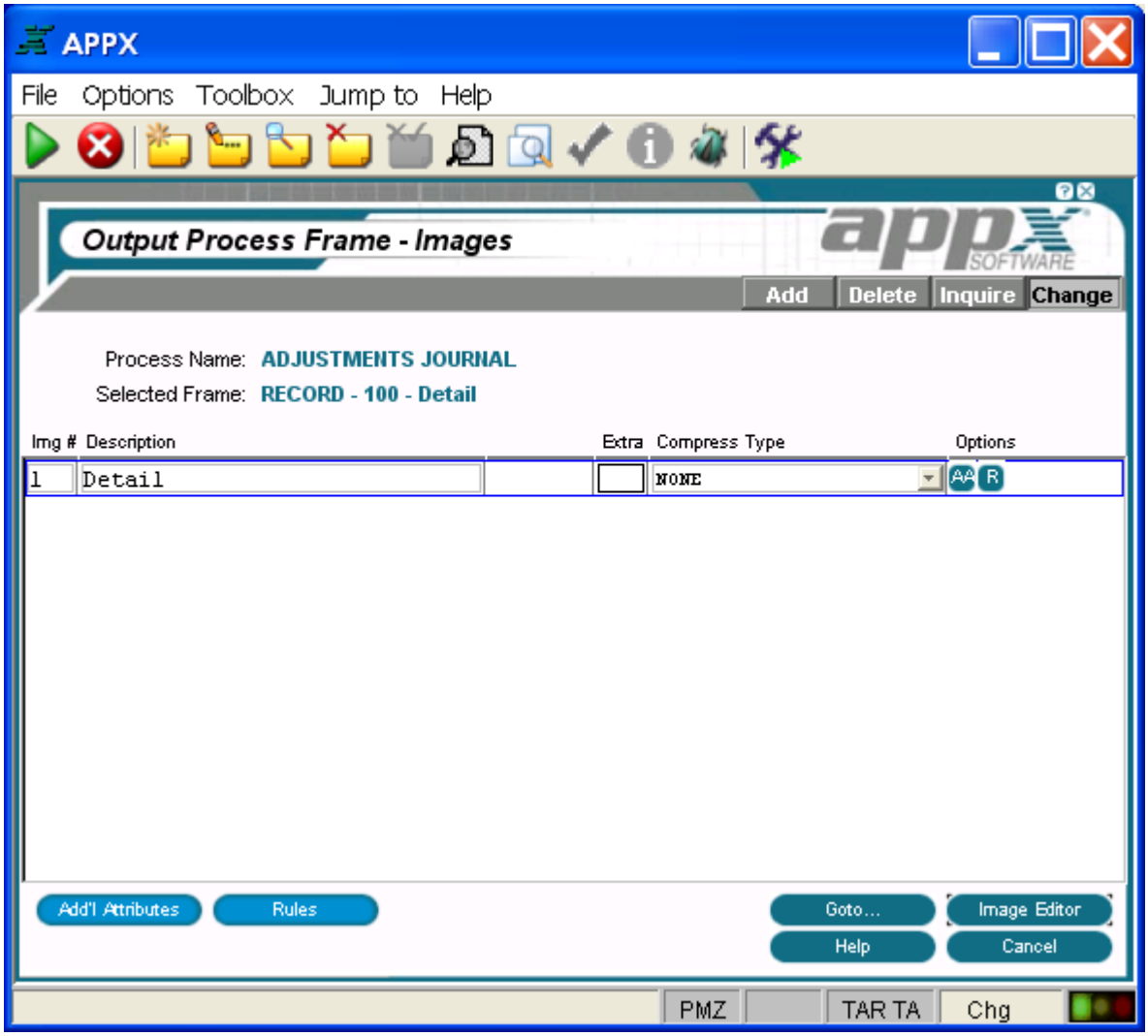


Figure 3-7-14

The non-scrolling output Images screen contains the following fields:

- ***Img #*** displays a sequence number that is defaulted by APPX. The image number uniquely identifies each alternate image for the frame and is referenced when determining which image, if any, is to display. The first image that is defined in add mode is assigned alternate image number *1*, and additional images are assigned a number equal to the highest existing image number, plus *1*. The default number can be overridden, but must be unique within the frame. Refer to the Images section in *Chapter 3-1* for a discussion of alternate images.
- ***Description*** identifies this alternate image. The description is for documentation purposes only.
- ***Extra*** indicates the number of extra lines or rows to add to the size of the image. For scrolling frames only, APPX computes the composite image size based upon the *Size* entry and this entry.
- ***Compress Type*** indicates whether or not to suppress blank lines in an image and, if so, the type of lines to compress. The *Compress Type* options are:

NONE ..... Do not compress.

EMBEDDED BLANK LINES ... Compress embedded blank lines.
(However, the image still occupies the full number of rows.)

TRAILING BLANK LINES ..... Compress trailing blank lines.

EMBEDDED/TRAILING ..... Compress embedded and trailing blank lines.

- ***Options*** are indicators that are present whenever the options they denote have been defined. Options applicable to this screen are AA (Additional Attributes) and R (Rules).

Buttons are provided at the bottom of each Images screen for accessing the following options:

- ADDITIONAL ATTRIBUTES - See the Non-Scrolling Output Image Additional Attributes Option for details.
- RULES - See the Output Image Rules Option for details. This overlay is the same for both scrolling and non-scrolling frames.
- GOTO... - Allows you to identify and retrieve another record by optionally entering a full or partial key.
- IMAGE EDITOR - See the Output Image in the Image Editor for details. This is the default option.
- HELP - Allows you to view online help text for a selected item.
- CLOSE or CANCEL - In Inquire mode, CLOSE allows you to end the current process. Otherwise, CANCEL allows you to end the current process without saving changes, if any, to the current record.

In addition, the menu bar provides the following pulldown menus:

- **Toolbox** - Same as the Output Frame Toolbox Pulldown Menu except that there is no GENERATE HEADINGS option at the image level.
- **Jump to** - Same as the Process Jump to Pulldown Menu described in Chapter 3-3.

Non-Scrolling Output Image Additional Attributes Option

When you select the ADDITIONAL ATTRIBUTES option on the Non-Scrolling Output Images screen, the non-scrolling output image Additional Attributes overlay is displayed as shown in *Figure 3-7-15 below,* where you can access additional parameters.

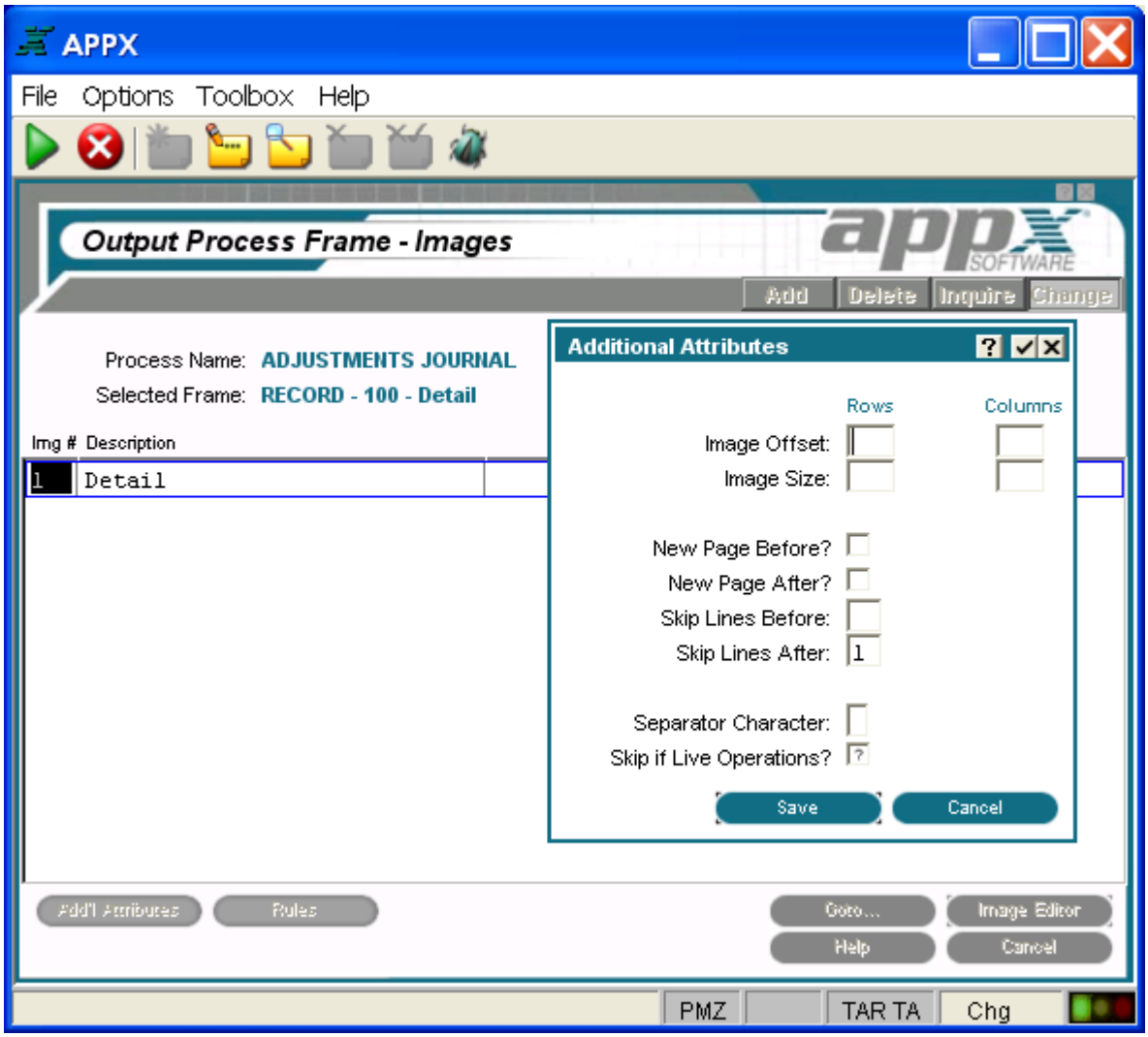


Figure 3-7-15. Non-Scrolling Output Image Additional Attributes Overlay

The non-scrolling output image Additional Attributes overlay contains the following options:

- ***Image Offset*** determines the placement of the rows and columns of this image, relative to the frame. Row entries designate positioning, relative to the top of the frame. Column entries designate positioning relative to the left of the frame. If blank, the APPX defaults to same row/column position as the frame. The default is blank. See Frame/Image Positioning in *Chapter 3-2* for more information.
- ***Image Size*** designates the number of rows and columns that comprise the displayed image. The image cannot be larger than the frame it belongs to. If blank, the size of the image defaults to the size of the frame. The default is blank.
- ***New Page Before?*** indicates whether or not a new page should be started before each time this image is printed. Click the checkbox to toggle between "yes" ☑ and "no" ☐ values.

- ***New Page After?*** indicates if a new page should be started after the image is printed. Click the checkbox to toggle between "yes" ☑ and "no" ☐ values.
- ***Skip Lines Before*** represents the number of lines to be skipped prior to the image being printed.
- ***Skip Lines After*** defines the number of lines to be skipped after the image is printed.
- ***Separator Character*** indicates whether a continuous line of a designated character (usually -, =, or *) should be following this image to separate it from the rest of the report. This field is optional and leaving it blank indicates that no separation line is to be printed.
- ***Skip If Live Operations?*** determines whether the frame is exclusive to the initial setup and recovery processing phases of operation. If "yes" ☑, the frame is not executed during live operations.

Output Image Rules Option

When you select the RULES option on either the Scrolling Output Images screen or the Non-Scrolling Output Images screen, the output image Event Points for Rules overlay is displayed as shown in *Figure 3-7-16 below*. This overlay provides options to access the ILF Editor for APPX event points. If statements are defined in an event point, the respective text on the screen is highlighted.

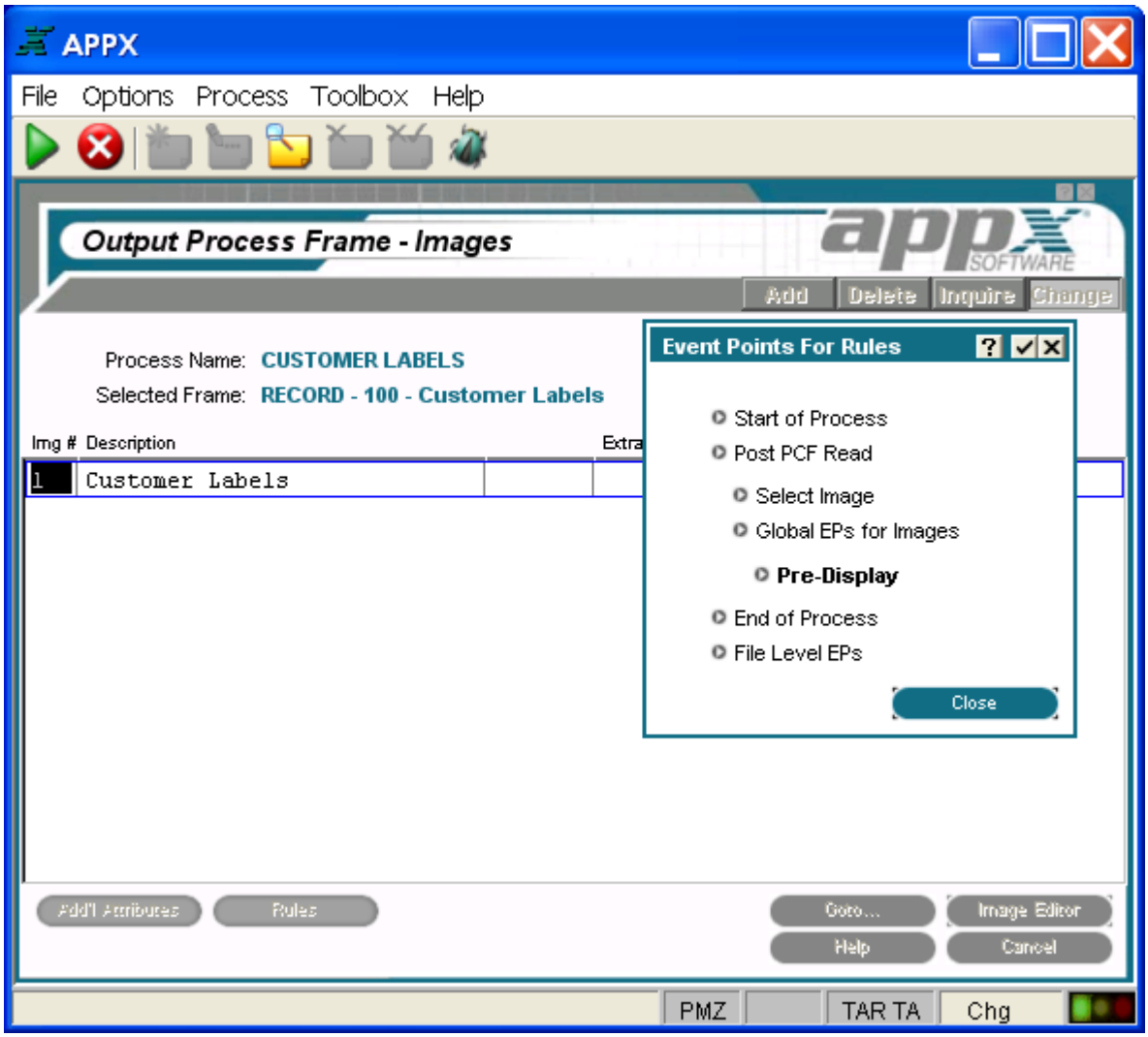


Figure 3-7-16. Output Image Event Points for Rules Overlay

The output image Event Points for Rules overlay contains the following options:

Start of Process

START OF PROCESS accesses the ILF editor for the *Start of Process* event point. Upon completion, you are returned to the Event Points overlay.

Post PCF Read

POST PCF READ accesses the ILF editor for the *Post PCF Read* event point. Upon completion, you are returned to the Event Points overlay.

Select Image

SELECT IMAGE accesses the ILF editor for the *Select Image* event point. Upon completion, you are returned to the Event Points overlay.

Global EP's For Images

GLOBAL EP'S FOR IMAGES calls the Global Event Points overlay shown in *Figure 3-7-17 below.*

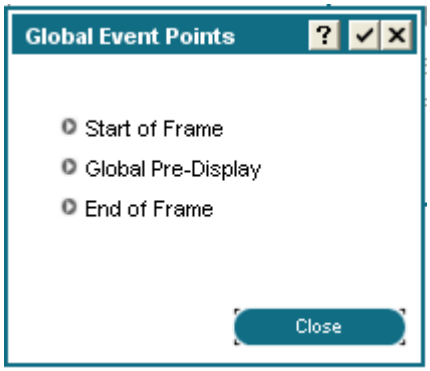


Figure 3-7-17. Global Event Points Overlay

This overlay provides options for three additional event points:

- START OF FRAME accesses the ILF editor for the *Start of Frame* event point. Upon completion, you are returned to the Global Event Points overlay.
- GLOBAL PRE-DISPLAY accesses the ILF editor for the *Global Pre-Display* event point. Upon completion, you are returned to the Global Event Points overlay.
- END OF FRAME accesses the ILF editor for the *End of Frame* event point. Upon completion, you are returned to the Global Event Points overlay.

Pre-Display

PRE-DISPLAY accesses the ILF editor for the *Pre-Display* event point. Upon completion, you are returned to the Event Points overlay.

End of Process

END OF PROCESS accesses the ILF editor for the *End of Process* event point. Upon completion, you are returned to the Event Points overlay.

File Level EPs

FILE LEVEL EPs provides access to the File Level Event Points for the PCF file. These event points are not directly associated with the process but with the PCF file. See the Files Rules Option *in Chapter 2-4* for more information on using these event points.

Refer to Chapter 3-16: Using Event Points for more information about event points.

Output Image in the Image Editor

After you complete specifications for an image on the Scrolling Output Images screen or the Non-Scrolling Output Images screen, you automatically enter the Image Editor when you press RETURN or select the IMAGE EDITOR option. The Image Editor, APPX's screen painting facility, is the facility used to format menus, inputs, outputs, and inquiry screens. *Figure 3-7-18 below shows a sample output image being edited in the Image Editor.*

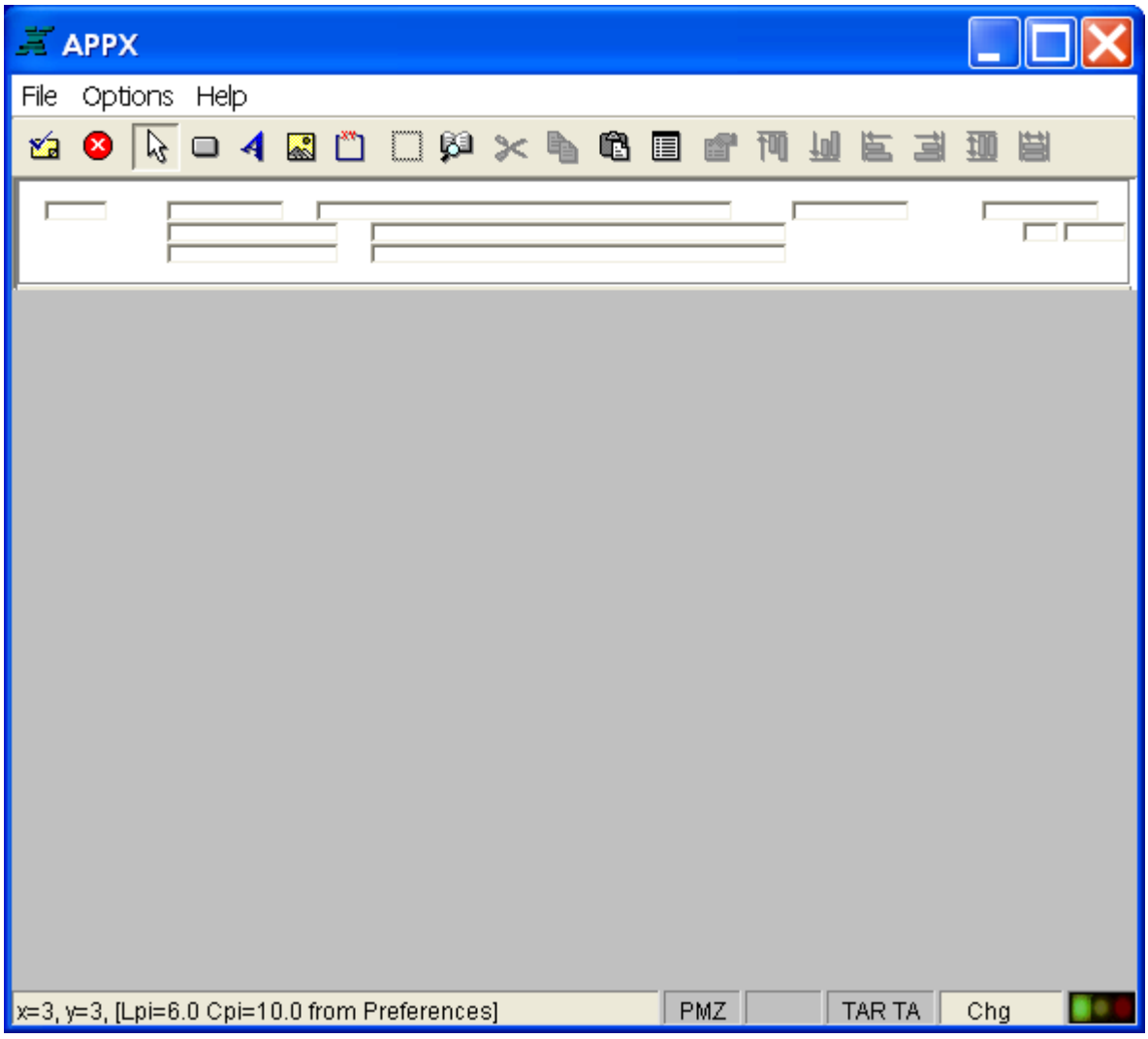


Figure 3-7-18. Output Image in the Image Editor

The standard features of the Image Editor can be employed in character mode using only a keyboard. However, you need a mouse to take advantage of the GUI features within the Image Editor. GUI tools provide designers with the ability to create screen images complete with pictures, buttons, icons, wallpapers, and other graphical features. Another GUI tool is a data palette offering drag and drop selection and placement of data fields.

Basic functionality of the Image Editor is fully explored in Chapter 3-14: Using the Image Editor and a complete discussion of its GUI features is presented in Chapter 3-15: Using GUI Features of the Image Editor. Not all the GUI attributes available for menus and inputs can be used for outputs and inquiries with the 4.2 release of APPX. However, widget control types that have been implemented for outputs and inquiries include Label, Box, and Picture. Data widget control types include Label, Raw Text, and Picture. Window properties and global window properties can also be defined.

To display an output image in the Image Editor as it will appear to your users, specify the PDF form in the *Preferred Form* field of the Output Process Additional Attributes. APPX will compute the lines per inch (lpi) and characters per inch (cpi) for display from the specifications for that form in System Administration. If there is no preferred form specified, APPX will use your client preferences, as in *Figure 3-7-18* above. See Designing GUI Output for more information.

In order to print graphical attributes, you must specify "PDF" in the *Print Format* field and identify a PDF form in the *Form* field on the print disposition screen. See Printing PDF Output for more information.

Designing GUI Output

One of the first things that you may notice is that the Image Editor looks different when you are designing GUI output or inquiry processes. *Figure 3-7-19* below shows a 21x80 image with the grid turned on.

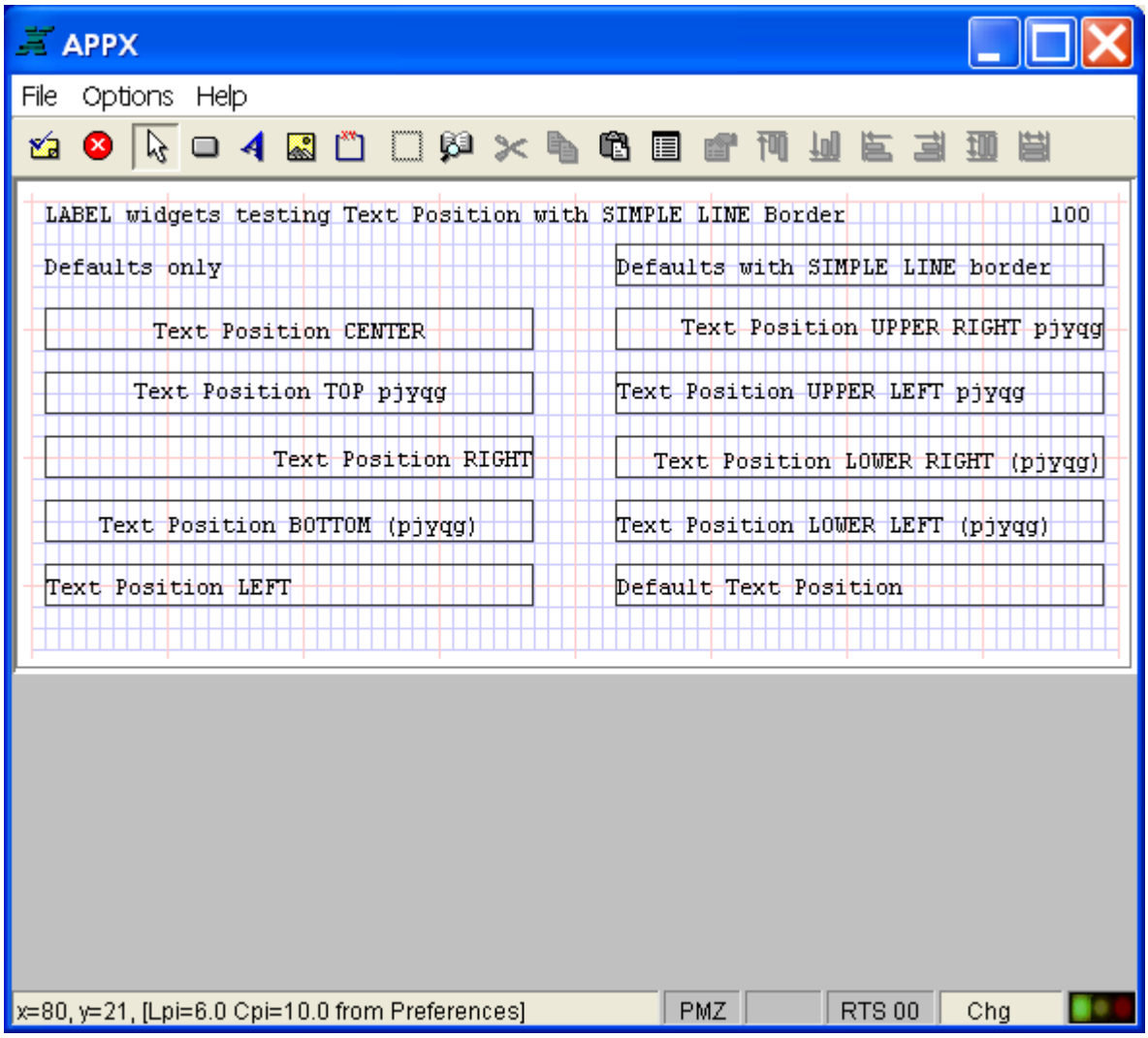


Figure 3-7-19. Output Image with Grid Turned On (21x80 at 6 lpi, 10 cpi)

Notice that only about 2/3 of the screen is used to display 21 rows. To some, the screen may look "squished" since 21 rows are displayed in about 2/3 of the space that is used to display 21 rows when designing an input or menu process. Why is this? Aspect ratio! We want the Image Editor to show the output image in exactly the same proportions that it will print at. For example, if you were to draw a box that looked square using the Image Editor, you would expect a square box to print. Notice on the status line that the output image is displayed at 6 lines per inch (lpi) and 10 characters per inch (cpi), anticipating that you will print the report using a form with those specifications.

Figure 3-7-20 below looks even more "squished". What happened??

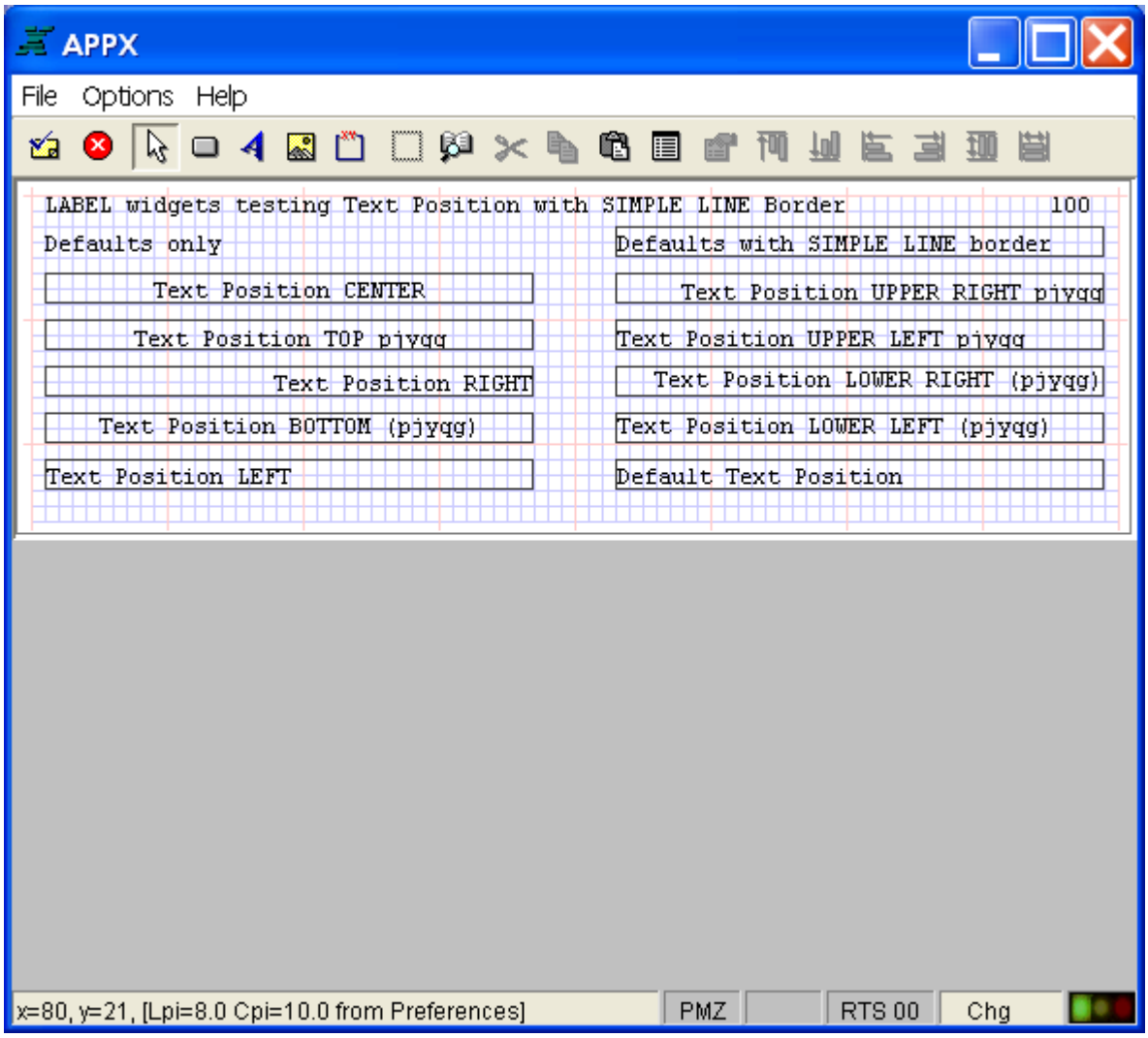


Figure 3-7-20. Output Image with Grid Turned On (21x80 at 8 lpi, 10 cpi)

Notice that the aspect ratio of this output image is based on 8 lpi and 10 cpi. But, WYSIWYG (what you see is what you get) prevails!

So, how can you control the aspect ratio of an output image in the Image Editor at design time? There are two ways. Effective with APPX 4.2, an output process has a field in Additional Attributes called *Preferred Form*. If you specify a FORM name in the *Preferred Form* field, then APPX will compute the lpi and the cpi from the definition for that FORM in System Administration. It can then display the output image using an aspect ratio that corresponds to the FORM specs. Or, while on the output image in Application Design, you can select the **File** menu on the menu bar and then select **Preferences**. The lpi and the cpi can be set as a client preference by double-clicking on the appropriate *Value* cell (*printRowsPerInch* and *printColsPerInch*, respectively). So, if you design all of your reports to print at 6 lpi and 12 cpi, simply set the preference and APPX will remember it from that point on.

Printing PDF Output

If you want to print an existing report as a PDF document, here is what you should do:

- Define a FORM in System Administration. Specify that the form type is PDF and fill in the form attributes. PDF documents are page oriented so the FORM attributes relate to a page of output. You will need to define the paper size. You can also specify left and right margins and top and bottom margins. If you take the paper size and subtract the margins, what's left is the printable area of the page. The length of the printable area will be divided by the number of lines specified to determine the lines per inch (lpi). The width of the printable area will be divided by the number of characters per line to determine the pitch (cpi). So if you specify 8.5" x 11" for the paper size and you specify a top and bottom margin of 0.5" and you specify that there are 60 printable lines on the page, APPX will figure out that the resulting lpi is 6. If you specify a left and right margin of 0.25" and you specify that there are 80 characters per line, APPX will figure out that the desired pitch is 10 cpi.
- On the print disposition screen, when you run any existing report, specify that the *Print Format* is "PDF" and select the appropriate PDF form definition (see the example in Figure 3-7-21 below).
- If you specify Print on Screen and you are connected via a remote client, the printer that you specify must be defined within APPX System Administration as a "LOCAL" printer because APPX will transfer the PDF document to the client and launch Acrobat Viewer in a separate viewer to display the report. Meanwhile, the APPX session will display the PDF Print on Screen Dialog discussed in the next section. The old APPX print on screen utility is not used to view PDF output. If you are connected via a remote client, you must be using the Java client if you expect to view a PDF document via Print on Screen. The win32 client is not able to perform the transfer of the PDF document or launch Acrobat Viewer on your remote desktop. However, if you are running the win32 client directly on the server, then you can print PDF documents to the screen as long as Acrobat Viewer is installed on the server.
- If you specify Print Hard copy, you may print to a Local printer or a server printer. But, obviously, you must have Acrobat Reader installed on your server and/or desktop if you expect to be able to print PDF output

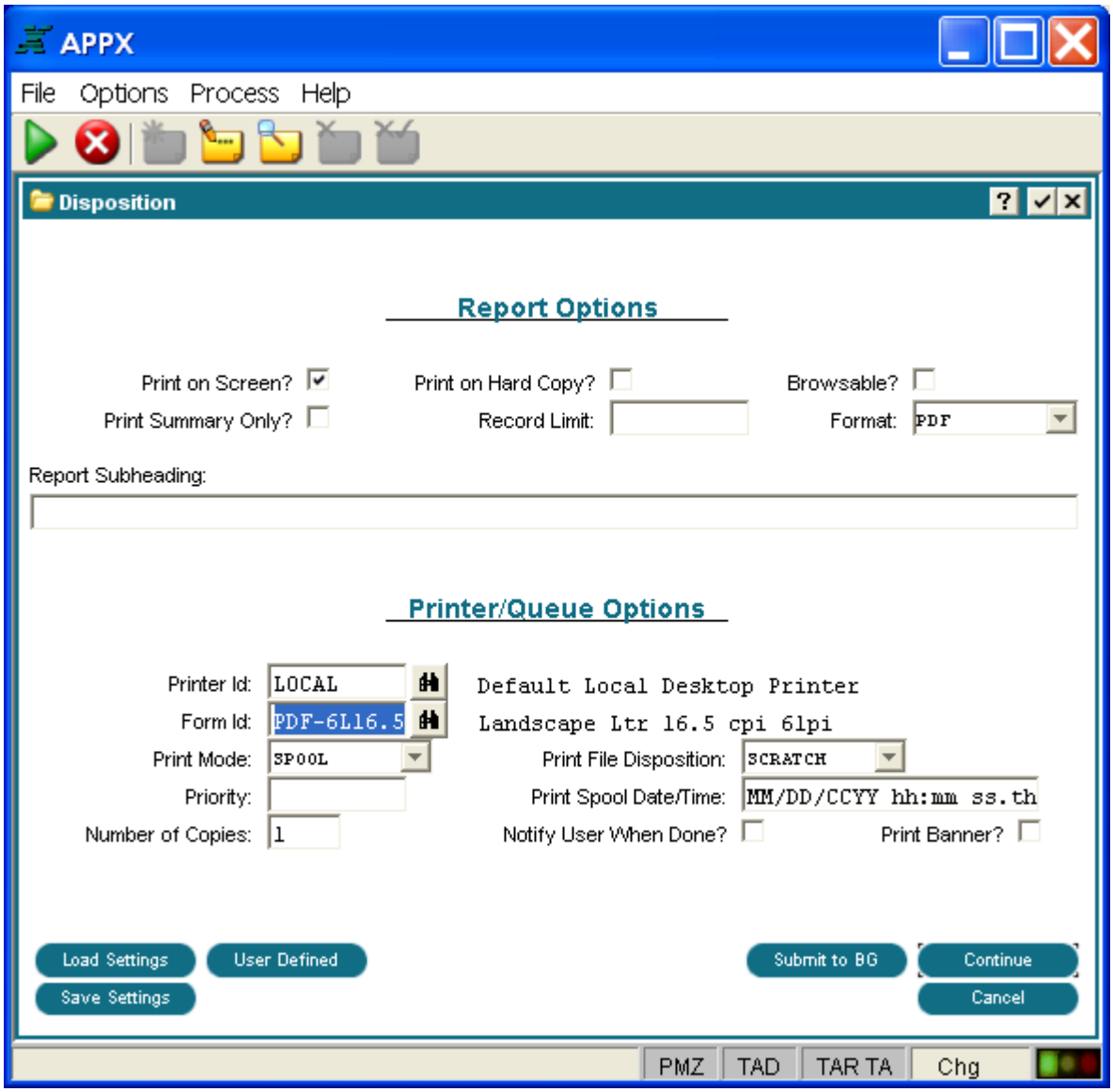


Figure 3-7-21. Print Disposition Screen for PDF Output

PDF Print on Screen Dialog

As discussed in the previous section, Printing PDF Output, if you specify Print on Screen when you are printing in PDF format, APPX will transfer the PDF document to the client and launch Acrobat Viewer in a separate viewer to display the report. APPX will then display the PDF Print on Screen Dialog, shown in Figure 3-7-22 below.

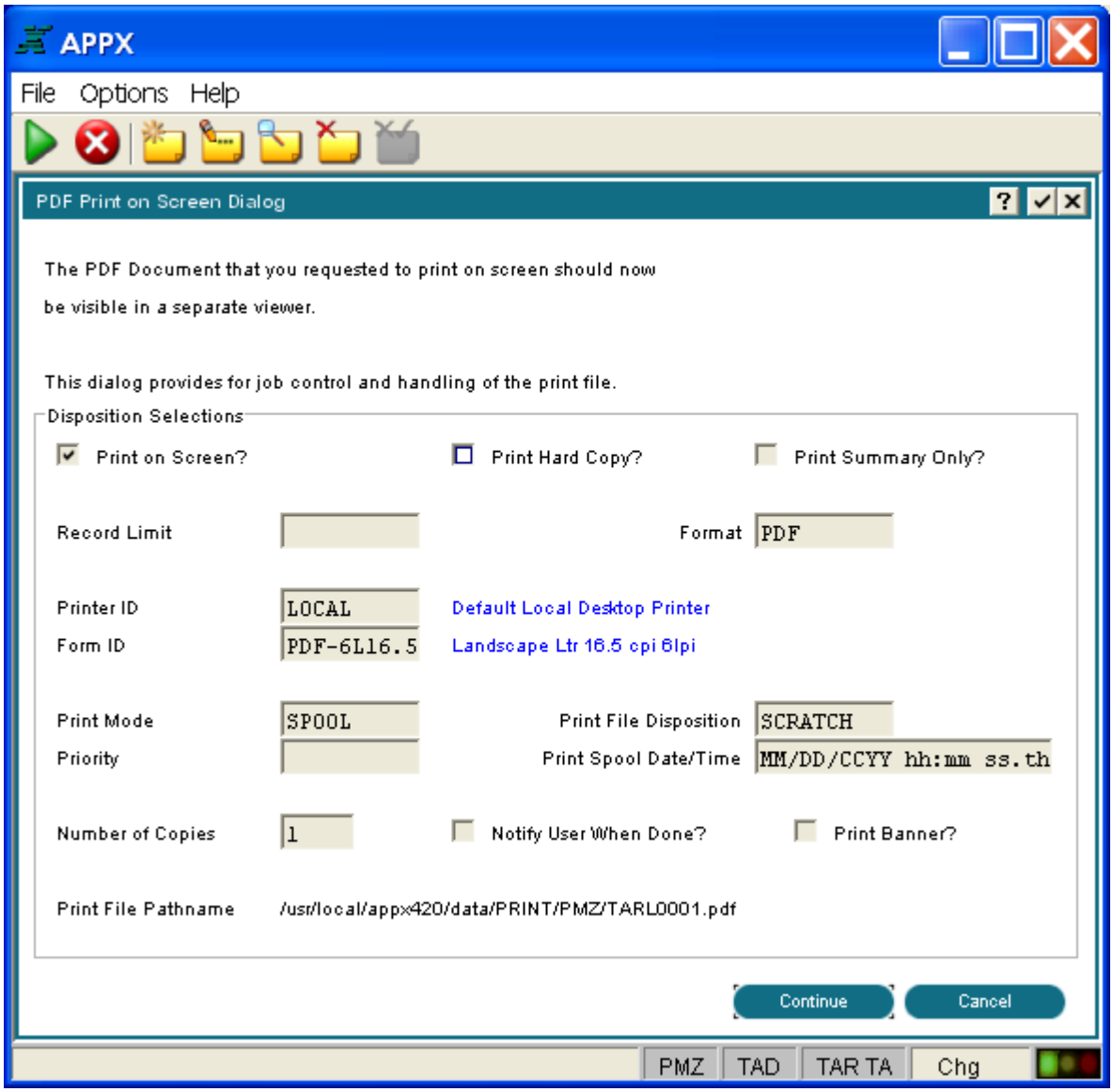


Figure 3-7-22. PDF Print on Screen Dialog

The CONTINUE and CANCEL buttons are provided to control the remainder of the job in the event the job has multiple steps. In previous releases of APPX, you could run a report to the screen, select CANCEL while viewing the report, and the APPX job would cancel without running the remainder of the steps. The PDF Print on Screen Dialog serves primarily to provide such functionality and maintain compatibility with existing jobs. While the PDF output appears on the screen using a PDF viewer, the dialog appears on the APPX screen for you to acknowledge or cancel.

Also, you can still control the print file by use of this screen. APPX will queue the PDF document to a server printer or upload a copy of the PDF document to your client and queue it on a local printer only after you acknowledge the Print on Screen dialog.

Keep in mind that the server prints the PDF document if you specify hard copy and select a server printer. The PDF document is only uploaded to the client if you are displaying it on the screen or if you are printing to a local printer. So, you could display it on screen and, after acknowledging the dialog, print the document on a server printer. While you may elect to print the document from within the PDF viewer, that is not necessarily the preferred method for printing. If you select hard copy within APPX, then the number of copies can be honored, the Print File disposition will relate to the copy of the print file on the server and can be honored, and Print Mode can be honored. When you close the PDF viewer on your client, the APPX Client will remove the PDF file. After printing to your local printer, the PDF file should also be removed.

Notice that the pathname of the server print file is provided on the dialog screen. The name of the print file after it is uploaded to the client is known only by the client. However, the titlebar of the Acrobat Viewer will show the client print file name.

CHAPTER 3-8

Update Processes

Introduction to Update Processes

Update processes modify a database by adding, deleting, or changing information in one or more files.

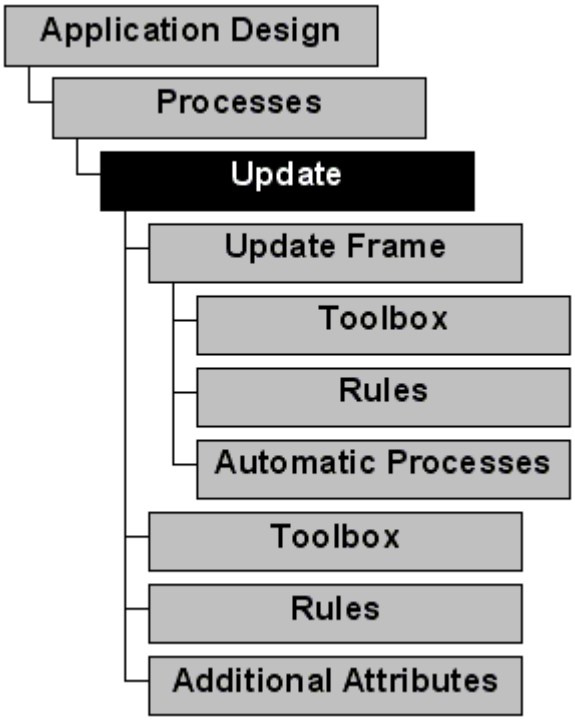


Figure 3-8-1. Update Process Definition

Update Process Specifications

Update process specifications establish the general attributes of an update process, including log file and PCF specifications. The Update Processes screen is shown in *Figure 3-8-2*.

Update Processes screen

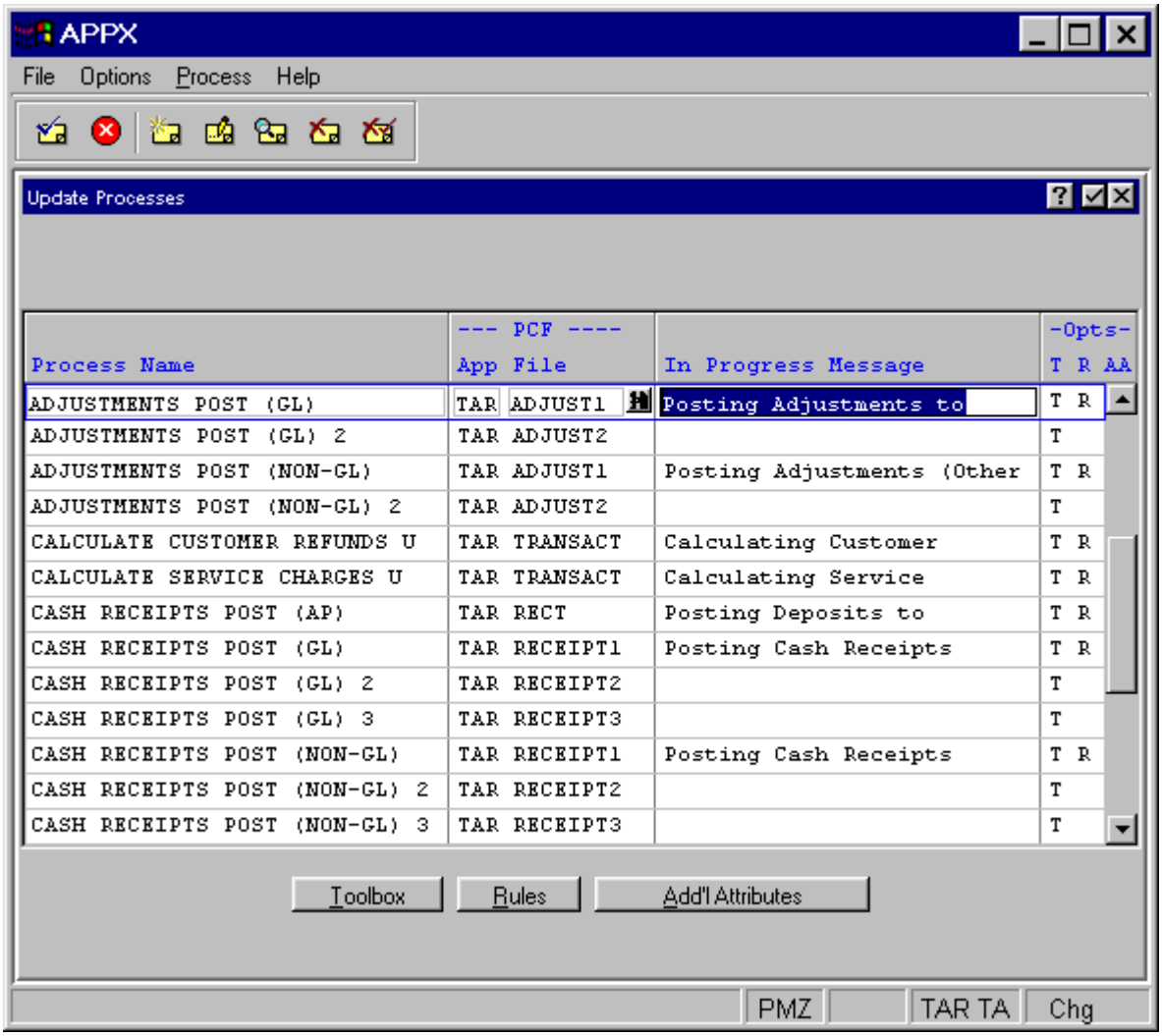


Figure 3-8-2

The Update Processes screen contains the following fields.

Process Name

Identifies a specific process and must be unique within type and application.

App/PCF Name

Identifies a file by application ID and file name that is to be automatically managed by APPX during the execution of this process. APPX automatically reads the process control file (PCF) during update process execution. However, ILF code must be used to write, rewrite, or delete records.

- The *In Progress Message* field may contain up to 60 alphanumeric characters and provides text for the message that displays on the screen while the job process executes in the foreground.

Opts

Are indicators that are present whenever documentation (T), rules (R) or additional attributes (AA) have been defined for this process.

The Update Processes screen contains the following options as described in the corresponding sections:

- Toolbox
- Rules
- Additional Attributes

Update Process Toolbox Option

The toolbox overlay appears on the screen when you invoke the TOOLBOX option from the Update process screen. The overlay contains a number of features such as Documentation Management and Security Management that will assist you with designing and invoking an update process. The options listed in the Toolbox overlay are described briefly below. See *Chapter 3-3, Process Definition* for a more detailed explanation of the options listed on the Toolbox overlay.

Documentation Option

When you select the DOCUMENTATION option from the Toolbox overlay, you will first see the Documentation overlay which allows you to select between user or technical documentation. Once you have entered your documentation, you are returned to the Update Processes screen.

Security Option

The SECURITY option allows you to identify the APPX security code to which a user must have rights in order to run the update process. The security code must be four numeric characters. If blank, there are no restrictions.

Database Management Option

The DATABASE MANAGEMENT option allows you to add new files or maintain existing ones without leaving update process design.

Process Management Option

The PROCESS MANAGEMENT option allows you to add other types of processes or maintain existing ones without leaving process design for update processes.

Design Transfer Option

The DESIGN TRANSFER option allows you to copy the attributes from a process that you have already designed and apply it to a new process.

Cross Reference Option

The CROSS REFERENCE option enables you to determine the location and frequency of a selected domain, file, field, application ID, process or ILF statement within a specific database.

Reference Browser Option

The REFERENCE BROWSER option runs a modified version of "Cross Reference – All" and produces a list of design components utilized by the currently selected update process. The results can be displayed interactively where links allow you to see the context of the references, edit the referenced component and drill down to the child process. This option can be used to identify, and optionally edit, the file level event points utilized by this update process.

Syntax Check Option

This feature allows you to create the Executable Module for the current update process without actually running it. If the Executable Module for the update process is valid this option is ignored when invoked. Note that this option (`OPTION` `97`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

View Audit Info Option

This feature allows you to see who created the currently selected update process and who changed it most recently. It also gives the date of creation and the date it was last changed. Note that this option (`OPTION` `98`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Invoke a Process Option

This feature is available to facilitate testing. It allows you to invoke any process; the currently selected update process using the Test Database defined in system administration is the default. Note that this option (`OPTION` `99`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Update Process Rules Option

Selecting the RULES option accesses the update process Event Points For Rules overlay which allows you to select from a variety of options that access the ILF Editor for APPX event points. The update Event Points overlay is shown in Figure 3-8-3 below.

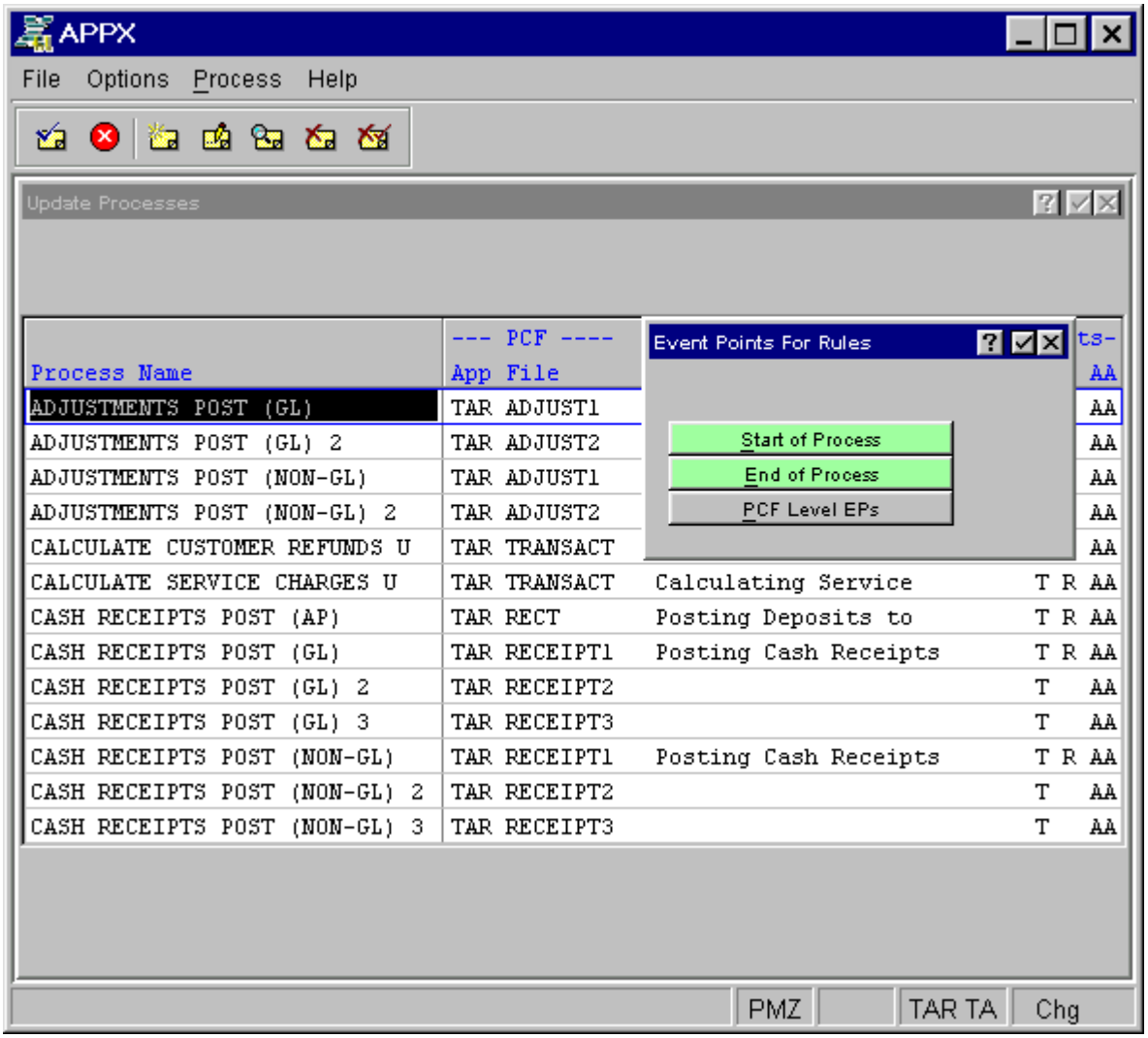


Figure 3-8-3. Update Event Points Overlay

The update Event Points overlay contains the following options:

Start of Process

START OF PROCESS, which is common to all types of processes, accesses the ILF editor for the *Start of Process* event point. Upon completion, you are returned to the process Event Points overlay.

End of Process

END OF PROCESS, which is common to all types of processes, accesses the ILF editor for the *End of Process* event point. Upon completion, you are returned to the process Event Points overlay.

PCF Level EPs

PCF LEVEL EPS provides access to the File Level Event Points for the PCF file. This overlay is shown in *Figure 3-8-3*. These Event Points are not directly associated with the process but with the PCF file. See the *Rules Option* section of *Chapter 2-4, Files and Fields* for more information on using PCF Level Event Points.

Refer to Chapter 3-16: Using Event Points for more information.

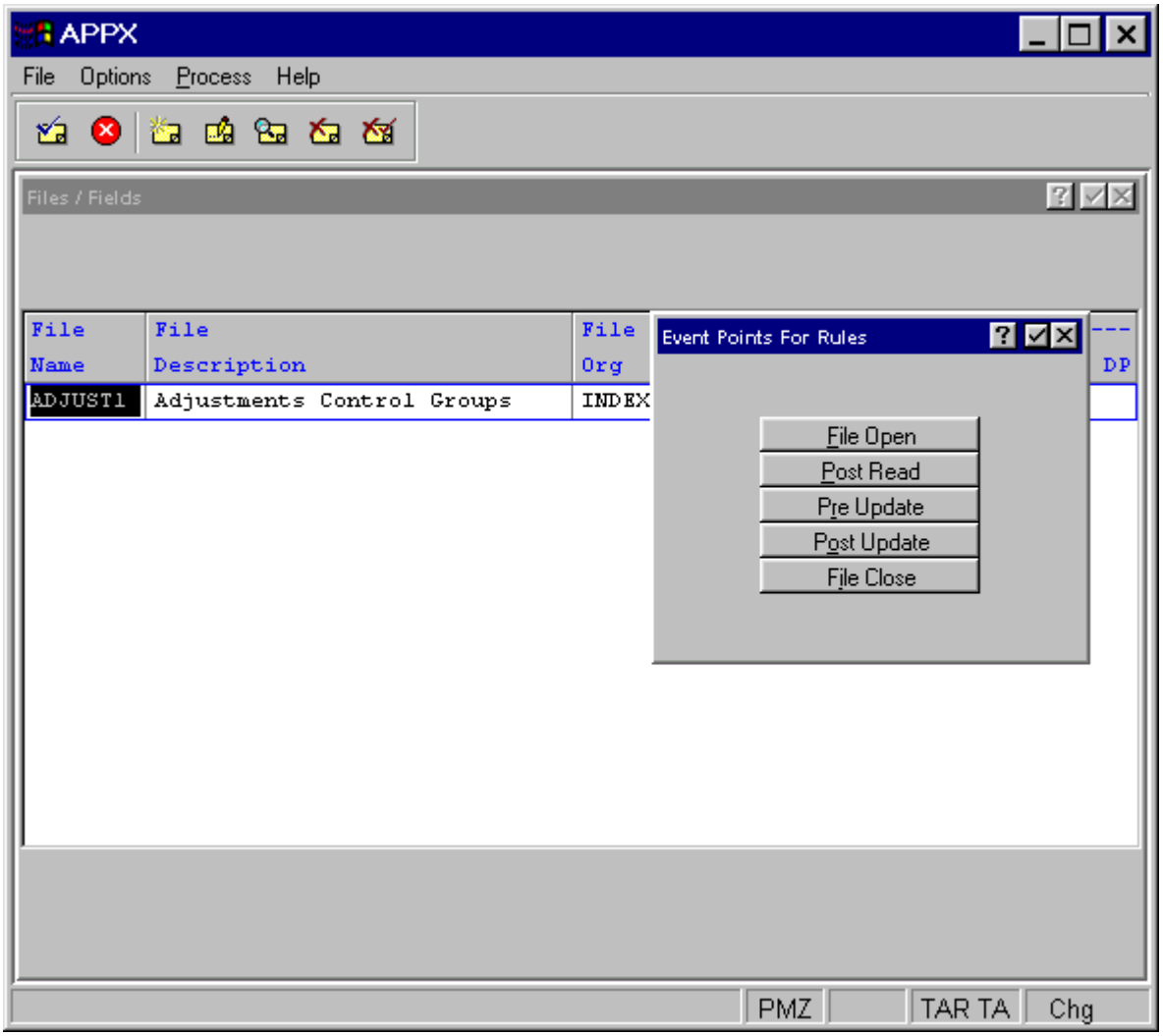


Figure 3-8-3. Event Points for Rules Overlay

Update Process Additional Attributes Option

The ADDITIONAL ATTRIBUTES option accesses the Update Process Additional Attributes overlay, shown in *Figure 3-8-4*, from which you can alter the standard EM attributes.

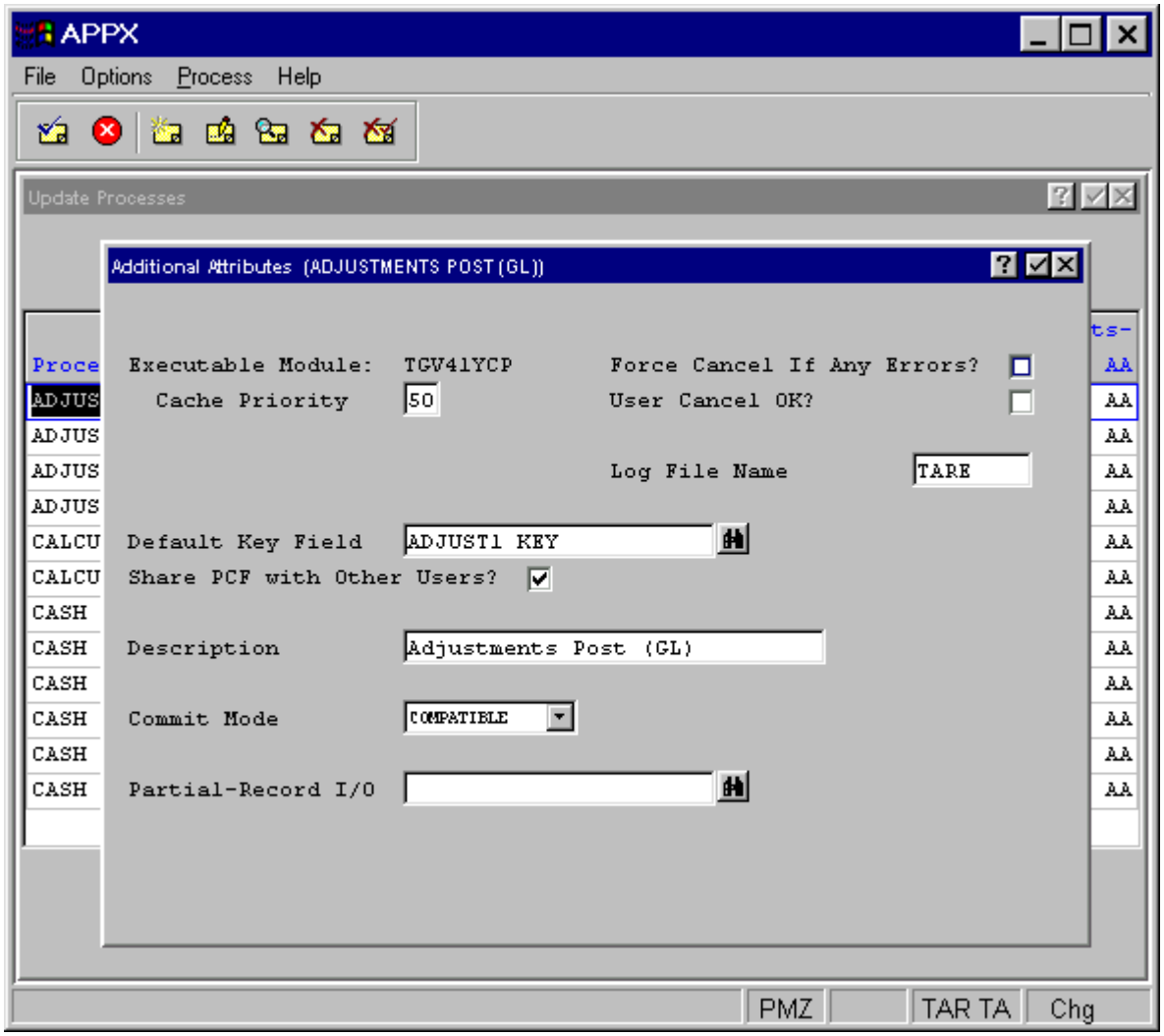


Figure 3-8-4. Update Processes Additional Attributes Overlay

The update Additional Attributes overlay contains the following fields:

Executable Module

Displays the name of the executable module (EM) that is automatically generated by APPX. APPX derives the file name from the date and time the currently selected update process was added. The name consists of a unique string of eight alphanumeric characters and cannot be changed. If APPX finds it necessary to recreate the EM, the new EM retains the same file name. See *Chapter 3-3* for more information.

Force Cancel if Any Errors?

Controls whether or not the user is forced to cancel the process if any errors are written to the log file. If "yes" ☑, the user is forced to cancel after the subprocess family completes.

Cache Priority

Establishes the likelihood that the process will remain in memory after execution, relative to other processes that the user has also completed executing. A process with a cache priority of *99* is most likely to remain in memory. A process with a cache priority of *1* is least likely to remain in memory. A cache priority of *0* indicates that the process is never cached. The default is *50*. This field is common to the Additional Attributes overlay for all types of processes. See *Chapter 3-3, Process Definitio*n, for more information.

User Cancel OK?

Contains either a “yes” ☑ or “no” ☐ and determines whether or not a user can select the CANCEL option to stop the execution of this process. The default is “yes”. You can control the user’s ability to select the CANCEL option with statements and predefined field CANCEL OK.

Log File Name

Establishes the name of the file that is used to store the list of error and warning messages generated during the execution of the update process family. If necessary to guarantee uniqueness, APPX may append one or more characters to the value entered here, or replace one or more of the ending characters. This field may contain up to eight alphanumeric characters. If two or more update processes *in the same subprocess family* are defined with identical *Log File Name* values, the processes share a common log file. They print as a single log, with continuous page numbers. If this specification is blank, APPX uses the last *Log File Name* that is active within the subprocess family. Refer to *Chapter 3-1* for a definition of a subprocess family.

Default Key Field

Designates the specific keypath to the PCF that is used in executing the defined update process. If you did not link this update process to a query process, the key determines the order the records process in. A default keypath that is defined for the invocation of a child process, if any, overrides this specification.

Share PCF With Other Users?

Contains either “yes” ☑ or “no” ☐ and determines whether or not other users can *update* the file concurrently. The default is "yes".

Description

Is incorporated in the message that displays on the screen when the currently selected update process is compiling. It is used to communicate with users since it provides a more descriptive and easily understood alternative to *Process Name*, which is the default if this field is left blank. See *Chapter 3-3, Process Definitio*n, for more information.

Commit Mode

Controls the commit mode for a process and can be selected from a drop-down list of available options. *Commit Mode* determines those points in time when APPX will commit changes made to the tables stored within an RDBMS. It is only applicable in conjunction with an RDBMS; it has no effect with APPXIO. Also, not all File Managers support this feature. Although initially set here for the process, it can be added or overridden by use of predefined field COMMIT MODE. The following options are available:

COMPATIBLE. Changes are committed as soon as all locks are freed. This provides maximum compatibility between APPXIO files and RDBMS tables.

AUTOCOMMIT*. APPX will commit each modification as it is made.
***This value will be implemented in a future release.**

PCF CYCLE. APPX will commit modifications as each PCF record is processed.

END. APPX will only commit modifications at the end of the process family.

MANUAL. APPX will not automatically commit changes. You must use the COMMIT statement to do so.

- ***Partial-Record I/O*** is used to specify a field from the PCF file so that instead of operating on the entire PCF record, APPX operates on the designated field only. If the field is a group header, then I/O will be performed on all members of the group. Utilizing partial-record I/O can greatly improve performance when the PCF is a table in an RDBMS. This is especially true when the table contains many columns, but relatively few are being utilized by the process. The SCAN option is available for field selection.

Update Frame Specifications

Frame specifications define attributes and statements unique to frames that manipulate fields from each record in a PCF, or perform start- or end-of-range processing. The update Frames overlay displays when you select ENTER. The update Frames overlay contains the following fields.

Frame Class

Defines the type of frame with which you are working. Valid options are *Range-Start* frames print immediately before the first record for which the value in a specific sort field (the range field) changes. *Record* frames manipulate information read from the process control file. *Range-End* frames execute immediately after the last record in a series of records that have the same value in a specific sort field (the range field).

Seq No

Displays a sequence number that is defaulted by APPX and used to identify the frame. The default sequence number can be overridden, but must be unique within each frame class for the process. Sequence numbers determine the order of the frames in the process structure. The first frame that is defined in add mode is assigned sequence number *100*, and additional frames are assigned a number equal to the highest existing sequence number, plus *100*.

Descriptive

Identifies this record frame. The field is for documentation purposes only.

Opts

Are indicators that are present whenever documentation (T), rules (R), or automatic processes (AP) have been defined for this frame.

The update Frames overlay contains the following options as described in the corresponding sections:

- Toolbox
- Rules
- Automatic Processes

If *Frame Class* is *Range-Start* or *-End*, the screen in *Figure 3-8-5* appears when you select ENTER.

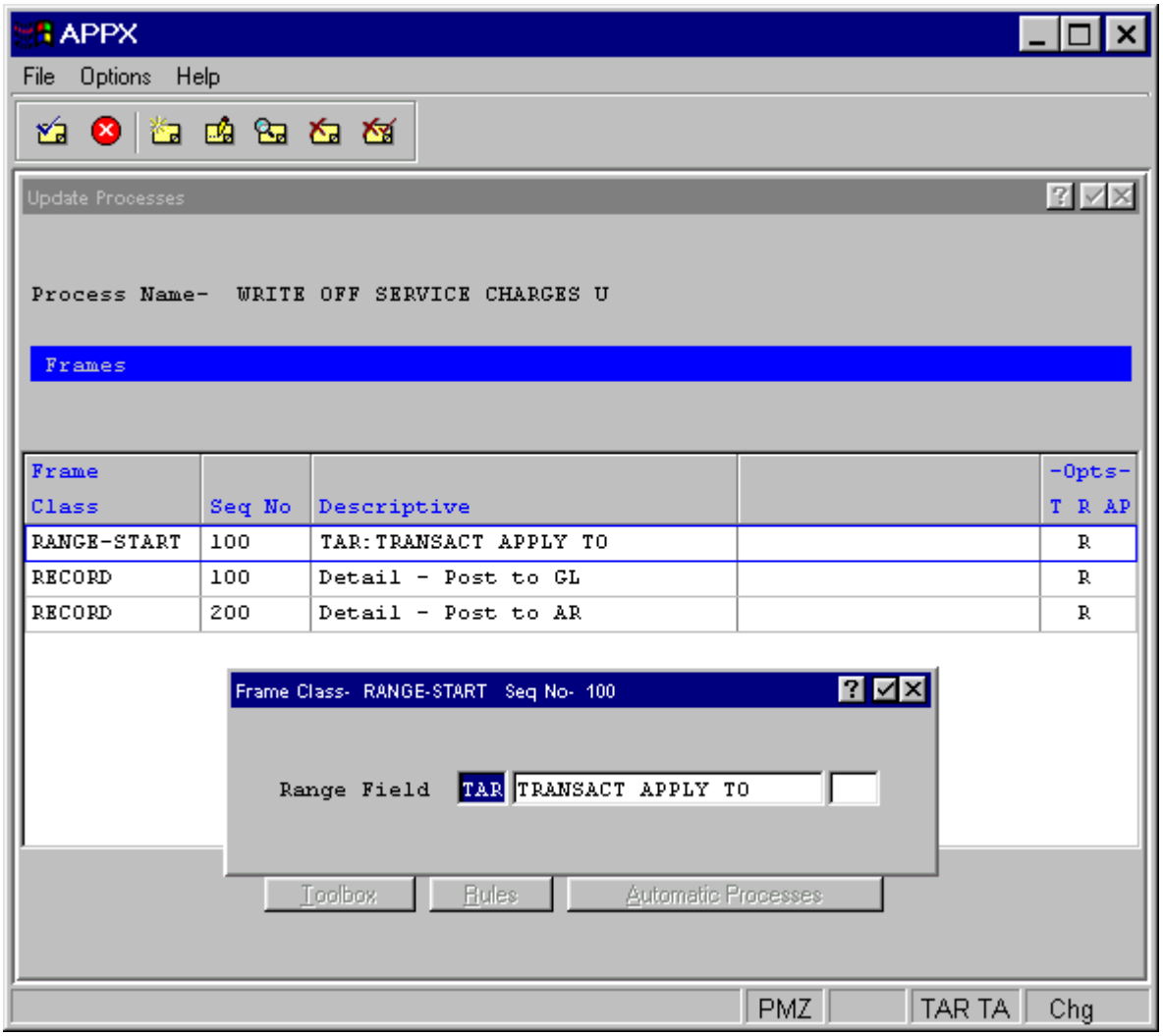


Figure 3-8-5. Update Range-Start and -End Frame Class Overlay

The update *Range-Start* and *-End* Frame Class overlay contains the following field.

- ***Range Field*** designates the application ID and field name of an APPX field in a query process that is used by this update process. The *Start of Range* and *End of Range* frames execute based on a change in the value of this field. This overlay contains three fields: one to identify the application ID, one to identify the field name, and one to identify the Seq No.

Update Frame Toolbox Option

The toolbox overlay appears on the screen when you invoke the TOOLBOX option from the update Frames overlay. This toolbox overlay contains a number of features such as Documentation Management and Security Management that will assist you with designing and invoking an update process. The options listed in the Toolbox overlay are described briefly below. See *Chapter 3-3, Process Definition* for a more detailed explanation of the options listed on the Toolbox overlay.

Database Management Option

The DATABASE MANAGEMENT option allows you to add new files or maintain existing ones without leaving update process design.

Process Management Option

The PROCESS MANAGEMENT option allows you to add other types of processes or maintain existing ones without leaving update process design.

Design Transfer Option

The DESIGN TRANSFER option allows you to copy the attributes from a process that you have already designed and apply it to a new process.

Syntax Check Option

This feature allows you to create the Executable Module for the current update process without actually running it. If the Executable Module for the update process is valid this option is ignored when invoked. Note that this option (`OPTION 97`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

View Audit Info Option

This feature allows you to see who created the currently selected update process and who changed it most recently. It also gives the date of creation and the date it was last changed. Note that this option (`OPTION 98`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Invoke a Process Option

This feature is available to facilitate testing. It allows you to invoke any process; the currently selected update process using the Test Database defined in system administration is the default. Note that this option (`OPTION 99`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Update Frame Rules Option

Selecting the RULES option accesses the update process frame Event Points For Rules overlay which allows you to select from a variety of options that access the ILF Editor for APPX event points. If statements were defined in the event point, the text on the overlay is highlighted. Upon completion, you are returned to the update Event Points For Rules overlay, which contains the following options:

Start of Process

START OF PROCESS accesses the ILF editor for the *Start of Process* event point. Upon completion, you are returned to the process Event Points overlay.

Start of Frame

START OF FRAME accesses the ILF editor for the *Start of Frame* event point. Upon completion, you are returned to the process Event Points overlay.

End of Process

END OF PROCESS accesses the ILF editor for the *End of Process* event point. Upon completion, you are returned to the process Event Points overlay.

PCF Level EPs

PCF LEVEL EPS provides access to the File Level Event Points for the PCF file. This overlay is shown in *Figure 3-8-3.* These Event Points are not directly associated with the process but with the PCF file. See the *Rules Option* section of *Chapter 2-4, Files and Fields* for more information on using PCF Level Event Points.

Refer to Chapter 3-16: Using Event Points for more information.

Update Frame Automatic Processes Option

The AUTOMATIC PROCESSES option accesses the Automatic Child Processes specification screen, which allows you to attach a specific process to the update process currently selected so that it invokes automatically when the update process is invoked. See Chapter 3-13: Automatic and Optional Children for more information.

CHAPTER 3-9

Query Processes

Introduction to Query Processes

Query processes:

- Determine the records that are available to a subsequent input, update, or output process.
- Establish the order selected records are retrieved in for display or processing.

Query processes have special rules with respect to "inheritance" of work fields and temporary work files among APPX processes. *(See Chapter 1-5, "Interprocess Communications"* for more information on inheritance) Please see the description under "PROCESS CODE", in chapter 4-7 (*Predefined Fields and Processes)* for a more detailed explanation of the special inheritance rules with respect to Query processing.

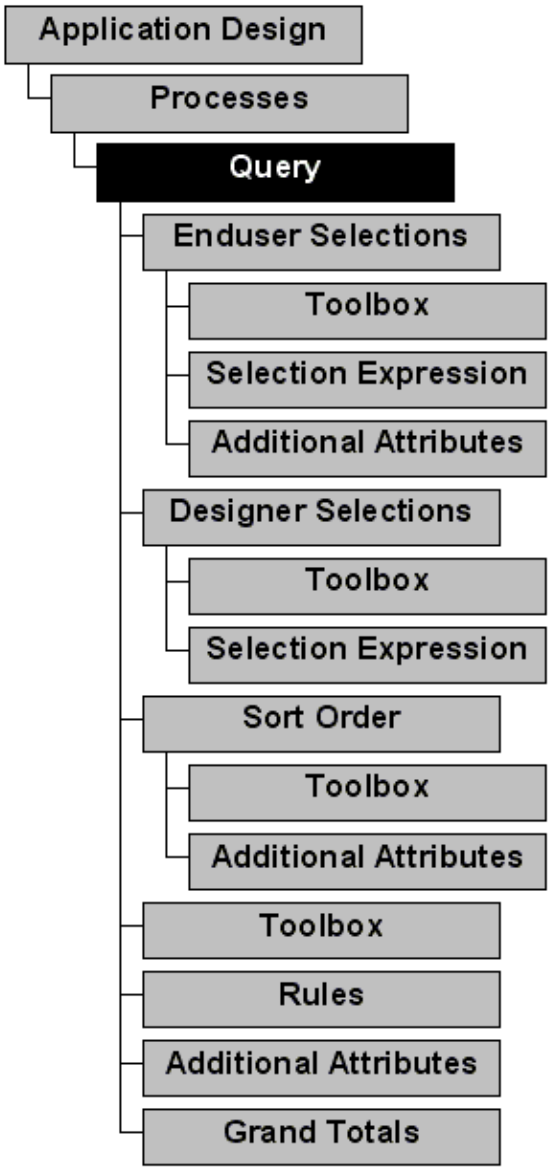


Figure 3-9-1. Query Process Definition

Query Process Specifications

Query process specifications establish the general attributes of a query process, including a process control file (PCF), and operational parameters.

Query Processes screen

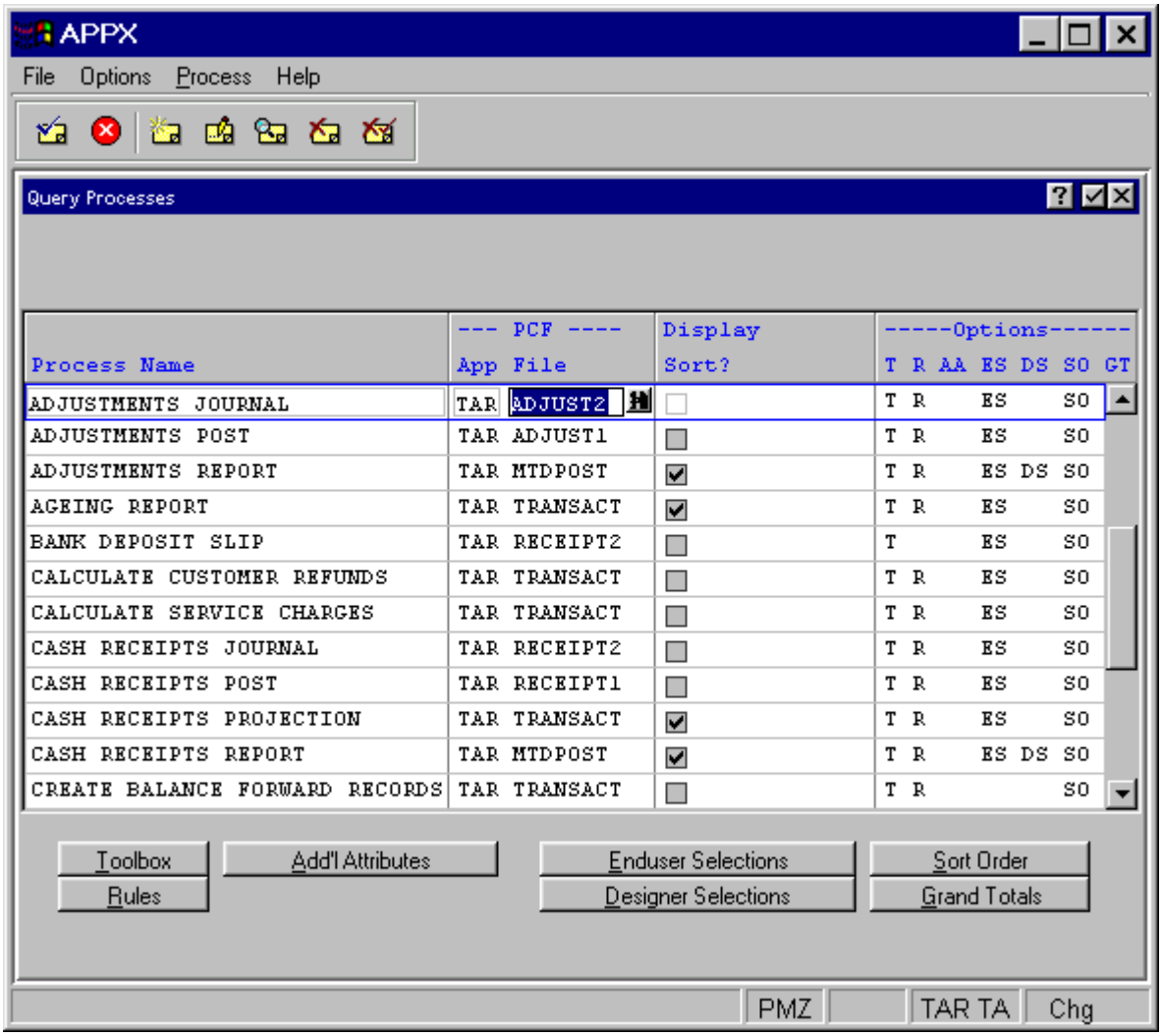


Figure 3-9-2

The Query Processes screen, shown in *Figure 3-9-2* above, contains the following fields:

Process Name

Identifies a specific process and consists of 30 uppercase alphanumeric characters that must be unique within type and application.

App/PCF Name

Identifies a file by application ID and file name that is to be automatically managed by APPX during the execution of this process. APPX automatically reads the process control file (PCF) during query process execution.

Display Sort?

Allows you to explicitly specify whether or not the sort order is displayed to the user. The default is "yes" ☑.

Options

Are indicators that are present whenever documentation (T), rules (R), enduser selections (ES), designer selections (DS), sort order (SO), or grand totals (GT) have been defined for this process.

The following options as described in the corresponding sections are available from the Query Processes screen:

- Toolbox
- Rules
- Additional Attributes
- Enduser Selections
- Designer Selections
- Sort Order
- Grand Totals

Query Process Toolbox Option

The toolbox overlay appears on the screen when you invoke the TOOLBOX option from the query frames overlay. This toolbox overlay contains a number of features such as Documentation Management and Security Management that will assist you with designing and invoking an query process. The options listed in the Toolbox overlay are described briefly below. See *Chapter 3-3, Process Definition* for a more detailed explanation of the options listed on the Toolbox overlay.

Documentation

When you select the DOCUMENTATION option from the Toolbox overlay, you will first see the Documentation overlay which allows you to select between user or technical documentation. Once you have entered your documentation, you are returned to the query Processes screen.

View Query

VIEW QUERY allows you to review all of the query options on a single screen.

Database Management

The DATABASE MANAGEMENT option allows you to add new files or maintain existing ones without leaving query process design.

Process Management

The PROCESS MANAGEMENT option allows you to add other types of processes or maintain existing ones without leaving query process design.

Design Transfer

The DESIGN TRANSFER option allows you to copy the attributes from a process that you have already designed and apply it to a new process.

Cross Reference

The CROSS REFERENCE option enables you to determine the location and frequency of a selected domain, file, field, application ID, process or ILF statement within a specific database.

Reference Browser Option

The REFERENCE BROWSER option runs a modified version of "Cross Reference – All" and produces a list of design components utilized by the currently selected job process. The results can be displayed interactively where links allow you to see the context of the references, edit the referenced component and drill down to the child process. This option can be used to identify, and optionally edit, the file level event points utilized by this job process.

Syntax Check

This feature allows you to create the Executable Module for the current query process without actually running it. If the Executable Module for the query process is valid this option is ignored when invoked. Note that this option (`OPTION` `97`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

View Audit Info

This feature allows you to see who created the currently selected query process and who changed it most recently. It also gives the date of creation and the date it was last changed. Note that this option (`OPTION` `98`), like any option with a value of 90 or greater, is available from any screen in Application Design without accessing the Toolbox.

Invoke a Process

This feature is available to facilitate testing. It allows you to invoke any process; the currently selected query process using the Test Database defined in system administration is the default. Note that this option (`OPTION` `99`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Query Process Rules Option

Selecting the RULES option accesses the query process frame Event Points For Rules overlay which allows you to select from a variety of options that access the ILF Editor for APPX event points. If statements were defined in the event point, the text on the overlay is highlighted. Upon completion, you are returned to the query Event Points For Rules overlay. The query Event Points overlay contains the following options:

Start of Process (Query Setup)

START OF PROCESS (QUERY SETUP) accesses the ILF editor for the *Start of Process* event point. Statements in this event point execute during query setup. Upon completion, you are returned to the query Event Points overlay.

End of Process (Query Setup)

END OF PROCESS (QUERY SETUP) accesses the ILF editor for the *End of Process* event point. Statements in this event point execute during query setup. Upon completion, you are returned to the process Event Points overlay.

Start of Query Execution

START OF QUERY EXECUTION accesses the ILF editor for the *Start of Query Execution* event point. Statements in this event point execute just before the query starts. Upon completion, you are returned to the query Event Points overlay.

Establish PCF Range

This event point provides explicit control for the designer to establish the range (or ranges) of records from the PCF file which are read by the query. This can significantly increase query performance by utilizing available key fields.

ESTABLISH PCF RANGE can be coded by the designer. Otherwise, it will be generated at runtime based on any Record Selection Beg At and End At questions. To explicitly limit customer records to those for a given state, code in the *Establish PCF Range* event point would consist of a simple BEG AT/END AT loop. These loops can be repeated multiple times to establish multiple ranges for the query to process.

Pre-User Selection

PRE-USER SELECTION accesses the ILF editor for the *Pre- User Selection* event point. Upon completion, you are returned to the query Event Points overlay.

Post User Selection

POST USER SELECTION accesses the ILF editor for the *Post User Selection* event point. Upon completion, you are returned to the query Event Points overlay.

End of Query Execution

END OF QUERY EXECUTION accesses the ILF editor for the *End of Query Execution* event point. Statements in this event point execute when the query is finished. Upon completion, you are returned to the query Event Points overlay.

PCF Level EPs

PCF LEVEL EPS provides access to the File Level Event Points for the PCF file. This overlay is shown in *Figure 3-9-3.* These Event Points are not directly associated with the process but with the PCF file. See the *Rules Option* section of *Chapter 2-4, Files and Fields* for more information on using PCF Level Event Points.

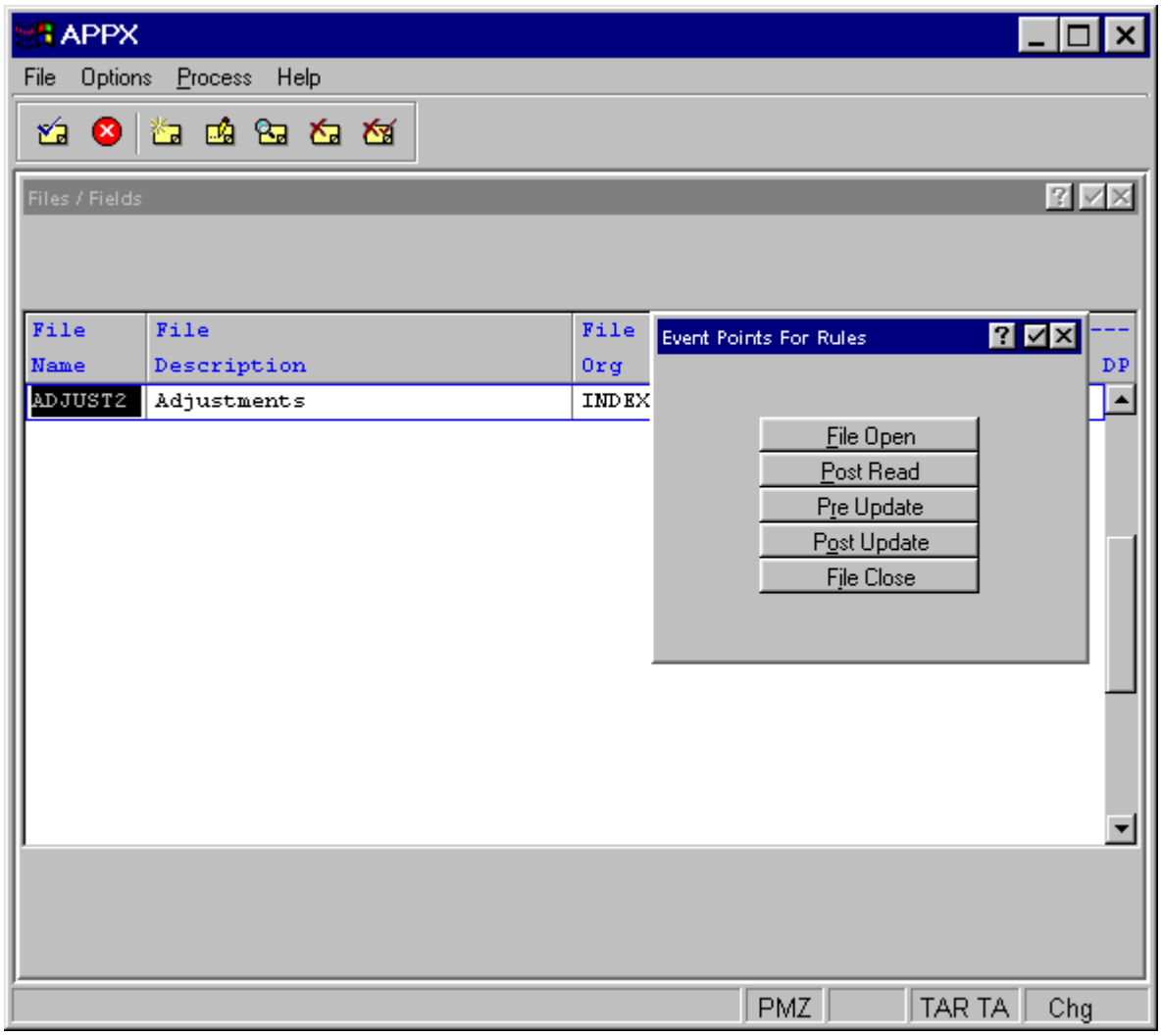


Figure 3-9-3. Event Points For Rules Overlay

Query Process Additional Attributes Option

ADDITIONAL ATTRIBUTES accesses the query process Additional Attributes overlay from which you can alter the standard EM attributes.

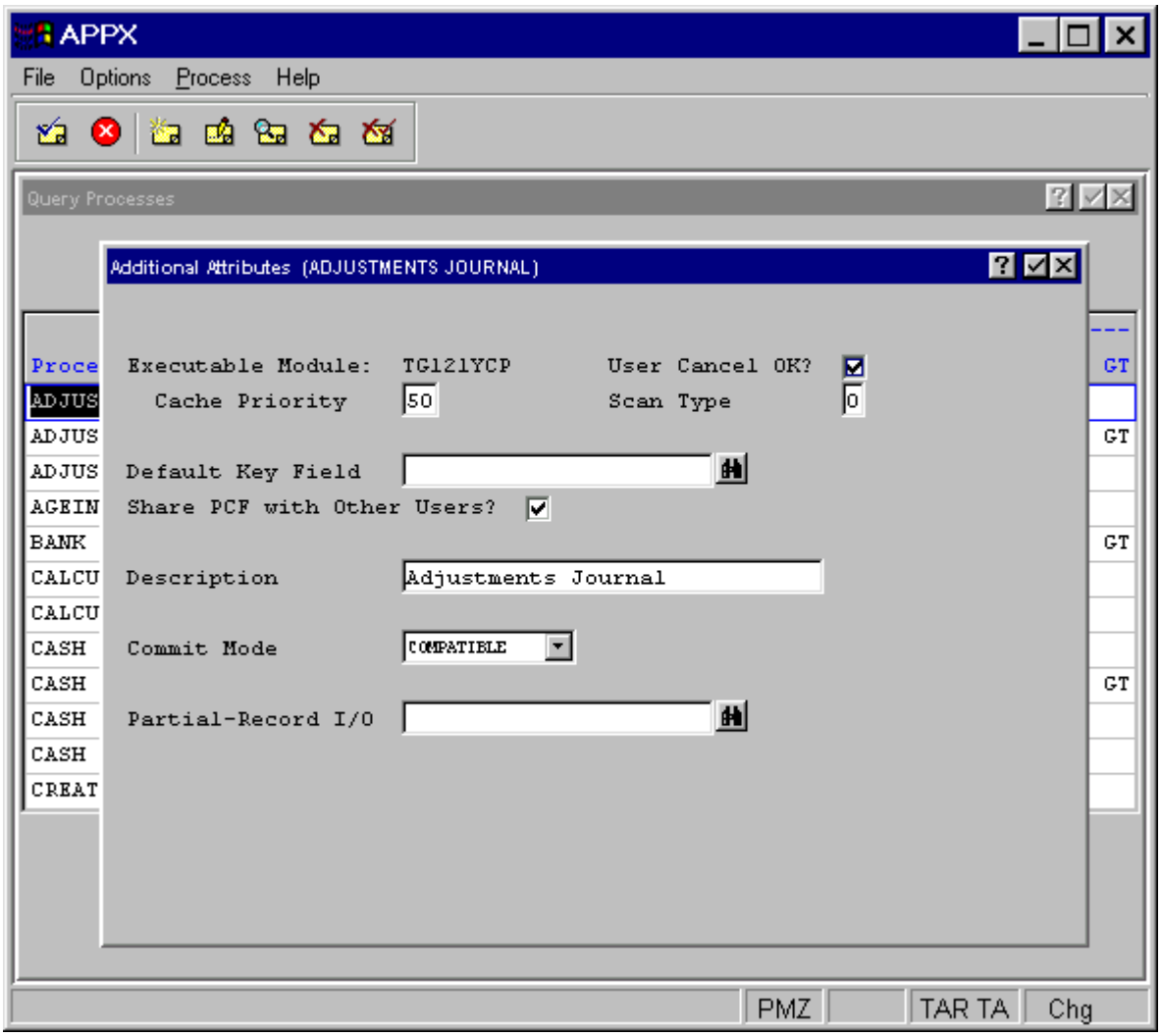


Figure 3-9-4. Query Process Additional Attributes Overlay

The query Additional Attributes overlay shown in *Figure 3-9-4* contains the following fields:

Executable Module

Displays the name of the executable module (EM) that is automatically generated by APPX. APPX derives the file name from the date and time the currently selected query process was added. The name consists of a unique string of eight alphanumeric characters and cannot be changed. If APPX finds it necessary to recreate the EM, the new EM retains the same file name. This field is common to the Additional Attributes overlay for all types of processes.

User Cancel OK?

Contains either “yes” or “no” and determines whether or not a user can select the CANCEL option key to stop the execution of this process. The default is “yes”. You can conditionally control the user’s ability to select the CANCEL option with statements and the predefined field CANCEL OK. This field is common to the Additional Attributes overlay for all types of processes.

Cache Priority

Establishes the likelihood that the process will remain in memory after execution, relative to other processes that the user has also completed executing. A process with a cache priority of *99* is most likely to remain in memory. A process with a cache priority of *1* is least likely to remain in memory. A cache priority of *0* indicates that the process is never cached. The default is *1*. This field is common to the Additional Attributes overlay for all types of processes.

Scan Type

Governs whether or not the application user is permitted to scan and further constrain the individual records in the query’s generated pointer file. The *Scan Type* options are ***0** Do Not Scan* which prohibits the user from scanning and individually constraining records and ***1** Scan* which permits the user to scan and constrain records. The default is ***0** Do Not Scan*.

Default Key Field

Designates the specific keypath that is initially presented to a user invoking this process. If blank, the primary key is used. This specification is overridden by the default keypath that is defined for the child process invocation of the process.

Share PCF with Other Users?

Indicates whether the designated PCF file is available to other users while this process is executing.

Description

Is incorporated in the message that displays on the screen when the query process is compiling. It is used to communicate with users since it provides a more descriptive and easily understood alternative to *Process Name*, which is the default if this field is left blank. The Additional Attributes overlay for all types of processes contain this field.

Commit Mode

Controls the commit mode for a process and can be selected from a drop-down list of available options. *Commit Mode* determines those points in time when APPX will commit changes made to the tables stored within an RDBMS. It is only applicable in conjunction with an RDBMS; it has no effect with APPXIO. Also, not all File Managers support this feature. Although initially set here for the process, it can be added or overridden by use of predefined field COMMIT MODE. The following options are available:

COMPATIBLE. Changes are committed as soon as all locks are freed. This provides maximum compatibility between APPXIO files and RDBMS tables.

AUTOCOMMIT*. APPX will commit each modification as it is made.
***This value will be implemented in a future release.**

PCF CYCLE. APPX will commit modifications as each PCF record is processed.

END. APPX will only commit modifications at the end of the process family.

MANUAL. APPX will not automatically commit changes. You must use the COMMIT statement to do so.

- ***Partial-Record I/O*** is used to specify a field from the PCF file so that instead of operating on the entire PCF record, APPX operates on the designated field only. If the field is a group header, then I/O will be performed on all members of the group. Utilizing partial-record I/O can greatly improve performance when the PCF is a table in an RDBMS. This is especially true when the table contains many columns, but relatively few are being utilized by the process. The SCAN option is available for field selection.

See the Establish PCF Range section in Chapter 3-16: Using Event Points for a discussion about the use of *Beg At/End At statement*s to further constrain records in a query process.

Enduser Selections Option

The ENDUSER SELECTIONS option allows quick access to interactive record selections. These are the selections that the operator sees at runtime. *Enduser Selections* specifications define data relationships that you can use to constrain the information that is available to a process. For example, you might define an output function to list information about domestic customers. A related query selects only those records with a domestic value in the country field. If an application user is permitted to see the record selection criteria, the frame is considered *interactive*, which qualifies it as an *Enduser Selection*. This does not imply that the user can modify the criteria. You can establish both *editable* and *non-editable* Enduser selection criteria.

The Enduser Selections overlay, shown in *Figure 3-9-5*, displays when you select the Enduser Selections option from the Query Processes screen.

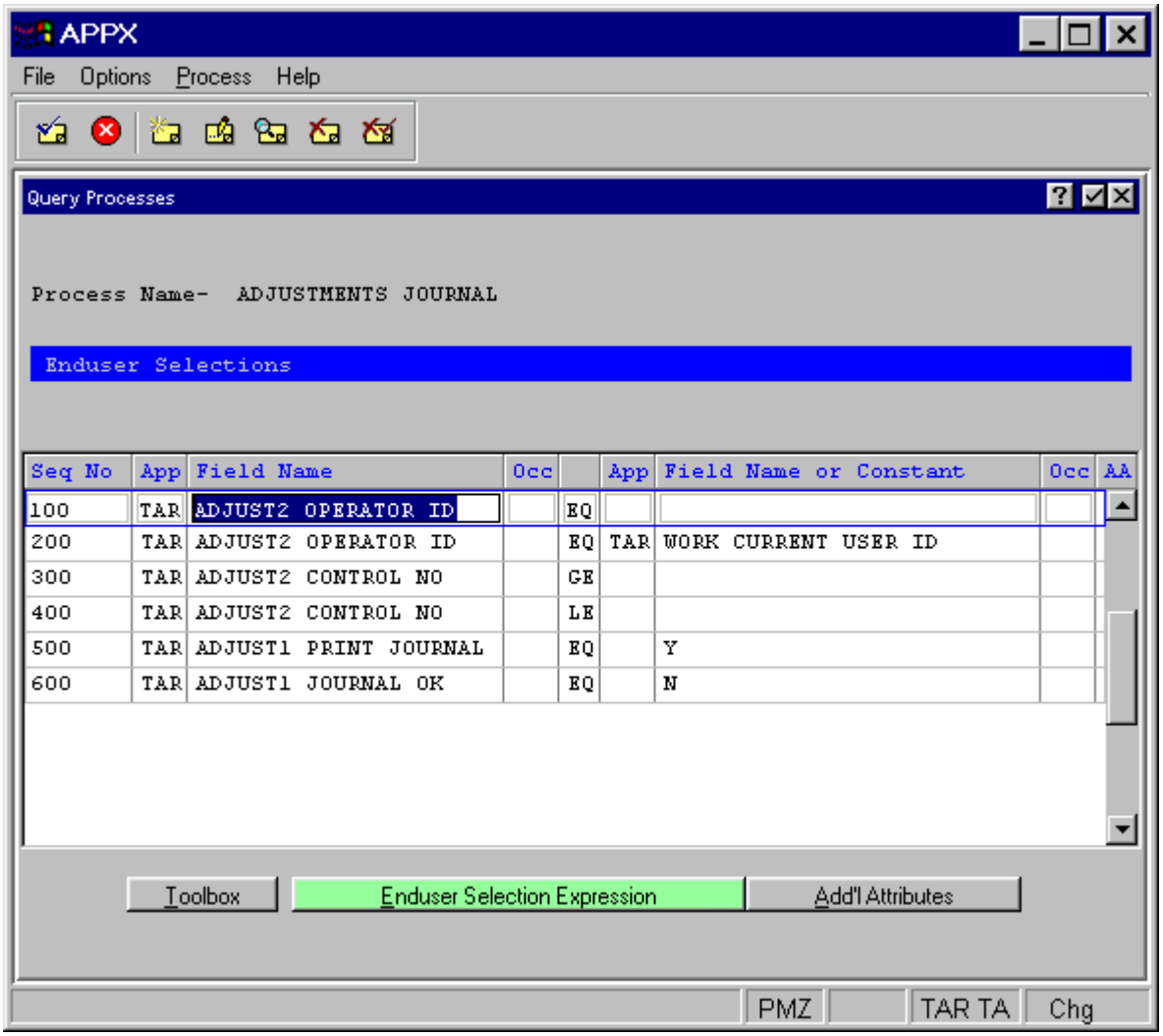


Figure 3-9-5. Enduser Selections Overlay

The Enduser Selections overlay contains the following fields:

Seq No

Displays a sequence number that is defaulted by APPX and used to reference the constraint in an *Enduser Selections Expression*. The sequence number controls the order that constraints display to a user. Unless an *Enduser Selections Expression* is entered on the Enduser Selections Expression specification overlay, all constraints are ANDed.

Left— App/Field Name/Occ

Identifies a field and optional occurrence that is to be compared with another field or constant. During query setup, these fields appear on the left-hand side of the screen.

Relation

Defines a relationship between the *App/Field Name/Occ* entered previously and the *App/Field Name or Constant/Occ* entered below. This field may contain up to two alphanumeric characters. The *Relation* options are:

- ***EQ*** (equal to)
- ***NE*** (not equal to)
- ***GT*** (greater than)
- ***LT*** (less than)
- ***GE*** (greater than or equal to)
- ***LE*** (less than or equal to)
- ***IN*** (includes)

Right— App/Field Name or Constant/Occ

Indicates the field or constant that serves as the object of the relation. For field entries other than from the PCF, the field must be explicitly made available with statements. The designer can also specify a NULL constant by entering " " (that's double-quote, space, double-quote) for alpha, numeric, date, and (effective with APPX 4.1.a) token fields.

AA

Is an indicator that is present when additional attributes have been defined for these enduser selections.

Three options are available on the Enduser Selections overlay and are described in the following sections: Toolbox, Enduser Selections Expression, and Additional Attributes.

Enduser Selections Toolbox Option

Several options that are described below are available from the Enduser Selections Toolbox overlay. See *Chapter 3-3, Process Definition* for additional information.

Database Management

The DATABASE MANAGEMENT option allows you to add new files or maintain existing ones without leaving query process design.

Process Management

The PROCESS MANAGEMENT option allows you to add other types of processes or maintain existing ones without leaving query process design.

Design Transfer

The DESIGN TRANSFER option allows you to copy the attributes from a process that you have already designed and apply it to a new process.

Syntax Check

This feature allows you to create the Executable Module for the current query process without actually running it. If the Executable Module for the query process is valid this option is ignored when invoked. Note that this option (`OPTION 97`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

View Audit Info

This feature allows you to see who created the currently selected query process and who changed it most recently. It also gives the date of creation and the date it was last changed. Note that this option (`OPTION 98`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Invoke a Process

This feature is available to facilitate testing. It allows you to invoke any process; the currently selected query process using the Test Database defined in system administration is the default. Note that this option (`OPTION 99`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Enduser Selections Expression Option

The ENDUSER SELECTIONS EXPRESSION option accesses the Enduser Selections Expression overlay, shown in *Figure 3-9-6. Enduser Selections Expression* specifications establish complex logical relationships between record selection criteria by using ANDs, ORs, and parentheses. You define separate selections expressions for Enduser and Designer record selections. APPX ANDs these two selections expressions.

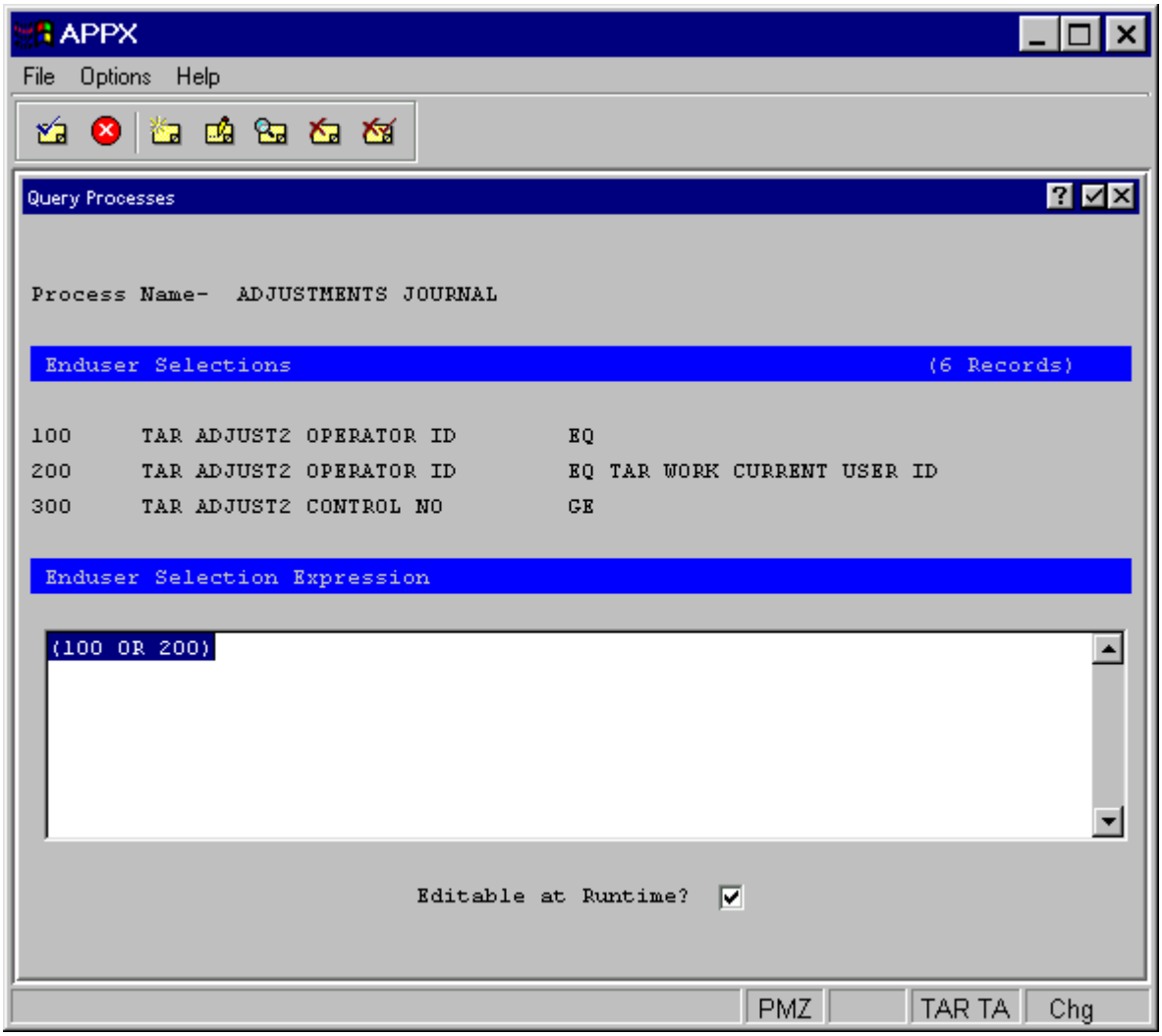


Figure 3-9-6. Enduser Selections Expression Overlay

The Enduser Selections Expression overlay contains two fields.

Enduser Selections Expression

Defines a logical expression that joins record selection constraints established in the Enduser Selections overlay. Constraints are identified by their sequence number. They can then be combined with other constraints by using ANDs, ORs, and parentheses. Valid entries in a selections expression are ***any sequence number*** that a constraint is defined for, AND, OR, and ( ).

- If *Enduser Selections Expression* is blank, all record selection constraints are ANDed. If either the *Relation* or the *App/Field Name or Constant/Occ* selections specification is blank, the relation is ignored (considered to be true). Any selections specification that is not explicitly referenced by a sequence number in a selections expression must be true for the selections expression to be true. A selections expression using all of the various *Selections Expression* entries might look something like the following:
- ((200 OR 300) AND 1000) OR (400 - 700)

Editable at Runtime?

Indicates whether or not a user can change the selections expression during query setup. The default is "yes" ☑ but can be toggled to "no" ☐ by clicking the checkbox.

Enduser Selections Add'l Attributes Option

The ADDITIONAL ATTRIBUTES option allows you to prevent the enduser from modifying the *Relational Operator*, the *Right Side Field or Operand*, or both, during runtime. In addition, the *Begin At?* and *End At?* flags allow you to constrain the reading of PCF records if desired.

When you select the *Additional Attributes* option from the Enduser Selections overlay, the overlay shown in *Figure 3-9-7* appears.

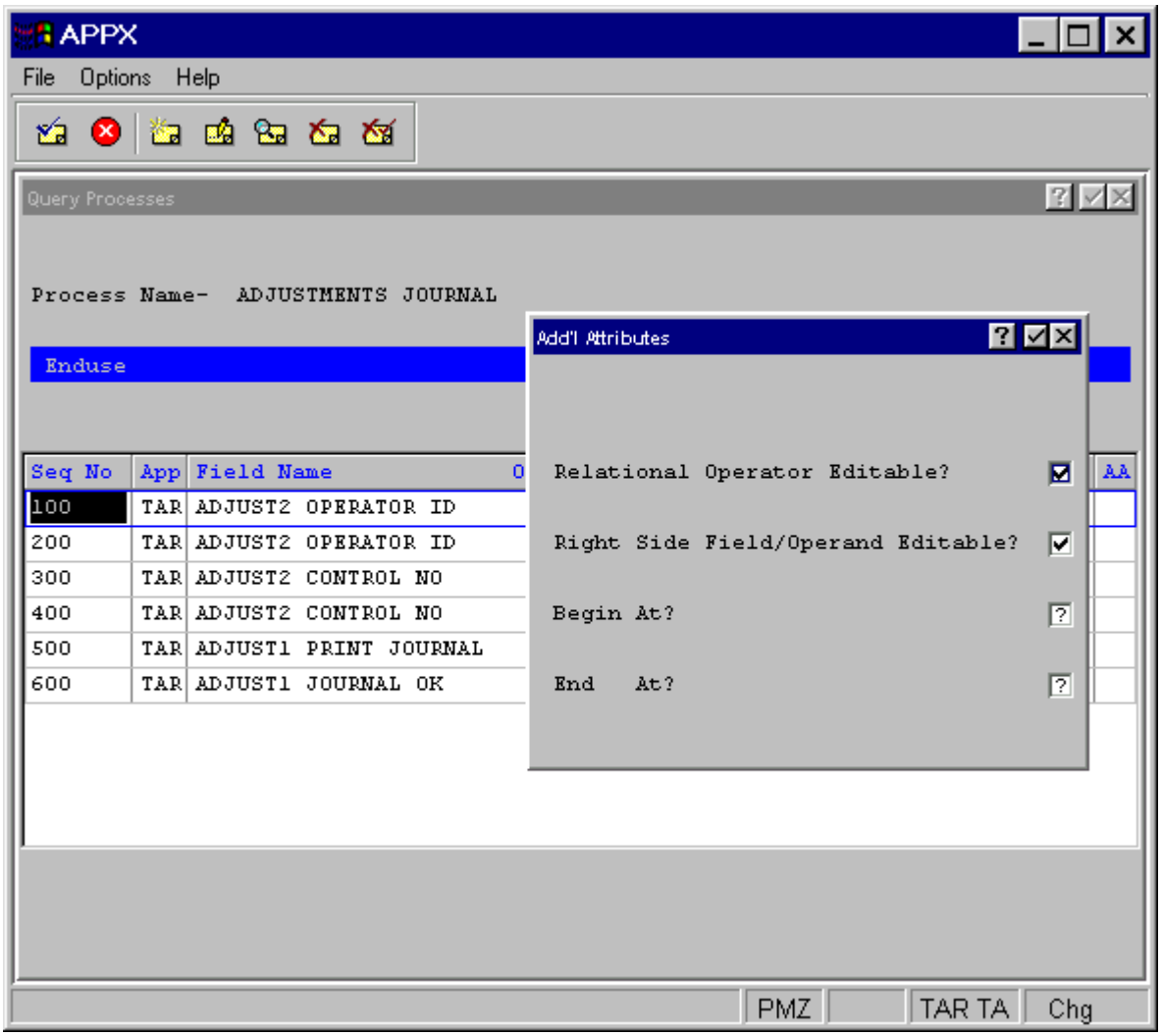


Figure 3-9-7. Enduser Selections Additional Attributes Overlay

The Enduser Selections Additional Attributes overlay contains the following fields.

Relational Operator—Editable?

Select "no" ☐ if you want to prevent the enduser from modifying the relational operator at runtime. The default is "yes" ☑.

Right Side Field or Operand—Editable?

Enter "no" ☐ if you want to prevent the enduser from modifying the right-side operand. The default is "yes" ☑.

Begin At?

Indicates whether or not the reading of PCF records should be constrained beginning at the value of this field's right side operand. This may significantly reduce runtime if the user intends to select just a subset of records within the PCF file. Click the checkbox to toggle between "yes" ☑ and "no" ☐.

End At?

Indicates whether or not the reading of PCF records should be constrained ending at the value of this field's right side operand. This may significantly reduce runtime if the user intends to select just a subset of records within the PCF file. Click the checkbox to toggle between "yes" ☑ and "no" ☐.

Designer Selections Option

The DESIGNER SELECTIONS option allows you to access non-interactive record selections. These function in exactly the same way as enduser selections, except that the operator never sees the runtime screens and, hence, cannot modify the operators. The Designer Selections overlay is shown in *Figure 3-9-8:*

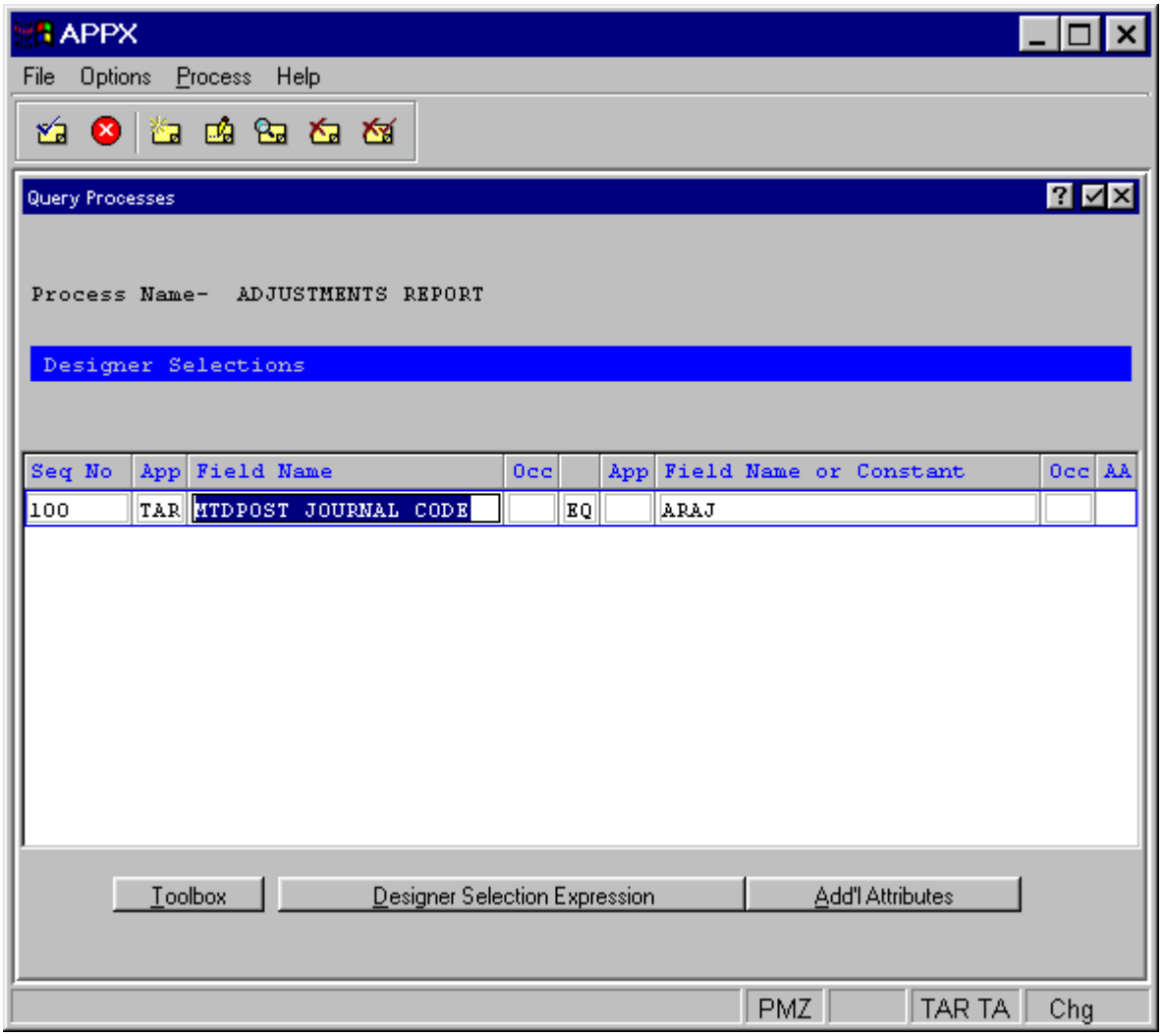


Figure 3-9-8. Designer Selections Overlay

Toolbox Option

The DATABASE MANAGEMENT option allows you to add new files or maintain existing ones without leaving query process design.

The PROCESS MANAGEMENT option allows you to add other types of processes or maintain existing ones without leaving query process design.

The DESIGN TRANSFER option allows you to copy the attributes from a process that you have already designed and apply it to a new process.

The SYNTAX CHECK feature allows you to create the Executable Module for the current query process without actually running it. If the Executable Module for the query process is valid this option is ignored when invoked. Note that this option, like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

The VIEW AUDIT INFO feature allows you to see who created the currently selected query process and who changed the currently selected query process most recently. It also gives the date of creation and the date it was last changed. Note that this option, like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

The INVOKE A PROCESS feature is available to facilitate testing. It allows you to invoke any process; the currently selected query process using the Test Database defined in system administration is the default. Note that this option, like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Designer Selections Expression Option

The DESIGNER SELECTIONS EXPRESSION option accesses the Designer Selections Expression overlay, shown in *Figure 3-9-9. Designer Selections Expression* specifications allow you to access non-interactive record selections. These function in exactly the same way as enduser selections, except that the operator never sees the runtime screens and, hence, cannot modify the operators.

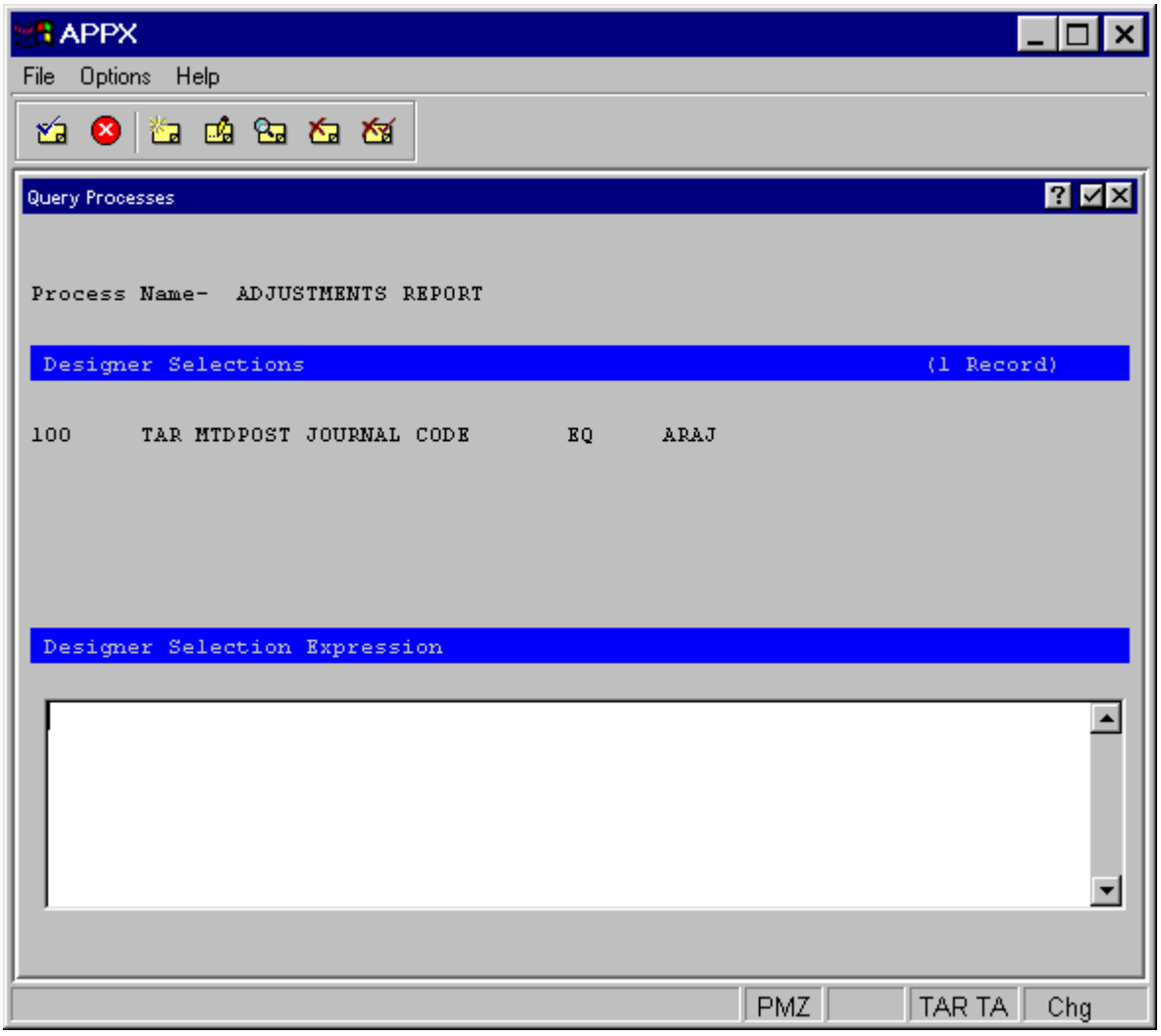


Figure 3-9-9. Designer Selections Expression Overlay

The Designer Selections Expression overlay contains the following field.

- ***Designer Selections Expression*** defines a logical expression that joins record selection constraints established in the Designer Selections overlay. Constraints are identified by their sequence number. They can then be combined with other constraints by using ANDs, ORs, and parentheses. Valid entries in a selections expression are ***any sequence number*** that a constraint is defined for, AND, OR, and ( ).
- If *Designer Selections Expression* is blank, all record selection constraints are ANDed. If either the *Relation* or the *App/Field Name or Constant/Occ* selections specification is blank, the relation is ignored (considered to be true). Any selections specification that is not explicitly referenced by a sequence number in a selections expression must be true for the selections expression to be true. A selections expression using all of the various *Selections Expression* entries might look something like the following:
- ((200 OR 300) AND 1000) OR (400 - 700)

> Note that the ADDITIONAL ATTRIBUTES option does not appear on the *Designer Selections* overlay like it does on the *Enduser Selections* overlay because it is not necessary to include the field ***Editable at Runtime?*** since the user is not allowed to modify these fields.

Sort Order Option

The SORT ORDER option provides quick access to the query process Sort Order overlay, shown in *Figure 3-9-10*. The Sort Order overlay also displays if you select ENTER , and then END from the Query Processes screen. Sort order specifications determine the order in which records are presented. You can establish a virtually unlimited number of sort items, with a maximum of 255 active at any one time. Active sort items display first by sequence number order, in their order of precedence. Non-active sort items can be activated by a user during query setup. Sort fields that are identified on the Sort Order overlay are related to range fields in *Start of Range* and *End of Range* frames in output, inquiry, and update processes.

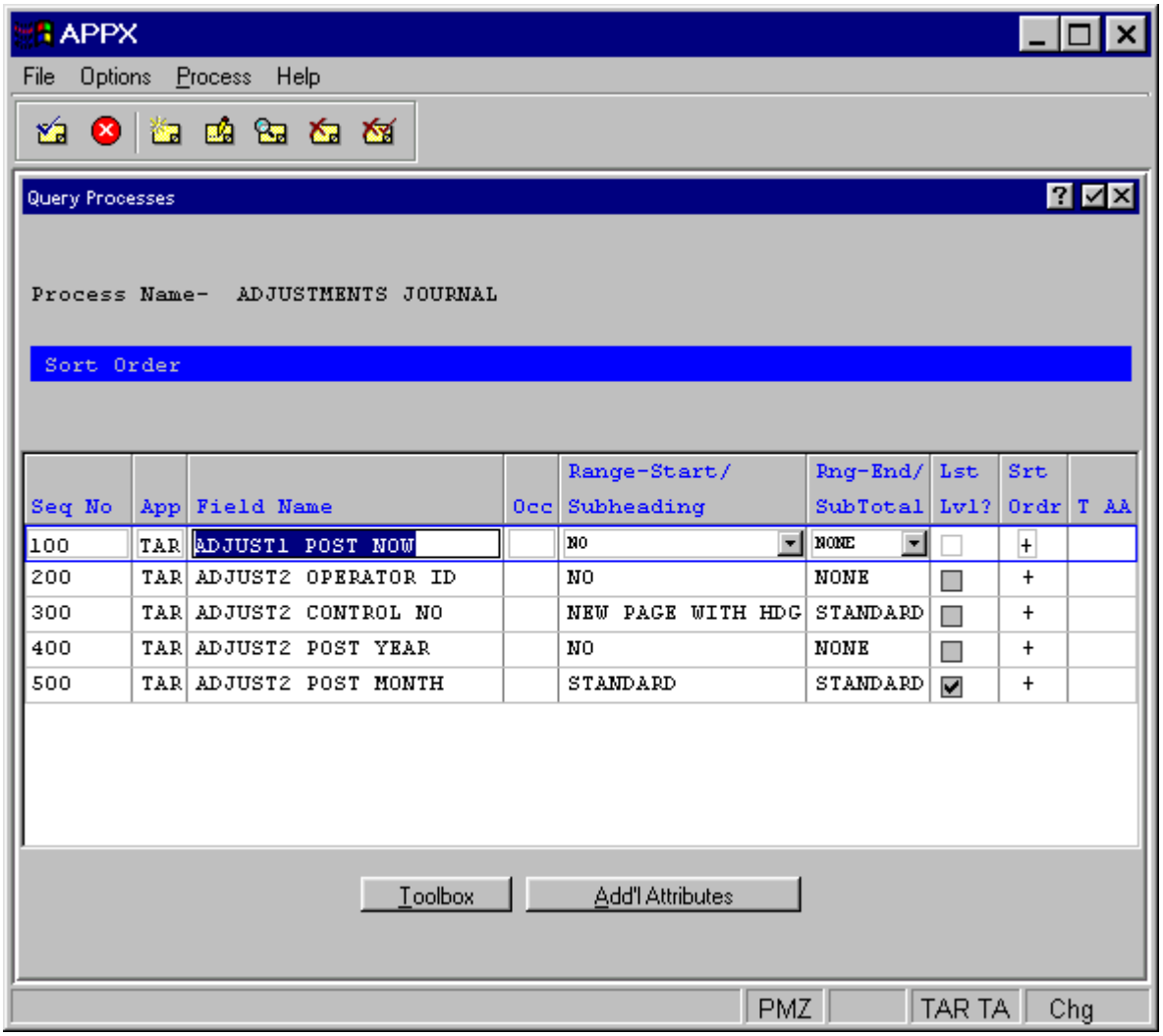


Figure 3-9-10. Query Processes Sort Order Overlay

The query Sort Order overlay contains the following fields.

Seq No

Defines the field's position within the sort order that is used to generate the report.

App/Field Name/Occ

Identifies a specific field on which to sort. This field must be maintained in the PCF or made available with statements.

Range-Start/Subheading

Allows you to choose whether or not to execute any *Range-Start* frames in a subsequent output or update process, for example, to print a subheading. The *Range-Start/Subheading* options are:

- *NO.* Any *Range-Start* frame should be ignored.
- *STANDARD.* A *Range-Start* frame will execute immediately after the previous *Record* or *Range-Start* frame.

- ***NEW PAGE.*** The report will skip to a new page before printing this *Range-Start* frame.
- ***NEW PAGE WITH MSG.*** The report will skip to a new page before printing this *Range-Start* frame, and will append a continuation message to it.

The default is *NO*.

Rng-End/SubTotal

Indicates whether or not to execute the *End of Range* frames, if any, for this sort field. The *Range-End/SubTotal* options are:

- ***NO.*** Any *Range-End* frame should be ignored.
- ***STANDARD.*** A *Range-End* frame will execute immediately after the previous *Record* or *Range-End* frame.
- ***NEW PAGE.*** The report will skip to a new page before printing this *Range-End* frame.

The default is *NO*.

Lst Lvl?

A "yes" in this checkbox designates the last field in the sort order. This may not necessarily be the last field defined since not all fields may be included in the sort. The default is "no" .

Sort Order (+/–)

Determines whether values are sorted in *ascending* (from lowest to highest value in the sort field), or *descending* (from highest to lowest value in the sort field) order. The default is + (ascending).

T

And ***AA*** are indicators that are present whenever documentation (T) or additional attributes (AA) have been defined for this process.

The following two options are available on the query Sort Order overlay:

Toolbox Option

The toolbox overlay appears on the screen when you invoke the TOOLBOX option from the query sort order overlay. The options listed in the Toolbox overlay are described briefly below. See *Chapter 3-3, Process Definition* for a more detailed explanation of the options listed on the Toolbox overlay.

The DATABASE MANAGEMENT option allows you to add new files or maintain existing ones without leaving query process design.

The PROCESS MANAGEMENT option allows you to add other types of processes or maintain existing ones without leaving query process design.

The SYNTAX CHECK option allows you to create the Executable Module for the current query process without actually running it. If the Executable Module for the query process is valid this option is ignored when invoked. Note that this option, like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

The VIEW AUDIT INFO feature allows you to see who created the currently selected query process and who changed it most recently. It also gives the date of creation and the date it was last changed. Note that this option, like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

The INVOKE A PROCESS feature is available to facilitate testing. It allows you to invoke any process; the currently selected query process using the Test Database defined in system administration is the default. Note that this option, like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Additional Attributes Option

The Sort Order Additional Attributes overlay contains the following fields:

Sort on This Field Required?

Select "yes" ☑ or "no" ☐ to control whether or not this field is required for sorting. The designer may force a user to sort on this field by selecting "yes". A user running the query may change the sort order, but a sort on this field is required. The default is "no"

Subheading NO Edit?

Allows you to specify whether a user can modify the subheading when the query is run. The default is ☑.

Subtotal NO Edit?

Allows you to specify whether a user can modify the subtotal when the query is run. The default is ☑.

Sort Order + Edit?

Allows you to specify whether a user can modify the sort order when the query is run. The default is ☑.

Grand Totals Option

GRAND TOTALS allows you to edit certain parameters that are related only to report grand totals. The Grand Totals overlay is shown in *Figure 3-9-11*.

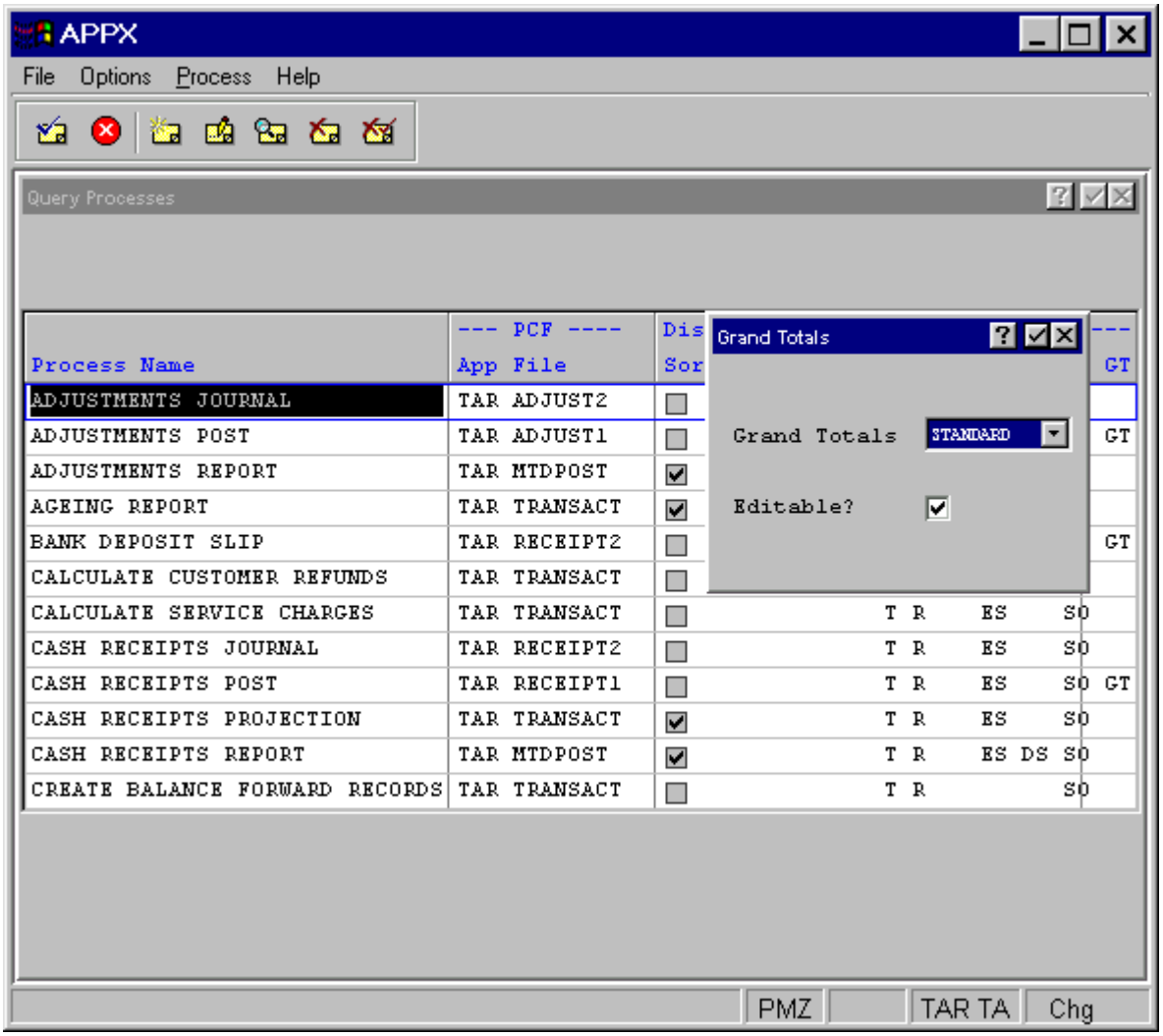


Figure 3-9-11. Query Process Grand Totals Overlay

The query process Grand Totals overlay contains the following fields.

- ***Grand Totals*** allows you to specify that a subsequent output process prints grand totals. The *Grand Totals* options are:
- ***NO.*** There will be no grand totals.
- ***STANDARD.*** The grand totals will print directly following the last *Record* or *Range-End* image.
- ***NEW PAGE.*** The report will skip to a new page before printing grand totals.

The default is *STANDARD*.

- ***Editable?*** field allows you to explicitly choose whether or not a user can modify this parameter when the query is run.

Populating Query Files

QPARM, QSLCT, and *QSORT* are temporary files that are established during query setup. They store the sort and record selection criteria that are established for this query execution. As a result, the records that generate into the pointer file during execution are directly determined by the content of these files.

The temporary files are initially populated with the sort and record selection criteria that are established in application design (in the query process' source files). When an application user modifies the defaults on sort and record selection displays, the modifications are made directly to these files.

If the query is active, the temporary files are already populated. If, within the confines of a subprocess or related process family, you execute another sort setup for this query, the contents of these existing files *can* be reused. Whether or not they are used, and how they are used, depends upon the mode the query process was invoked in and the *Query Update Type.* In Chapter 3-13: Automatic and Optional Children, refer to Child Additional Attributes Option for information about *Override Default Mode*. Also, see Child Use Query Option for a discussion of *Query Update Type*.

CHAPTER 3-10

Inquiry Processes

Introduction to Inquiry Processes

Inquiry Processes convey information from the database, usually to the screen, although the hard copy is available. The format is designer-defined. Customer Transactions and account balance information are examples of inquiry processes. Inquiry processes share many of the features of output processes, with one exception.: An inquiry process automatically generates a Key Entry frame, which allows the user to enter the key to the desired record. APPX assumes that an inquiry process is a means to quickly access a specific record, or set of records, rather than to sift the database for a range of records.

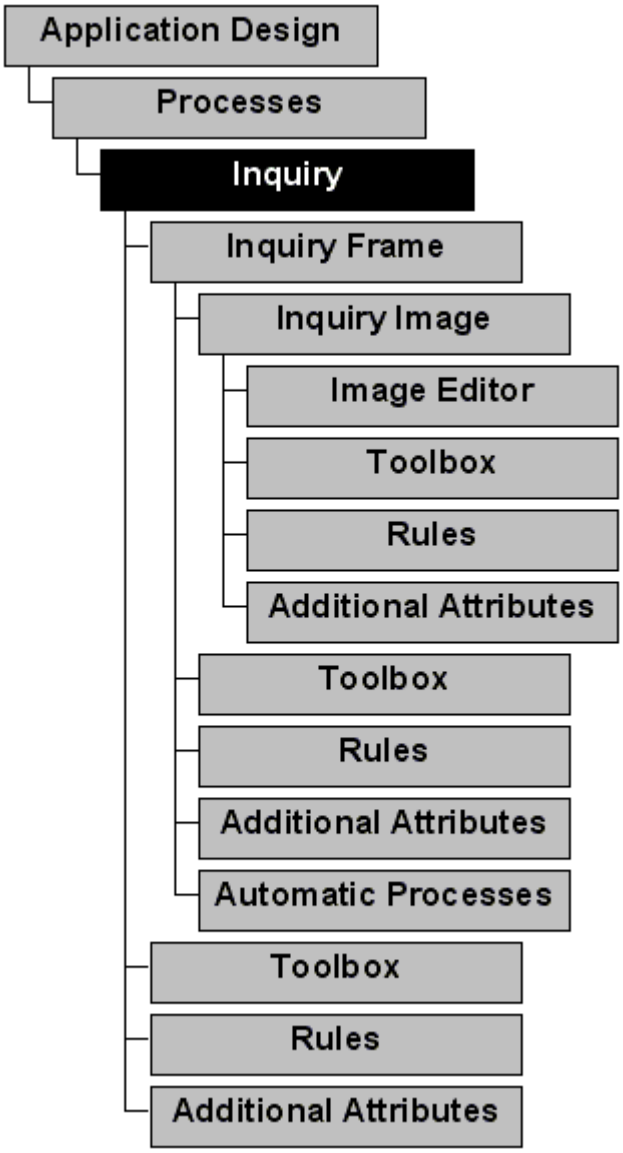


Figure 3-10-1. Inquiry Process Definition

Inquiry Process Specifications

Inquiry process specifications establish the general attributes of an inquiry process. They include such specifications as the inquiry print file, if any, the process control file, the name of the inquiry, and format characteristics. The Inquiry Processes screen is shown in *Figure 3-10-2*.

Inquiry Processes screen

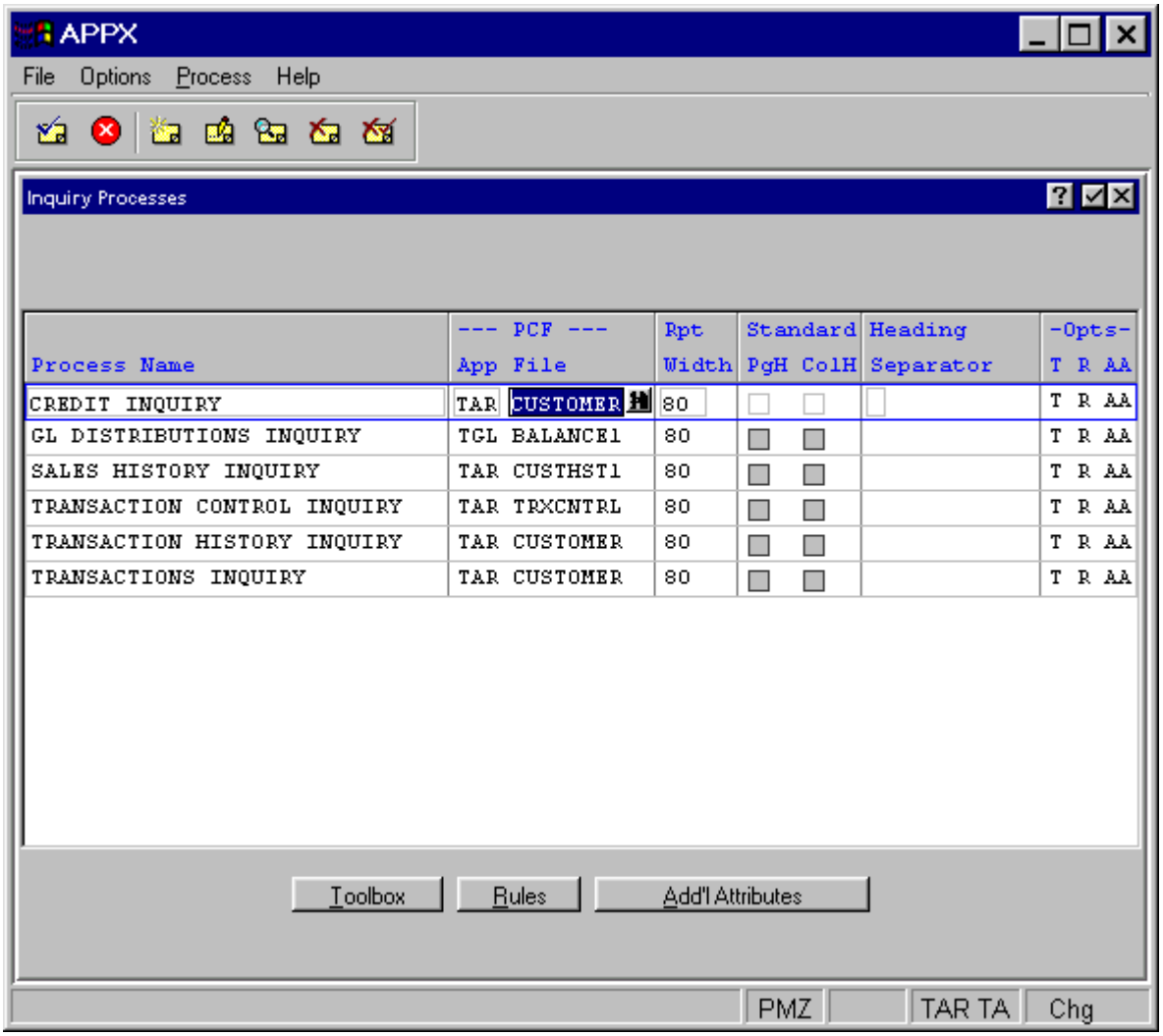


Figure 3-10-2

The Inquiry Processes screen contains the following inquiry-specific fields.

Process Name

Identifies a specific process that is unique within type and application.

App/PCF Name

Identifies a file by application ID and file name that is to be automatically managed by APPX during the execution of this process. APPX automatically reads the process control file (PCF) during inquiry process execution.

Rpt Width

Indicates the maximum number of characters across, or in other words the number of columns, to allocate for each row in the report format. APPX uses the value you enter here to produce standard images of the proper width. If blank, the default width of the form that the report prints on defaults to the form width which is defined in system administration. If the form is narrower than the width of the formatted report, APPX truncates the inquiry. If the form is wider than the width of the report, however, APPX uses the larger width for standard images, but does not expand the format of the report itself. The default is *80*.

Standard PgH

Determines whether or not APPX automatically prints a page heading standard image. If "yes" ☑, this image prints before any *Start of Page* frames for the report. Be careful not to also explicitly define a *Start of Page* frame with a page heading standard image, which could cause the standard image to print twice at the top of each page in the report. If "no" ☐, you can explicitly define a *Start of Page* frame with a page heading standard image or, alternatively, define your own page heading using a *Start of Page* frame. The default is "yes".

Standard ColH

Determines whether or not APPX automatically prints a *Start of Page* frame with a column heading standard image. APPX generates column headings based upon specifications in the data dictionary for each item included in the inquiry image. If "yes" ☑, this image prints after any *Start of Page* frames for the report. Be careful not to also explicitly define a *Start of Page* frame with a column heading standard image, which would cause the standard image to print twice for each page in the report. If "no" ☐, you can explicitly define a *Start of Page* frame with a column heading standard image or, alternatively, define your own column headings using a *Start of Page* frame. The default is "yes". Refer to Standard Column Heading Image for additional information.

Heading Separator

Identifies one alphanumeric character that prints continuously across the report to separate standard page headings, page frames, and standard column headings from the body of the report. If blank, APPX does not print a heading separator. The default is =.

Opts

Are indicators that are present whenever documentation (T), rules (R), or additional attributes (AA) have been defined for this process.

The following options as described in the corresponding sections are available on the Inquiry Processes screen:

- Toolbox
- Rules
- Additional Attributes

Inquiry Process Toolbox Option

The toolbox overlay appears on the screen when you invoke the TOOLBOX option from the inquiry processes screen. The overlay contains a number of features such as Documentation Management and Security Management that will assist you with designing and invoking an inquiry process. The options listed in the Toolbox overlay are described briefly below. See *Chapter 3-3, Process Definition* for a more detailed explanation of the options listed on the Toolbox overlay.

Documentation

When you select the DOCUMENTATION option from the Toolbox overlay, you will first see the Documentation overlay which allows you to select between user or technical documentation. Once you have entered your documentation, you are returned to the Inquiry Processes screen.

Security

The SECURITY option allows you to identify the APPX security code to which a user must have rights in order to run the inquiry process. The security code must be four numeric characters. If blank, there are no restrictions.

Database Management

The DATABASE MANAGEMENT option allows you to add new files or maintain existing ones without leaving inquiry process design.

Process Management

The PROCESS MANAGEMENT option allows you to add other types of processes or maintain existing ones without leaving inquiry process design.

Design Transfer

The DESIGN TRANSFER option allows you to copy the attributes from a process that you have already designed and apply it to a new process.

Cross Reference

The CROSS REFERENCE option enables you to determine the location and frequency of a selected domain, file, field, application ID, process or ILF statement within a specific database.

Reference Browser Option

The REFERENCE BROWSER option runs a modified version of "Cross Reference – All" and produces a list of design components utilized by the currently selected inquiry process. The results can be displayed interactively where links allow you to see the context of the references, edit the referenced component and drill down to the child process. This option can be used to identify, and optionally edit, the file level event points utilized by this inquiry process.

Syntax Check

This feature allows you to create the Executable Module for the current inquiry process without actually running it. If the Executable Module for the inquiry process is valid this option is ignored when invoked. Note that this option (`OPTION 97`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

View Audit Info

This feature allows you to see who created the currently selected inquiry process and who changed the currently selected inquiry process most recently. It also gives the date of creation and the date it was last changed. Note that this option (`OPTION 98`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Invoke a Process

This feature is available to facilitate testing. It allows you to invoke any process; the currently selected inquiry process using the Test Database defined in system administration is the default. Note that this option (`OPTION 99`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Inquiry Process Rules Option

Selecting the RULES option accesses the inquiry process Event Points For Rules overlay which allows you to select from options that access the ILF Editor for APPX event points. The inquiry process Event Points overlay contains the following options:

Start of Process

START OF PROCESS, which is common to all types of processes, accesses the ILF editor for the *Start of Process* event point. Upon completion, you are returned to the process Event Points overlay.

Post PCF Read

POST PCF READ accesses the ILF editor for the *Post PCF Read* event point. Upon completion, you are returned to the process Event Points overlay.

End of Process

END OF PROCESS, which is common to all types of processes, accesses the ILF editor for the *End of Process* event point. Upon completion, you are returned to the process Event Points overlay.

PCF Level EPs

PCF LEVEL EPS provides access to the File Level Event Points for the PCF file. This overlay is shown in *Figure 3-10-3.* These Event Points are not directly associated with the process but with the PCF file. See the *Rules Option* section of *Chapter 2-4, Files and Fields* for more information on using PCF Level Event Points.

Refer to Chapter 3-16: Using Event Points for more information.

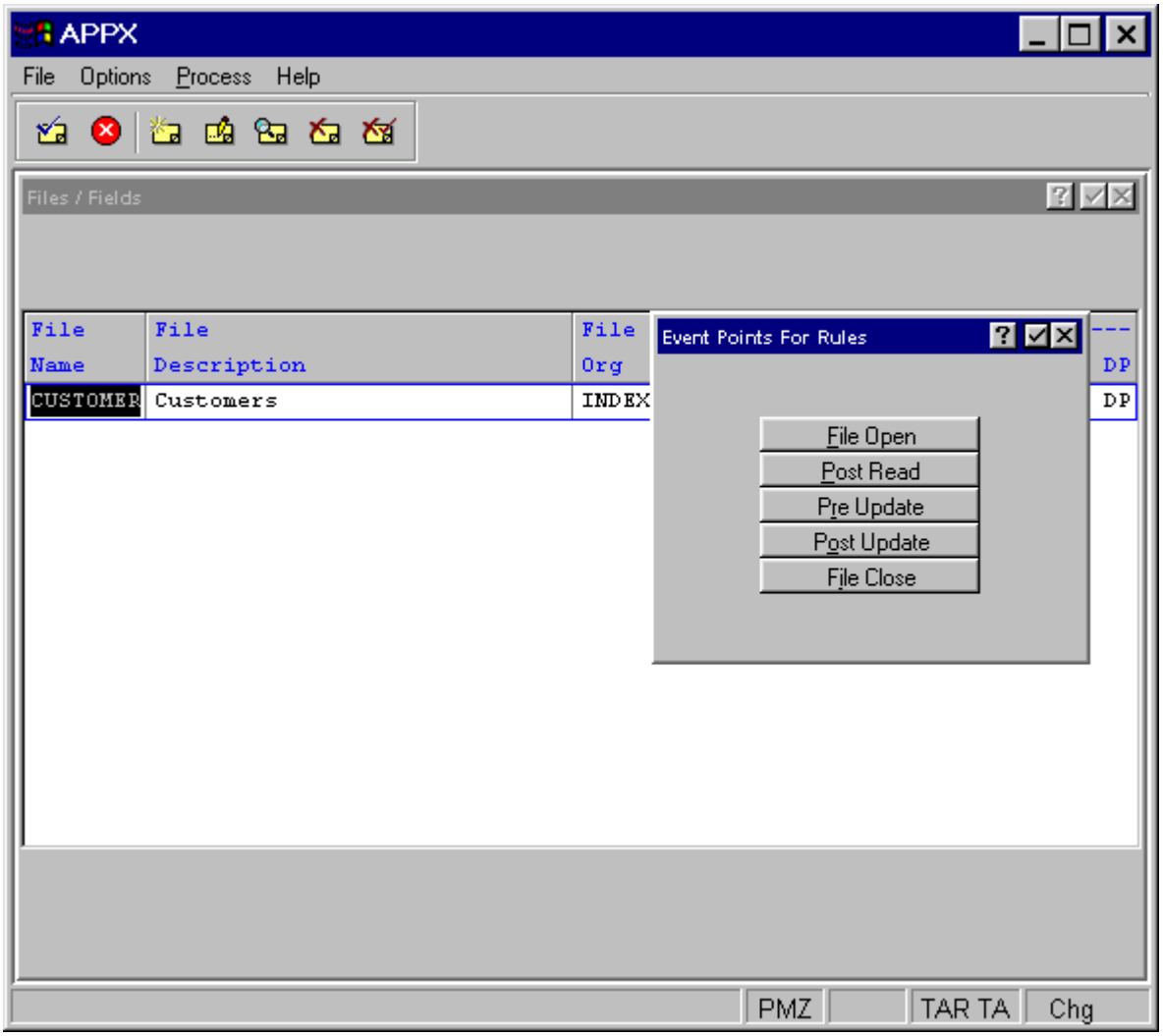


Figure 3-10-3. File Level Event Points for Rules Overlay

Inquiry Process Additional Attributes Option

The ADDITIONAL ATTRIBUTES option accesses the Inquiry Process Additional Attributes overlay, shown in *Figure 3-10-4*, from which you can alter the standard EM attributes.

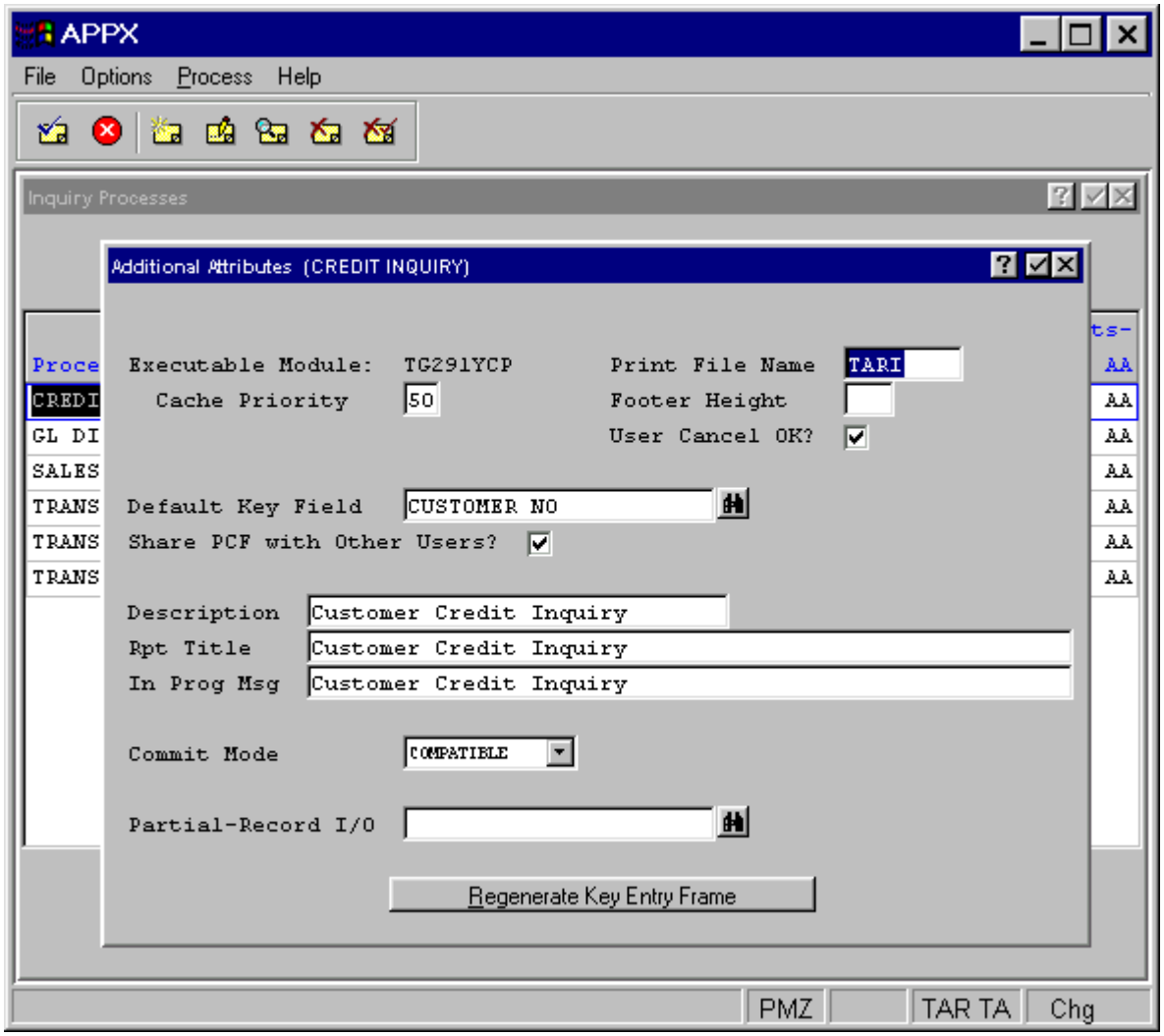


Figure 3-10-4. Inquiry Process Additional Attributes Overlay

The Additional Attributes overlay for Inquiry processes contain the following fields.

Executable Module

Displays the name of the executable module (EM) that is automatically generated by APPX. APPX derives the file name from the date and time the currently selected inquiry process was added. The name consists of a unique string of eight alphanumeric characters and cannot be changed. If APPX finds it necessary to recreate the EM, the new EM retains the same file name. This field is common to the Additional Attributes overlay for all types of processes. See *Chapter 3-3, Process Definition*, for more information.

Print File Name

Establishes the name of the file that is used to store the contents of the printed inquiry. If necessary to guarantee uniqueness, APPX may append one or more characters to the value you enter here, or replace one or more of the ending characters. If two or more output/inquiry processes in the same subprocess family are defined with identical *Print File* values, the processes share a common print file. They print as a single report, with continuous page numbers. If this specification is blank, APPX uses the last *Print File* that is active within the subprocess family.

Cache Priority

Establishes the likelihood that the process will remain in memory after execution, relative to other processes that the user has also completed executing. A process with a cache priority of *99* is most likely to remain in memory. A process with a cache priority of *1* is least likely to remain in memory. A cache priority of *0* indicates that the process is never cached. The default is *50*. This field is common to the Additional Attributes overlay for all types of processes. See *Chapter 3-3, Process Definitio*n, for more information.

Footer Height

Indicates the amount of space, in number of rows, to reserve on each page for printing the page footer. APPX uses this value to determine placement of *End of Page* frames. If blank, APPX does not reserve space for a page footer.

User Cancel OK?

Contains either “yes” ☑ or “no” ☐ and determines whether or not a user can select the CANCEL option key to stop the execution of this process. The default is “yes”. You can conditionally control the user’s ability to select the CANCEL option with statements and the predefined field CANCEL OK. This field is common to the Additional Attributes overlay for all types of processes.

Default Key Field

Designates the specific keypath to the PCF that is used in executing the defined inquiry. If blank, the primary key is used. A default keypath that is defined for the invocation of a child process overrides this specification.

Share PCF with Other Users?

Indicates whether other users can access the designated PCF file when this process is executing. Default is “yes” ☑ but can be toggled to “no” ☐ by clicking the checkbox.

Description

Is incorporated in the message that displays on the screen when the inquiry process is compiling. It is used to communicate with users since it provides a more descriptive and easily understood alternative to *Process Name*, which is the default if this field is left blank. The Additional Attributes overlay for all types of processes contain this field. See *Chapter 3-3, Process Definitio*n, for more information.

Rpt Title

Defines the text that is used to name the report. This text automatically displays in the standard image for a *Start of Page* frame which is printed at the top of each page of the inquiry.

In Prog Msg

Provides text for the message that displays to the screen while the report is executing in the foreground.

Commit Mode

Controls the commit mode for a process and can be selected from a drop-down list of available options. *Commit Mode* determines those points in time when APPX will commit changes made to the tables stored within an RDBMS. It is only applicable in conjunction with an RDBMS; it has no effect with APPXIO. Also, not all File Managers support this feature. Although initially set here for the process, it can be added or overridden by use of predefined field COMMIT MODE. The following options are available:

COMPATIBLE. Changes are committed as soon as all locks are freed. This provides maximum compatibility between APPXIO files and RDBMS tables.

AUTOCOMMIT*. APPX will commit each modification as it is made.
***This value will be implemented in a future release.**

PCF CYCLE. APPX will commit modifications as each PCF record is processed.

END. APPX will only commit modifications at the end of the process family.

MANUAL. APPX will not automatically commit changes. You must use the COMMIT statement to do so.

- ***Partial-Record I/O*** is used to specify a field from the PCF file so that instead of operating on the entire PCF record, APPX operates on the designated field only. If the field is a group header, then I/O will be performed on all members of the group. Utilizing partial-record I/O can greatly improve performance when the PCF is a table in an RDBMS. This is especially true when the table contains many columns, but relatively few are being utilized by the process. The SCAN option is available for field selection.

The following option is available on the inquiry Additional Attributes overlay.

Regenerate Key Entry Frame Option

The REGENERATE KEY ENTRY FRAME option allows you to re-create the automatically generated, but modifiable, *Key Entry* frame that makes inquiry unique and different from output. If you have modified the *Key Entry* frame for any reason and wish to return to the standard frame, select this option.

Inquiry Frame Specifications

Inquiry frame specifications define attributes and statements unique to frames that manipulate the keys from each record in a PCF. The inquiry frames overlay, shown in *Figure 3-10-5*, is automatically generated after you select ENTER from the Inquiry Processes screen.

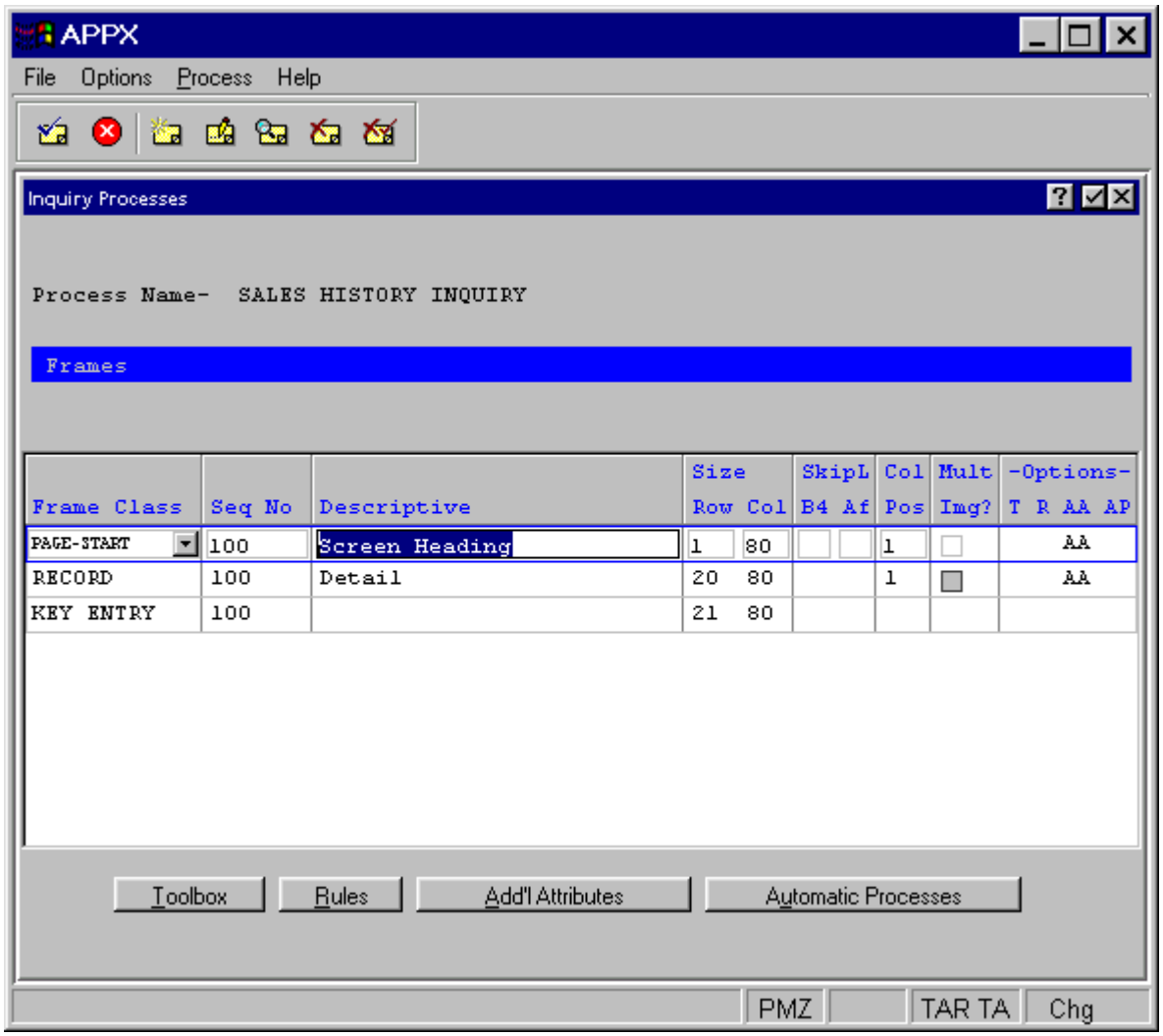


Figure 3-10-5. Inquiry Frames Overlay

The inquiry process Frames overlay contains the following fields.

- ***Frame Class*** defines the type of frame with which you are working. Valid options are:
- ***Report-Start*** frames print once at the beginning of a report. To print on a separate page, the designer should set *New Page After?* in Inquiry Frame Additional Attributes for the last *Report-Start* frame.
- ***Page-Start*** frames print once at the beginning of each page of a report. They are primarily used for page headings and column headings. Standard images for both of these are optionally available. Refer to Standard Column Heading Image for additional information.

- ***Range-Start*** frames print immediately before the first record for which the value in a specific sort field (the range field) changes. They are primarily used for subheadings and lower-level column headings. Standard images for both of these are optionally available. Refer to Standard Column Heading Image for additional information.
- ***Record*** frames manipulate and print information read from the process control file. They comprise the body of a report. *Record* frames can be defined to scroll.
- ***Range-End*** frames print immediately after the last record in a series of records that have the same value in a specific sort field (the range field). They are primarily used for subtotals, and a standard subtotal image is optionally available.
- ***Page-End*** frames print once at the conclusion of each page of a report.
- ***Report-End*** frames print once at the conclusion of a report. To print on a separate page, the designer should set *New Page Before?* in Inquiry Frame Additional Attributes for the first *Report-End* frame.
- ***Key-Entry*** frames manipulate the keys from each record in a PCF.
- ***Seq No*** displays a sequence number that is defaulted by APPX and used to identify the frame. This default sequence number can be overridden, but must be unique within the frame class for the process. Sequence numbers determine the order of the frames in the process structure and may contain up to seven numeric characters. The first frame that is defined in add mode is assigned sequence number *100*, and additional frames are assigned a number equal to the highest existing sequence number, plus *100*.
- ***Descriptive*** identifies this frame. The description is used for documentation purposes only.
- ***Size—Row/Col*** designates the number of rows and columns that compose the frame. The default is *21*, *80*.
- ***SkipL B4***, displayed for all Inquiry frame classes except Key Entry, indicates the number of blank lines to be printed before executing this frame. For optional or repeating frames, it may be more appropriate to specify lines to skip at the image level in Inquiry Image Additional Attributes. The default is *0*.
- ***SkipL Af***, displayed for all Inquiry frame classes except Key Entry, indicates the number of blank lines to print subsequent to executing this frame.. For optional or repeating frames, it may be more appropriate to specify lines to skip at the image level in Inquiry Image Additional Attributes. The default is *0*.
- ***Col Pos*** designates how many columns over from the left side of the image that the report will print. See Frame/Image Positioning in *Chapter 3-2* for more information.
- ***Mult Img?*** This field indicates that this output frame contains more than one image if checked “yes” . If “no” is entered, you will be unable to add more than one entry at the image level. The default is “no”.
- ***Options*** are indicators that are present whenever documentation (T), rules (R), additional attributes (AA), or automatic processes (AP) have been defined for this process.

The following options as described in the corresponding sections are available on the inquiry Frames overlay:

- Toolbox
- Rules
- Additional Attributes
- Automatic Processes

Inquiry Frame Toolbox Option

The toolbox overlay appears on the screen when you invoke the TOOLBOX option from the Inquiry Frames overlay. The Inquiry Frames Toolbox overlay contains a number of features such as Documentation Management and Security Management that will assist you when designing and invoking an inquiry process. The options listed in the Toolbox overlay are described briefly below. See *Chapter 3-3, Process Definition* for a more detailed explanation of the options listed on the Toolbox overlay.

Database Management Option

The DATABASE MANAGEMENT option allows you to add new files or maintain existing ones without leaving inquiry process design.

Process Management Option

The PROCESS MANAGEMENT option allows you to add other types of processes or maintain existing ones without leaving inquiry process design.

Design Transfer Option

The DESIGN TRANSFER option allows you to copy the attributes from a process that you have already designed and apply it to a new process.

Syntax Check Option

This feature allows you to create the Executable Module for the current inquiry process without actually running it. If the Executable Module for the inquiry process is valid this option is ignored when invoked. Note that this option (`OPTION 97`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

View Audit Info

This feature allows you to see who created the currently selected inquiry process and who changed it most recently. It also gives the date of creation and the date it was last changed. Note that this option (`OPTION 98`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Invoke a Process

This feature is available to facilitate testing. It allows you to invoke any process; the currently selected inquiry process using the Test Database defined in system administration is the default. Note that this option (`OPTION 99`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Inquiry Frame Rules Option

Selecting the RULES option accesses the inquiry frame Event Points For Rules overlay which allows you to select from options that access the ILF Editor for APPX event points. If statements were defined in the event point, the text on the screen is highlighted. Refer to Chapter 3-16: Using Event Points for more information.

The Inquiry Frames Event Points overlay contains the following options:

Start of Process

START OF PROCESS, which is common to all types of processes, accesses the ILF editor for the *Start of Process* event point. Upon completion, you are returned to the process Event Points overlay.

Post PCF Read

POST PCF READ accesses the ILF editor for the *Post PCF Read* event point. Upon completion, you are returned to the process Event Points overlay.

Select Image

SELECT IMAGE accesses the ILF editor for the *Select Image* event point. Upon completion, you are returned to the inquiry frame Event Points overlay.

Global EP's For Images

GLOBAL EP'S FOR IMAGES overlay allows you to access three additional event points which are described below.

- START OF FRAME accesses the ILF editor for the *Start of Frame* event point. Upon completion, you are returned to the inquiry frame Event Points overlay.

- GLOBAL PRE-DISPLAY accesses the ILF editor for the *Global Pre-Display* event point. Upon completion, you are returned to the inquiry frame Event Points overlay.
- END OF FRAME accesses the ILF editor for the *End of Frame* event point. Upon completion, you are returned to the inquiry frame Event Points overlay.

End of Process

END OF PROCESS accesses the ILF editor for the *End of Process* event point. Upon completion, you are returned to the process Event Points overlay.

PCF Level EPs

PCF LEVEL EPS provides access to the File Level Event Points for the PCF file. This overlay is shown in *Figure 3-10-3.* These Event Points are not directly associated with the process but with the PCF file. See the *Rules Option* section of *Chapter 2-4, Files and Fields* for more information on using PCF Level Event Points.

Refer to Chapter 3-16: Using Event Points for more information.

Inquiry Frame Additional Attributes Option

The ADDITIONAL ATTRIBUTES option accesses the inquiry frame Additional Attributes overlay, which varies between frames with different Frame Classes. The Report and Page-Start and -End Frame Additional Attributes is shown in *Figure 3-10-6*. The Record frame Additional Attributes overlay is shown in *Figure 3-10-7*. The Inquiry Record Frames Additional Attributes overlay is shown in *Figure 3-10-8*.

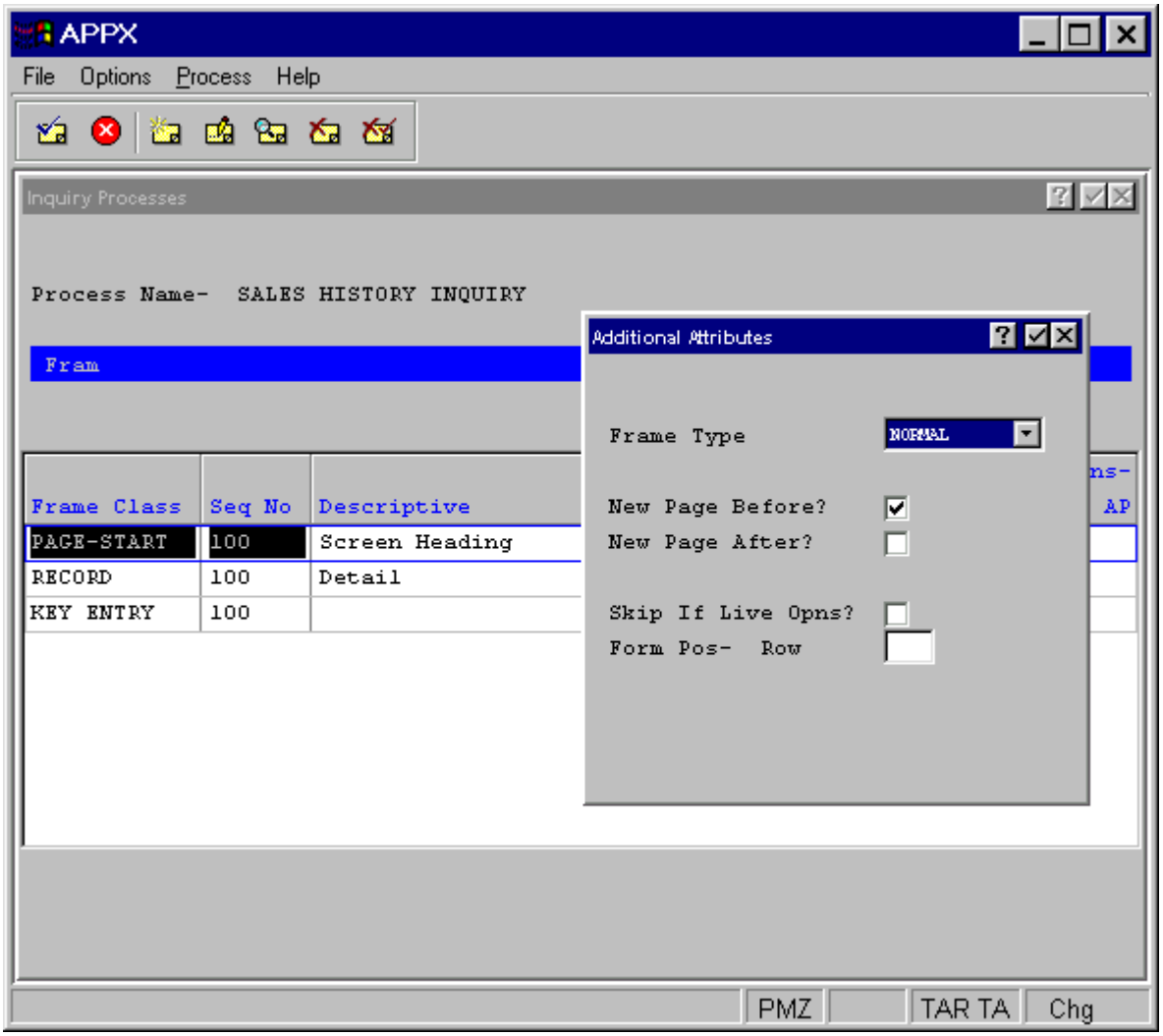


Figure 3-10-6. Inquiry Report and Page-Start and -End Frames Additional Attributes Overlay

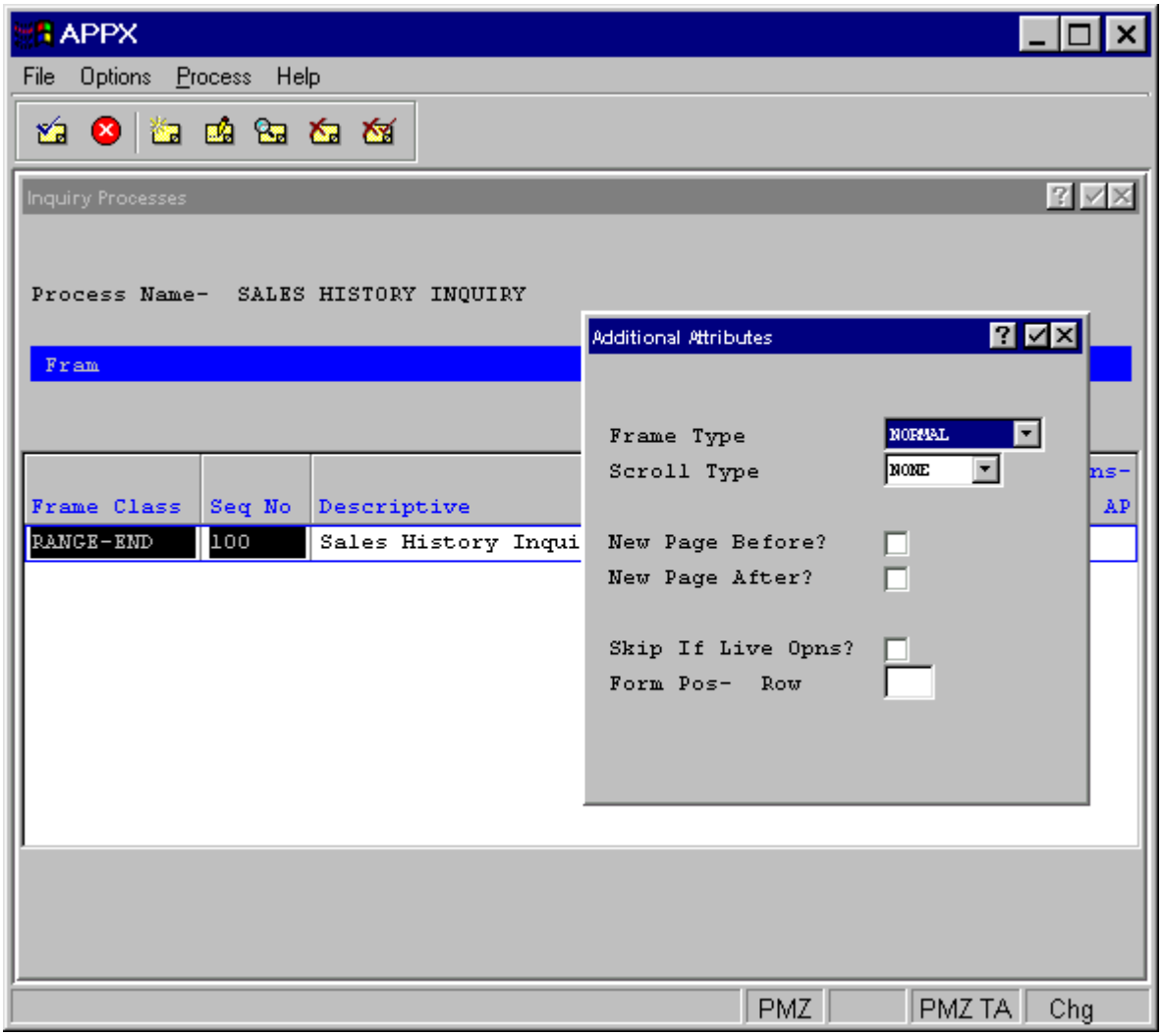


Figure 3-10-7. Inquiry Range-Start and -End Frames Additional Attributes Overlay

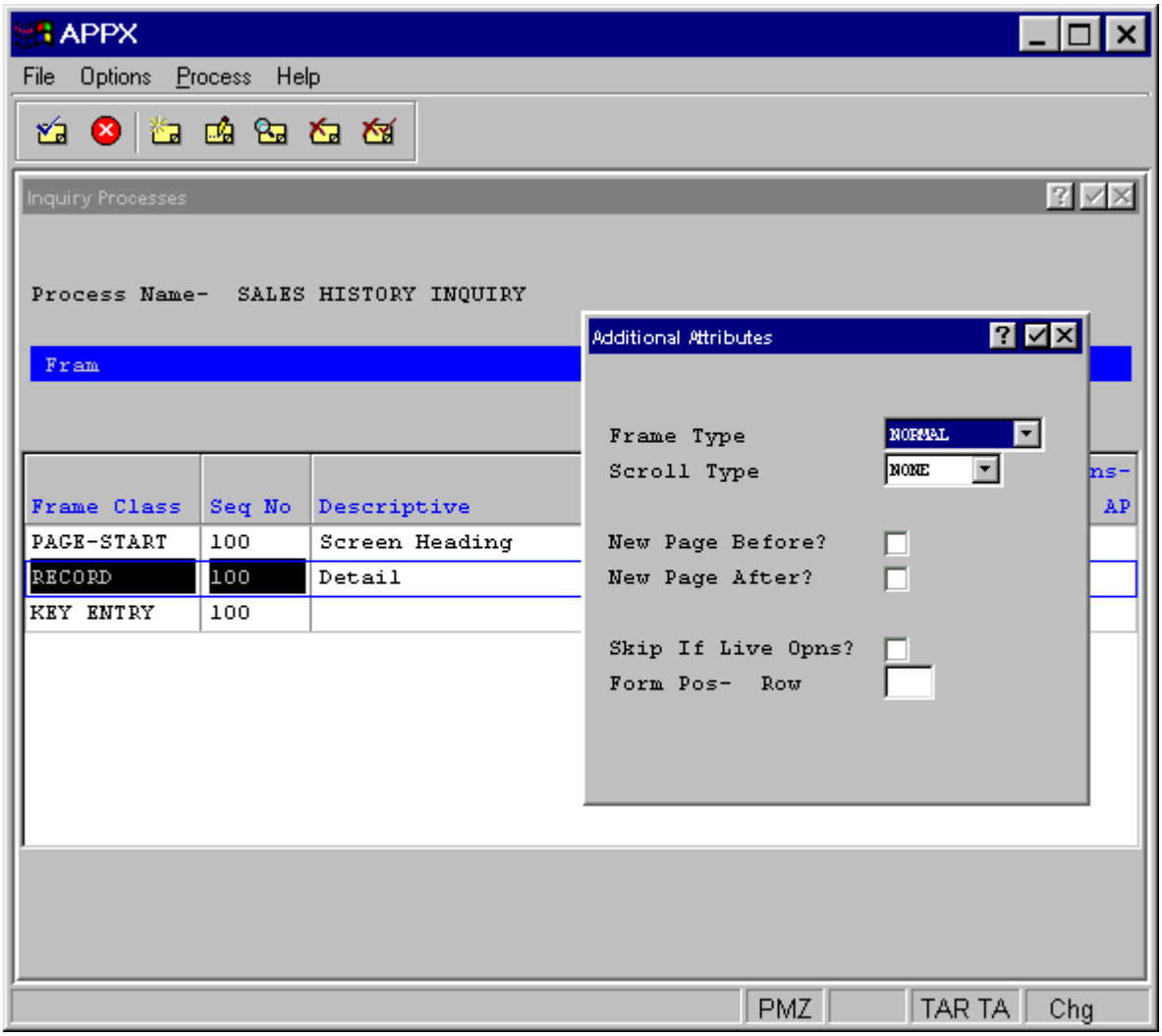


Figure 3-10-8. Inquiry Record Frame Additional Attributes Overlay

The following fields are available on the inquiry frame Additional Attributes overlay for all frame classes except Key Entry.

- ***Frame Type*** governs the execution of an image (or images) for the frame. The *Frame Type* options are:

NORMAL. Executes the first (lowest-numbered) alternate image unless another image was explicitly selected with statements and the predefined field ALTERNATE IMAGE NUMBER.

OPTIONAL. Executes the image that is identified by the *Normal Frame Type* above, with the following exception: If the value in the predefined field ALTERNATE IMAGE NUMBER is *0*, do not execute an image.

REPEATING. Executes the image repeatedly until the value in the ALTERNATE IMAGE NUMBER is set to *0* by statements.

The default is *NORMAL*.

New Page Before?

Determines whether or not to begin a new page before executing this frame. If the report is between pages before this frame executes, “yes” ☑ does not cause a blank page to print. The default is “no” ☐.

New Page After?

Determines whether or not to begin a new page subsequent to executing this frame. If the report is between pages after this frame executes, “yes” ☑ does not cause a blank page to print. The default is “no” ☐.

Skip If Live Opns?

Determines whether or not the frame is exclusive to the initial setup and recovery processing phases of operation. If “yes” ☑ , the frame does not execute during the live operations phase. If “no” ☐, the frame executes during all phases of operation. The default is “no” ☐.

Form Pos— Row

Designates the placement of the frame on the display. When implemented, the row is defined relative to the top of the display. This field may contain up to three numeric characters. See Frame/Image Positioning in *Chapter 3-2* for more information.

The Record and Range-Start and -End frames Additional Attributes overlay contains the following unique field:

Scroll Type

Indicates whether or not the images on the frame scroll and, if so, the type of scrolling to be performed. The *Scroll Type* options are:

- ***NONE.*** The image on the frame does not scroll.
- ***DOWN.*** APPX divides the frame into a grid of equal-sized regions, based upon a calculated composite image size. The first image is placed into the upper left region in the frame. Succeeding images are placed in the region below the previous image until the column is complete. As each column is completed, the next image is placed at the top of the column immediately to the right of the completed column.
- ***ACROSS.*** APPX divides the frame into a grid of equal-sized regions, based upon a calculated composite image size. The first image is placed into the upper left region in the frame. Succeeding images are placed in the region to the right of the previous image until the row is complete. As each row is completed, the next image is placed in the far left-hand position of the row immediately below the completed row.

The default is blank.

The inquiry *Key Entry* frame Additional Attributes overlay is shown in *Figure 3-10-9*:

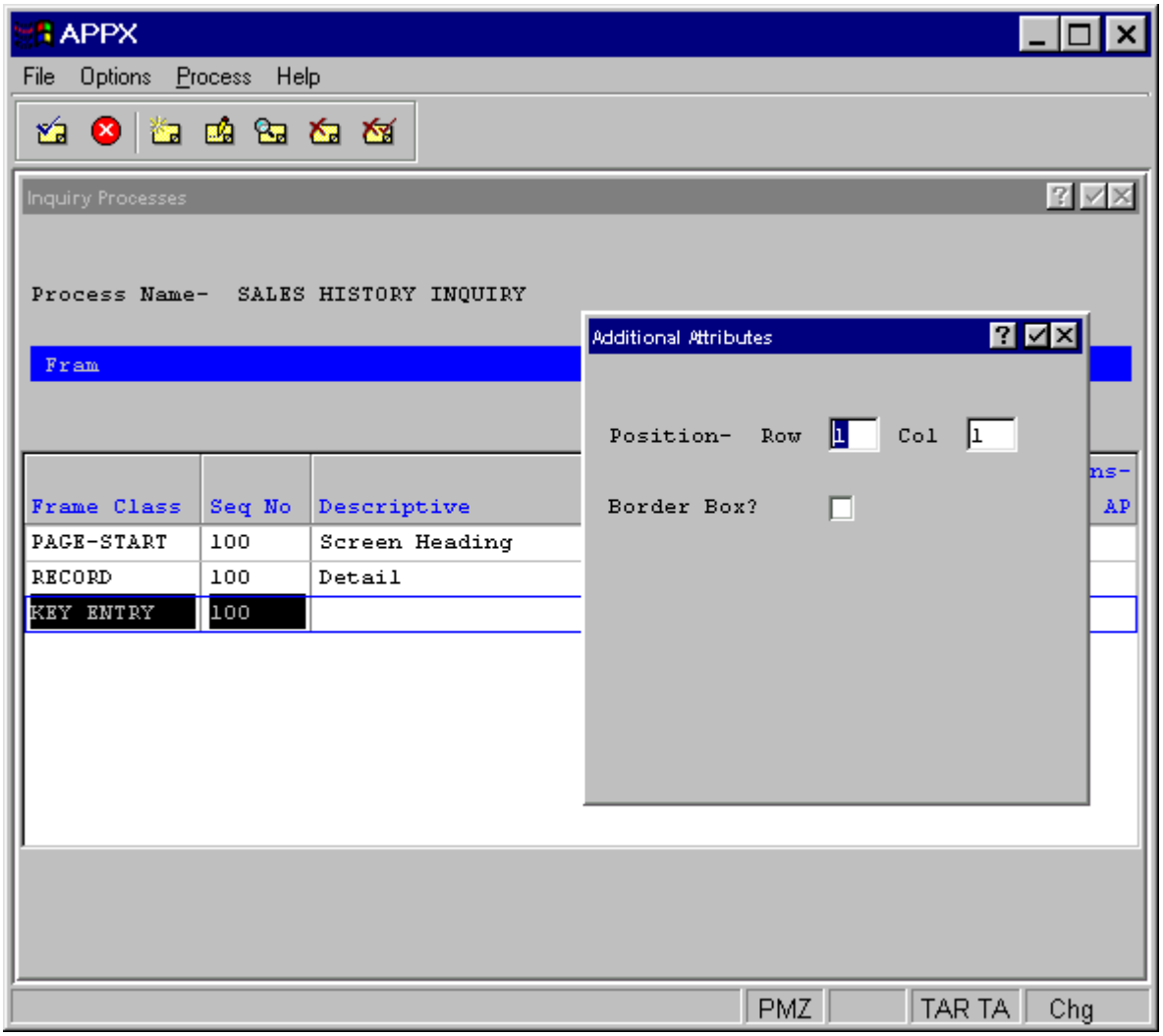


Figure 3-10-9. Inquiry Key Entry Frame Additional Attributes Overlay

The following fields are available on the inquiry *Key Entry* frame Additional Attributes overlay:

Position— Row/Col

Determines the placement of the frame on the display. Row entries designate positioning relative to the top of the display and column entries define the positioning relative to the far left-hand position on the display. The default is *1,1*.

Border Box?

Indicates whether or not APPX draws a box around this frame before it displays to a user. The border does not consume rows or columns within the space that is defined for the frame, but does consume space on the screen. The default is “no” ☐.

Inquiry Frame Automatic Processes Option

The AUTOMATIC PROCESSES option, available on the Inquiry Frame overlay for all frame-classes except Key Entry, accesses the Automatic Child Process overlay. This allows you to attach a specific process to the inquiry process currently selected so that it invokes automatically when the inquiry process is invoked. Refer to Chapter 3-13: Automatic and Optional Children for more information.

Inquiry Image Specifications

Inquiry image specifications define the visual and operational attributes of the image. The Images overlay automatically displays after you select ENTER from the inquiry Frames overlay. The Images overlay for *Report-Start* and *-End*, *Page-Start* and *-End*, *Range-Start* and *-End* and *Record*-type Frame Classes is shown in *Figure 3-10-10*, and the Images overlay for *Key Entry*-type frame classes is shown in *Figure 3-10-11*.

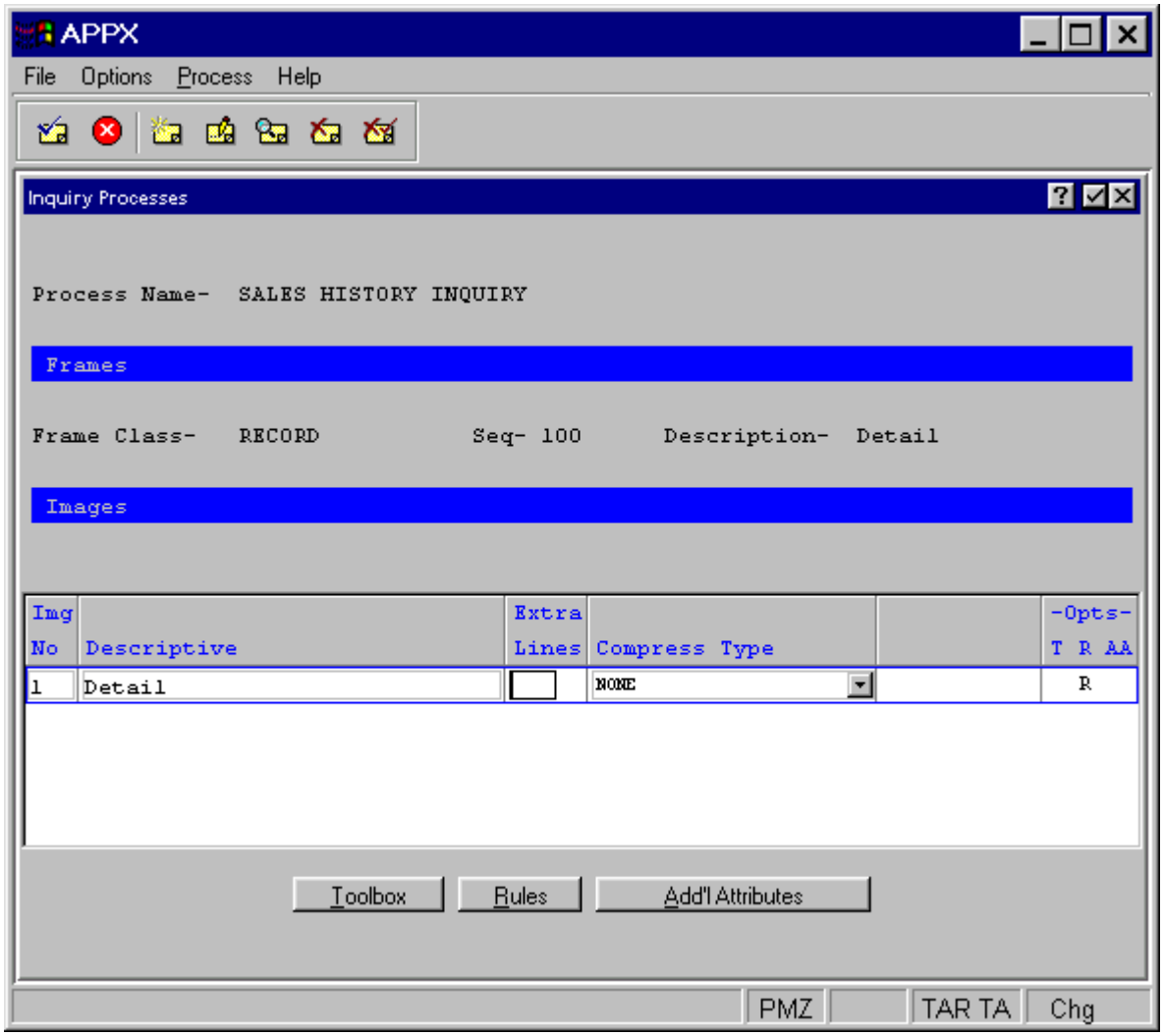


Figure 3-10-10. Inquiry Images Overlay for Report-Start and -End, Page-Start and -End, Range-Start and -End and Record type Frame Classes

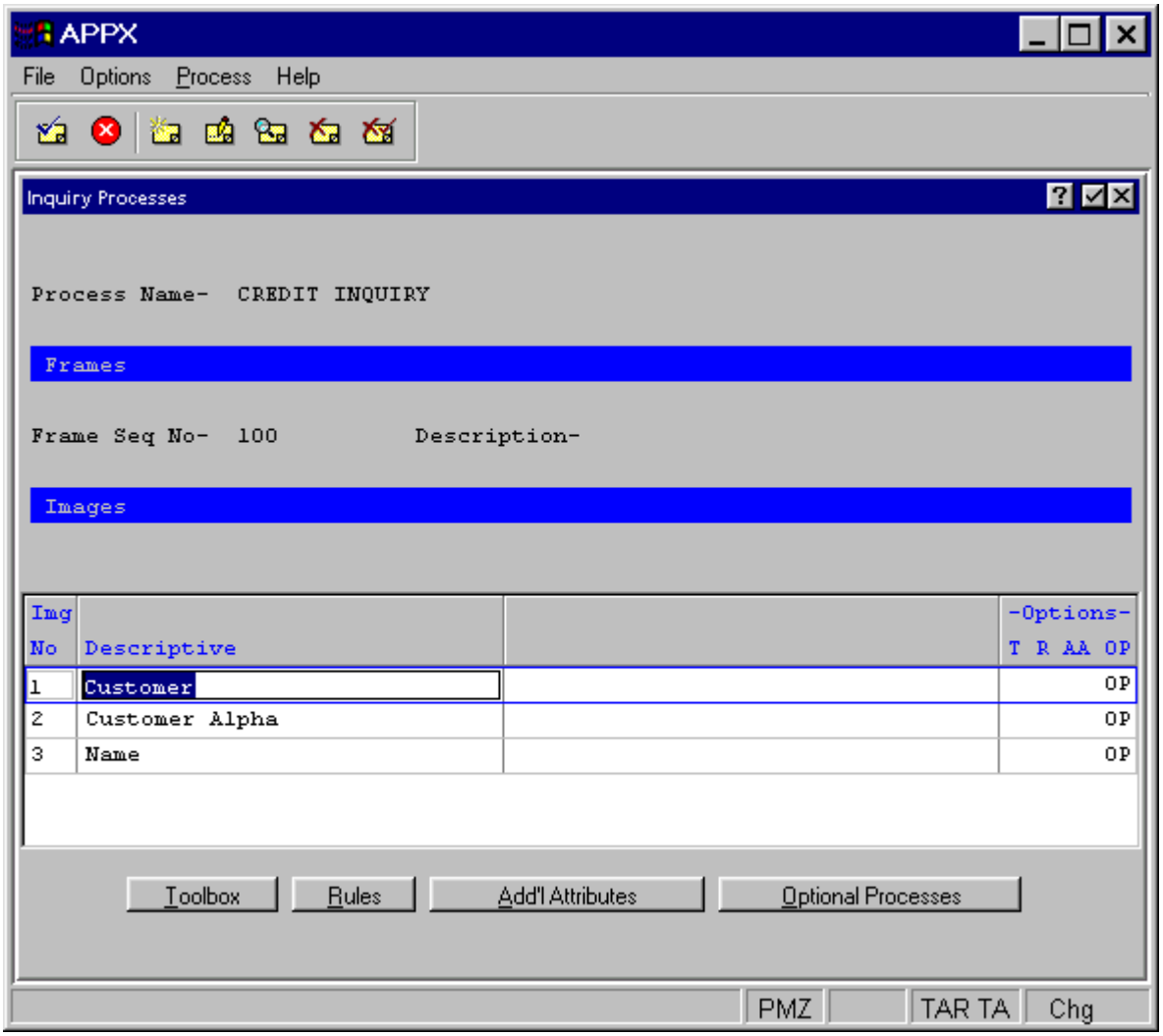


Figure 3-10-11. Inquiry Images Overlay for Key Entry type Frame Classes

Both inquiry Images overlays contain the following fields:

Img No

Displays a sequence number that is defaulted by APPX. The image number uniquely identifies each possible image for the frame and is referenced when determining which image, if any, is to display. The first image that is defined in add mode is assigned alternate number *1*, and additional frames are assigned a number equal to the highest existing image number, plus *1*. This default alternate number can be overridden, but must be unique within the frame. Refer to *Chapter 3-1, Images,* for a discussion of alternate images in APPX.

Descriptive

Identifies this alternate image. This field is used for documentation purposes only.

Opts

Are indicators that are present whenever documentation (T), rules (R), additional attributes (AA), or optional processes (OP--for key entry images only) have been defined for this process.

The inquiry Images overlays for *Report-Start* and *-End*, *Page-Start* and *-End*, *Range-Start* and *-End* and *Record* type Frame Classes also contain the following fields:

Extra Lines

Indicates a number of additional rows to add to the size of the image. This field may contain up to three numeric characters. For non-scrolling frames only, this entry defines the additional number of rows to reserve on the page when determining page breaks. For example, you can use this specification to ensure sufficient room to print at least one record frame beneath a subheading.

Compress Type

Indicates whether or not to suppress blank lines in an image and, if so, the type of lines to compress. The *Compress Type* options are:

- ***NONE.*** Do not compress.
- ***EMBEDDED BLANK LINES.*** Compress embedded blank lines. (Image still occupies the full number of rows.)
- ***TRAILING BLANK LINES.*** Compress trailing blank lines.
- ***EMBEDDED/TRAILING.*** Compress embedded and trailing blank lines.

The default is *NONE.*

The following options as described in the corresponding sections are available on the inquiry Image overlays:

- Toolbox
- Rules
- Additional Attributes
- Optional Processes

Inquiry Image Toolbox Option

The following options are available from the Inquiry Image Toolbox overlay:

Database Management

The DATABASE MANAGEMENT option allows you to add new files or maintain existing ones without leaving inquiry process design.

Process Management

The PROCESS MANAGEMENT option allows you to add other types of processes or maintain existing ones without leaving inquiry process design.

Design Transfer Option

The DESIGN TRANSFER option allows you to copy the attributes from a process that you have already designed and apply it to a new process.

Syntax Check

The SYNTAX CHECK option allows you to create the Executable Module for the current inquiry process without actually running it. If the Executable Module for the inquiry process is valid this option is ignored when invoked. Note that this option (`OPTION 97`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

View Audit Info

The VIEW AUDIT INFO option allows you to see who created the currently selected inquiry process and who changed the currently selected inquiry process most recently. It also gives the date of creation and the date it was last changed. Note that this option (`OPTION 98`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Invoke a Process

The INVOKE A PROCESS option is available to facilitate testing. It allows you to invoke any process; the currently selected inquiry process using the Test Database defined in system administration is the default. Note that this option (`OPTION 99`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Inquiry Image Rules Option

Selecting the RULES option accesses the inquiry image Event Points For Rules overlay which allows you to select from options that access the ILF Editor for APPX event points. If statements were defined in the event point, the text on the screen is highlighted. Refer to Chapter 3-16: Using Event Points for more information. The inquiry Event Points overlay contains the following options:

Start of Process

START OF PROCESS, which is common to all types of processes, accesses the ILF editor for the *Start of Process* event point. Upon completion, you are returned to the process Event Points overlay.

Post PCF Read

POST PCF READ accesses the ILF editor for the *Post PCF Read* event point. Upon completion, you are returned to the process Event Points overlay.

Select Image

SELECT IMAGE accesses the ILF editor for the *Select Image* event point. Upon completion, you are returned to the inquiry frame Event Points overlay.

Global EP's For Images

GLOBAL EP'S FOR IMAGES overlay allows you to access four additional event points. See *Page 164* for more information.

Pre-Display

PRE-DISPLAY accesses the ILF editor for the *Pre-Display* event point. Upon completion, you are returned to the inquiry image Event Points overlay.

Pre-Display/Verify

PRE-DISPLAY/VERIFY, available only for images with *key entry* type frame-class, accesses the ILF editor for the *Pre-Display/Verify* event point. Upon completion, you return to the inquiry frame Event Points overlay.

Option Intercept

OPTION INTERCEPT, available only for images with *key entry* type frame-class, accesses the ILF editor for the *Option Intercept* event point. Upon completion, you return to the inquiry frame Event Points overlay.

Verify

VERIFY, available only for images with *key entry* type frame-class, accesses the ILF editor for the *Verify* event point. Upon completion, you are returned to the inquiry frame Event Points overlay.

End of Process

END OF PROCESS accesses the ILF editor for the *End of Process* event point. Upon completion, you are returned to the process Event Points overlay.

PCF Level EPs

PCF LEVEL EPS provides access to the File Level Event Points for the PCF file. This overlay is shown in *Figure 3-10-3.* These Event Points are not directly associated with the process but with the PCF file. See the *Rules Option* section of *Chapter 2-4, Files and Fields* for more information on using PCF Level Event Points.

Refer to Chapter 3-16: Using Event Points for more information.

Inquiry Image Additional Attributes Option

The ADDITIONAL ATTRIBUTES option accesses the inquiry process image Additional Attributes overlay. This overlay may vary depending on the type of frame to which the image belongs. The Inquiry Images Additional Attributes Overlay for *Report-Start* and *-End*, *Page-Start* and *-End*, *Range-Start* and *-End* and *Record*-type Frame Classes is shown in *Figure 3-10-12*, and the Inquiry Images Additional Attributes Overlay for *Key Entry*-type Frame Classes is shown in *Figure 3-10-13*.

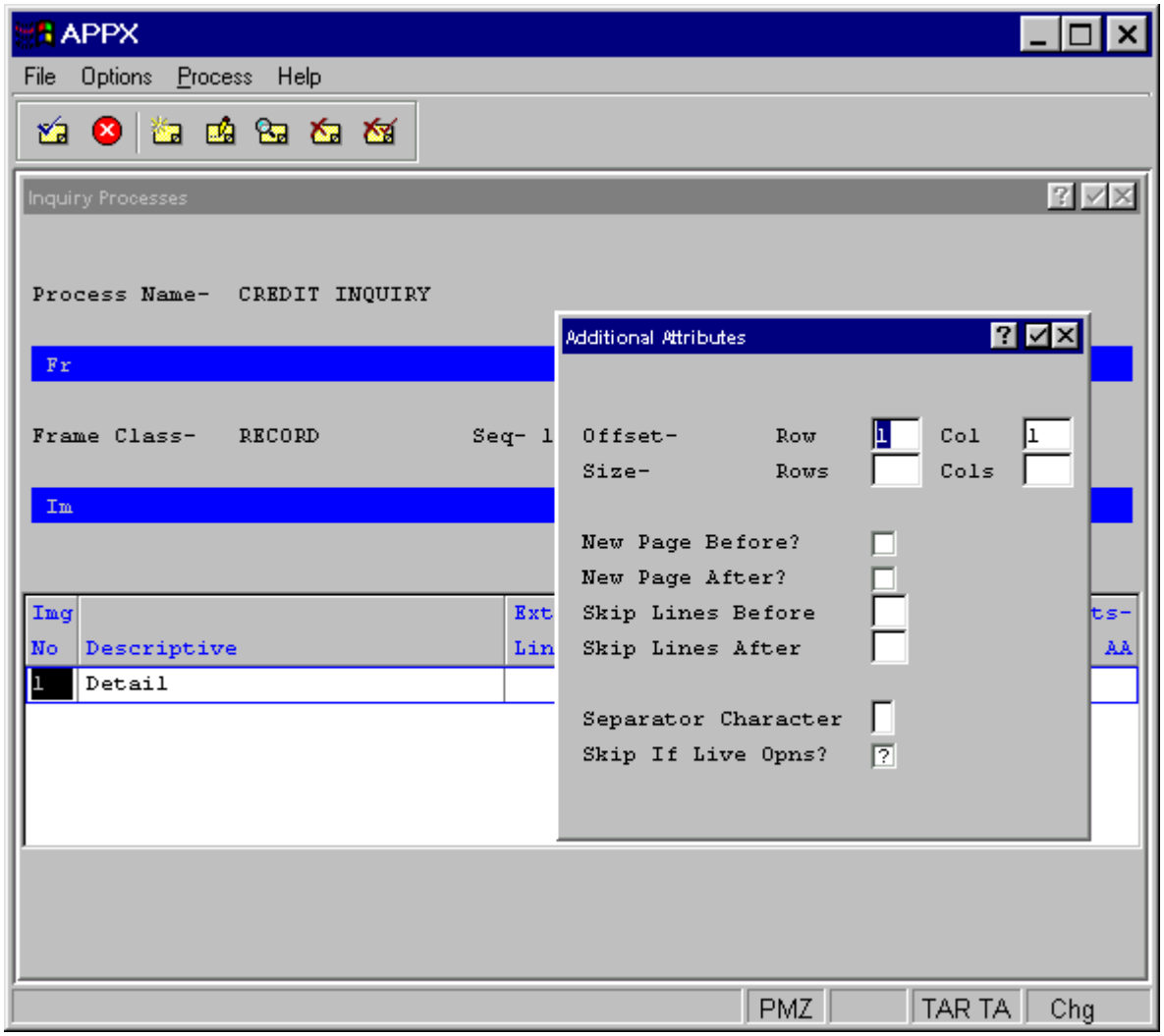


Figure 3-10-12. Inquiry Images Additional Attributes Overlay for Report-Start and -End, Page-Start and -End, Range-Start and -End and Record type Frame Classes

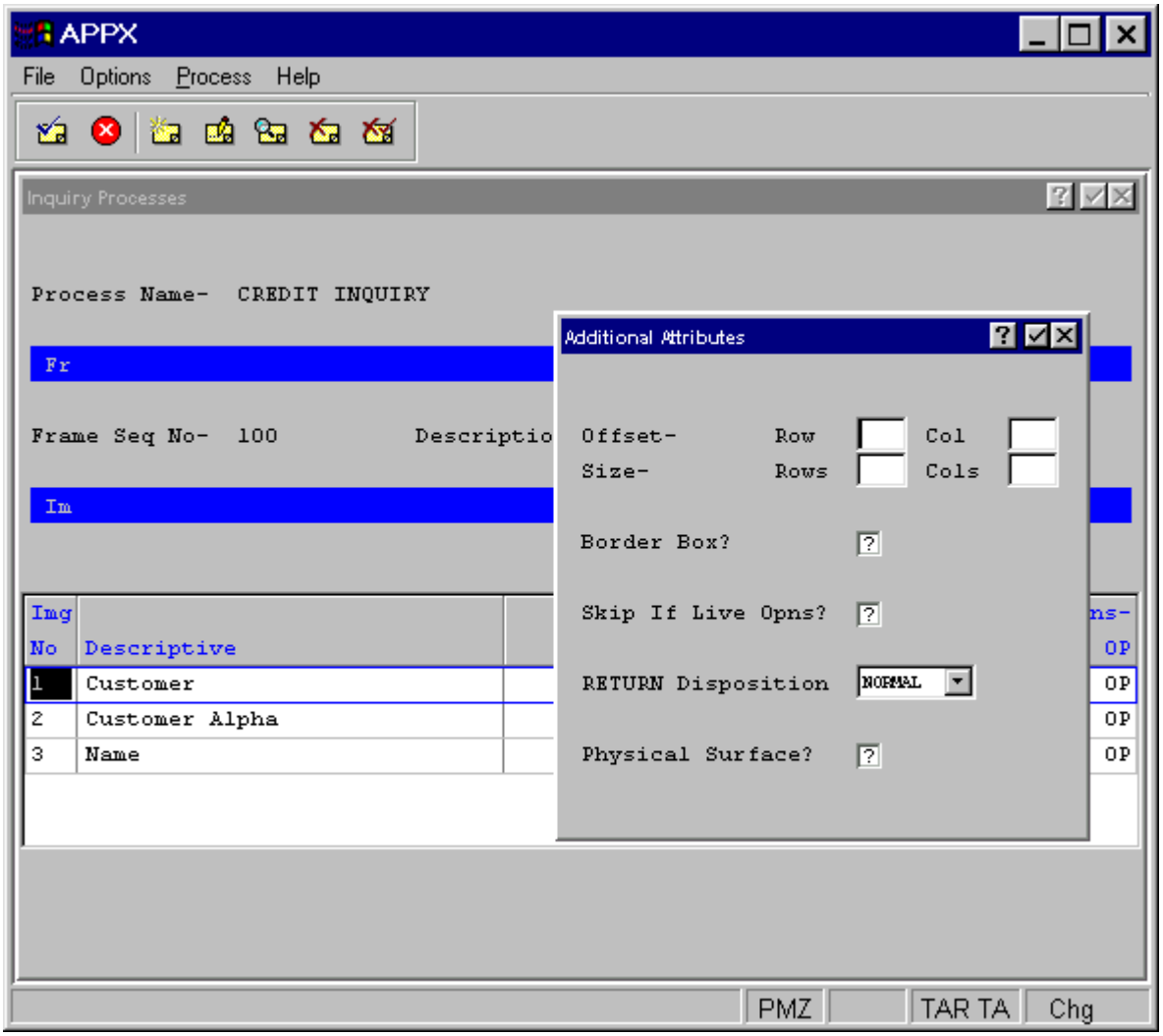


Figure 3-10-13. Inquiry Key Entry Image Additional Attributes Overlay

The following fields are available on Inquiry Images Additional Attributes Overlay for *Report-Start* and *-End*, *Page-Start* and *-End*, *Range-Start* and *-End* and *Record*-type Frame Classes:

Offset— Row/Col

Determines the placement of the rows and columns of this image, relative to the frame. Row entries designate positioning relative to the top of the frame. Column entries designate positioning relative to the left of the frame. If blank, the APPX defaults to same row/column position as the frame. The default is blank. See Frame/Image Positioning in *Chapter 3-2* for more information.

Size— Rows/Cols

Designates the number of rows and columns that comprise the displayed image. The image cannot be larger than the frame it belongs to. If blank, the size of the image defaults to the size of the frame. The default is blank.

New Page Before?

Determines whether or not to begin a new page before executing this image in a nonscrolling frame. If the report is between pages before this image executes, “yes” ☑ does not cause a blank page to print. The default is “no” ☐.

New Page After?

Determines whether or not to begin a new page subsequent to executing this image in a nonscrolling frame. If the report is between pages after this image executes, “yes” ☑ does not cause a blank page to print. The default is “no” ☐.

Skip Lines Before

Indicates the number of blank lines to be printed before executing this image. If Print Beside is in effect, APPX only skips lines for the columns that are occupied by the image. The default is *0*.

Skip Lines After

Indicates the number of blank lines to print subsequent to executing this image. If Print Beside is in effect, APPX only skips lines for the columns that are occupied by the image. The default is *0*.

Separator Character

Determines the character that prints continuously across the entire image to separate it from the rest of the report. If blank, APPX does not print a heading separator. The default is blank.

Skip If Live Operations?

Determines whether or not the image is exclusive to the initial setup and recovery processing phases of operation. If “yes” ☑, the image is not executed during the live operations phase. The default is “unknown" ☒.

The following fields are available on the inquiry *Key Entry* image Additional Attributes overlay.

Offset— Row/Col

Determines the placement of the rows and columns of this image, relative to the frame. Row entries designate positioning, relative to the top of the frame. Column entries designate positioning relative to the left of the frame. If blank, the APPX defaults to same row/column position as the frame. The default is blank. See Frame/Image Positioning in *Chapter 3-2* for more information.

Size— Rows/Cols

Designates the number of rows and columns that comprise the displayed image. The image cannot be larger than the frame it belongs to. If blank, the size of the image defaults to the size of the frame. The default is blank.

Border Box?

Indicates whether or not APPX draws a box around this image before it displays to a user. The border does not consume rows or columns within the space that is defined for the image, but does consume space on the screen. If the box specification is combined with an overline, an underline, or both, APPX replaces the top and/or bottom of the box with the overline and/or underline. This minimizes the space that is consumed by the border. The physical space on the screen that is consumed by the box is system-specific.

Skip If Live Operations?

Determines whether or not the image is exclusive to the initial setup and recovery processing phases of operation. If "yes" ☑, the image is not executed during the live operations phase.

Return Disposition

Option designates the effect on the image displayed for this frame of a user selecting ENTER . There are three ***RETURN** Disposition* options.

- ***HOLD.*** Causes APPX to automatically control the disposition of the frame. The image for the frame displays until the process cycle completes (until all frames execute for one PCF record), at which time the image is removed.
- ***CLEAR.*** Causes the contents of the frame to clear, but the image for the frame still exists and still obscures the contents of any prior images that it overlays on the screen.
- ***REMOVE.*** Causes the frame itself to be removed, and will no longer obscure the contents of any prior frames that it overlays on the screen.

The default is *HOLD.*

Physical Surface?

Controls whether or not this image is "attached" to the physical surface. Click the checkbox to toggle between values of "yes" ☑, "no" ☐, or "unknown" ?. "Yes" indicates that the image will always display in front of images that are not attached to the physical surface and that panning options do not apply. The image will always be "seen" by the user even if the logical surface is panned far to the right or other images not attached to the physical surface are executed.

Inquiry Image Optional Processes Option

The OPTIONAL PROCESSES option, which is available from the Inquiry Images Additional Attributes Overlay for only *Key Entry*-type Frame Classes, accesses the inquiry image Optional Child Processes specification screens that define attributes to govern the discretionary invocation of another APPX process from an image in an inquiry process. These specifications include the name of the child

process, the invocation type, and whether or not various activities at the child level affect the parent process. Upon completion, you are returned to the inquiry Images overlay. See Chapter 3-13: Automatic and Optional Children for detailed information about defining optional child processes.

CHAPTER 3-11

Status Processes

Introduction to Status Processes

Status processes are used to provide various error reporting options.

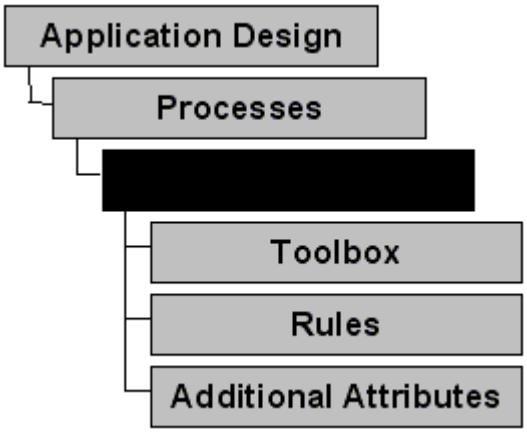


Figure 3-11-1. Status Process Definition

Status Process Specifications

Status process specifications establish the general attributes of a status process. The Status Processes screen is shown in *Figure 3-11-2.*

Status Processes screen

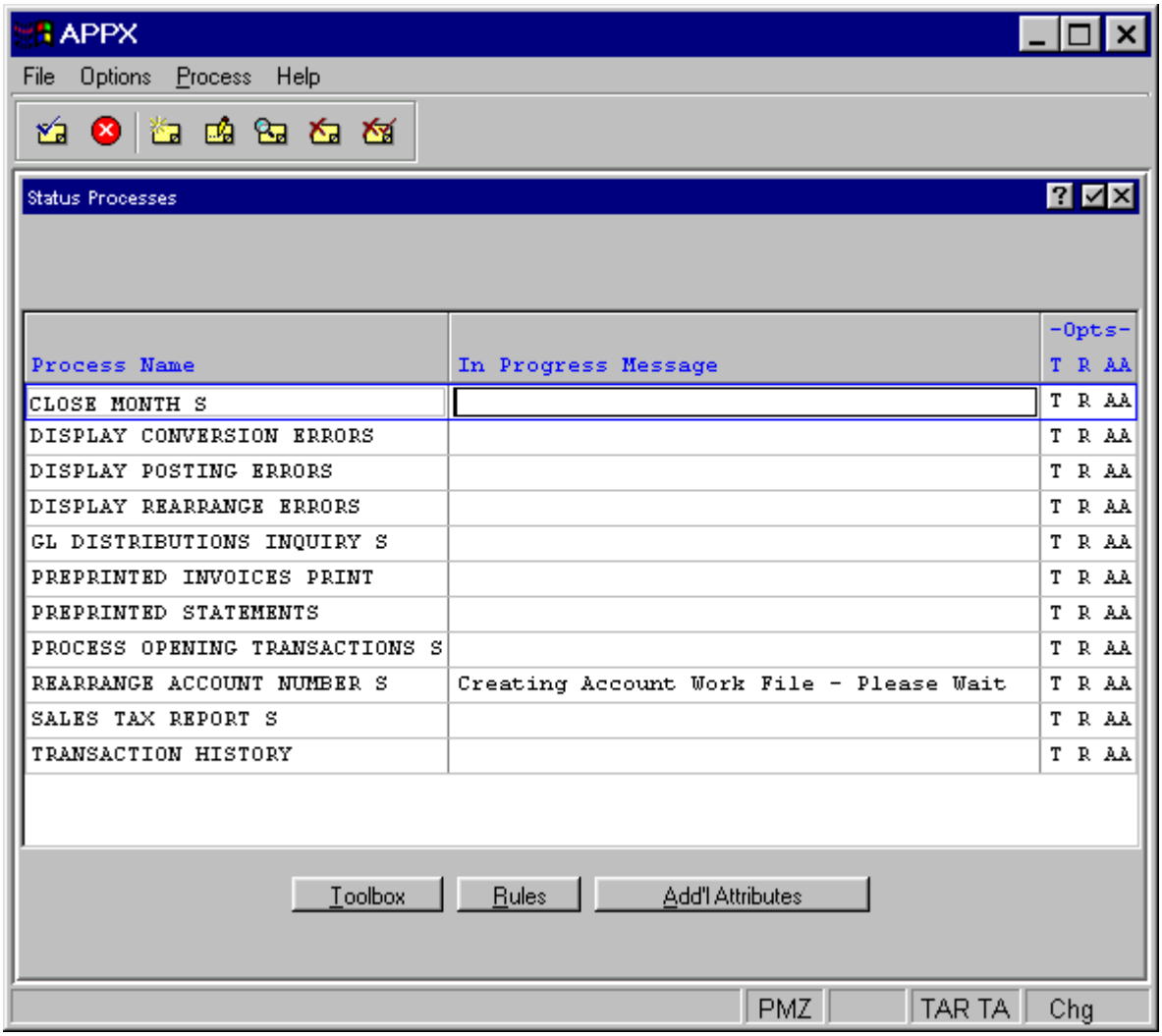


Figure 3-11-2

The Status Processes screen, shown in *Figure 3-11-2,* contains the following fields:

Process Name

Identifies a specific process that must be unique within type and application.

In Progress Message

Is the message that is displayed to the user while this process is running.

Opts

Are indicators that are present whenever documentation (T), rules (R), or additional attributes (AA) have been defined for this process.

The Status Processes screen contains the following options as described in the corresponding sections:

- Toolbox

- Rules
- Additional Attributes

Status Process Toolbox Option

The toolbox overlay appears on the screen when you invoke the TOOLBOX option from any process screen, including an status process screen. The overlay contains a number of features such as Documentation Management and Security Management that will assist you with designing and invoking an status process. The options listed in the Toolbox overlay are described briefly below. See *Chapter 3-3, Process Definition* for a more detailed explanation of the options listed on the Toolbox overlay.

Documentation

When you select the DOCUMENTATION option from the Toolbox overlay, you will first see the Documentation overlay which allows you to select between user or technical documentation. Once you have entered your documentation, you are returned to the Status Processes screen.

Security

The SECURITY option allows you to identify the APPX security code to which a user must have rights in order to run the status process. The security code must be four numeric characters. If blank, there are no restrictions.

Database Management

The DATABASE MANAGEMENT option allows you to add new files or maintain existing ones without leaving status process design.

Process Management

The PROCESS MANAGEMENT option allows you to add other types of processes or maintain existing ones without leaving status process design.

Design Transfer

The DESIGN TRANSFER option allows you to copy the attributes from a process that you have already designed and apply it to a new process.

Cross-Reference

The CROSS REFERENCE option enables you to determine the location and frequency of a selected domain, file, field, application ID, process or ILF statement within a specific database.

Reference Browser Option

The REFERENCE BROWSER option runs a modified version of "Cross Reference – All" and produces a list of design components utilized by the currently selected status process. The results can be displayed interactively where links allow you to see the context of the references, edit the referenced component and drill down to the child process. This option can be used to identify, and optionally edit, the file level event points utilized by this status process.

Syntax Check

The SYNTAX CHECK option allows you to create the Executable Module for the current status process without actually running it. If the Executable Module for the status process is valid this option is ignored when invoked. Note that this option (`OPTION 97`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

View Audit Info

The VIEW AUDIT INFO option allows you to see who created the currently selected status process and who changed the currently selected status process most recently. It also gives the date of creation and the date it was last changed. Note that this option (`OPTION 98`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Invoke a Process

The INVOKE A PROCESS option is available to facilitate testing. It allows you to invoke any process; the currently selected status process using the Test Database defined in system administration is the default. Note that this option (`OPTION 99`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Status Process Rules Option

Selecting the RULES option accesses the status process Event Points For Rules overlay, shown in *Figure 3-11-3*, which allows you to enter the ILF Editor and finish the status process. The status Event Points overlay contains the following option:

Start of Process

START OF PROCESS, which is common to all types of processes, accesses the ILF editor for the Start of Process event point. Upon completion, you are returned to the process Event Points overlay.

Refer to Chapter 3-16: Using Event Points for more information.

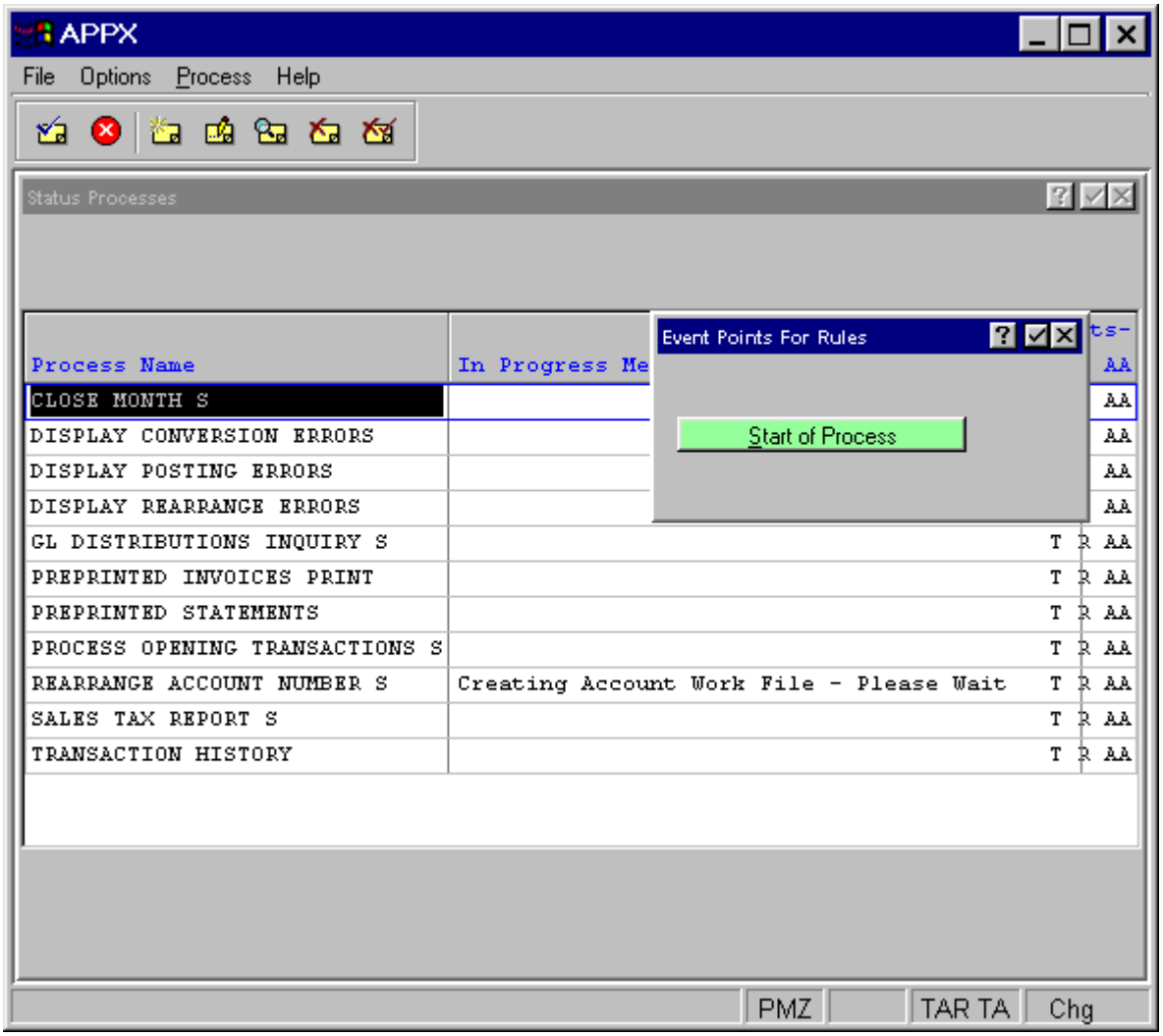


Figure 3-11-3. Event Points For Rules Overlay

Status Process Additional Attributes Option

The ADDITIONAL ATTRIBUTES overlay for Status processes is shown in *Figure 3-11-4*:

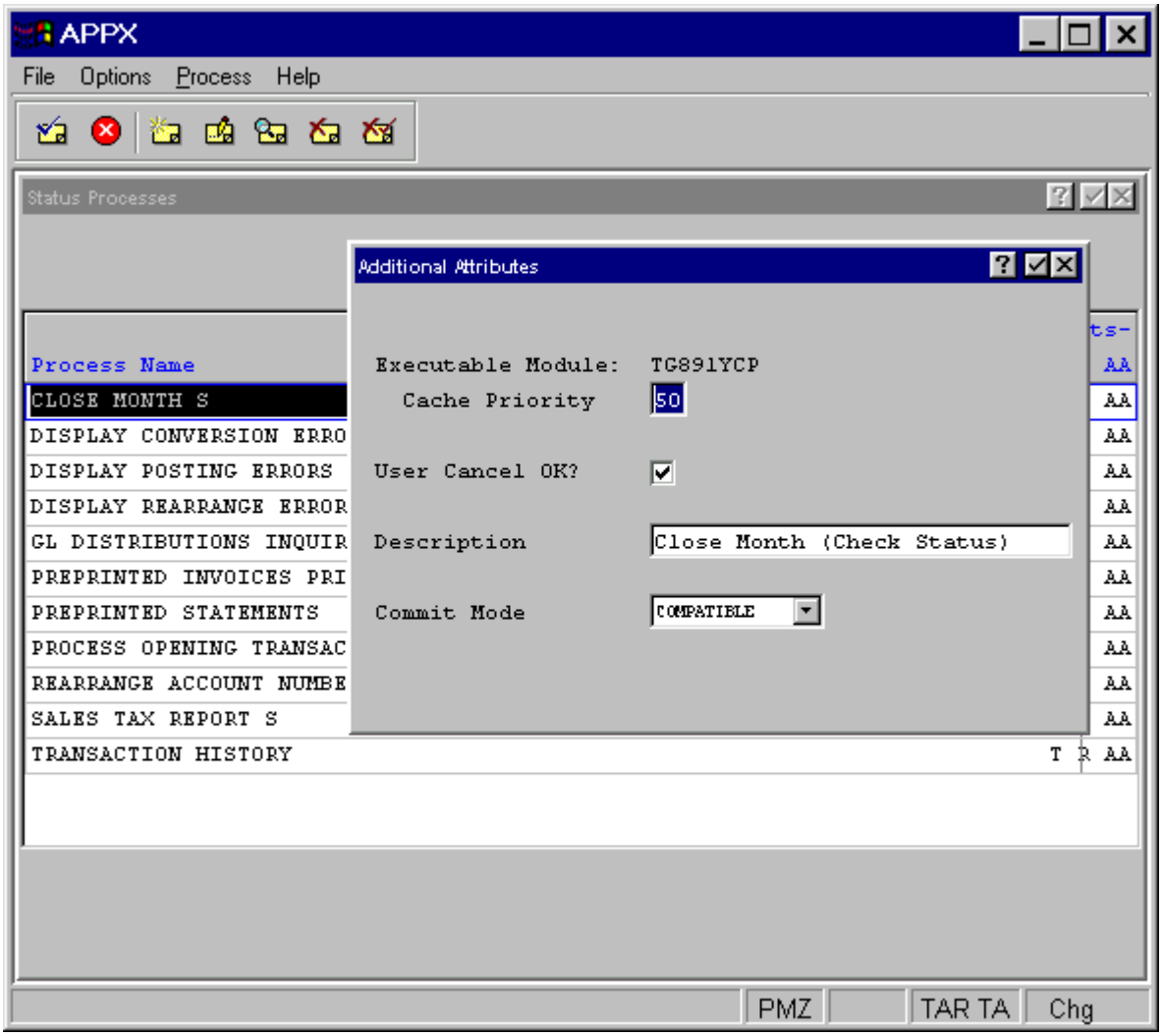


Figure 3-11-4. Status Process Additional Attributes Overlay

The ADDITIONAL ATTRIBUTES overlay for Status processes contains the following fields:

Executable Module

Displays the name of the executable module (EM) that is automatically generated by APPX. APPX derives the file name from the date and time the currently selected status process was added. The name consists of a unique string of eight alphanumeric characters and cannot be changed. If APPX finds it necessary to recreate the EM, the new EM retains the same file name.

Cache Priority

Establishes the likelihood that the process will remain in memory after execution, relative to other processes that the user has also completed executing. A process with a cache priority of *99* is most likely to remain in memory. A process with a cache priority of *1* is least likely to remain in memory. A cache priority of *0* indicates that the process is never cached. The default is *50*.

User Cancel OK?

Determines whether or not a user can select the CANCEL option to stop the execution of this process. The default is "yes" . You can conditionally control the user's ability to select the CANCEL option with statements and the predefined field CANCEL OK. This field is common to the Additional Attributes overlay for all types of processes.

Description

Is incorporated in the message that displays on the screen when the status process is compiling. It is used to communicate with users since it provides a more descriptive and easily understood alternative to *Process Name*, which is the default if this field is left blank. The Additional Attributes overlay for all types of processes contain this field.

Commit Mode

Controls the commit mode for a process and can be selected from a drop-down list of available options. *Commit Mode* determines those points in time when APPX will commit changes made to the tables stored within an RDBMS. It is only applicable in conjunction with an RDBMS; it has no effect with APPXIO. Also, not all File Managers support this feature. Although initially set here for the process, it can be added or overridden by use of predefined field COMMIT MODE. The following options are available:

COMPATIBLE. Changes are committed as soon as all locks are freed. This provides maximum compatibility between APPXIO files and RDBMS tables.

AUTOCOMMIT*. APPX will commit each modification as it is made.
***This value will be implemented in a future release.**

PCF CYCLE. APPX will commit modifications as each PCF record is processed.

END. APPX will only commit modifications at the end of the process family.

MANUAL. APPX will not automatically commit changes. You must use the COMMIT statement to do so.

CHAPTER 3-12

Subroutine Processes

Introduction to Subroutine Processes

Subroutine processes are reusable procedural programs that can be executed from other APPX processes. A program that converts one unit of measure to another, for example, can be created as a subroutine.

A subroutine process does not have frames, images, or automatic or optional children. After completing the specifications on the Processes screen to identify the subroutine process, you are automatically taken to the statement editor to define or modify ILF statements for the *Subroutine* event point.

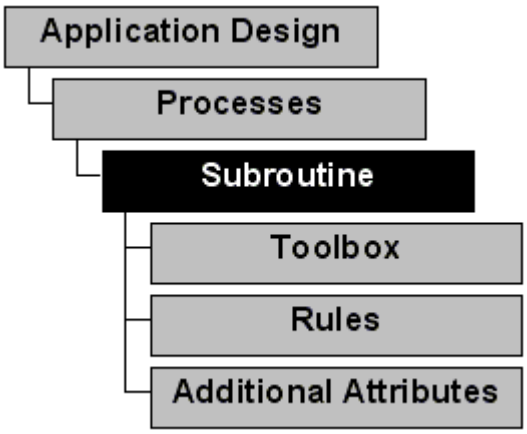


Figure 3-12-1. Subroutine Process Definition

Subroutine Process Specifications

The Subroutine Processes screen accesses the APPX statement editor for the *Subroutine* event point. This screen automatically displays after you select the Subroutine option from the Processes overlay. The Subroutine Processes screen is shown in *Figure 3-12-2.*

Subroutine Process screen

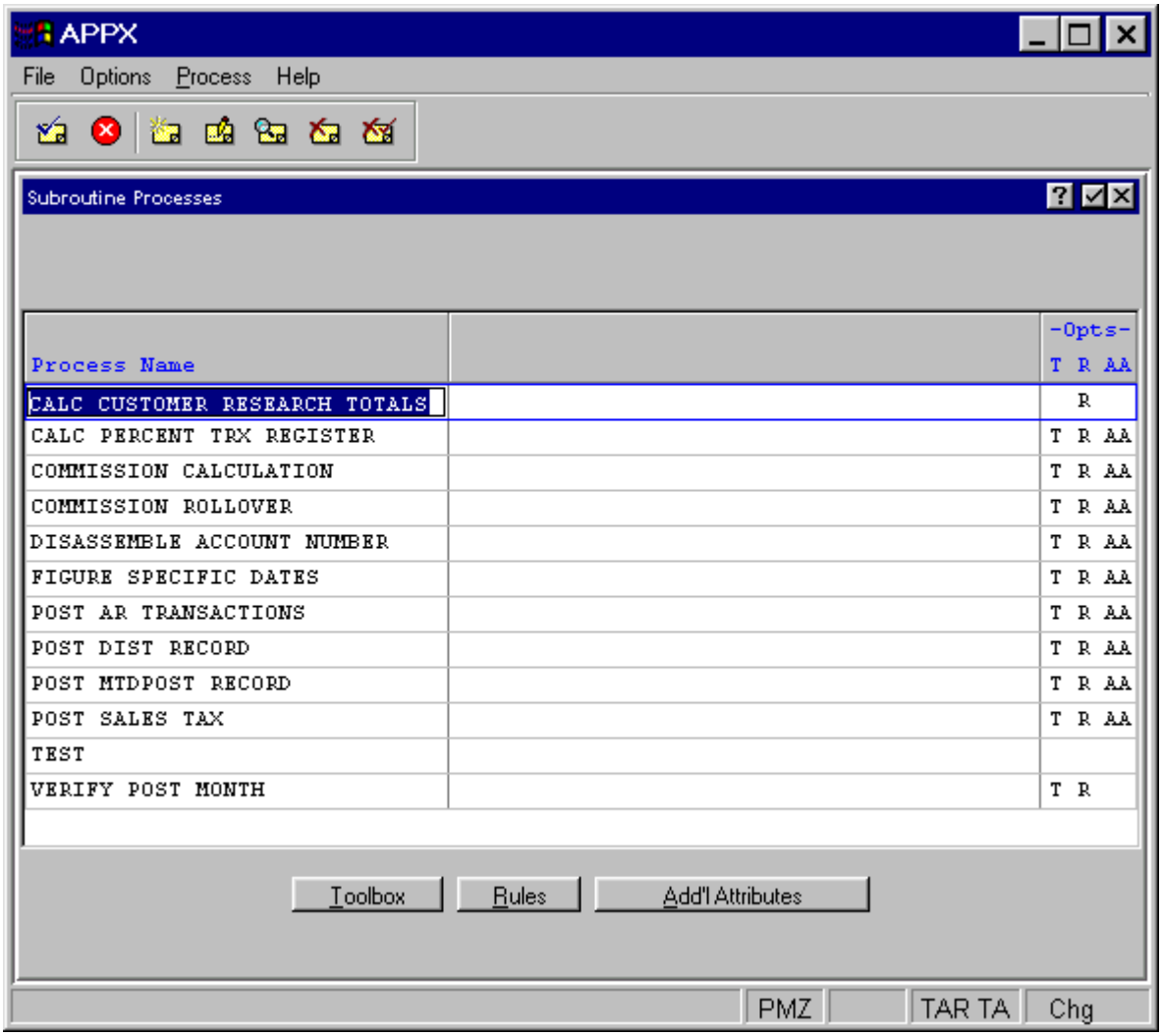


Figure 3-12-2

The Subroutine Processes screen contains the following field.

Process Name

Identifies a specific process that is unique within type and application.

Opts

Are indicators that are present whenever documentation (T), rules (R), or additional attributes (AA) have been defined for this process.

The Subroutine Processes screen contains the following options as described in the corresponding sections:

- Toolbox
- Rules

• Additional Attributes

Subroutine Process Toolbox Option

The toolbox overlay appears on the screen when you invoke the TOOLBOX option from the subroutine process screen. The overlay contains a number of features such as Documentation Management and Security Management that will assist you with designing and invoking an subroutine process. The options listed in the Toolbox overlay are described briefly below. See *Chapter 3-3, Process Definition* for a more detailed explanation of the options listed on the Toolbox overlay.

Documentation Option

When you select the DOCUMENTATION option from the Toolbox overlay, you will first see the Documentation overlay which allows you to select between user or technical documentation. Once you have entered your documentation, you are returned to the Subroutine Processes screen.

Database Management Option

The DATABASE MANAGEMENT option allows you to add new files or maintain existing ones without leaving subroutine process design.

Process Management Option

The PROCESS MANAGEMENT option allows you to add other types of processes or maintain existing ones without leaving subroutine process design.

Design Transfer Option

The DESIGN TRANSFER option allows you to copy the attributes from a process that you have already designed and apply it to a new process.

Cross Reference Option

The CROSS REFERENCE option enables you to determine the location and frequency of a selected domain, file, field, application ID, process or ILF statement within a specific database.

Reference Browser Option

The REFERENCE BROWSER option runs a modified version of “Cross Reference – All” and produces a list of design components utilized by the currently selected subroutine process. The results can be displayed interactively where links allow you to see the context of the references, edit the referenced component and drill down to the child process. This option can be used to identify, and optionally edit, the file level event points utilized by this subroutine process.

Syntax Check Option

This feature allows you to create the Executable Module for the current subroutine process without actually running it. If the Executable Module for the subroutine process is valid this option is ignored when invoked. Note that this option (`OPTION 97`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

View Audit Info Option

This feature allows you to see who created the currently selected subroutine process and who changed it most recently. It also gives the date of creation and the date it was last changed. Note that this option (`OPTION 98`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Invoke a Process Options

This feature is available to facilitate testing. It allows you to invoke any process; the currently selected subroutine process using the Test Database defined in system administration is the default. Note that this option (`OPTION 99`), like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Subroutine Process Rules Option

Selecting the RULES option accesses the subroutine process Event Points For Rules overlay, shown in *Figure 3-12-3,* which allows you to enter the ILF Editor and finish the subroutine

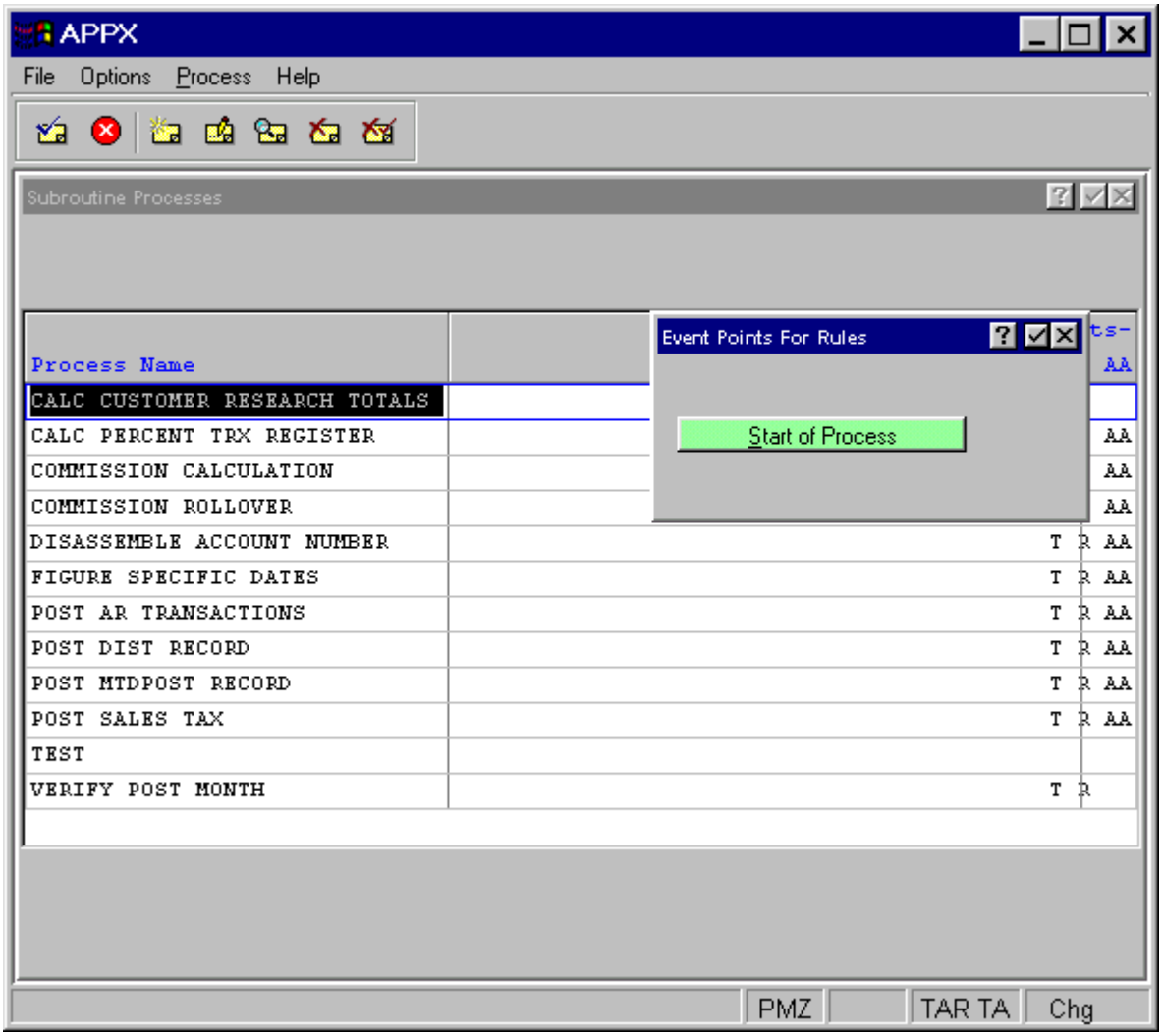


Figure 3-12-3. Event Points For Rules Overlay For Subroutine Processes

The subroutine Event Points overlay contains the following option:

Start of Process

START OF PROCESS, which is common to all types of processes, accesses the ILF editor for the *Start of Process* event point. Upon completion, you are returned to the process Event Points overlay.

Refer to Chapter 3-16: Using Event Points for more information.

Subroutine Process Additional Attributes Option

The ADDITIONAL ATTRIBUTES option accesses the Subroutine Process Additional Attributes overlay from which you can alter the standard EM attributes. The Additional Attributes overlay for Subroutine processes is shown in *Figure 3-12-4*:

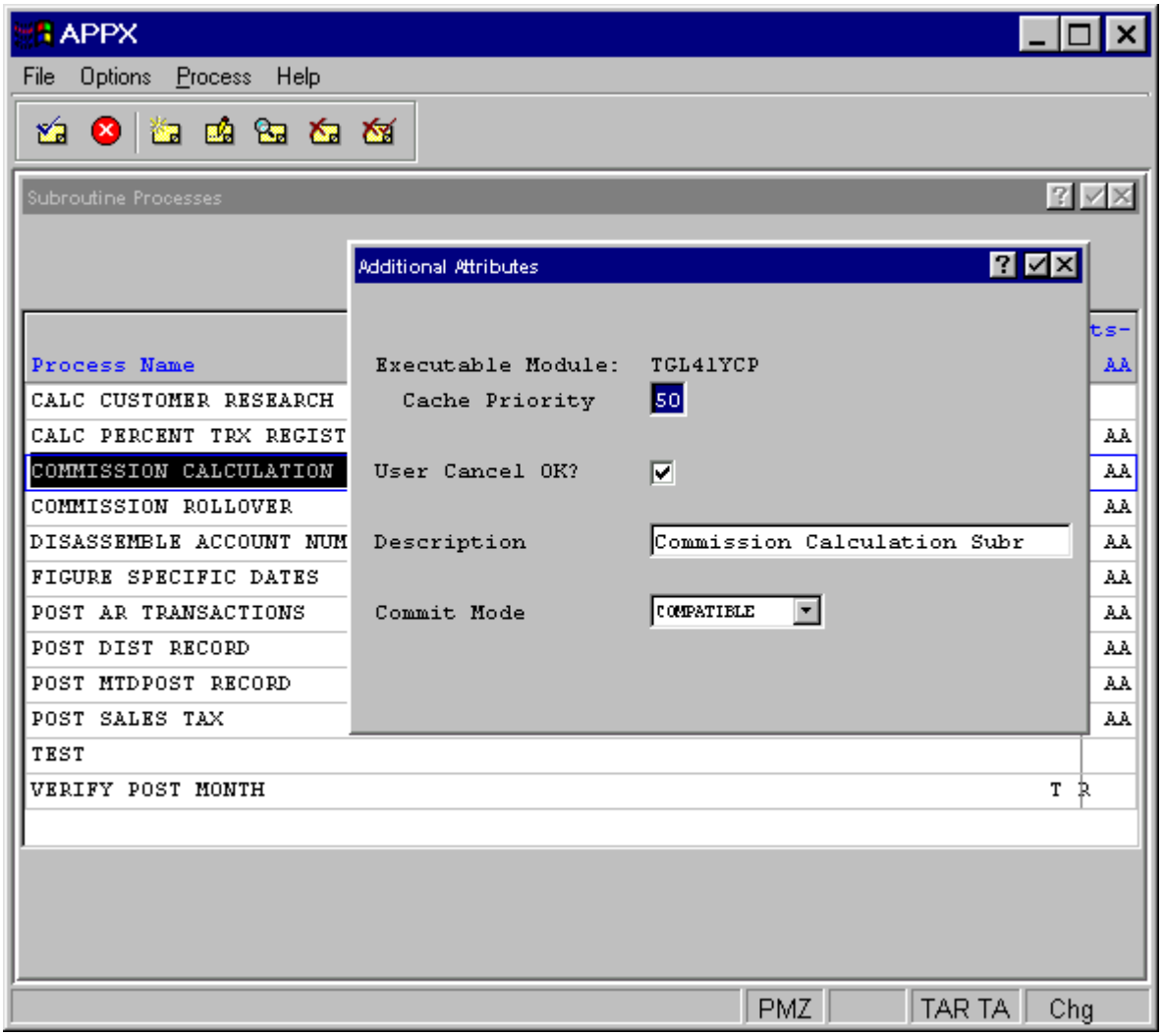


Figure 3-12-4. Subroutine Process Additional Attributes Overlay

The Additional Attributes overlay for Subroutine processes contains the following fields:

Executable Module

Displays the name of the executable module (EM) that is automatically generated by APPX. APPX derives the file name from the date and time the currently selected subroutine process was added. The name consists of a unique string of eight alphanumeric characters and cannot be changed. If APPX finds it necessary to recreate the EM, the new EM retains the same file name. This field is common to the Additional Attributes overlay for all types of processes.

Cache Priority

Establishes the likelihood that the process will remain in memory after execution, relative to other processes that the user has also completed executing. A process with a cache priority of *99* is most likely to remain in memory. A process with a cache priority of *1* is least likely to remain in memory. A cache priority of *0* indicates that the process is never cached. The default is *50*. This field is common to the Additional Attributes overlay for all types of processes.

User Cancel OK?

Contains either “yes” ☑ or “no” ☐ and determines whether or not a user can select the CANCEL option key to stop the execution of this process. The default is “yes”. You can conditionally control the user’s ability to select the CANCEL option with statements and the predefined field CANCEL OK. This field is common to the Additional Attributes overlay for all types of processes.

Description

Is incorporated in the message that displays on the screen when the subroutine process is compiling. It is used to communicate with users since it provides a more descriptive and easily understood alternative to *Process Name*, which is the default if this field is left blank.

Commit Mode

Controls the commit mode for a process and can be selected from a drop-down list of available options. *Commit Mode* determines those points in time when APPX will commit changes made to the tables stored within an RDBMS. It is only applicable in conjunction with an RDBMS; it has no effect with APPXIO. Also, not all File Managers support this feature. Although initially set here for the process, it can be added or overridden by use of predefined field COMMIT MODE. The following options are available:

COMPATIBLE. Changes are committed as soon as all locks are freed. This provides maximum compatibility between APPXIO files and RDBMS tables.

AUTOCOMMIT*. APPX will commit each modification as it is made.
***This value will be implemented in a future release.**

PCF CYCLE. APPX will commit modifications as each PCF record is processed.

END. APPX will only commit modifications at the end of the process family.

MANUAL. APPX will not automatically commit changes. You must use the COMMIT statement to do so.

CHAPTER 3-13

Automatic and Optional Children

Introduction to Automatic and Optional Children

Children are simply processes that are invoked from another process (the *parent* process). If they are invoked as a result of an option selection by the user, they are optional children. If they are invoked by APPX without user intervention, they are automatic children.

APPX does, however, impose certain restrictions on the types of processes that can invoke children. Optional children can only be invoked from interactive images in menu and input processes. Automatic children can be invoked from frames in job, input, output, update, and inquiry processes. See the Automatic and Optional Child Invocations section of Chapter 3-1: Overview of APPX Processes for more on invocation.

Child Specifications

The Child specification screen is a scrolling display that identifies APPX processes to be invoked either automatically or optionally from another APPX process. The specifications that follow are a composite of those you see when defining automatic and optional children. If a specification only displays for a specific type of child, the definition indicates so.

You access the Automatic Child Processes specification screen, shown in *Figure 3-13-1*, by selecting AUTOMATIC PROCESSES from the Frames specification screen for input, output, update, and inquiry processes, or by selecting STEPS from a job processes screen. You access the Optional Child Processes specification screen, shown in *Figure 3-13-2*, by selecting OPTIONAL PROCESSES from the Images specifications screen for input processes, or by selecting OPTIONAL PROCESSES from the Menu Processes screen.

Automatic Child Processes screen

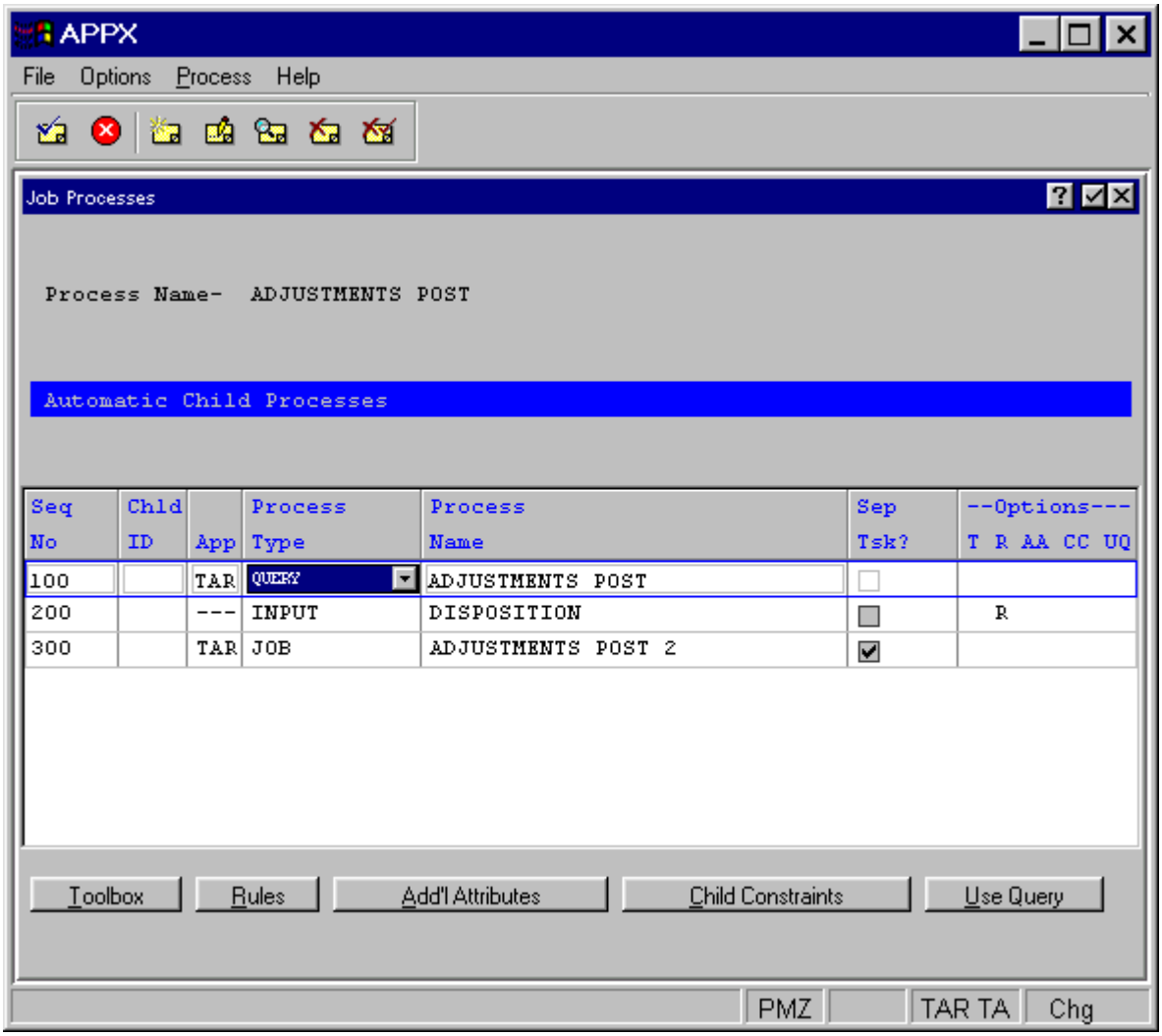


Figure 3-13-1

Optional Child Processes screen

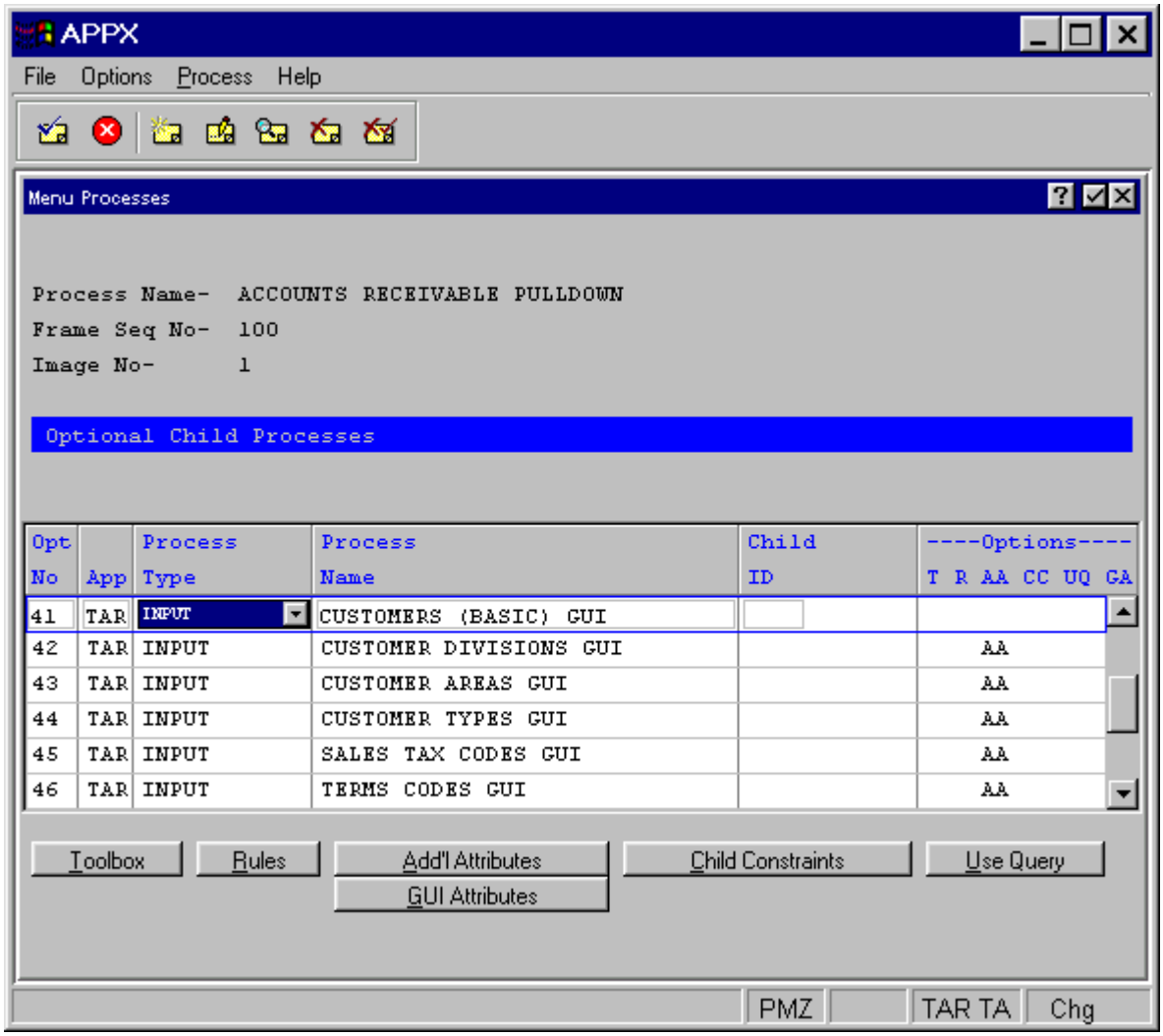


Figure 3-13-2

The child screens display one of the following two fields, depending upon whether the child process is automatic or optional.

Seq No

For an automatic child, establishes the order in which this child is invoked for this frame. You can explicitly alter this default execution sequence with statements and the *Child ID* specification. The first automatic child that is defined in add mode is assigned sequence number *100*, and additional fields are assigned a number equal to the highest existing sequence number, plus *100*. This default sequence number can be overridden, but must be unique for each automatic child that is invoked from the same parent process frame.

Opt No

For optional children, represents the number of the option that a user must select to invoke this optional child process from a menu or an input image. The number must be unique for each optional child that is invoked from the same parent process image. There is no sequential relationship that is established between any two or more optional children.

The following specifications uniquely identify a process to be invoked as an automatic or optional child.

Child ID

Can be used to control execution sequence for automatic children. For both automatic and optional children (that do not execute in any predetermined sequence), the field can also be used to pass four characters of information to the child process with the CHILD ID predefined field. If the automatic children are to invoke in *Seq No* order, leave this specification blank. If the automatic children can be invoked in other than *Seq No* order, the *Child ID* functions as a label. You use this specification in conjunction with statements and the predefined field NEXT CHILD ID to modify the execution sequence. Refer to Chapter 3-16: Using Event Points for additional information.

Process Type

Identifies the type assigned to the process that is to be invoked as an automatic or optional child. The *Process Type* options are Menu, Job, Input, Output, Update, Query, Inquiry, Status, and Subroutine. See *Chapter 3-1* for an explanation of the different process types.

App

Identifies the application that contains the process to be invoked as an automatic or optional child.

Process Name

Designates the process that is to be invoked as an automatic or optional child. To invoke the startup menu for a selected database, define the process type as *MENU*, and leave the application ID and process name blank. To invoke the startup menu for a particular application, fill in the application ID and leave the process name blank.

Separate Task?

For automatic child processes, select “yes” ☑ if you want the child process to be run as a separate task in background. A *task* is defined as a single APPX session. If “no” ☐, the process cannot start a separate task. If “yes”, the process will start a separate task if the operating system supports background processing and if the operator selected the *Background* option from a disposition screen. This task is independent of, and runs concurrently with, any subsequent child processes. If you want to run several processes within a single background session, you must create another job, and define the processes in that job as NOT being separate tasks. You must then invoke that job as a child process with *Separate Task?* ☑ in place of the originally invoked child processes. The default is “no”.

Options

Are indicators that are present whenever documentation (T), rules (R), additional attributes (AA), child constraints (CC), use query (UQ), or GUI attributes (GA) have been defined for this child.

The following options as described in the corresponding sections are available on one or both of the child screens:

- Toolbox
- Rules
- Additional Attributes
- Child Constraints
- Use Query
- GUI Attributes

Child Toolbox Option

The toolbox overlay appears on the screen when you invoke the TOOLBOX option from the child process screen. The overlay contains a number of features such as Documentation Management and Security Management that will assist you with designing and invoking a child process. The options listed in the Toolbox overlay are described briefly below. See *Chapter 3-3, Process Definition* for a more detailed explanation of the options listed on the Toolbox overlay.

Database Management

The DATABASE MANAGEMENT option allows you to add new files or maintain existing ones without leaving process design for automatic and optional processes.

Process Management

The PROCESS MANAGEMENT option allows you to add other types of processes or maintain existing ones without leaving process design for automatic and optional processes.

GO Add

This feature allows you to add any type of process without leaving child process design. Note that this option, like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

GO Delete

This feature allows you to delete any type of process without leaving child process design. Note that this option, like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

GO Inquire

This feature allows you to search for any type of process without leaving child process design. Note that this option, like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

GO Change

This feature allows you to change the specifications of any type of process without leaving child process design. Note that this option, like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Syntax Check

This feature allows you to create the Executable Module for the current automatic or optional process without actually running it. If the Executable Module for the process is valid this option is ignored when invoked. Note that this option, like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

View Audit Info

This feature allows you to see who created the currently selected automatic or optional process and who changed the process most recently. It also gives the date of creation and the date it was last changed. Note that this option, like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Invoke a Process

This feature is available to facilitate testing. It allows you to invoke any process; the currently selected automatic or optional process using the Test Database defined in system administration is the default. Note that this option, like any option with a value of 90 or greater, is available for use from any screen in Application Design without having to go to the Toolbox.

Child Rules Option

RULES accesses the Event Points For Rules overlay, shown in *Figure 3-13-3*. If statements were defined in the event point for the option that is currently active on the screen, the event point text is highlighted.

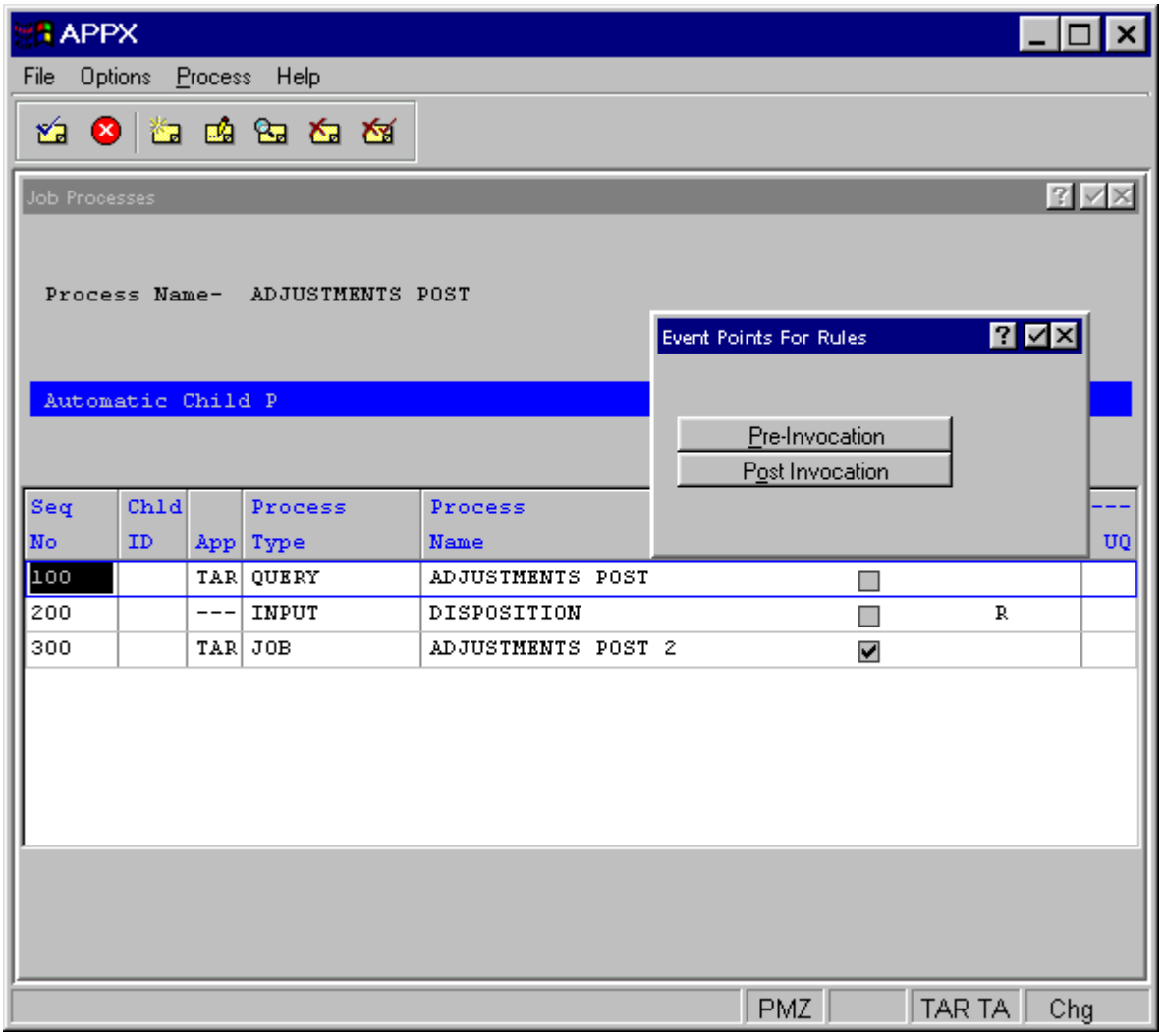


Figure 3-13-3. Child Event Points For Rules Overlay

The child Event Points For Rules overlay contains the following options:

Pre-Invocation

PRE-INVOCATION accesses the ILF editor screen for the *Pre-Invocation* event point. Upon completion, you are returned to the child Event Points overlay.

Post Invocation

POST INVOCATION accesses the ILF editor screen for the *Post Invocation* event point. Upon completion, you are returned to the child Event Points overlay.

Refer to *Chapter 3-16, Using Event Points,* for additional information.

Child Additional Attributes Option

ADDITIONAL ATTRIBUTES allows entry of invocation type, parent, and cancel dispositions. The child Additional Attributes overlay is shown in *Figure 3-13-4*.

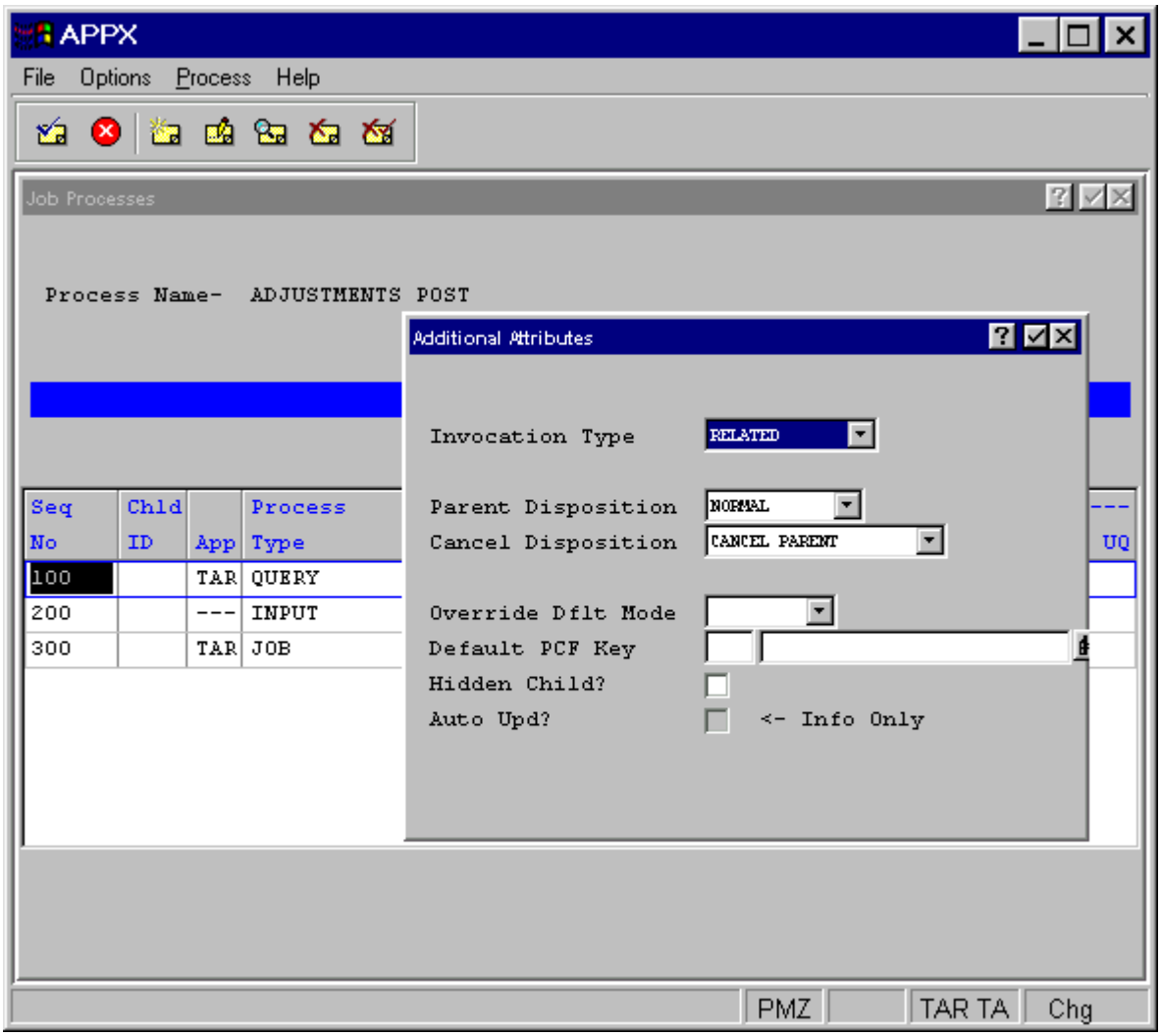


Figure 3-13-4. Child Additional Attributes Overlay

The child Additional Attributes overlay contains the following fields:

Invocation Type

Defines the boundaries of shared information between the parent and child process, or how tightly coupled the child is to the parent. It determines the types of information (such as a record area, a file position or the contents of a temporary file) that the parent process can pass to and share with the child process it invokes. Whether or not a specific item of information is passed or shared depends upon *Pass* and *Share* specifications that were entered in the data dictionary, or defined explicitly with statements. The options (ranked from most to least tightly coupled) are *SUBPROCESS, RELATED, DETACHED*. If you enter a partial value, APPX selects the option with the next highest value. For example, an entry of *REL* or *R* is equivalent to entering *RELATED*. See *Chapter 1-5* for more information about Interprocess Communication.

Parent Disposition

Determines what happens to the parent process when a child process is invoked. The *Parent Disposition* options are*:*

- **NORMAL.** The parent process resumes after the child process completes. For example, after a line item entry process (the child) completes, the user returns to the order header entry process (the parent) to enter the next order.
- **END BEFORE.** The parent process completes immediately before the child process invokes. The parent's *End of Process* event point executes and the parent process ends before the child is invoked. After the child process completes, the user is positioned wherever they would have gone if the parent process had ended normally.
- **END AFTER.** The parent process completes immediately after the child process ends. After the child's *End of Process* event point executes, the parent's *End of Process* event point executes and the parent process ends.

The default is *NORMAL.*

Cancel Disposition

Governs the effect of a user electing to terminate the child process with the APPX CANCEL option. The *Cancel Disposition* options are:

- **DISABLE.** The user is not permitted to cancel the child process.
- **RETURN TO PARENT.** The parent process resumes after the user cancels the child process.
- **END PARENT.** The parent process is ended if the user cancels the child process.
- **CANCEL PARENT.** The parent process is also cancelled if the user cancels the child process.

The default is *Return to Parent* for optional children and *Cancel Parent* for automatic children.

- ***Override Dflt Mode.*** A child process automatically defaults to the same mode as the parent process unless you enter a specific mode here. In the case of query child processes, note that this specification can affect how a query is processed. Usually, mode is left blank or specified as ADD for

a query. However, if you intend to reuse existing query temporary files while repeating query setup, enter CHANGE here in order to preserve any runtime modifications to the initial query setup. See Populating Query Files and Child Use Query Option for more information.

Default PCF Key

Determines the specific keypath that is initially presented to a user for this child process. This specification overrides the keypath that is defined at the process level. If keypath specifications are blank at all levels, the primary key is the default keypath.

Hidden Child?

Indicates whether or not this child should be excluded from generated KEATerm displays. Click the checkbox to toggle between "yes" ☑ and "no" ☐.

Auto Upd?

For informational purposes only, this field indicates whether or not the child record will be automatically updated or deleted to correspond with updates/deletes of the parent record. The checkbox will contain "yes" ☑ or "no" ☐.

Child Constraints Option

CHILD CONSTRAINTS allows entry of child selection criteria specifications. The Child Constraints overlay, shown in *Figure 3-13-5*, is a scrolling display.

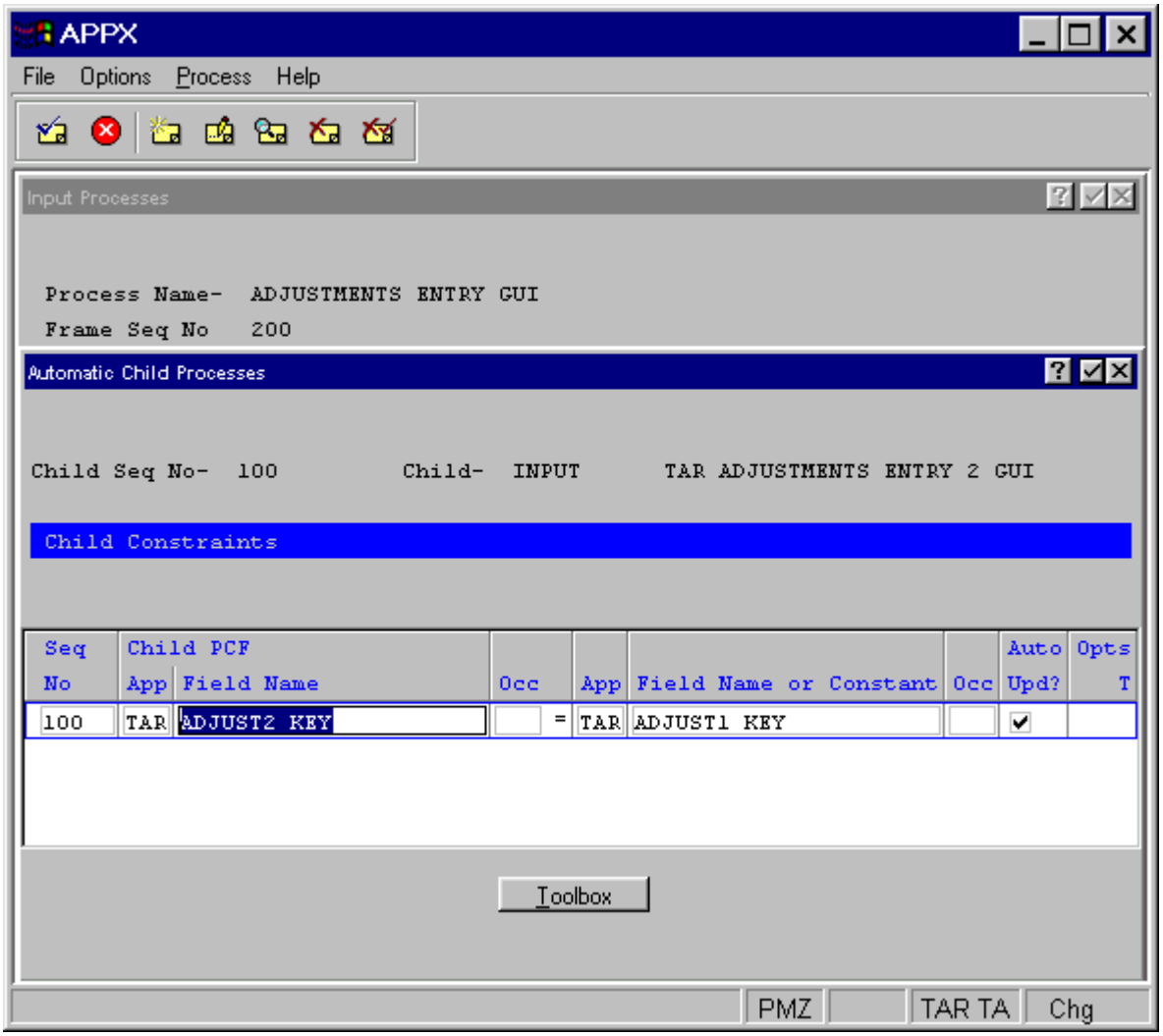


Figure 3-13-5. Child Constraints Overlay

The Child Constraints Overlay contains some or all of the following fields.

Seq No

Displays a sequence number that is defaulted by APPX. This default sequence number can be overridden, but must uniquely identify each constraint. Since all of the constraints are ANDed together, the order the constraints are applied in is not significant. The first field that is defined in add mode is assigned sequence number *100*, and additional fields are assigned a number equal to the highest existing sequence number, plus *100*.

Child PCF/App/Field Name/Occ

Designates the field (by application ID, field name, and occurrence number) from the child's process control file, which must be equal to the *Field Name or Constant* entered beside it. This entry is not restricted to a key field.

App/Field Name or Constant/Occ

Identifies a field or constant. The field you entered above must be *equal to* this field or constant. Other relations (for example, *LE* or *GT*) are not permitted.

Auto Upd?

For automatic child processes invoked from input processes, indicates that this constraint should be used as a basis for automatic data updates or automatic record deletions. For example, assume that you create an APPX program to edit back orders, including a parent input process for the order header and a child input process for the line items. You constrain the child process so that the line item order number equals the order number in the parent. If "yes" ☑, the following updates will occur: If you delete the parent record, all constrained child records are deleted. In addition, if you change the constraining field (usually the key), APPX will change the corresponding field in each of the child records. If "no" ☐, these updates do not occur.

Child Use Query Option

For automatic child processes, the Use Query option identifies a specific query process by *Application* and *Process Name* that determines the records available to this child process. This specification affects input, output, and update processes. The child Use Query overlay is shown in *Figure 3-13-6* below.

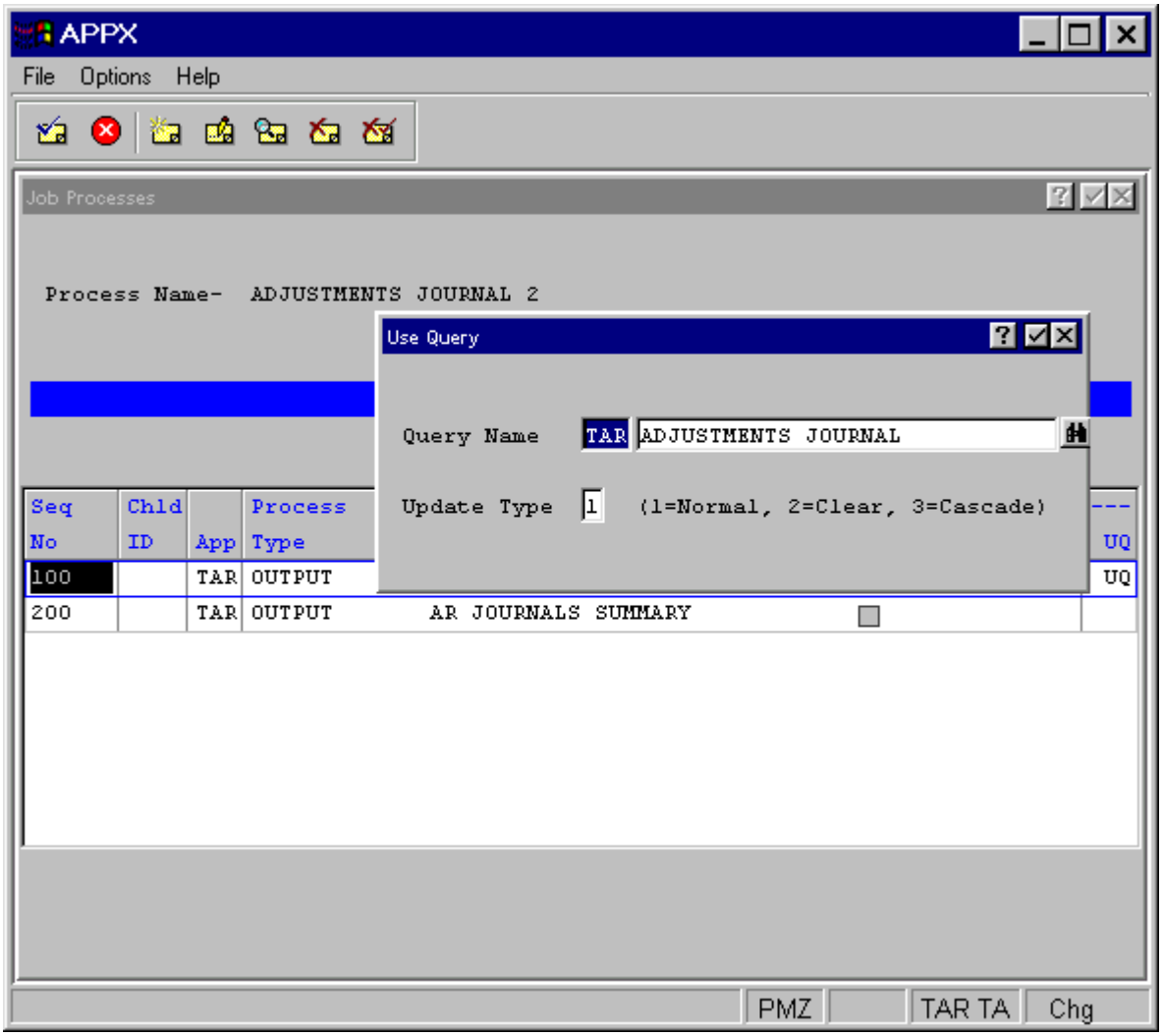


Figure 3-13-6. Child Use Query Overlay

The child Use Query overlay contains the following fields.

- *Query Name.* The PCF of the query that is identified by this specification must match the PCF of the child process.
- ***Update Type*** determines the disposition of any existing pointer file for the query process to be used. The query *Update Type* options are:
- **1 - NORMAL.** Generate the pointer file if it does not yet exist. If the pointer file exists, use it without re-executing the query.
- **2 - CLEAR.** Discard any existing pointer file and generate a new one.
- **3 - CASCADE.** Apply any new constraints to further restrict the contents of an existing pointer file.

> Note that if the new constraints are less restrictive than the constraints that were used to generate the existing pointer file, the *CASCADE* option will not add records to the pointer file. In such a case, you should use the *CLEAR* option.

Also, if you intend to reuse an existing pointer file and yet repeat the query setup, be sure that you specify CHANGE as the Override Default Mode in the Additional Attributes for the second query.

The default is *2 (CLEAR)*.

Child GUI Attributes Option

Graphical Attributes are available for optional child processes only. These user options can be listed and displayed in a variety of ways on menus and input images including pull-down and right-click context menus instead of the traditional option number or button-driven user options. This feature is discussed in the *Defining Graphical Attributes for Optional Children* section of *Chapter 3-15: Using GUI Features of the Image Editor*.

CHAPTER 3-14

Using the Image Editor

Image Editor

The Image Editor, APPX's screen painting facility, is the means used to format menu, input, output, and inquiry process images. With Image Editor options described in the next section, you can design both the visual presentation of each item on an image as well as the background characteristics of the image itself. As you design the image, you are seeing the image exactly as it will appear to your users. Options are also provided to control cursor positioning and to perform data lookups.

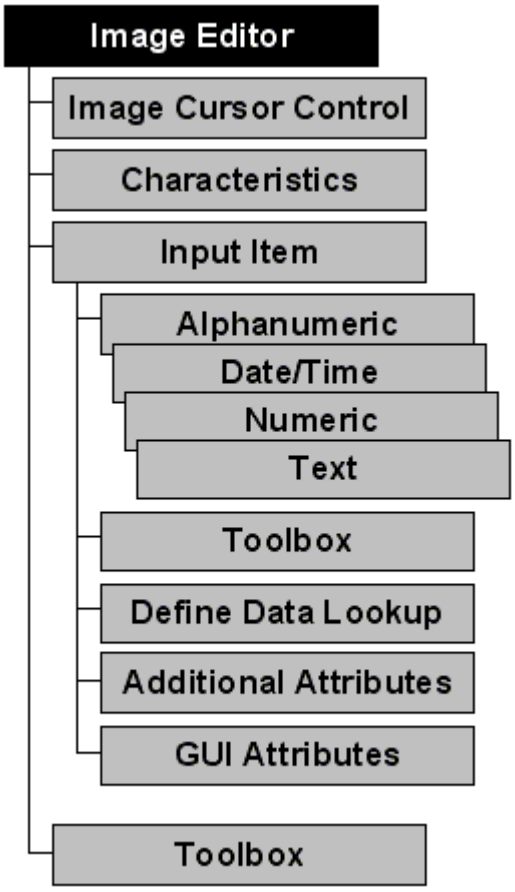


Figure 3-14-1. Image Editor

APPX automatically invokes the Image Editor when you press ENTER after completing the appropriate specifications for menu, input, output, or inquiry images. When you access the layout for an existing image, APPX displays the image as it appears to a user, and you can edit its items, text, and background characteristics. If you are creating a new image, APPX displays a blank screen on which you can paint items, text, and background characteristics. In either case, APPX creates a vertical and a horizontal line on the screen to indicate the bottom and right edges of the image, based upon your row and column specifications.

This chapter addresses the standard features of the Image Editor which are common to both character-based and GUI applications. However, if you are designing for a GUI-enabled environment, you will probably want to paint items using the GUI features of the Image Editor described in Chapter 3-15. Most of the icons on the Image Editor toolbar apply to the GUI features and are not discussed in this chapter. You can use both standard and GUI features on the same image and switch back and forth interchangeably without leaving the image.

Figure 3-14-2 below shows an input image being edited in the Image Editor. This image uses only standard features of the editor and is suitable for a character-based application. See Figure 3-15-1 in Chapter 3-15 for a similar input image using GUI features being edited in the Image Editor.

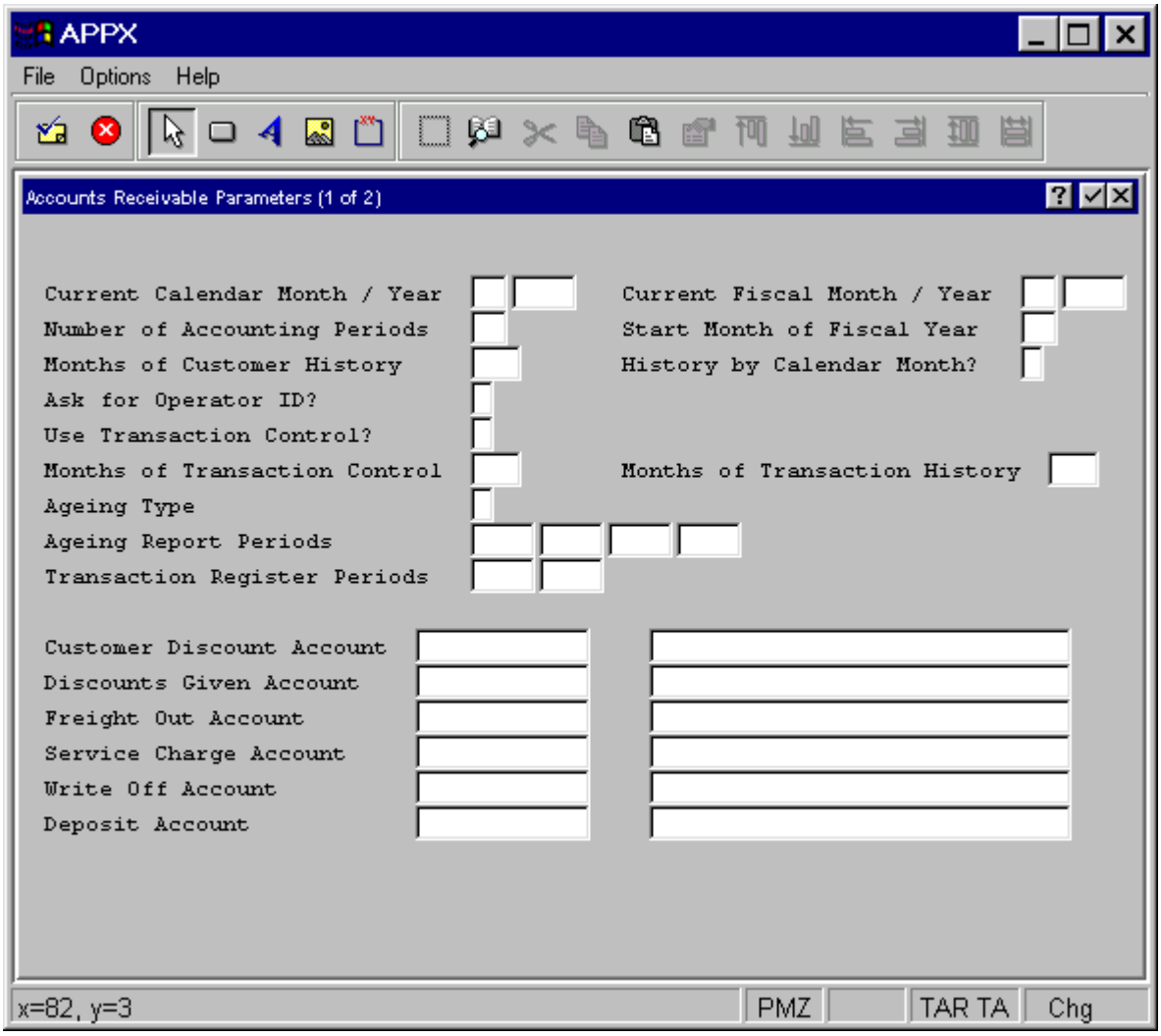


Figure 3-14-2. Input Image Using Standard Features of the Image Editor

Image Editor Options

APPX provides a number of tools in the Image Editor for use in designing a new image or redesigning an existing image. As mentioned earlier, most of the icons on the Image Editor toolbar apply to GUI features and require the use of a mouse. However, the standard features of the Image Editor can be employed from the keyboard using the cursor movement keys and various function and option keys. These options are listed on the Image Editor Options help screen, shown in *Figure 3-14-3,* and are summarized below. The help screen is available online by pressing F1 (HELP) from the Image Editor. **If option numbers are not displayed and you wish to see them, select SHOW OPTION NUMBERS from the Options pulldown menu.**

The Image Editor Options help screen is for informational purposes only; buttons, icons, and options are not operable on this screen. To perform these operations, you must first return to the Image Editor screen by pressing ENTER, selecting the CLOSE option, or selecting END from the titlebar.

Image Editor Options Help screen

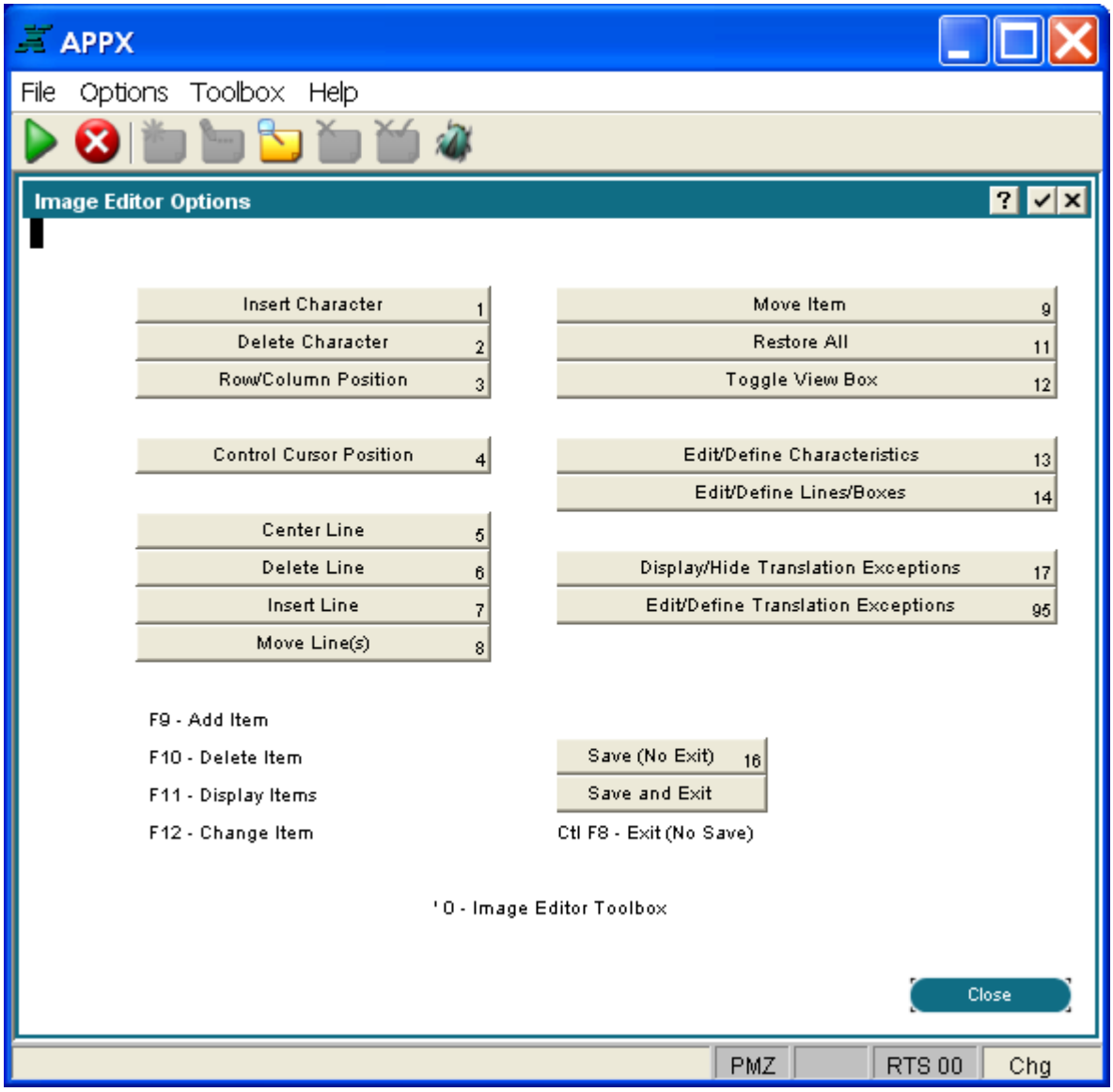


Figure 3-14-3

The standard feature options within the Image Editor include the following:

- ***Insert Character*** (`OPTION` `1`) inserts a space immediately before the current position of your cursor.
- ***Delete Character*** (`OPTION` `2`) deletes the character (or space) you cursor is currently positioned on.
- ***Row/Column Position*** (`OPTION` `3`) displays an overlay that contains the current row/column position of your cursor. To clear the display, select `ENTER`.
- ***Control Cursor Position*** (`OPTION` `4`) accesses the Control Cursor Position overlay that controls cursor positioning when a menu or input image displays to a user. See the Control Cursor Position Option section for details.

- ***Center Line*** (OPTION 5) centers or re-centers an existing line of characters, based on the number of columns in an image. You must use this option if you edit a centered line and change its position or length.
- ***Delete Line*** (OPTION 6) deletes the entire line your cursor is currently positioned on.
- ***Insert Line*** (OPTION 7) inserts a new line immediately above the line your cursor is currently positioned on.
- ***Move Line(s)*** (OPTION 8) moves one or more lines, starting with the line your cursor is currently positioned on. After selecting this option, you are prompted to identify the ending line you want to move, and the new position for the line or lines you want to move.
- ***Move Item*** (OPTION 9) moves an item from its current position on an image and places it in a position that is established by the placement of your cursor. Position the cursor on the item before selecting this option. After you select the option, pseudo-blanks for the item disappear. Move the cursor to indicate the new starting position for the item, and select ENTER ☑.
- ***Restore All*** (OPTION 11) reverts to the last saved image.
- ***Toggle View Box*** (OPTION 12) repositions an overlay such as the prompt for moving lines on a different part of the image.
- ***Edit/ Define Characteristics*** (OPTION 13) edits or defines image characteristics. Such characteristics include bright and inverse video for menu and input images, bold and expanded print for output and inquiry images, plus underlining and foreground/background colors for any images. See *Defining Image Characteristics* for additional procedural information.
- ***Edit/Define Lines/Boxes*** (OPTION 14) edits or defines lines and boxes. Once you have established the boundaries of the line or box (using the same procedure described in *Defining Image Characteristics for* OPTION 13*), a Box Properties overlay appears. See the Painting Boxes or Lines section of Chapter 3-15 for specification details on the Box Properties overlay.*
- ***Display/Hide Translation Exceptions*** (OPTION 17) toggles between highlighting and removing the highlighting from any words, phrases or sentences that have translation exceptions defined for them. This feature is relevant only if the Language Translator feature is in use at your installation. Refer to the *Language Translation Manual* for more information.
- ***Edit/Define Translation Exceptions*** (OPTION 95) allows you to select words, phrases or sentences on the image and define translations exceptions for them for APPX to use when performing language translations. This feature is only used if the Language Translator feature is in use at your installation. Refer to the separate *APPX Language Translator Manual* for more information.
- F9 - ***Add Item*** defines an item representing a single appearance of a logic, numeric, alphanumeric, or data/time field. Position the cursor where the new item will start and press F9. Refer to *Defining Items for Input, Output, and Inquiry Images* for additional procedural information.
- F10 - ***Delete Item*** removes an item from an existing image. Position your cursor on the item to be deleted and press F10.
- F11 - ***Display Items*** calls the Items Display overlay, shown in *Figure 3-14-4* below, which lists the items defined on the current image along with many of their attributes. Certain attributes can be

changed directly from this overlay. Item attributes are discussed in the *Image Item Specifications or in the various Additional Attributes* sections. You can press F11 at any time to access the *Items Display*.

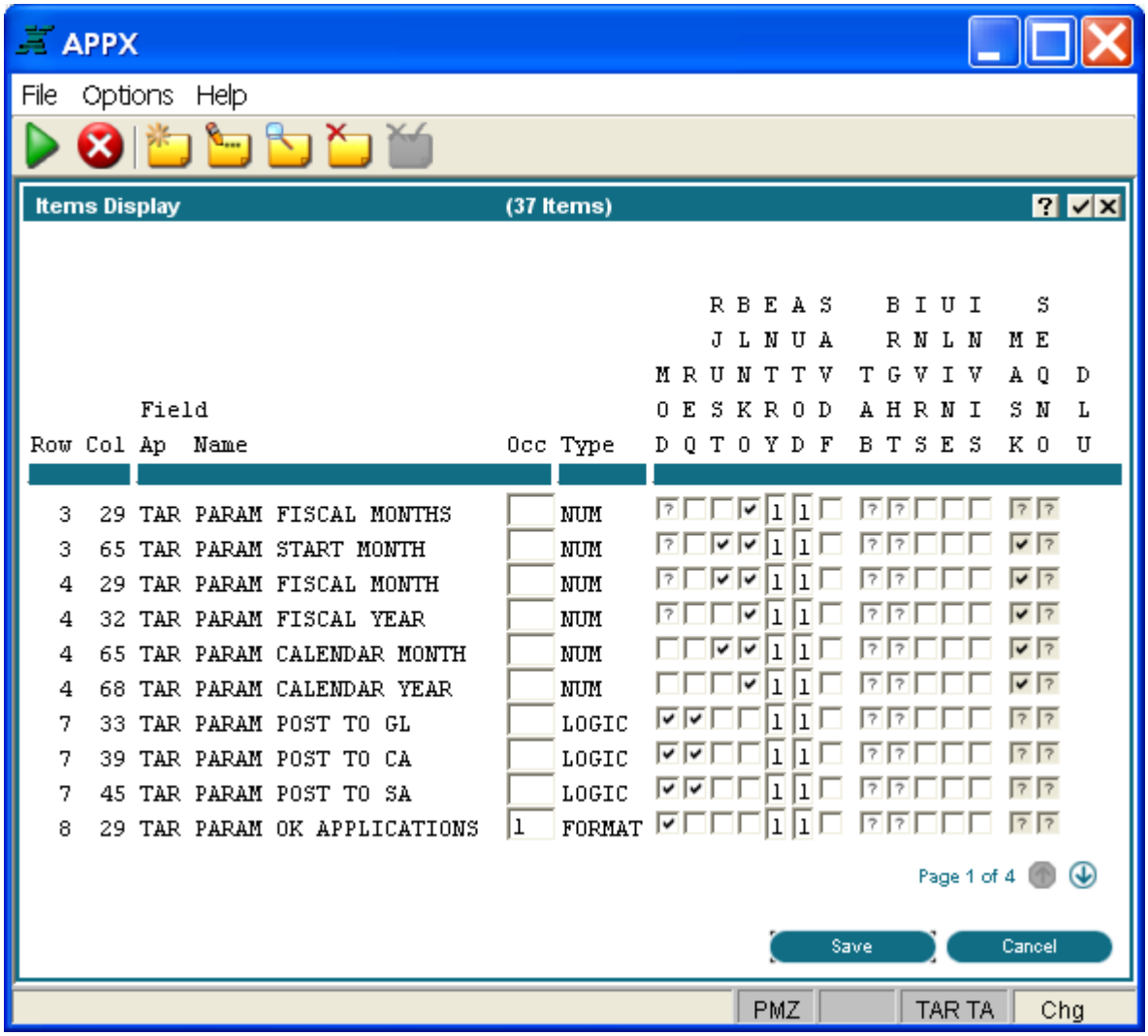


Figure 3-14-4. Image Editor Items Display

- F12 - ***Change Item*** edits an existing item. Position the cursor on the item to be edited and press F12.
- ***Save (No Exit)*** (OPTION 16) saves everything you added or changed on a painted image. APPX does not exit the image editor after saving an image.

Save and Exit

Automatically saves all additions or changes to an image and leaves the image editor environment. Select the SAVE AND EXIT icon from the toolbar.

Exit (No Save)

Exits the image editor and returns you to the previous screen without saving the additions or changes you made to an image. Select the CANCEL icon from the toolbar.

- *Image Editor Toolbox* (OPTION 0) calls the Image Editor Toolbox overlay from which you can access the data dictionary, the event points for the current process, or the optional processes for the current image. See the Image Editor Toolbox Option section for details.

Control Cursor Position Option

The CONTROL CURSOR POSITION option from the Image Editor, for input or menu images only, calls an overlay containing specifications to control cursor positioning when such images display to a user. The Control Cursor Position overlay is shown in *Figure 3-14-5*:

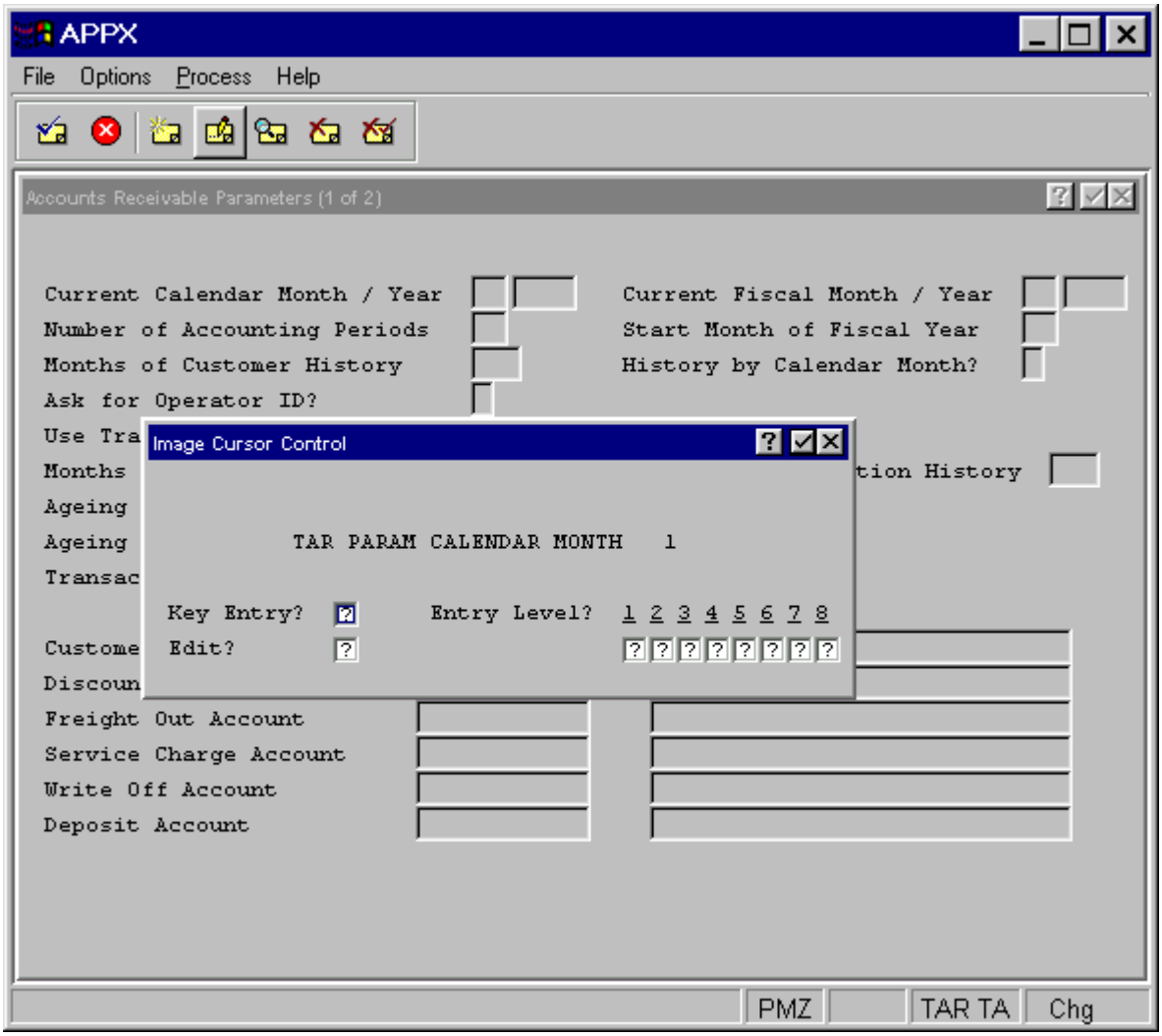


Figure 3-14-5. Control Cursor Position Overlay

The Cursor Control Position overlay contains the following fields:

Key Entry?

Indicates whether or not the cursor is to be positioned on this field during key entry. The default is "unknown" ?. Click the checkbox to toggle between "yes" ☑ and "no" ☐.

Edit?

Indicates whether or not the cursor is to be positioned on this field during data modification. The default is "unknown" ?.

Entry Level?

Indicates, for entry levels 1 through 8, whether or not the cursor is to be positioned on this field at that entry level. The default is "unknown" ?.

Image Editor Toolbox Option

The TOOLBOX option from the Image Editor screen calls the Image Editor Toolbox overlay shown in *Figure 3-14-6* below:

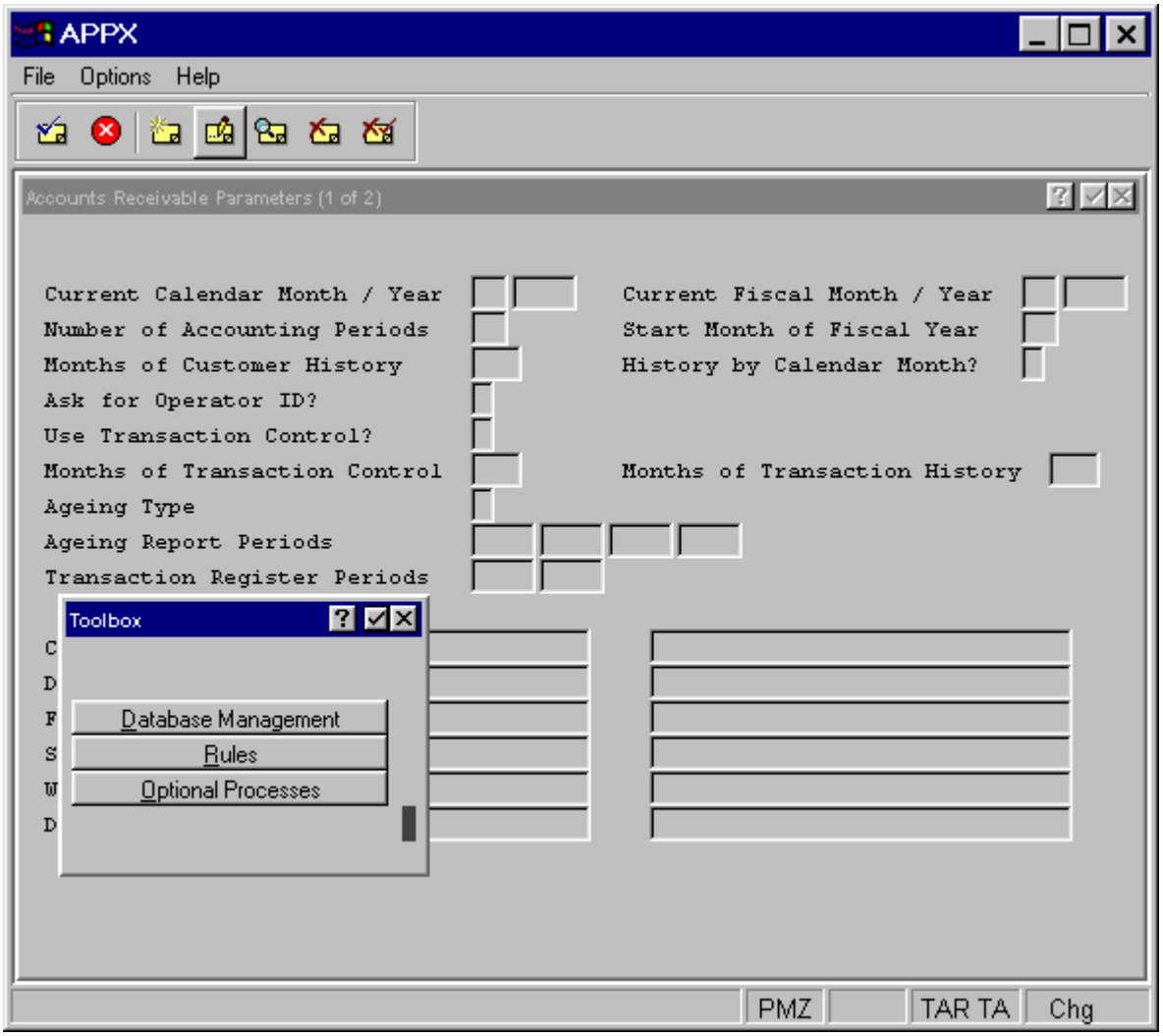


Figure 3-14-6. Image Editor Toolbox Overlay

The following options are available on the Image Editor Toolbox overlay:

Database Management Option

The DATABASE MANAGEMENT option allows you to add new files or maintain existing ones without leaving the image editor. See Chapter 2-1: Accessing the Data Dictionary for additional information.

Rules Option

The RULES option calls an Event Points overlay which provides options that allow you to access the ILF Editor for applicable event points. If statements exist in an event point, the event point option is highlighted. Refer to the Image Editor Toolbox Rules Option section as well as Chapter 3-16: Using Event Points for more information.

Optional Processes Option

This option accesses the Optional Children specification screen for the current image. These specifications include the name of the child process, the invocation type, and whether or not various activities at the child level affect the parent process. See Chapter 3-13: Automatic and Optional Children for detailed information on defining optional child processes.

Upon completion of any option, you are returned to the Image Editor.

Image Editor Toolbox Rules Option

The RULES option from the Image Editor Toolbox presents an Events Point overlay with options to access the ILF Editor for applicable event points. If statements are defined in the event point, the event point option is highlighted. Refer to Chapter 3-16: Using Event Points for more information.

The Image Editor Toolbox Event Points overlay may contain some or all of the following options, depending on the type of process:

- START OF PROCESS
- SET RUN-TIME DEFAULTS
- POST PCF READ
- SELECT IMAGE
- GLOBAL EP'S FOR IMAGES allowing access to four additional event points:
- START OF FRAME
- GLOBAL PRE-DISPLAY
- GLOBAL OPTION INTERCEPT
- END OF FRAME
- PRE-DISPLAY
- PRE-DISPLAY/VERIFY
- OPTION INTERCEPT
- VERIFY
- PRE-PCF UPDATE
- POST PCF UPDATE
- END OF PROCESS
- FILE LEVEL EP'S allowing access to five additional event points:
- FILE OPEN
- POST READ
- PRE-UPDATE
- POST UPDATE

- FILE CLOSE

Upon completion of any event point, you are returned to the Event Points overlay. **Press ENTER, select the CLOSE option, or select END from the titlebar to return to the Image Editor.**

Editing or Defining Items on Images

An item is a single appearance of an APPX field that is painted on a menu, input, output, or inquiry image. You can edit or define an item using the keyboard as itemized below. However, if you are designing for a GUI-enabled environment, you will probably want to edit and define items using the GUI features of the Image Editor described in Chapter 3-15. Most of the icons on the Image Editor toolbar apply to the GUI features and are not discussed here.

- **F9** - To paint an item on an image, position the cursor on the desired row/column and press F9.
- **F12** - To edit an existing item, position the cursor on the item and press F12.
- **F10** - To delete an existing item, position the cursor on the item and press F10.

Image Item Specifications

If you are editing an existing item or adding a new item, APPX presents a number of specification screens that establish various operational and display attributes, as well as data lookup definitions.

The item screen contains specifications that relate to the appearance of a data dictionary field on an input, output, or inquiry image. The specifications that follow are a composite of those you see when defining an input, output, or inquiry item. If a specification only displays for a specific type of process, the definition indicates so. The Item screen, shown in *Figures 3-14-7, 3-14-8, 3-14-9*, and *3-14-10*, displays when you choose to place a field on an image by selecting add or change mode for item definition.

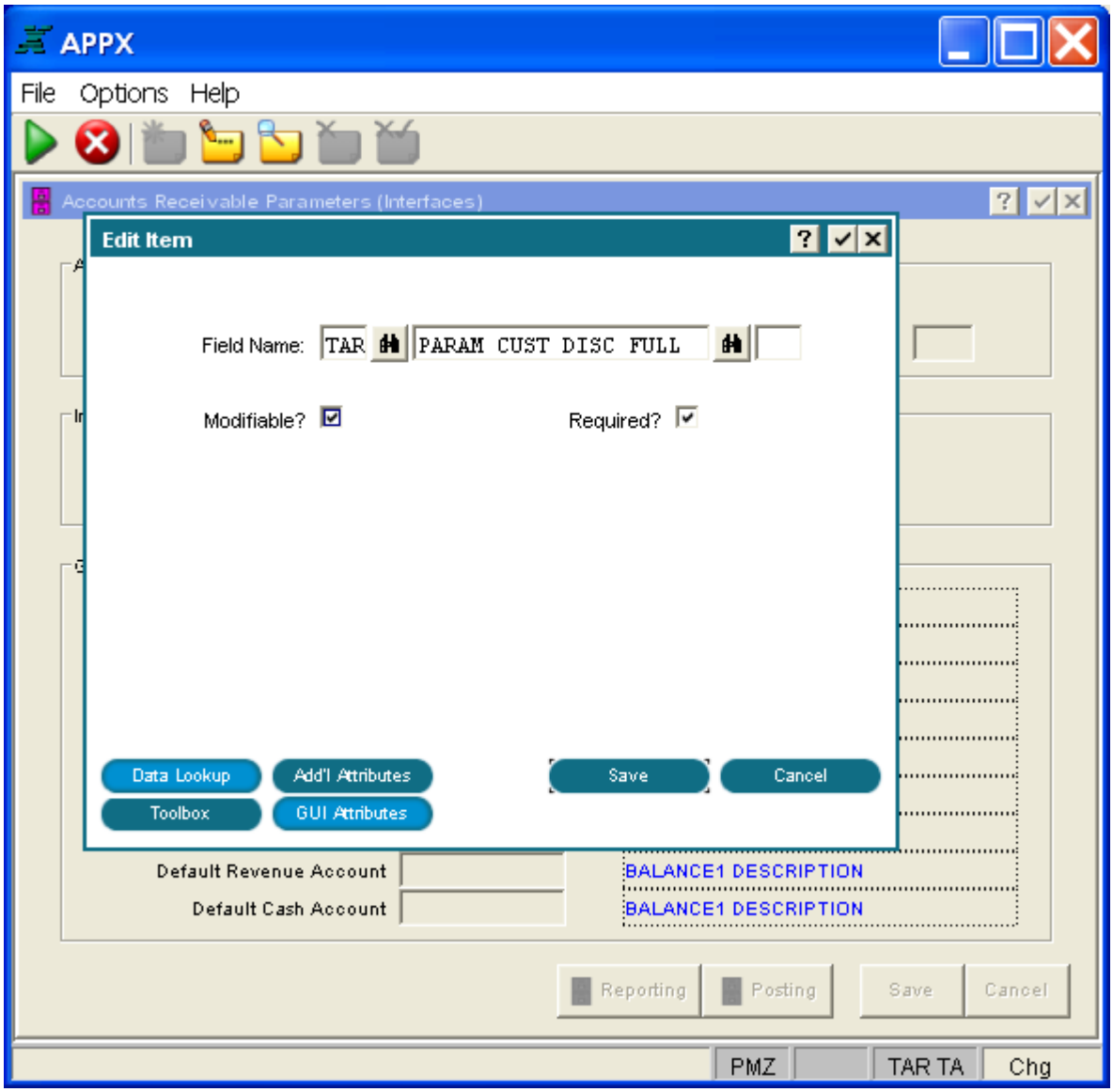


Figure 3-14-7 Input or Inquiry Key Entry Item Screen for Alphanumeric, Date, and Logic Fields

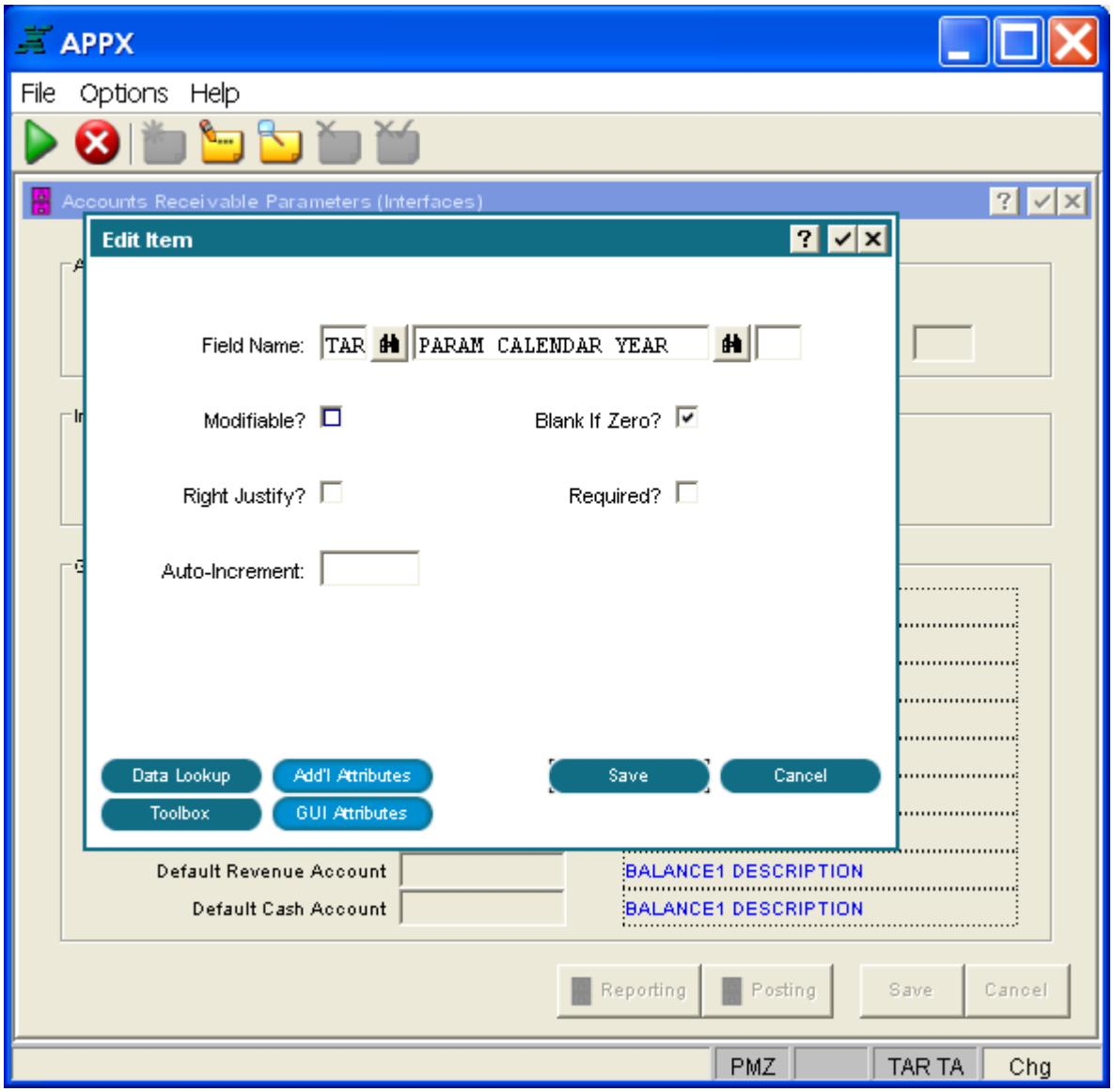


Figure 3-14-8. Input or Inquiry Key Entry Item Screen for Numeric Fields

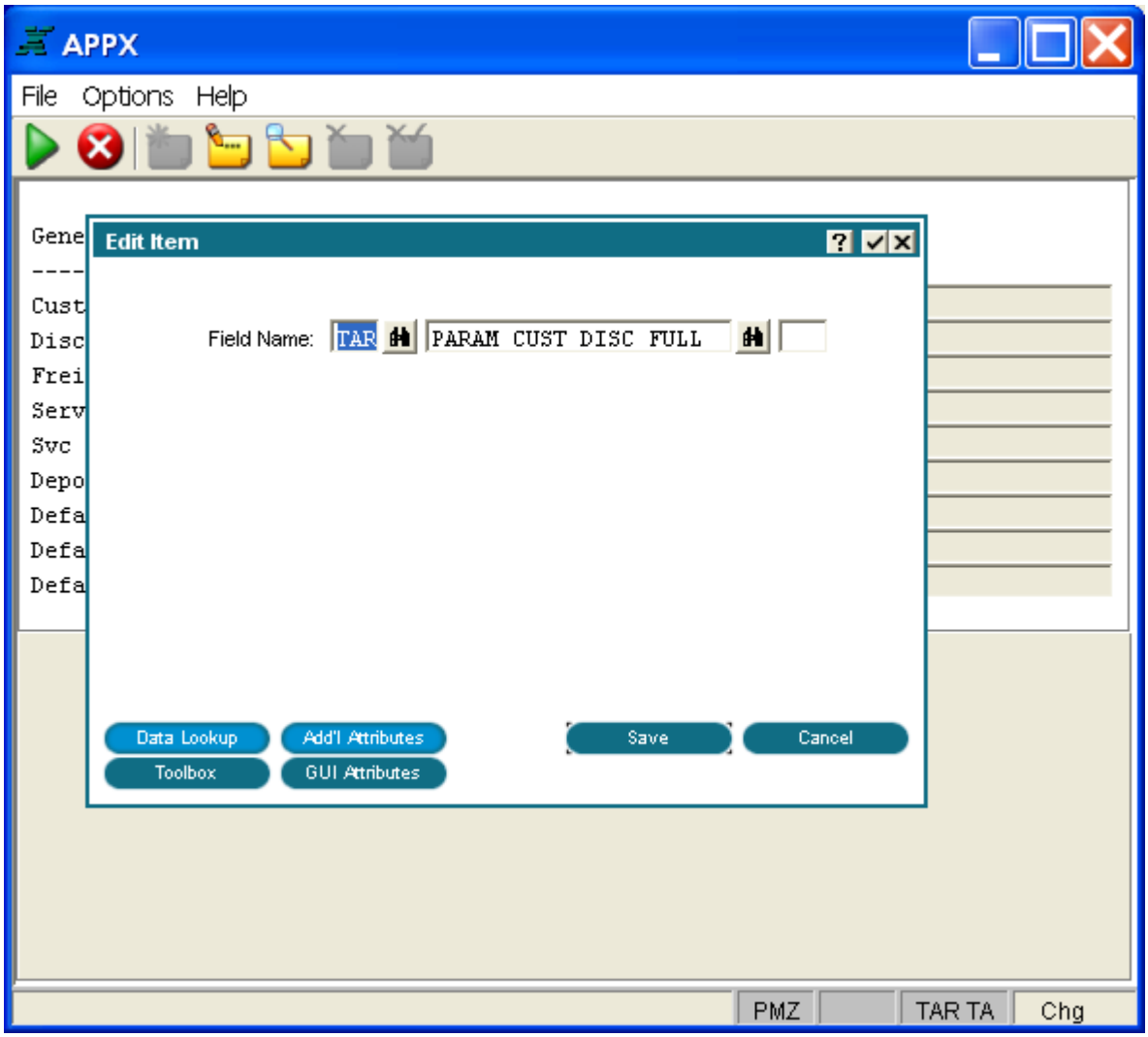


Figure 3-14-9. Output or Inquiry Item Screen for Alphanumeric, Date, or Logic Fields

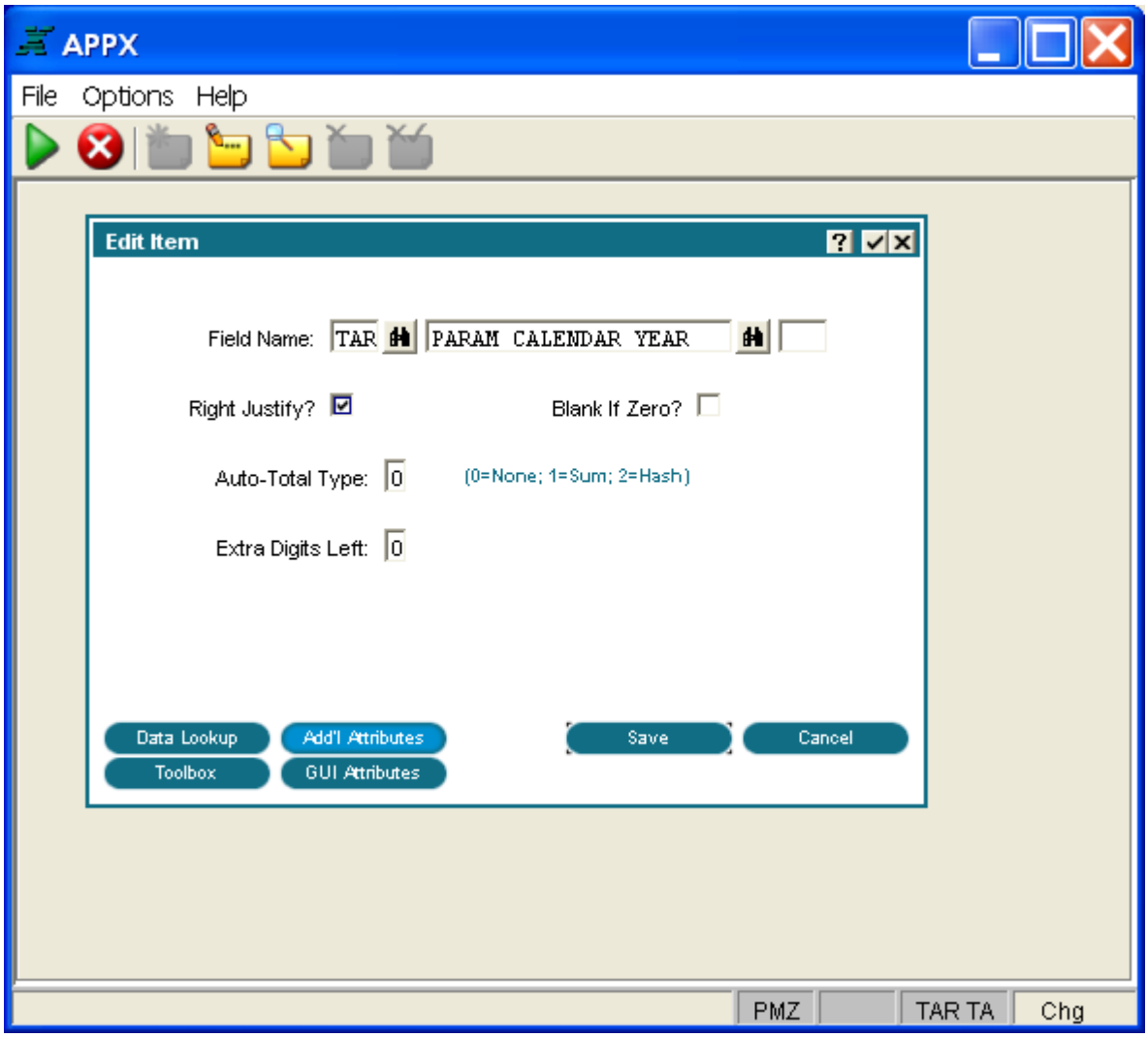


Figure 3-14-10. Output or Inquiry Item Screen for Numeric Fields

The item screens contain some of the following fields, depending on the type of process.

- ***Field Name*** identifies a specific field (by application ID, field name, and occurrence) to be painted on an image. Predefined fields can also be painted on an image; their application ID is ---.
- ***Modifiable?*** for input processes and inquiry *Key Entry* images, governs whether or not a user is permitted to change the value that displays. If "no" (blank), APPX determines whether the item is modifiable, based upon the operational phase. If the operational phase is live operations, the item is not modifiable. If the operational phase is initial setup or recovery processing, the item is modifiable. The default is "yes" ☑.
- ***Required?*** for input processes and inquiry *Key Entry* images, determines whether or not a user is required to enter a value. The default is "no" ☐.
- ***Blank If Zero?*** indicates whether or not a value of 0 should display as a blank (rather than a 0) to a user. The default is "no" ☐.

- ***Right Justify?*** governs whether or not the contents of the item should align with the far right-hand character position in the mask. If “no” ☐, the contents are aligned with the far left-hand character position in the mask. For input processes, the default is “no” ☐. For output processes, the default is "yes" ☑.
- ***Auto Increment***, for input processes, indicates that the default value for the item on each record added should be automatically incremented by the value that you enter here.
- ***Auto-Total Type***, for output processes, indicates whether or not the item is to be automatically totaled by APPX and, if so, the type of total. The *Auto Total Type* options are:
- **0 = None**. The value in this numeric item for each record is not automatically totaled by the output process.
- **1 = Sum**. The value in this numeric item for each record is automatically added together, and the total is printed for any subtotals or grand totals in the output.
- **2 = Hash**. The value in the first 16 characters of this numeric item for each record is automatically added together, and the total is printed for any subtotals or grand totals in the output. Hash totals ignore overflow. For example, if an item is five characters in length, adding one to 99999 results in a value of 0. Numeric items that are hash-totaled must be defined as packed decimal storage type in the data dictionary.

The default is *0 = None*.

- ***Extra Digits Left***, for output processes, designates a number of digits (in addition to the digits in the data dictionary) to allow for the potentially greater size of totaled fields.

The following options are available at the bottom of every Item screen:

- ***Data Lookup - See Data Lookup Option for details.***
- ***Additional Attributes - See Image Item Additional Attributes Option for details.***
- ***Toolbox - See Image Item Toolbox Option for details.***
- ***GUI Attributes*** varies depending on the type of field (alphanumeric, numeric, date, or logic) and the type of process. See the Defining Graphical Attributes for Image Fields section in Chapter 3-15: Using GUI Features of the Image Editor.

Image Item Toolbox Option

The TOOLBOX option from the Image Item Specifications screen accesses the Image Item Toolbox overlay shown in *Figure 3-14-11*.

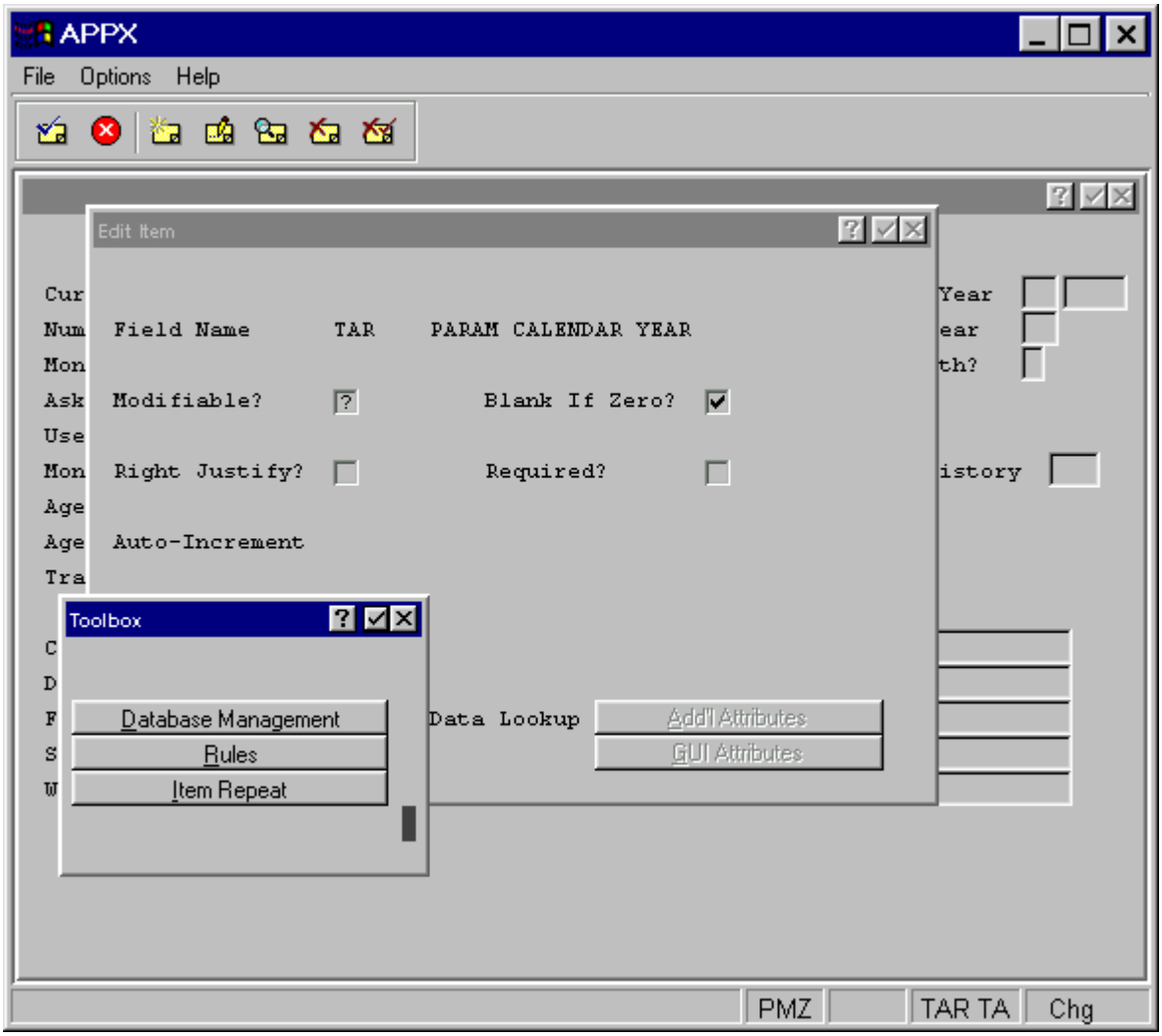


Figure 3-14-11. Image Item Toolbox Overlay

The following options are available on the Image Item Toolbox overlay:

Database Management Option

The DATABASE MANAGEMENT option allows you to add new files or maintain existing ones without leaving the image item screen. See *Chapter 3-3, Process Definition* for additional information.

Rules Option

The RULES option calls an Event Points overlay which provides options that allow you to access the ILF Editor for applicable event points. If statements exist in an event point, the event point option is highlighted. Refer to the Image Item Toolbox Rules Option section as well as Chapter 3-16: Using Event Points for more information.

Item Repeat

The ITEM REPEAT option provides for repeating items on the screen and is described in the Item Repeat Option section later in this chapter.

Image Item Toolbox Rules Option

The RULES option from the Image Item Toolbox presents an Events Point overlay with options to access the ILF Editor for applicable event points. If statements are defined in an event point, the event point option is highlighted. The Image Item Toolbox Events Point overlay contains the following options:

- START OF PROCESS accessed the ILF editor for the *Start of Process* event point. Upon completion, you are returned to the Image Item Screen.
- POST PCF READ, accesses the ILF editor for the *Post PCF Read* event point. Upon completion, you are returned to the Image Item Screen.
- SELECT IMAGE accesses the ILF editor for the *Select Image* event point. Upon completion, you are returned to the Image Item Screen.
- GLOBAL EP'S FOR IMAGES overlay allows you to access four additional entry points.
- PRE-DISPLAY accesses the ILF editor for the *Pre-Display* event point. Upon completion, you are returned to the Image Item Screen.
- PRE-DISPLAY/VERIFY accesses the ILF editor for the *Pre-display/Verify* event point. Upon completion, you are returned to the Image Item Screen.
- OPTION INTERCEPT accesses the ILF editor for the *Option Intercept* event point. Upon completion, you are returned to the Image Item Screen.
- VERIFY accesses the ILF editor for the *Verify* event point. Upon completion, you are returned to the Image Item Screen.
- END OF PROCESS accesses the ILF editor for the *End of Process* event point. Upon completion, you are returned to the Image Item Screen.
- PCF LEVEL EPS provides access to the File Level Event Points for the PCF file. These Event Points are not directly associated with the process but with the PCF file. See the *Rules Option* section of *Chapter 2-4, Files and Fields* for more information on using PCF Level Event Points.

Refer to Chapter 3-16: Using Event Points for more information.

Item Repeat Option

The ITEM REPEAT option from the Image Item Toolbox accesses the Repeat Item overlay shown in *Figure 3-14-12*.

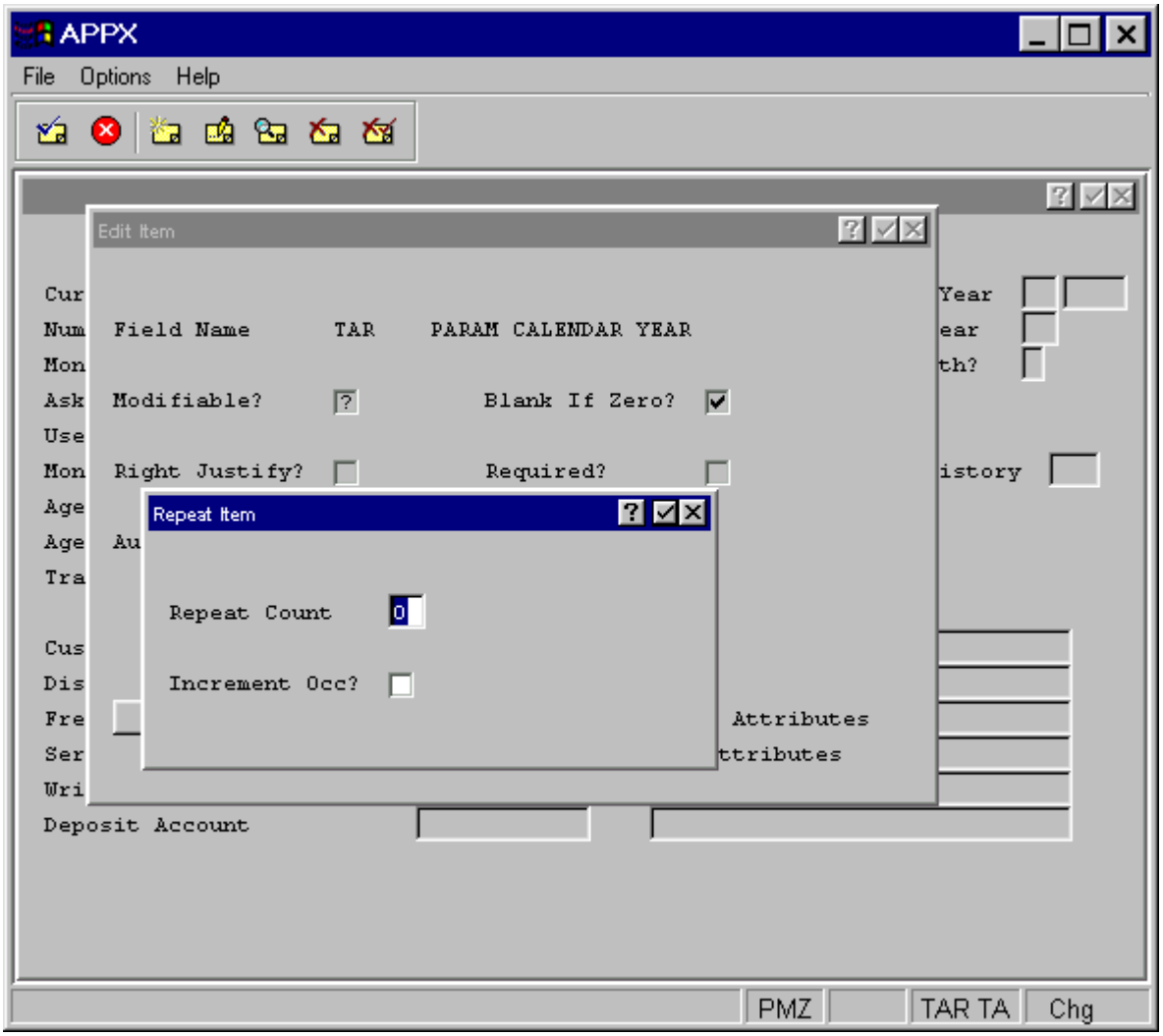


Figure 3-14-12. Repeat Item Overlay

The Repeat Item overlay contains the following fields:

Repeat Count

Field allows you to repeat an item on a screen any number of times that you specify, up to 99 times.

Increment Occ?

Tells APPX to automatically increment occurrence numbers for the repeated item.

Data Lookup Option

Data lookups link a field in one file to additional data in a related file by means of a key field in the related file. In other words, they tell APPX how to access another file during process execution. Lookups are defined for items on images in input, output, and inquiry processes. If the key field is a group, the order that the components are painted on the screen is significant. Parts of the key are assembled from left to right, and top to bottom.

You can define the same field(s) and data lookup more than once on a single image. Each time you define the field, you can designate a data lookup. For example, you may have a Product File Maintenance screen that permits entry of three alternate suppliers. Each Supplier ID is used to perform data lookup in the Supplier file.

Values in the lookup file can display on an image automatically. *Auto displays* tell APPX which of the values from the lookup file should display on an image. For example, if you define a data lookup to a customer file with a customer number key field, you can automatically display such additional fields as the customer name, address, phone, and contact name without having to execute a DISPLAY statement for each field. If a field is not auto displayed, you must define a DISPLAY statement before the field will appear on the image. (For auto display purposes, the order that fields are painted in on an image is also significant if multiple lookups to the same file are defined. The auto displayed fields must be part of the file that is defined in the preceding data lookup definition, determined from left to right, top to bottom.)

Automatic actions can also be specified if the lookup record is not on file. For example, you can specify whether this should be considered an error or a warning condition.

The Define Data Lookup overlay includes two mutually exclusive overlays that are used to define a data lookup. One screen is for the painted item that represents the first part of the key for a lookup from a specific file and a second is for additional items, if any, that comprise the rest of the key.

When you select the DEFINE DATA LOOKUP option from the Image Item Specifications screen, the overlay shown in *Figure 3-14-13* appears.

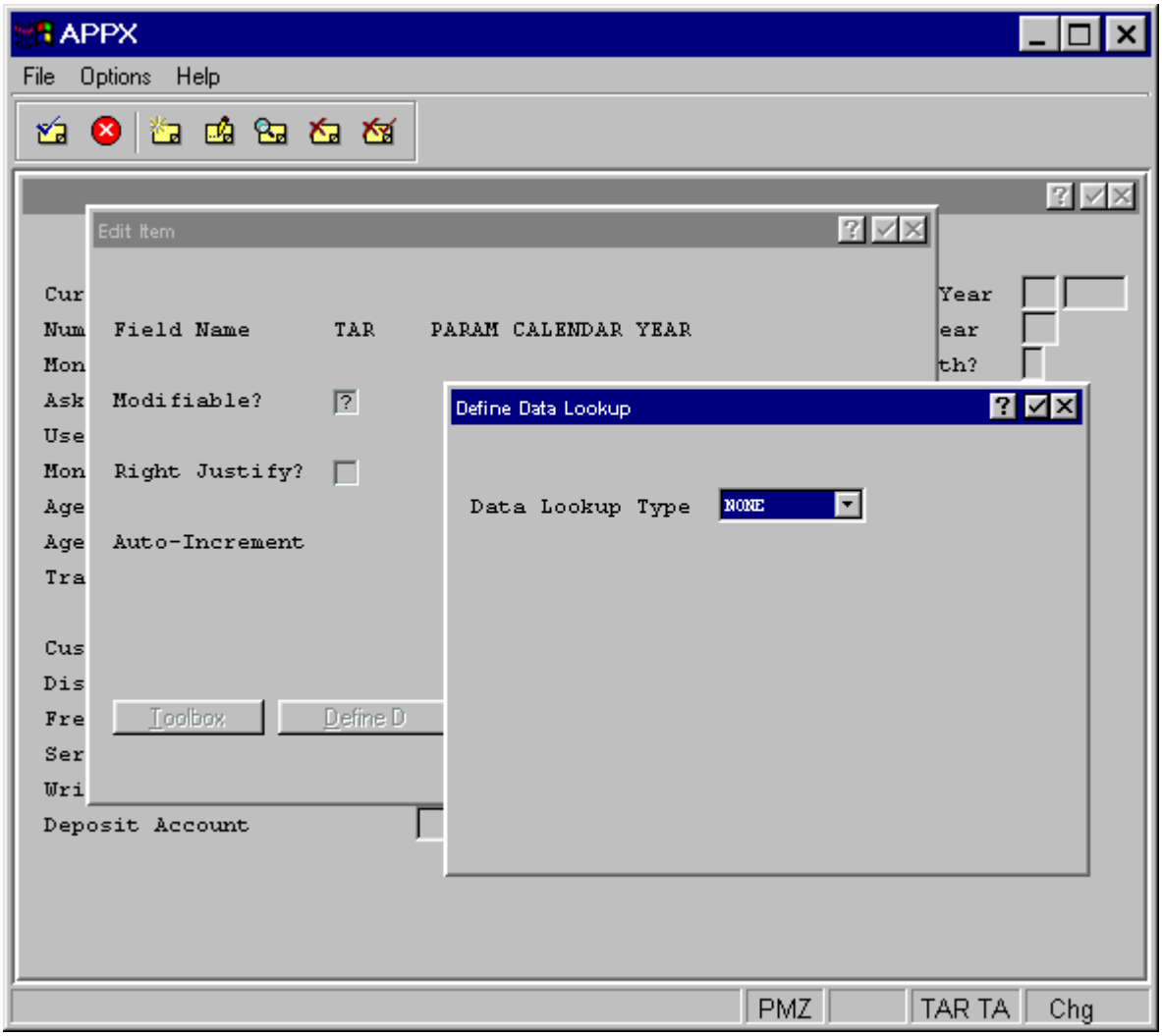


Figure 3-14-13. Define Data Lookup Overlay

The initial Define Data Lookup overlay contains one field. This field appears on all other Define Data Lookup overlays, as well.

- ***Data Lookup Type*** governs whether or not the value of this item is used as a key, or part of a key, to automatically look up a record in another file. If so, this screen is followed by a specification screen that defines the data lookup. The Data Lookup Type options are:
- **NONE.** There is no data lookup associated with this item.
- **DEFINE.** The value of this item establishes the first, or only, portion of the lookup key value.
- **CONTINUE.** The value of this item establishes the second (or other subsequent) portion of the lookup key value. It must be preceded by an item that is created with a value of Define.

The default value is *None*.

If you enter *Define* for the *Data Lookup Type*, the overlay shown in *Figure 3-14-14* appears.

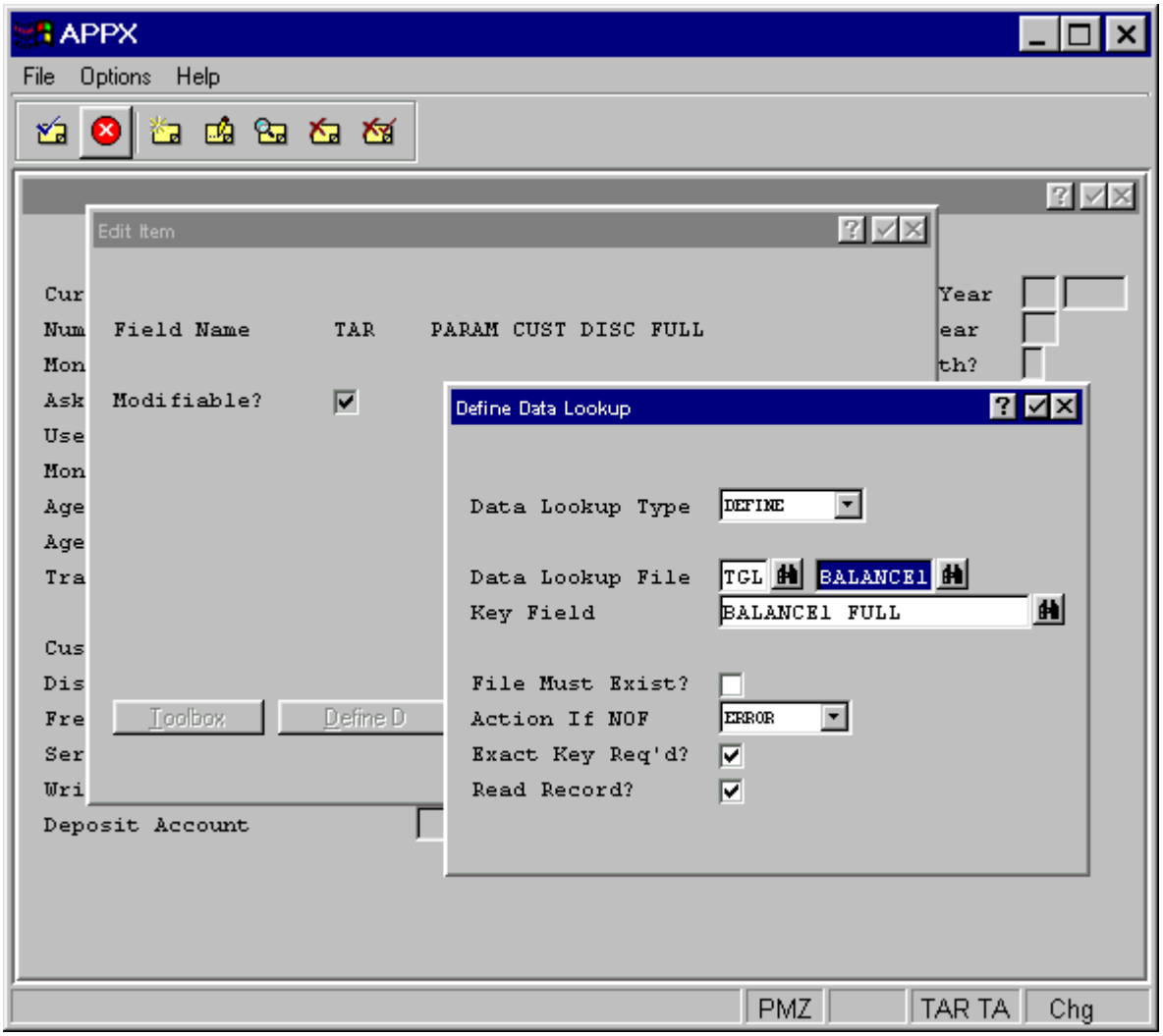


Figure 3-14-14. Define Data Lookup Overlay, Define

If you enter *Continue* for the *Data Lookup Type*, the overlay shown in *Figure 3-14-15* appears.

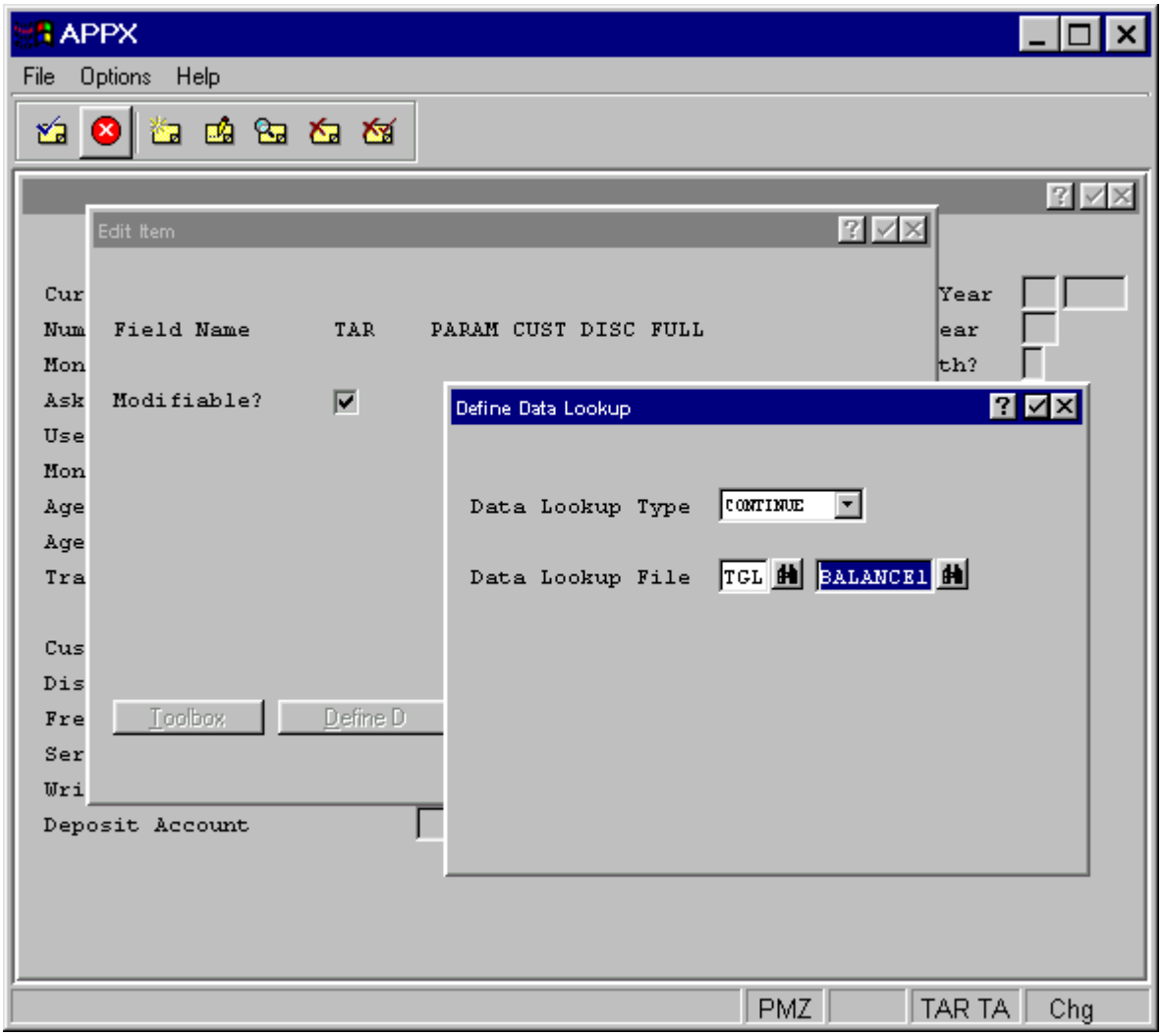


Figure 3-14-15. Define Data Lookup Overlay, Continue

The *Define Data Lookup* overlays for *Define* and *Continue* contain the following field in common:

Data Lookup File

Identifies the application and file that is to be read. The following additional fields only display for the *Define Data Lookup*, *Define* overlay.

The remaining fields appear on the *Define* overlay only:

Key Field

Determines the key field in the *Data Lookup File* that serves as the keypath for a lookup. This key field is identified by its *Field Name*.

File Must Exist?

Indicates whether or not the file that is entered in *Data Lookup File* above must exist. If "yes" ☑ and the file does not exist, APPX reports an error condition. If "no" ☐ and the file does not exist, the lookup is not performed. The default is "yes" ☑.

Action if NOF

Governs APPX's response if the referenced record is not on file (NOF) in the *Data Lookup File*. The *Action if NOF* options are:

- **NONE.** Takes no action with regard to the NOF condition.
- **WARNING.** Displays a warning message to the user.
- **ERROR.** Displays an error message to the user.

For output processes, APPX prints warnings and errors to the report file. They do not display on the screen. The default is *None*.

Exact Key Req'd?

Determines whether or not a user must enter a key value that exactly matches a value contained in the file defined for the lookup. If "no" ☐, the user can enter a partial key value, and APPX will search for the first record that matches the partial key entry. The user can also enter a complete key value and, if not on file, APPX will retrieve the record with the next highest key value. The default is "yes" ☑.

Read Record?

Governs whether or not the data lookup record is read into the current record area for this file. If "no" ☐, APPX performs the lookup and auto displays fields in the lookup record. However, the record read does not affect the current record area in memory, and is not available with statements. This is useful, for example, when you want to establish a lookup to the process control file but do not want to modify the current PCF record. The default is "yes" ☑.

Image Item Additional Attributes Option

The ADDITIONAL ATTRIBUTES option from the Image Item Specifications screen varies depending on the type of process and the type of field (alphanumeric, numeric, date, or logic). For output and inquiry processes, this option varies depending on whether or not it is an *End of Range* frame. The following sections show the various *Additional Attributes* overlays and describe each overlay attribute.

Alphanumeric Item Additional Attributes

The alphanumeric item Additional Attributes overlay, shown in *Figures 3-14-16, 3-14-17* and *3-14-18,* contain specifications that are unique to the appearance of an alphanumeric field painted as an item on an image.

The options that follow are a composite of those you see when defining alphanumeric items for frames in input, output, and inquiry processes. If an option only displays for a specific type of process, the definition indicates so.

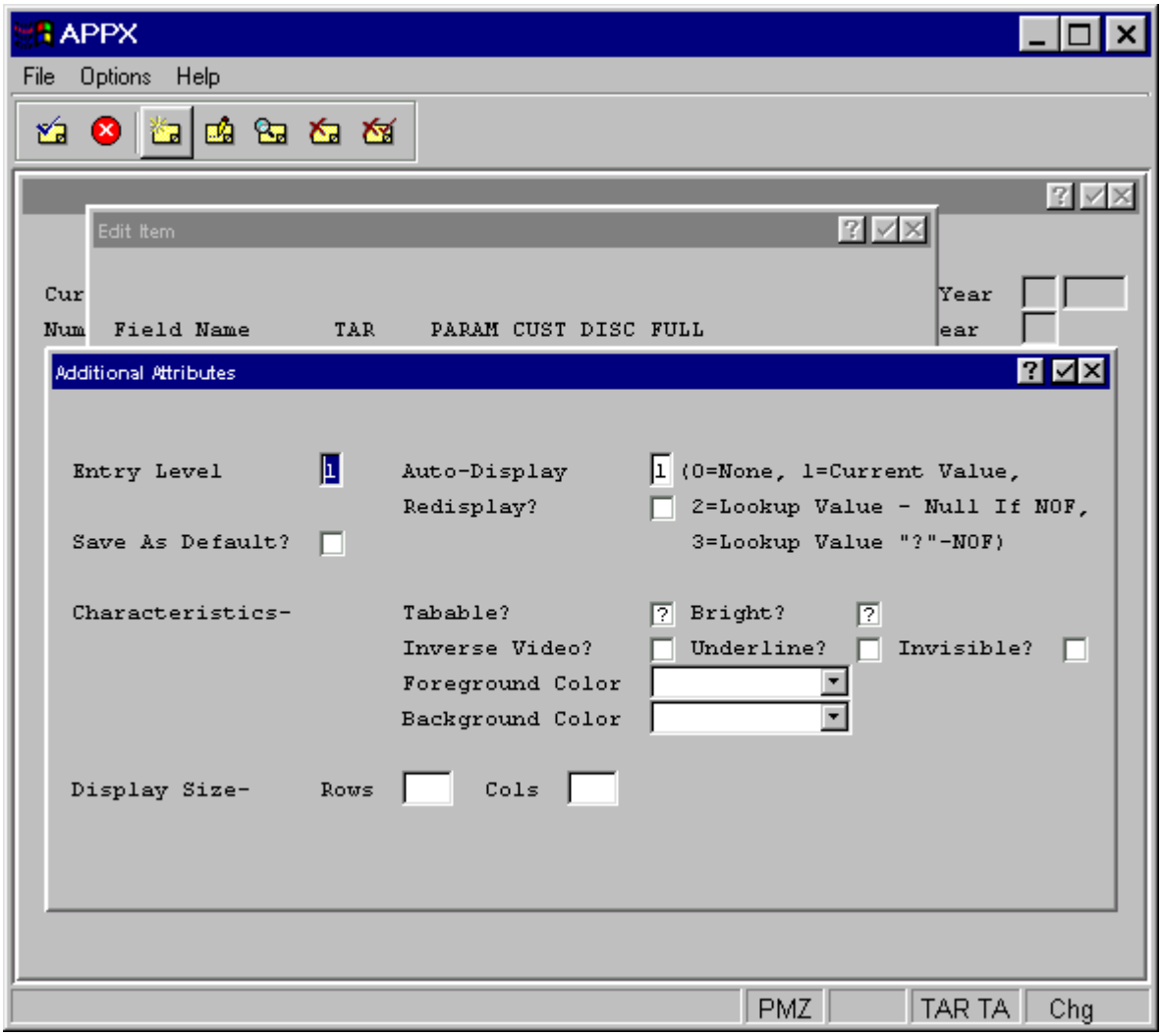


Figure 3-14-16. Alphanumeric Input Additional Attributes Overlay

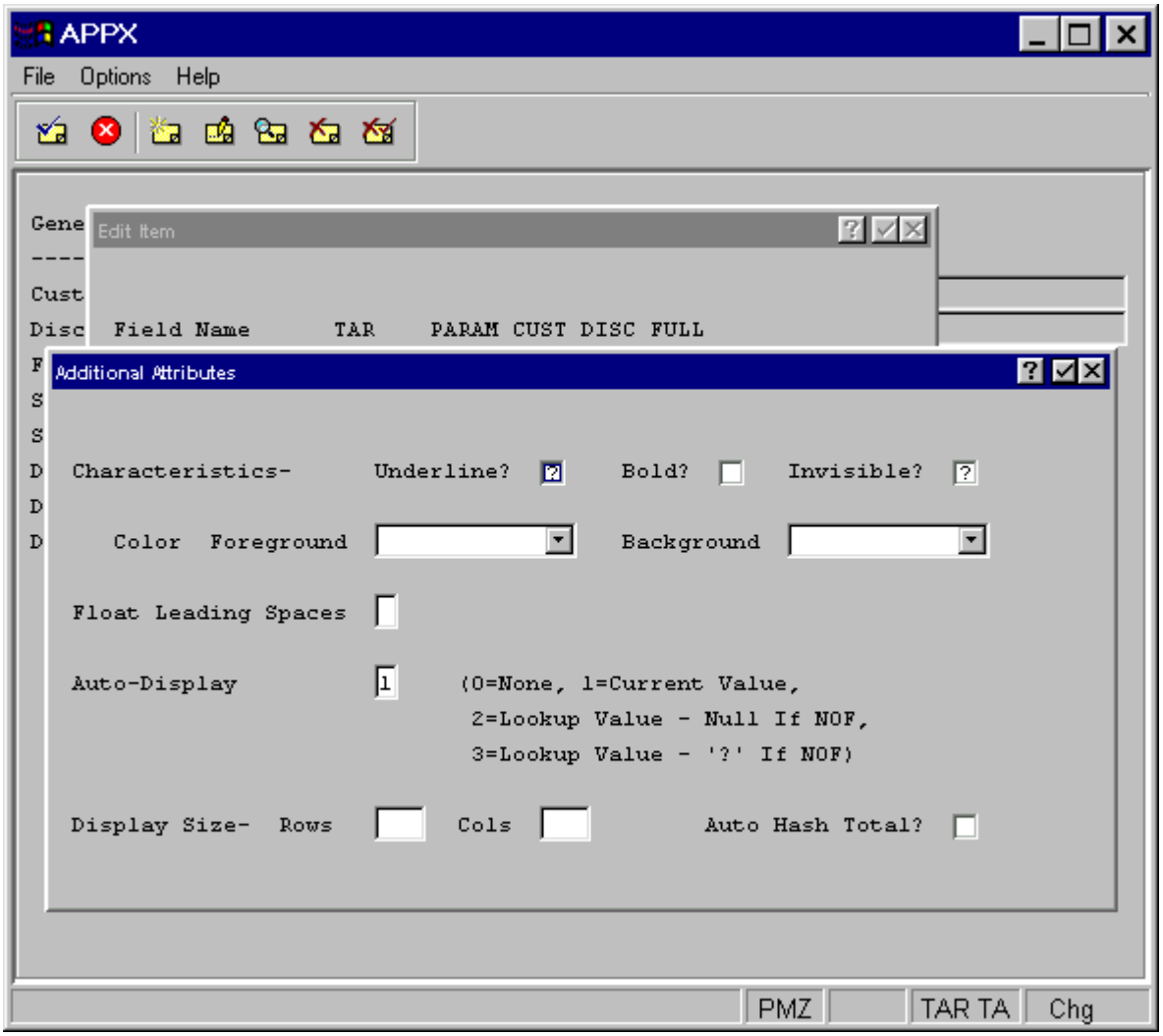


Figure 3-14-17. Alphanumeric Output Additional Attributes Overlay

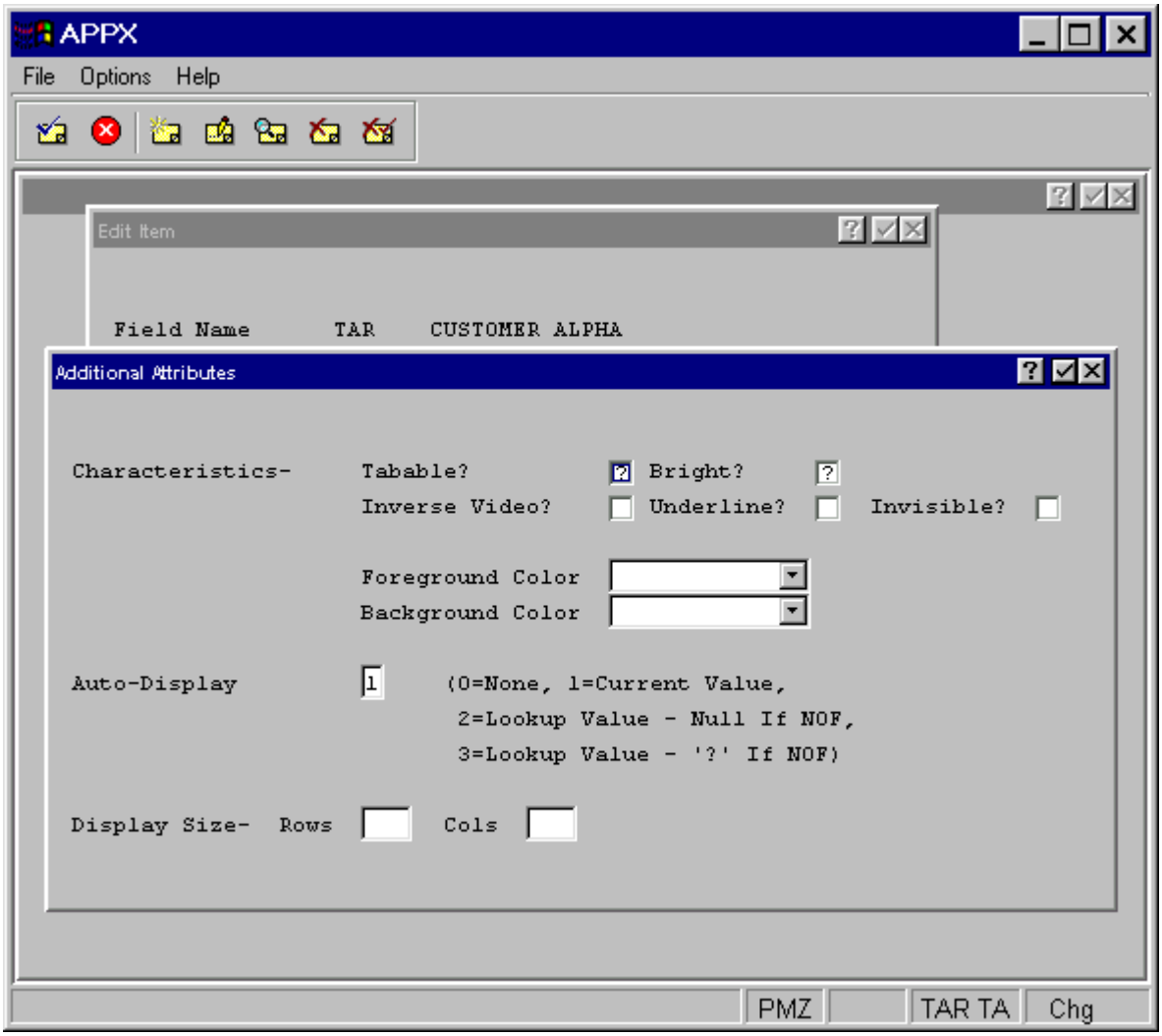


Figure 3-14-18. Alphanumeric Inquiry Key Entry Additional Attributes Overlay

Alphanumeric item Additional Attributes overlays contain some or all of the following fields, which may be common to date/time, numeric, and *End of Range* Additional Attributes overlays, as well.

Entry Level

For input processes, designates the entry level this item belongs to. If *Entry level* is *0* (and *Modifiable?* is "yes"), this field is always able to be edited, even during key entry. If nonzero, the user first accesses the field in the data entry level that is represented by this nonzero value.

Auto-Display

Indicates whether or not the value in this item automatically displays and, if so, both the source of the value and the action to be taken if the information is not on file. The *Auto-Display* options are:

- **0 = None.** The value for this item will not automatically display. Its appearance is explicitly controlled by the designer with statements.
- **1 = Current Value.** Displays the value of the field in the record currently in memory.

- **2 = Lookup Value - Null If NOF.** Displays the value of the field in the record that is defined in the data lookup. If the information is not on file, the field displays blanks.
- **3 = Lookup Value - '?' If NOF.** Displays the value of the field in the record that is defined in the data lookup. If the information is not on file, the field displays question marks (???).

The default is *1*.

Redisplay

Indicates whether or not the value in this item should be redisplayed when it changes. If a user enters a partial value (as with *token* fields), the complete value is displayed before allowing the user to continue with data entry. Note that if the field is set with ILF code, *Redisplay* will not cause the screen to redisplay and it is necessary to execute a *Display* statement to cause a redisplay of the screen.

Save As Default?

For input processes, establishes the default value for the *Set Entry Attributes* option. If "yes" ☑, a value entered in this item in add mode serves as a default value for the next record that is added during the session. The default is "no" ☐. Refer to the *User Manual* for a discussion of the *Set Entry Attributes* option.

- ***Characteristics.*** The alphanumeric Additional Attributes overlays contain some or all of the following *Characteristics* fields.

Tabable?

For input process and inquiry Key Entry images, indicates whether or not a user can automatically move the cursor to the first character in this item from the preceding item by pressing TAB. If blank, the item is tabable if modifiable, and non-tabable if not modifiable. The default is "unknown" but can be toggled to "yes" or "no" by clicking the checkbox.

Bright?

For menu and input processes, and inquiry Key Entry images, designates whether or not the appearance of this item on the screen is highlighted. If blank, the entry is bright if modifiable, and dim if not modifiable. The default is "unknown" but can be toggled to "yes" or "no" by clicking the checkbox.

Bold?

For output and inquiry processes, designates whether or not the appearance of this item on the report is bolded. The default is "unknown" but can be toggled to "yes" or "no" by clicking the checkbox.

Inverse Video?

For menu and input processes, and inquiry Key Entry images, designates whether or not the background and foreground colors as displayed on the screen are reversed for this item only. The default is "no" ☐.

Underline?

Governs whether or not APPX adds a single underline below the value displayed for this item when the item appears on the screen or in hard copy. The default is “no” ☐.

Invisible?

For menu and input processes, indicates whether or not the appearance of this item is hidden from the user. The default is “no” ☐.

Foreground Color

Indicates what color the value of this field should be displayed in. Colors can be selected from a drop-down list. Leave this field blank to use the default foreground color.

Background Color

Indicates what color the background of this field box should be displayed in. Colors can be selected from a drop-down list. Leave this field blank to use the default background color.

Float Leading Spaces

For output and inquiry processes, controls positioning of the item. If nonblank, the position of the item is not absolute. It can move to the left, based on the position of the last character in the preceding item and the number you enter in this specification. An entry of *0* through *9* represents the number of spaces that must appear between this item and the last character of the preceding item. If blank, the item does not float. The default is blank.

Display Size - Rows/Cols

Determines the number of rows, and the number of columns in each row, within which to display an item. The total number of characters that are defined by these specifications can be more or less that the actual length of a field in the data dictionary. If either one of these specifications is blank, APPX computes its value by dividing the value of the other specification into the item's length as defined in the data dictionary. For example, if you specify 30 columns for a 120-character field, APPX assumes a row entry of *4*.

- If both specifications are blank, APPX determines the display format with reference to the field's data dictionary length and the current cursor position on the image. A column specification equal to the available positions between the current cursor position and the far right-hand position on the image is assumed. A row specification is calculated based upon the field length and the assumed column specification. If the field length is smaller that the available positions on the screen or output, the item displays as a single row. For nonblank specifications, a change to the field length in the data dictionary has no effect on the *Display Size*.

Auto Hash Total?

For output processes and inquiry except for key entry, governs whether or not values in the alphanumeric field are to be hash-totaled. Nonnumeric characters are treated as *0*, except for trailing spaces. Hash totals ignore overflow. For example, if your item is five characters in length, adding one to 99999 results in a value of 00000. The default is “no” ☐.

Date/Time Item Additional Attributes

The date/time item Additional Attributes overlays, shown in *Figures 3-14-19, 3-14-20,* and *3-14-21,* contain specifications that are unique to the appearance of a date/time field painted as an input, output, or inquiry *Key Entry* image. This overlay displays when you select the Additional Attributes option from the Item definition screen.

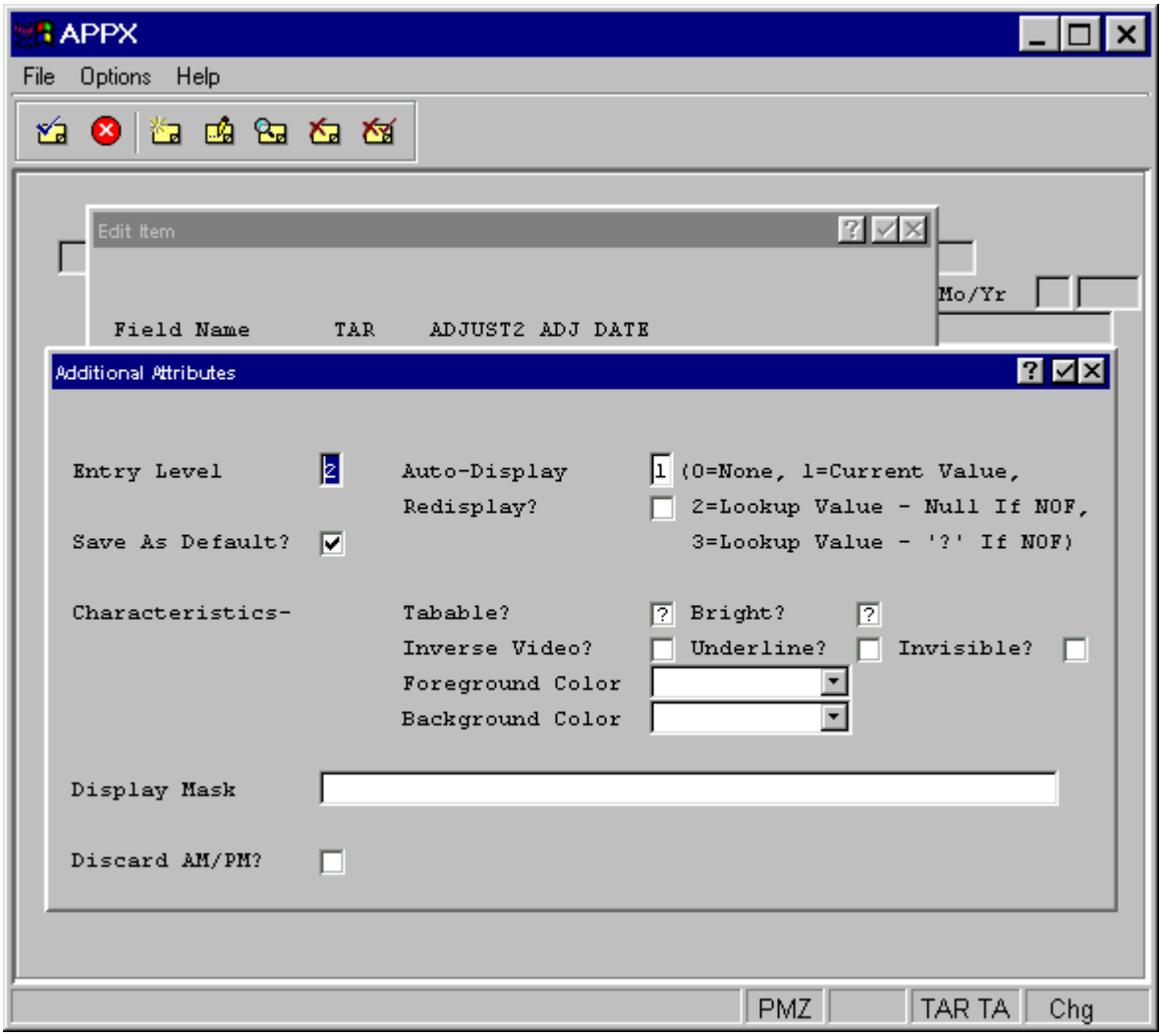


Figure 3-14-19. Date/Time Input Additional Attributes Overlay

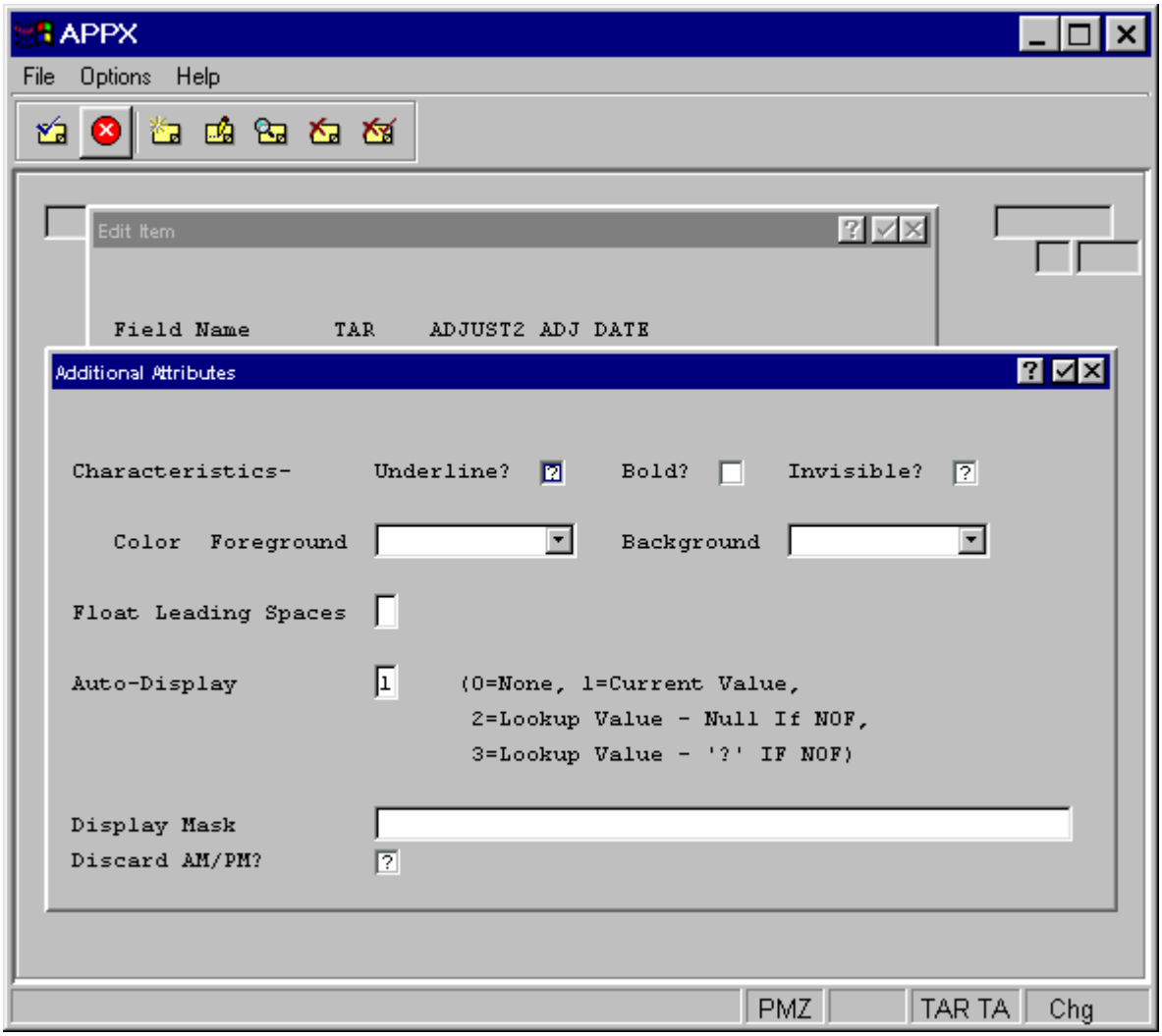


Figure 3-14-20. Date/Time Output Additional Attributes Overlay

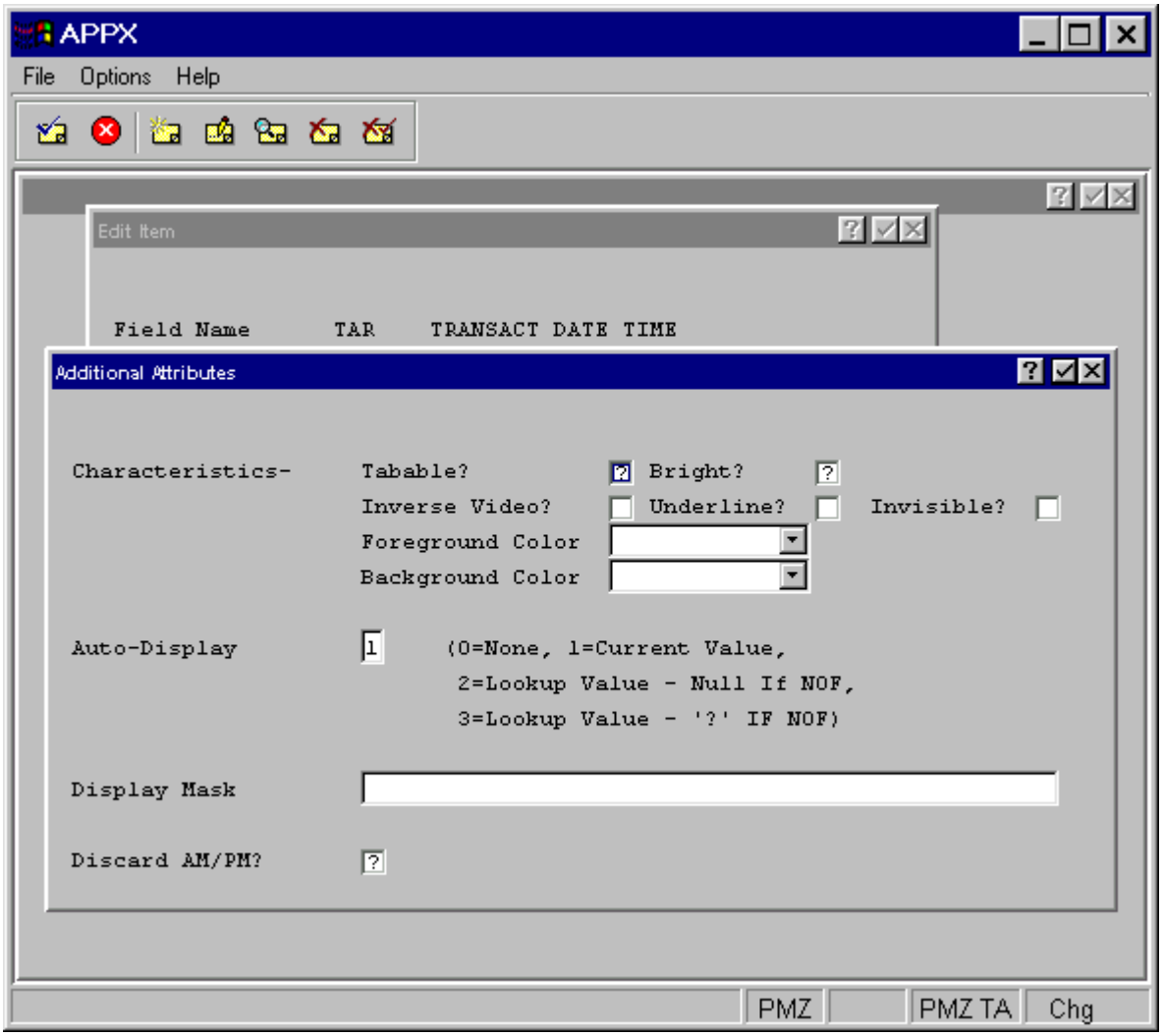


Figure 3-14-21. Date/Time Inquiry Key Entry Additional Attributes Overlay

The date/time item Additional Attributes overlays contain some of the same fields that the alphanumeric item Additional Attributes overlays do, as well as the following date/time-specific fields. These fields may also be common to the numeric Additional Attributes overlays.

Display Mask

Determines which parts included in the data dictionary *Store/thru* specification display on the image. See the Display Mask (Date/Time) section for details.

Discard AM/PM?

Establishes whether or not the AM or PM portion of a date/time mask that is specified in system administration should be discarded. It applies only to date/time fields that include an hour component. If the value in the date/time field represents an elapsed time, set this specification to “yes” ☑. If the value represents a specific point in time, set the specification to “no” ☐. The default is “no” ☐.

Display Mask (Date/Time)

Display Mask for a date/time item determines which parts included in the data dictionary *Store/thru* specification (month, year, or hour, for example) display on an image. This entry can include more or less parts than the data dictionary, and can be presented in a different order. If more parts are included, those parts that are not present in the data dictionary are discarded from the mask before presentation to a user. Each position in a mask may contain a part placeholder, a pad character, a special character, text, or a separator character.

- ***Part placeholder*** designates the parts of a date that are physically stored. Valid options are:

88	Century
77	Year
66	Month
55	Day
44	Hour
33	Minute
22	Second
11	Tick (hundredth of a second)

- ***Pad character*** of *0* to force the display of a leading *0* in any part that contains a single-digit value. For example, 04 could be used to indicate that you wish to store hours and wish for a leading zero to be printed whenever the hours are less than 10.
- ***Special Characters***
- ‘ ’ or “ ” (constant enclosed) — Any characters enclosed within quotation marks, including special characters, are considered text and display on an image exactly as they appear on a mask.
- \ (constant next character) — Any single character immediately following the back-slash, including a special character, is considered text and displays on an image exactly as it appears on a mask.
- ~ (reserved for APPX’s internal use) — This character should never be used, unless enclosed in quotation marks or immediately preceded by a back-slash.
- ***Text*** including, but not limited to, information in the categories listed below can be inserted anywhere on a mask.
- Month names, such as January or Jan (the first three characters of a month)
- Day names, such as Monday or Mon (the first three characters of the day)
- AM/PM or am/pm designations
- Punctuation, such as , or .
- Numeric suffixes, such as th or rd
- Literals enclosed in “ ”
- ***Separator characters***, such as :, /, or -

APPX automatically translates month and day names based on the actual month and day stored in the field. This text translation is case-sensitive. For example, if you enter *MONDAY* on a mask and the stored data is actually *Wednesday*, APPX translates it as *WEDNESDAY*, rather than *Wednesday*.

The text defined on a date/time display mask can include any characters with the exception of the special characters: quotation marks (" "), back-slashes (\), and tilde marks (~). To include any of these characters as text on a mask, each special character must be preceded by a back-slash (\). Special characters do not occupy space when a field displays on an image.

To include descriptions for a day of the week or month of the year on a display mask, simply enter any day or month description. APPX translates your entry into the appropriate day or month description based upon the data to be displayed. For example, a display mask defined as *Monday, January 05, 0807*, displays any of the following, depending on the data in the field: *Sunday, January 1, 1989*, or *Thursday, April 6, 1989*, or *Saturday, August 18, 1990*. The display expands or contracts as necessary to accommodate varying lengths of the possible day and month descriptions. (To avoid translation of *Monday*, for example, you must enclose the text in quotation marks: *"Monday"*.)

APPX's description translations are case-sensitive (APPX maintains the capitalization that is defined on a mask), and are also sensitive to abbreviations. If you enter *Monday* on a mask, APPX translates as *Tuesday* or *Wednesday* or *Thursday*, etc. If you enter *Mon* on a mask, APPX translates as *Tue* or *Wed* or *Thu*, etc. (The short version of a month or day name is always represented by the first three characters in the name.)

If you wish to display time information on a 12-hour rather than a 24-hour basis, you can include an *am* or *pm* designation anywhere on a mask. APPX translates the characters according to the stored data. If a mask includes *am* and the data is *3:00* in the afternoon, APPX substitutes *pm* for *am*. These translations are also case-sensitive (*am* versus *AM*, for example).

It is possible to encounter data with less or more digits than what is defined on a mask. If the data has more characters than defined on a display mask, the screen or printed output displays only the characters that are defined on the mask.

If the display mask defines more parts of a date or time than the *Store thru* parameter, any parts not stored are ignored when the mask is displayed.

APPX's date/time masking facility is designed to provide maximum flexibility. Any mask format, if not specifically excluded below, is permitted.

- All part placeholders must be entered as two characters.
- An am/pm designation is meaningful only in a mask that contains an hour placeholder. Otherwise, it is treated as text.
- Masks cannot have noncontiguous part placeholders, such as a year and an hour placeholder. This should not be interpreted to place restrictions on the order that the parts display in the mask.
- Month names or day names are only valid in masks that include sufficient part placeholders to identify a specific month or day within a specific year.

Logic Item Additional Attributes

There are no additional specifications to enter for a logic item. - Yes, there are!

Numeric Item Additional Attributes

The numeric item Additional Attributes overlays, shown in *Figures 3-14-22, 3-14-23,* and *3-14-24*, contain specifications that are unique to the appearance of a numeric field painted as an item on an image. The specifications that follow are a composite of those you see when defining a numeric input, output, or inquiry item. If a specification only displays for a specific type of process, the definition indicates so.

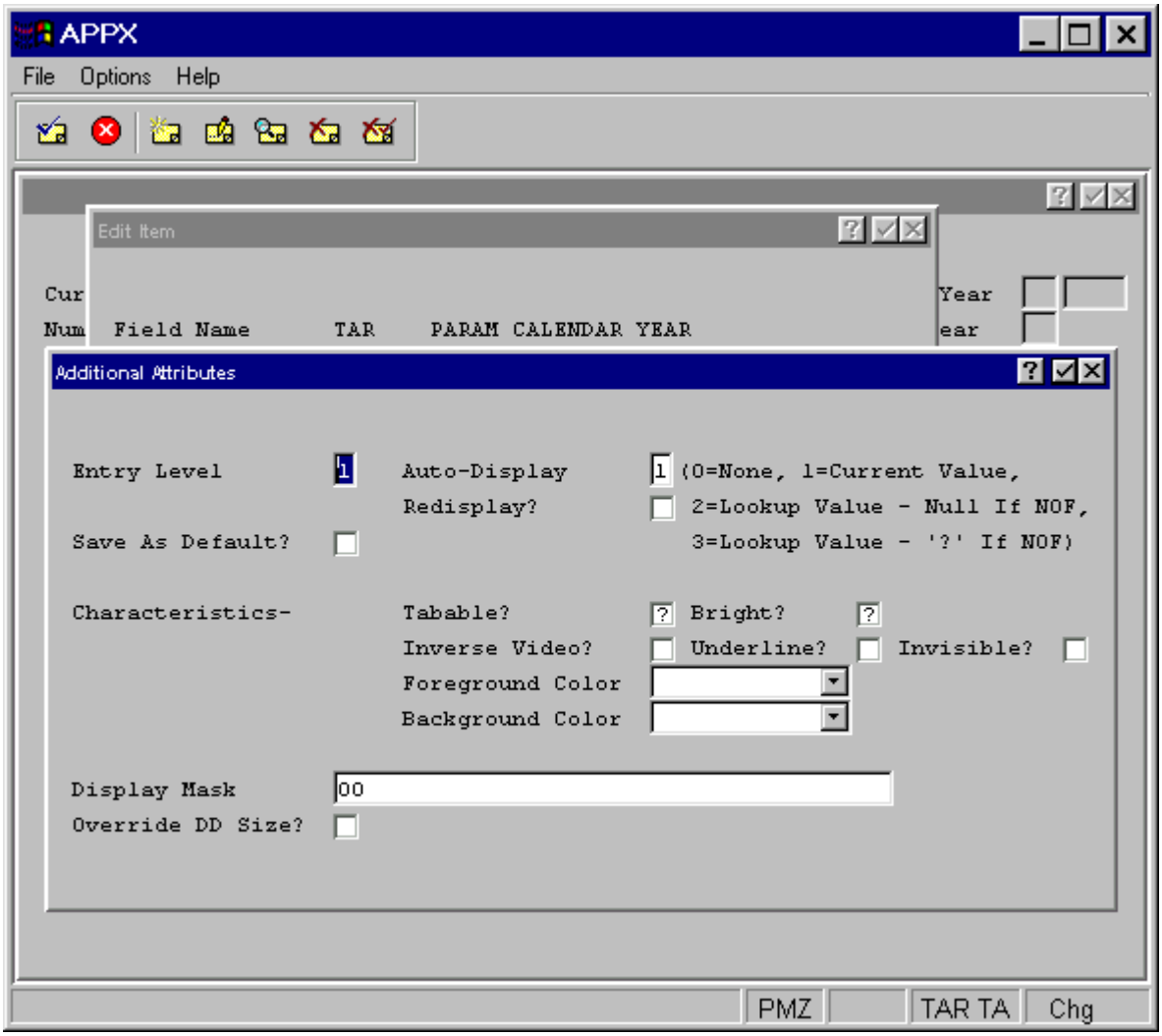


Figure 3-14-22. Numeric Input Additional Attributes Overlay

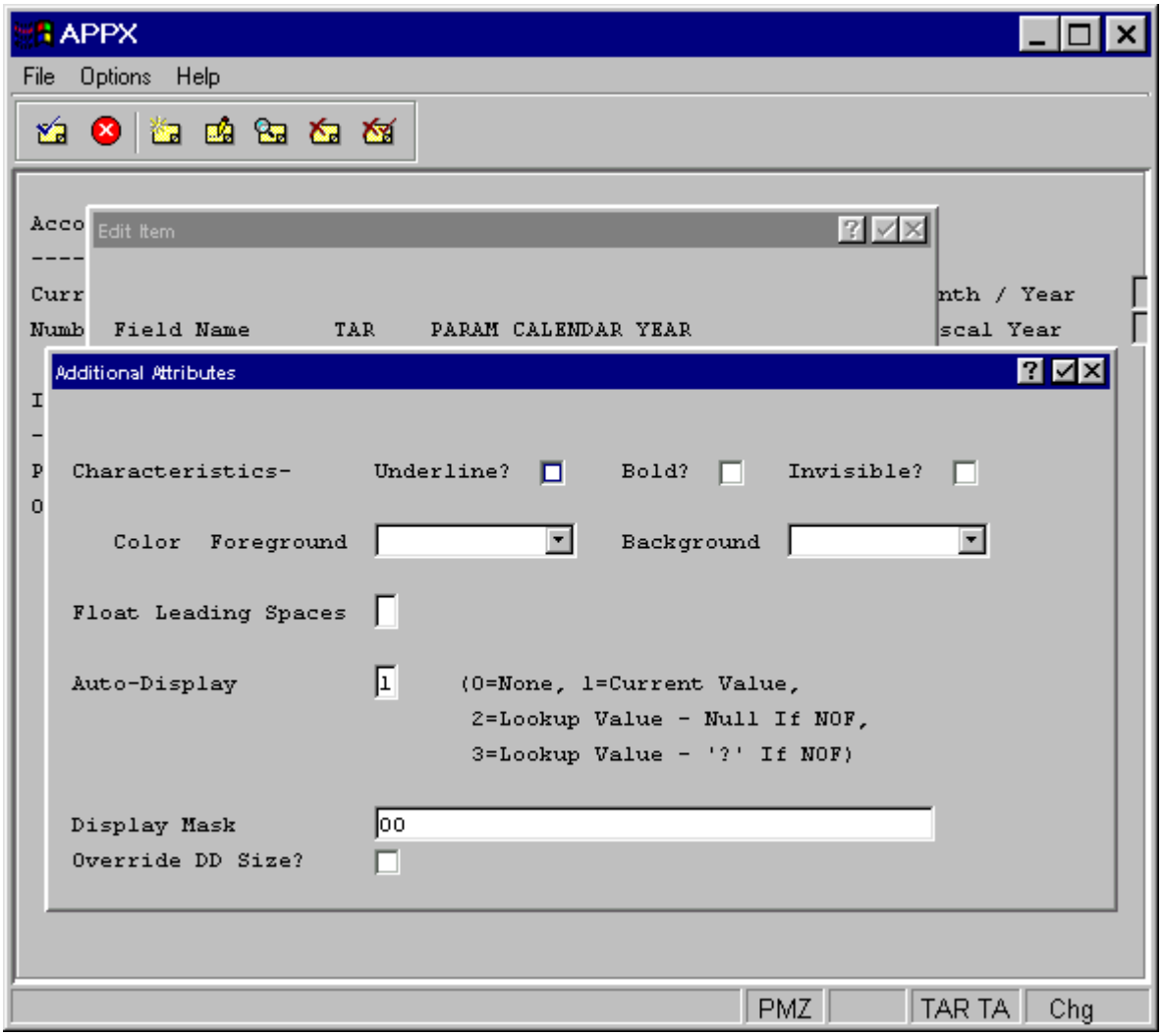


Figure 3-14-23. Numeric Output Additional Attributes Overlay

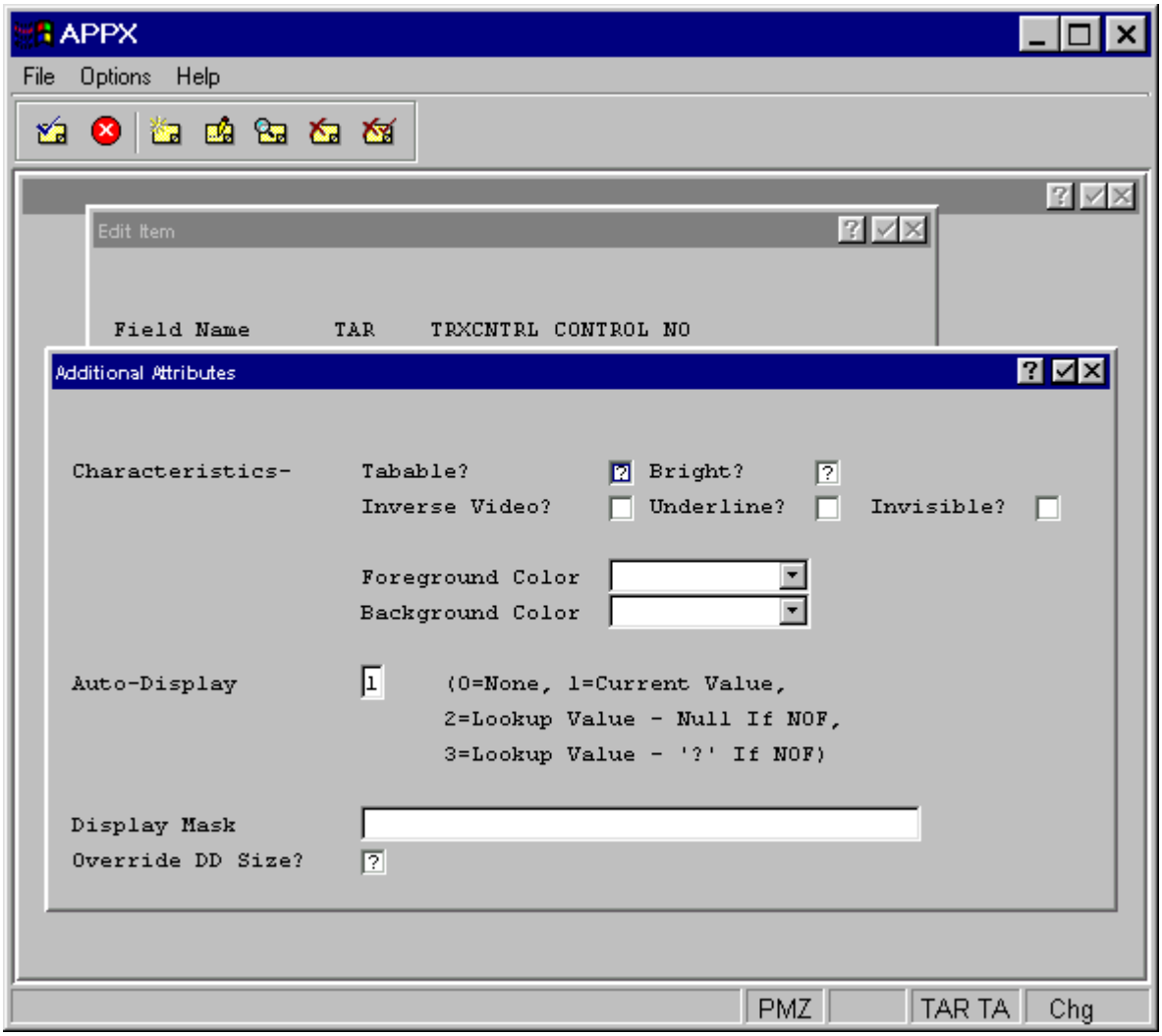


Figure 3-14-24. Numeric Inquiry Key Entry Additional Attributes Overlay

The numeric item Additional Attributes overlays contain some of the same fields that the alphanumeric and date/time item Additional Attributes overlays do, as well as the following numeric-specific fields.

- *Auto-Display* indicates whether or not the value in this item automatically displayed and, if so, both the source of the value and the action to be taken if the information is not on file. The options are:
- **0 = None.** The value for this item will not automatically display. Its appearance is explicitly controlled by the designer with statements.
- **1 = Current Value.** Displays the value of the field in the record currently in memory.
- **2 = Lookup Value - Null if NOF.** *D*isplays the value of the field in the record that is defined in the data lookup. If the information is not on file, the field displays blanks.
- **3 = Lookup Value '?' if NOF.** *D*isplays the value of the field in the record that is defined in the data lookup. If the information is not on file, the field displays question marks (???).

The following option only displays for End of Range frames in an output process.

- *4 = Range Total Value.* Displays the sum of the values in this field for each record in the range.

The default is *0 = None.* Refer to *Automatic Data Lookup and Display* and *Data Lookup Specifications* for additional information.

Display Mask

Designates the format that data should display in to a user, such as on screens and reports. Refer to *Chapter 2-3* for a complete explanation of the various types of numeric display masks.

Override DD Size?

Select “yes” ☑ if you want the defined display mask to override the field size entered in the data dictionary for this input item. This mask can be either larger or smaller than the actual field size. The default is “no” ☐.

- ***Re-Display.*** A “yes” ☑ in this field instructs APPX to display the new value in a field if that value has changed. If a user enters a partial value, APPX will display the value before allowing the user to continue with data entry.

Defining Image Characteristics

In addition to defining display attributes for items on images, APPX permits you to establish display attributes (*characteristics*) for various regions of an image. Such characteristics include bright and inverse video for menu and input images, bold and expanded print for output and inquiry images, as well as underlining and foreground/background colors for all images. Of course, more sophisticated graphical displays are also possible by using the GUI features of the Image Editor described in Chapter 3-15.

To define the characteristics of a region on an image (or edit an existing region), position the cursor at the starting row/column and, using the keyboard, select `OPTION` `13`. APPX prompts you to position the cursor at the ending row/column and press `RETURN`. This establishes the size of the region. APPX then displays a Define Characteristics overlay. *Figure 3-14-25* below shows the overlay displayed for a menu or input image while *Figure 3-14-26 shows the overlay presented for an output or inquiry image.* You can select the characteristics you want to apply by toggling the appropriate checkboxes between "yes" ☑ and "no" ☐ values. To set foreground and/or background colors, you can pick a color from the drop-down lists of available choices. When you have made your selections, press `RETURN` or select `ENTER` to apply the changes and return to the Image Editor.

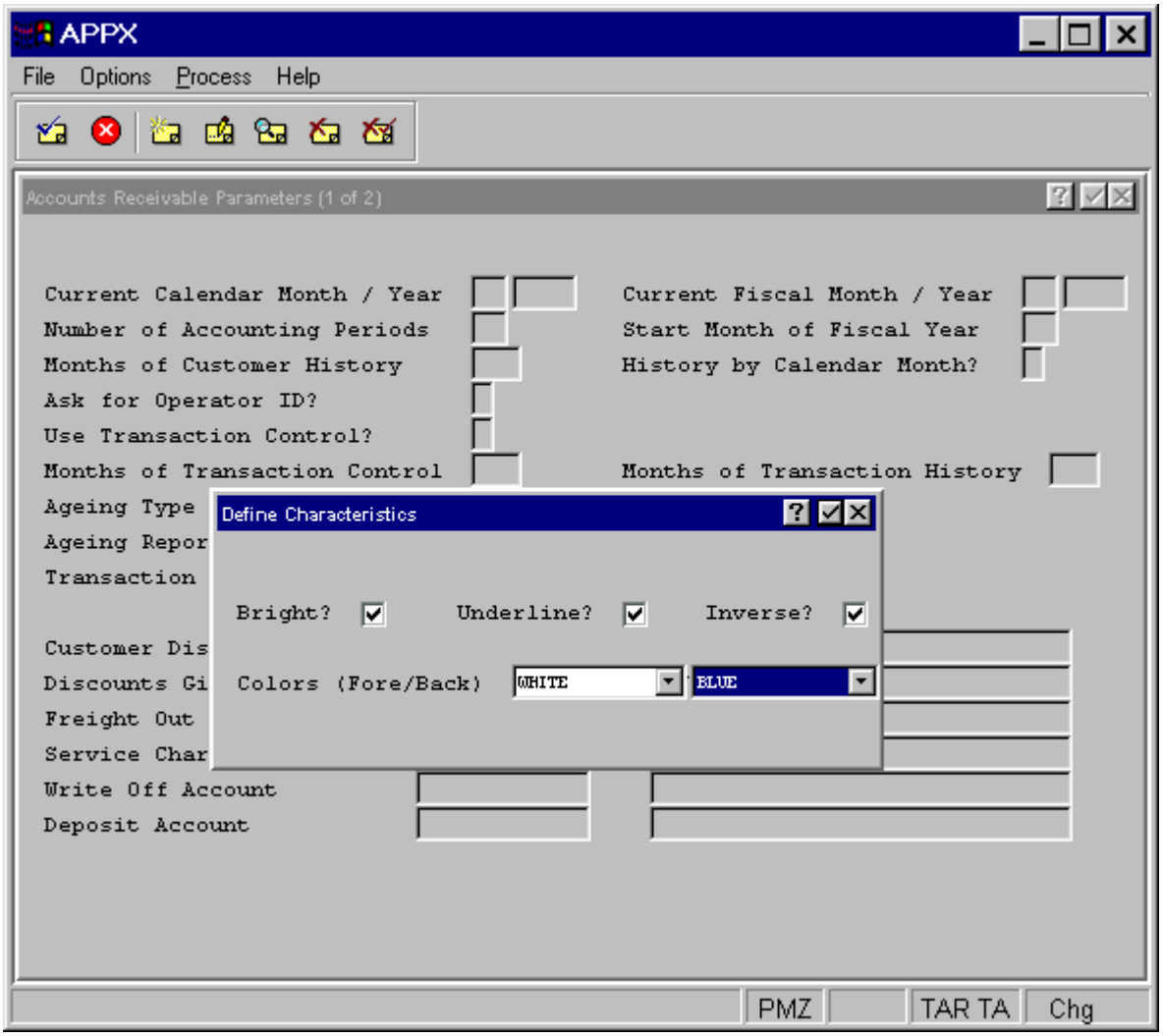


Figure 3-14-25. Define Characteristics Overlay for Menus and Input Images

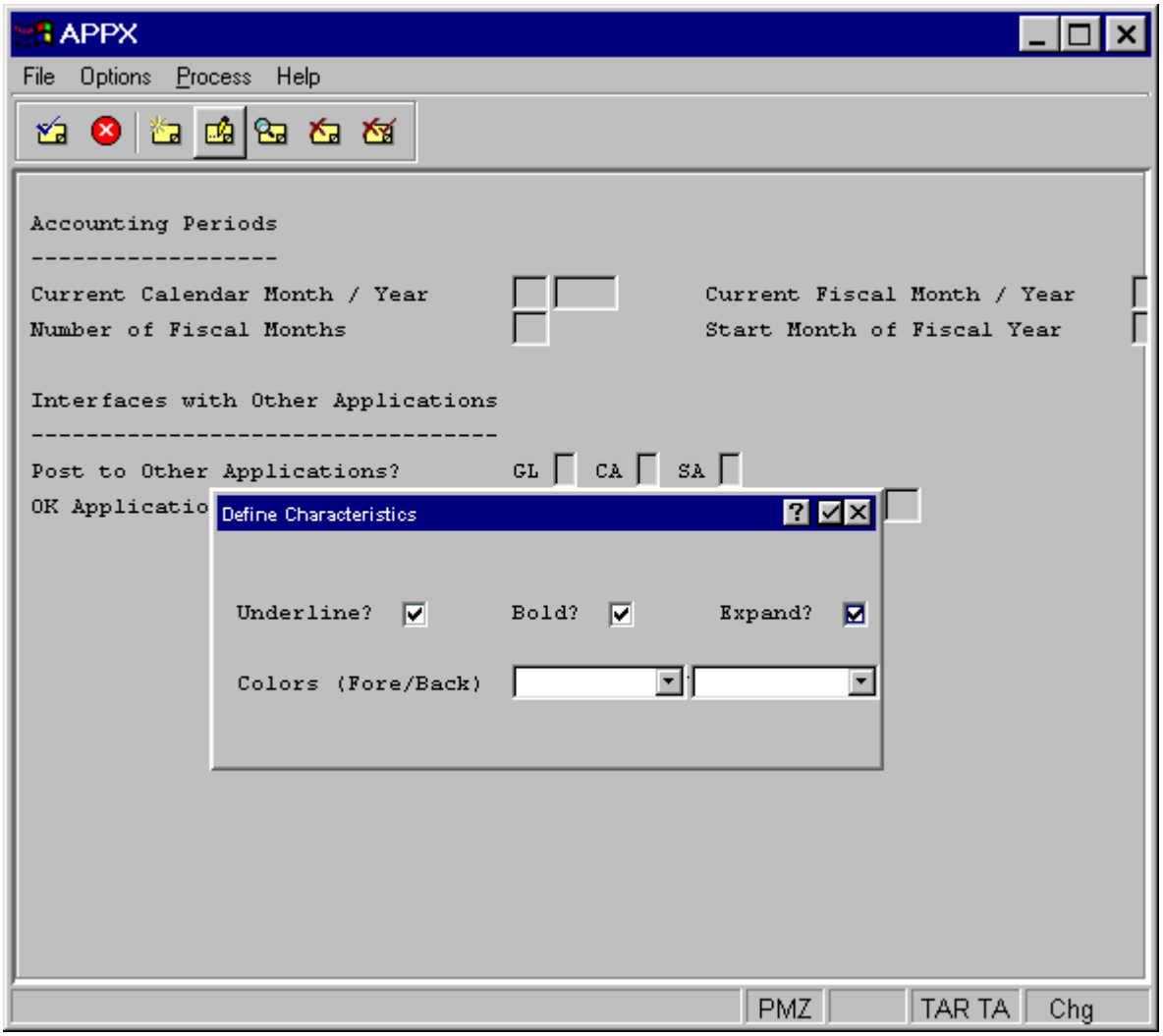


Figure 3-14-26. Define Characteristics Overlay for Output and Inquiry Images

The specifications that follow are a composite of those you see on the respective Define Characteristics overlays. If a specification only displays for a specific type of image, the definition indicates so.

Bright?

For menu and input images, determines whether or not text within the region appears bright on the screen.

Underline?

Governs whether or not text within the region appears with an underline.

Inverse?

For menu and input images, determines whether or not text within the region appears with its light and dark areas reversed on the screen.

Bold?

For output and inquiry images, indicates whether or not text within the region appears bold in the output.

Expand?

For output and inquiry images, controls whether or not text prints in expanded format.

Foreground Color

Indicates the color the field value is displayed in. Select from a drop-down list of colors. Leave this field blank to use the default foreground color.

Background Color

Indicates the color the field box is displayed in. Select from a drop-down list of colors. Leave this field blank to use the default background color.

The default for these fields is blank.

CHAPTER 3-15

Using GUI Features of the Image Editor

GUI Features of the Image Editor

The Image Editor allows you to create menu, input, output, and inquiry images containing a variety of graphical components for use in GUI-enabled environments. Such images are designed using the mouse as the primary design tool instead of the keyboard. One of the Image Editor's most important GUI features is the APPX *Data Palette* which allows you to interactively view the dictionary of database components, select them, and "drag" them onto your screen image.

The APPX Image Editor works much like many other Windows-based screen design tools. You can make selections, move and resize items, and perform cut/copy/paste operations in a similar manner. Tools for many of the GUI features can be accessed from the Image Editor toolbar discussed in the next section. Graphical components can be added to menu, input, output, and inquiry processes at the frame, image, or field level. These components include wallpapers, titlebars, menu bar items, toolbar items, icons, and other features that are not part of the actual "image" you paint.

Figure 3-15-1 below shows an input image using GUI features being edited in the Image Editor. See Figure 3-14-2 in Chapter 3-14 for a similar input image using only standard features being edited in the Image Editor.

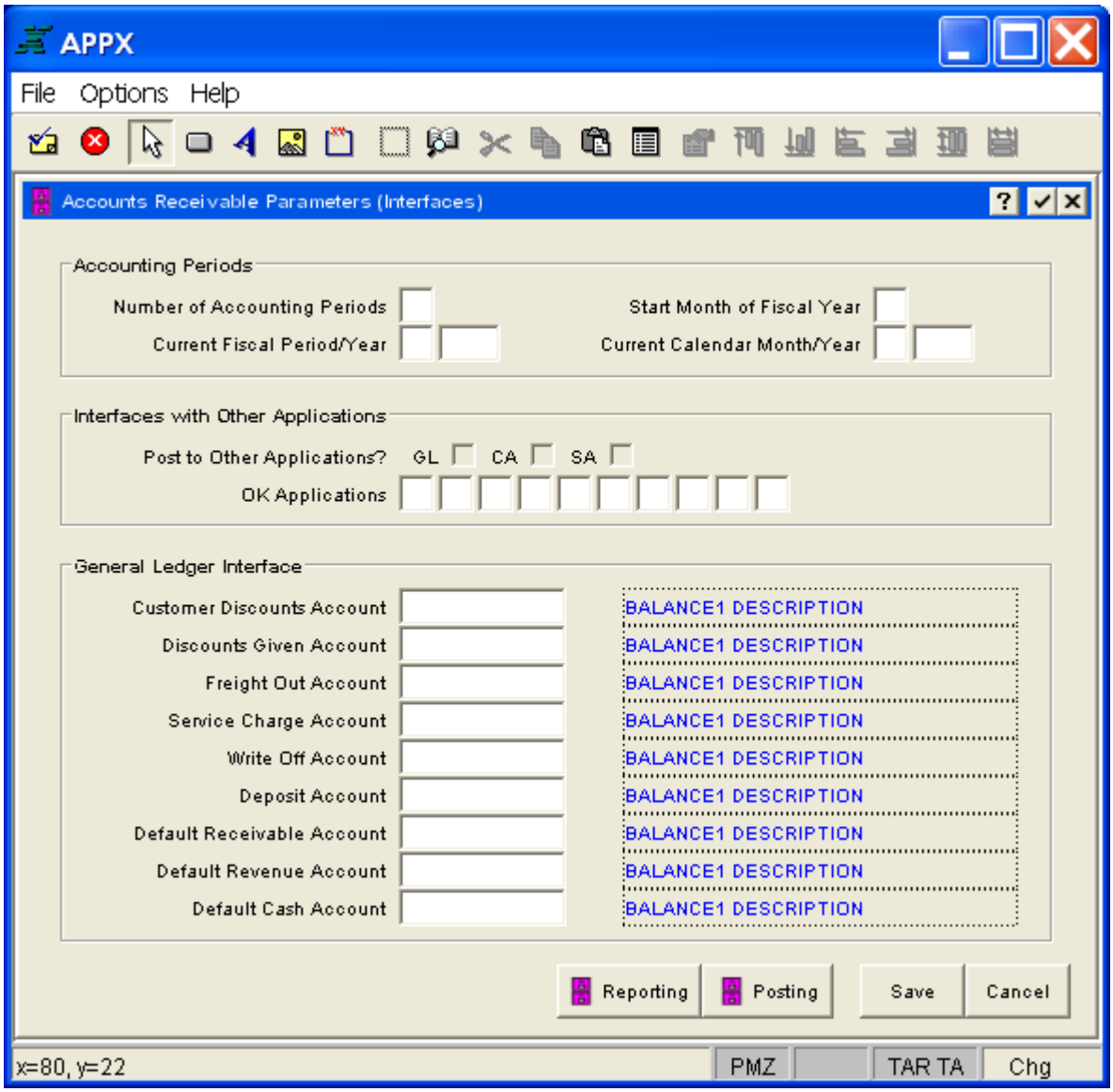


Figure 3-15-1. Input Image Using GUI Features of the Image Editor

Image Editor Toolbar

The Image Editor toolbar contains many design tools not found on the standard APPX runtime toolbar. These tools allow the designer to easily perform a variety of image design functions.

The items on the toolbar are grouped in four categories:

APPX Options

- SAVE AND EXIT to save the image and exit the image editor.
- CANCEL to discard the changes made to the image and exit the image editor.

Paint Object Tools

- SELECT A REGION defines the region of the image you want to perform your design work in.
- PAINT A BUTTON is used to place a button on a menu or input image and define its properties.
- PAINT A LABEL is used to place a label (text) on the image and define its properties.
- PAINT A PICTURE is used to place a picture (jpg or gif file, for example) on the image and define its properties.
- PAINT A BOX OR LINE is used to place a box or line on the image and define its properties.
- SHOW OBJECT BOUNDS is used to display the boundaries of an object, useful for objects with no borders.

For more information regarding the Paint Object Tools, see Painting Buttons, Labels, Pictures, and Boxes or Lines.

Data Palette, Cut/Copy/Paste, and Edit Tools

- DATA PALETTE opens the APPX Data Palette and allows you to select and add data objects to the image from the Data Dictionary.
- CUT SELECTION is used to remove a selection from the image. It becomes available to be "pasted" elsewhere on the image, if desired.
- COPY SELECTION is used to copy a selection from the image so that it can be "pasted" or duplicated elsewhere.
- PASTE SELECTION is used to place a previously "cut" or "copied" selection elsewhere on the image. Note that this icon cannot be selected until a selection has been "cut" or "copied".
- EDIT WINDOW PROPERTIES is used to edit properties of the current image or global properties of all images for the current frame.
- EDIT OBJECT PROPERTIES is used to edit the properties of a previously defined and selected button, label, picture, box, line, or item.

Alignment and Sizing Icons

- ALIGN TOPS aligns the tops of two or more selected objects.
- ALIGN BOTTOMS aligns the bottoms of two or more selected objects.
- ALIGN LEFT SIDES aligns the left sides of two or more selected objects.
- ALIGN RIGHT SIDES aligns the right sides of two or more selected objects.

- MAKE SAME SIZE (VERTICALLY) makes two or more selected objects the same height.
- MAKE SAME SIZE (HORIZONTALLY) makes two or more selected objects the same width.

For details on selecting objects, see Making Selections.

Making Selections

The following rules apply when using the mouse to select individual objects or groups of objects on an image:

- Click an object to select it and clear any previous selections.
- CTRL-click an object to select it and retain any previous selections. This is required when selecting two or more objects to align or size. When two or more objects are selected in this manner, the selection border around the last item selected will be blue. All other selected items will have a selection border that is red. The item with the blue selection border is used by the alignment and sizing tools to determine how to align or size the other selected items.
- Drag a rectangle around any combination of the above to select all objects within the rectangle. This will also clear any previous selections.

Using Guide Bars

Guide bars are used by the APPX designer to align objects on an image. They can be created and moved using these instructions:

- Use the right-click menu (see the Image Editor Right-Click Menu) to create vertical and horizontal guide bars.
- When the mouse is positioned over a horizontal guide bar, the cursor changes to an up/down move cursor that you can use to click and drag the guide bar to a new location on the image.
- When the mouse is positioned over a vertical guide bar, the cursor changes to a left/right move cursor that you can use to click and drag the guide bar to a new location.
- You can remove a guide bar (either vertical or horizontal) by dragging it off the screen.

Cursor Types

When moving the mouse over a selection, the cursor changes based on its position within the selection. If there are multiple selections active on the image, the cursor becomes a move cursor only. Otherwise, see Table 3-15-1 below for information on how mouse location affects cursor type.

Table 3-15-1. Cursor Types

Mo Mouse Location	Cursor Type

NW drag handle	Upper-left drag cursor
N drag handle	Up/down drag cursor
NE drag handle	Upper-right drag cursor
E drag handle	Left/right drag cursor
SE drag handle	Lower-right drag cursor
S drag handle	Up/down drag cursor
SW drag handle	Lower-left drag cursor
W drag handle	Left/right drag cursor
All other locations	Crosshair (move cursor)

Moving and Resizing

Single or multiple selections of buttons, labels, pictures, and lines or boxes can be moved and/or resized. Moving and resizing selections are subject to these guidelines:

- A single selection is resized by dragging a drag handle.
- A selection (single or multiple) is moved by dragging the cursor if it is a move cursor, or by use of the arrow keys.
- Dragging any selection when multiple objects are selected moves all selections.
- Only one selection can be resized at a time using a drag handle. Multiple selections cannot be resized by dragging. However, you can make multiple selections the same size vertically or horizontally by using the sizing icons on the Image Editor Toolbar or the sizing menu items on the Image Editor Right-Click Menu.
- If no objects are selected, the arrow keys move the cursor as expected.

Image Editor Right-Click Menu

The *Right-Click Menu*, shown in *Figure 3-15-2*, is used to perform many of the same functions available on the Image Editor toolbar.

Image Editor's Right-Click menu

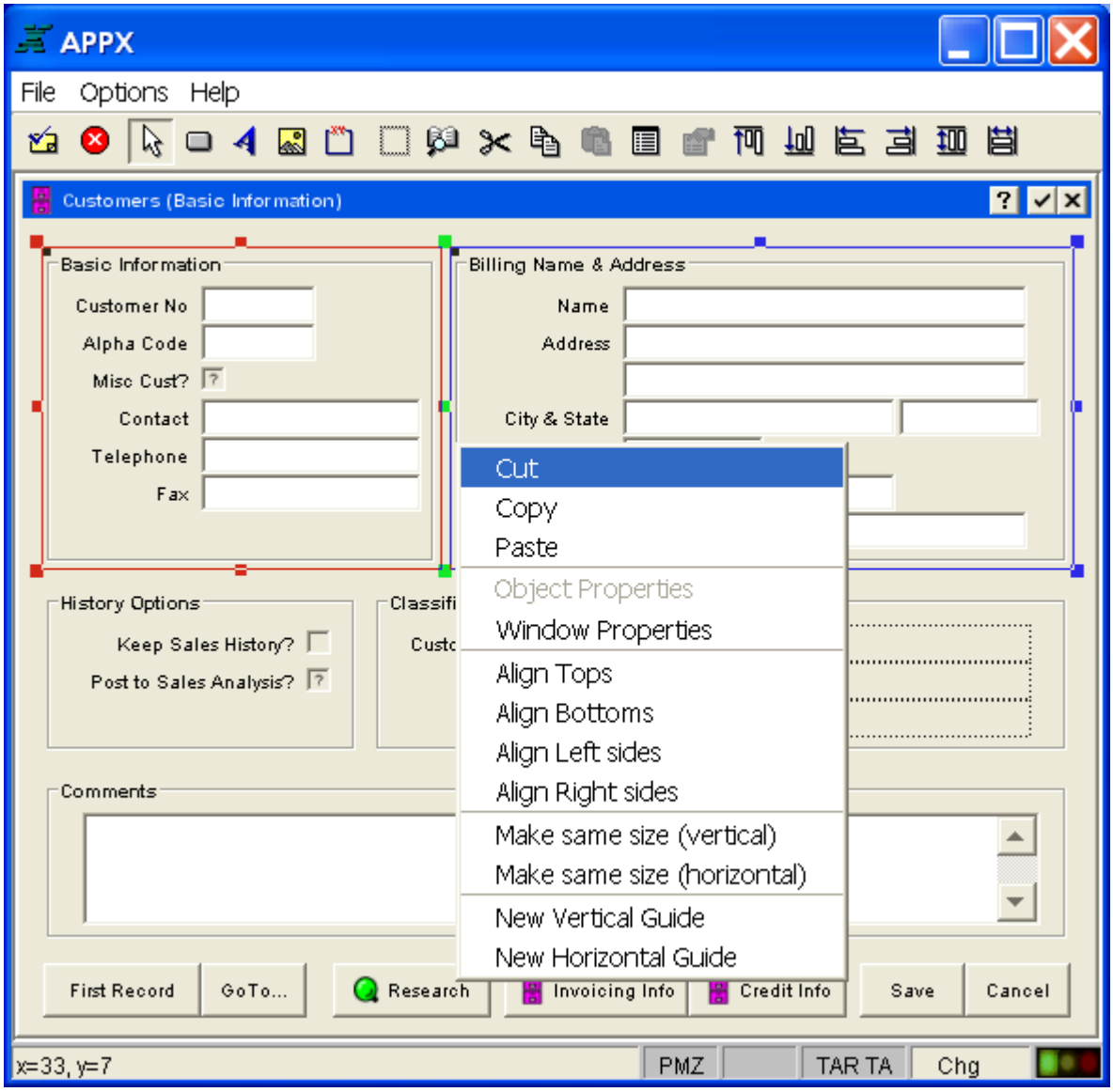


Figure 3-15-2

To access the *Right-Click Menu,* simply click your mouse's right button. Note that positioning of your pointer is important as the top of the menu appears directly below where the mouse pointer is positioned. These functions are available:

- **Cut/Copy/Paste operate as** described in the *Data Palette, Cut/Copy/Paste, and Edit Tools* section of the Image Editor Toolbar topic.
- **Object Properties** *edits* properties of the selected object as described under *Painting Buttons, Labels, Pictures, and Boxes or Lines*.
- **Window Properties edits** graphical attributes of the current image or all images for the current frame.

- **Alignment/Sizing** *operate a*s described in the *Alignment and Sizing Icons* section of the Image Editor Toolbar topic.
- **New Vertical/Horizontal Guide** creates vertical and horizontal guide bars as described under *Using Guide Bars*.

APPX Data Palette

The APPX *Data Palette* is used to paint APPX data objects on the image when using the GUI features of the Image Editor. To access the Data Palette from the Image Editor, just select the DATA PALETTE tool from the Image Editor Toolbar.

The APPX Data Palette overlay appears as shown in *Figure 3-15-3*. You can move the Data Palette Overlay by clicking the Data Palette's titlebar and dragging the overlay to the desired location. You can also resize the overlay by clicking and dragging any corner or side of the overlay's border, thus enlarging or shrinking the overlay.

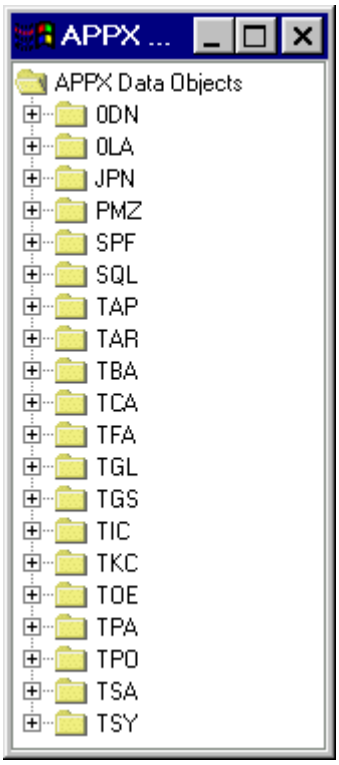


Figure 3-15-3. APPX Data Palette Overlay

APPX Data Objects

The APPX Data Objects section of the Data Palette provides you with an expandable list of data items that are available for placement on your image. See *Figure 3-15-4*. Note that different symbols tell you the data type (alphanumeric, numeric, date, etc.) of each field. To select an item and paint it on your image, just select the item and drag it to the image and its desired position.

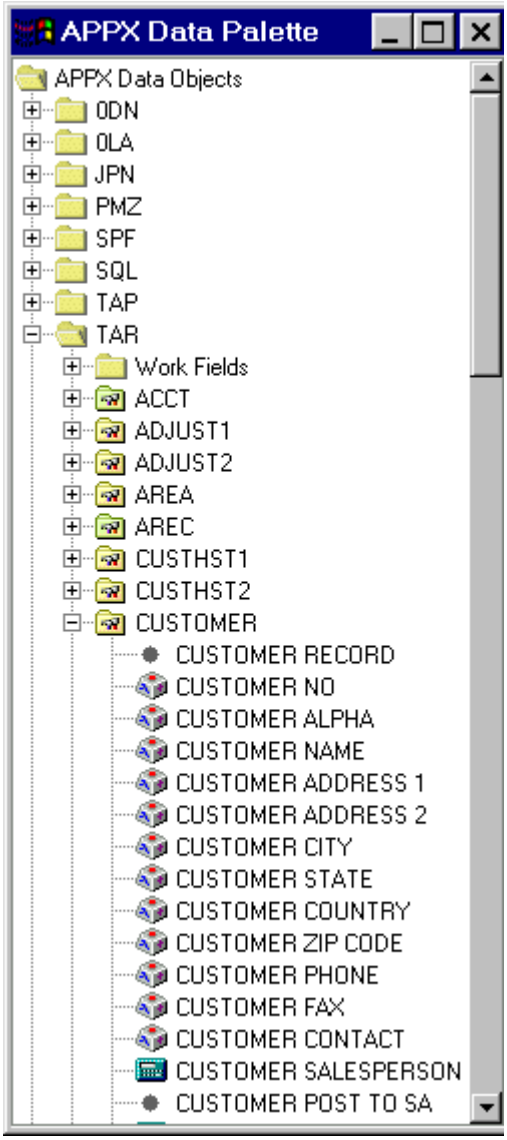


Figure 3-15-4. APPX Data Palette Data Objects List

Editing or Defining Data Objects for Images

The Data Palette is only used to add new data objects to an image. Activate the Data Palette by selecting the DATA PALETTE option from the Image Editor toolbar. Available APPX applications are listed in the APPX Data Options sections of the Data Palette. Double-click one of the applications listed to expand the list to *Work Fields* and file names. Double-click *Work Fields* to view a list of the application's work fields, or double-click a file name to view a list of fields defined within the chosen file. Using the left button of your mouse, click the desired field name (see *Figure 3-15-5*) and drag it to the desired location on the image (position the pointer to where you want the first character of the field to be placed). Release the mouse button to insert the field. *Figure 3-15-6 shows the image after inserting the new field and clicking on it to select it.*

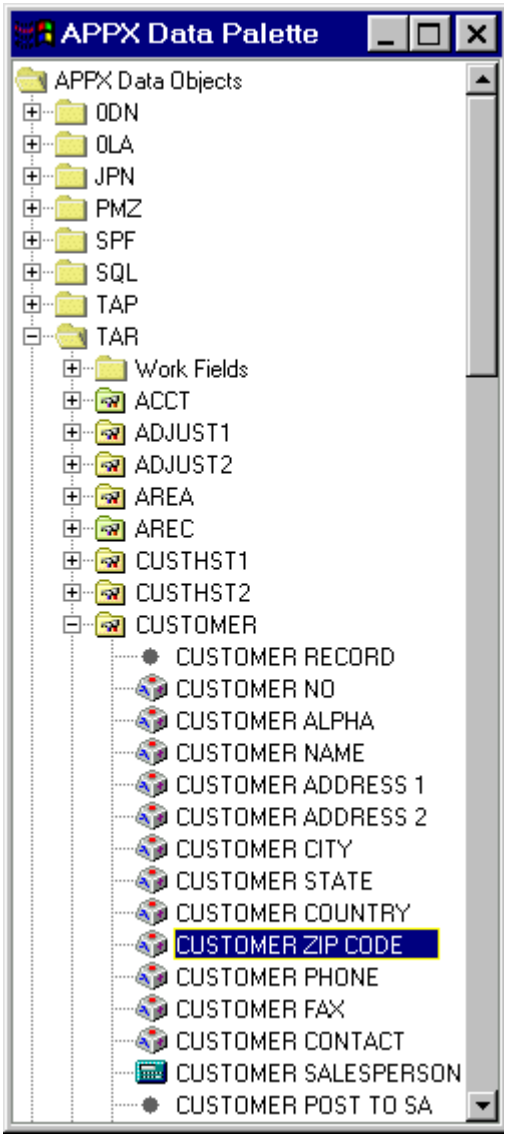


Figure 3-15-5. Selecting Data Object

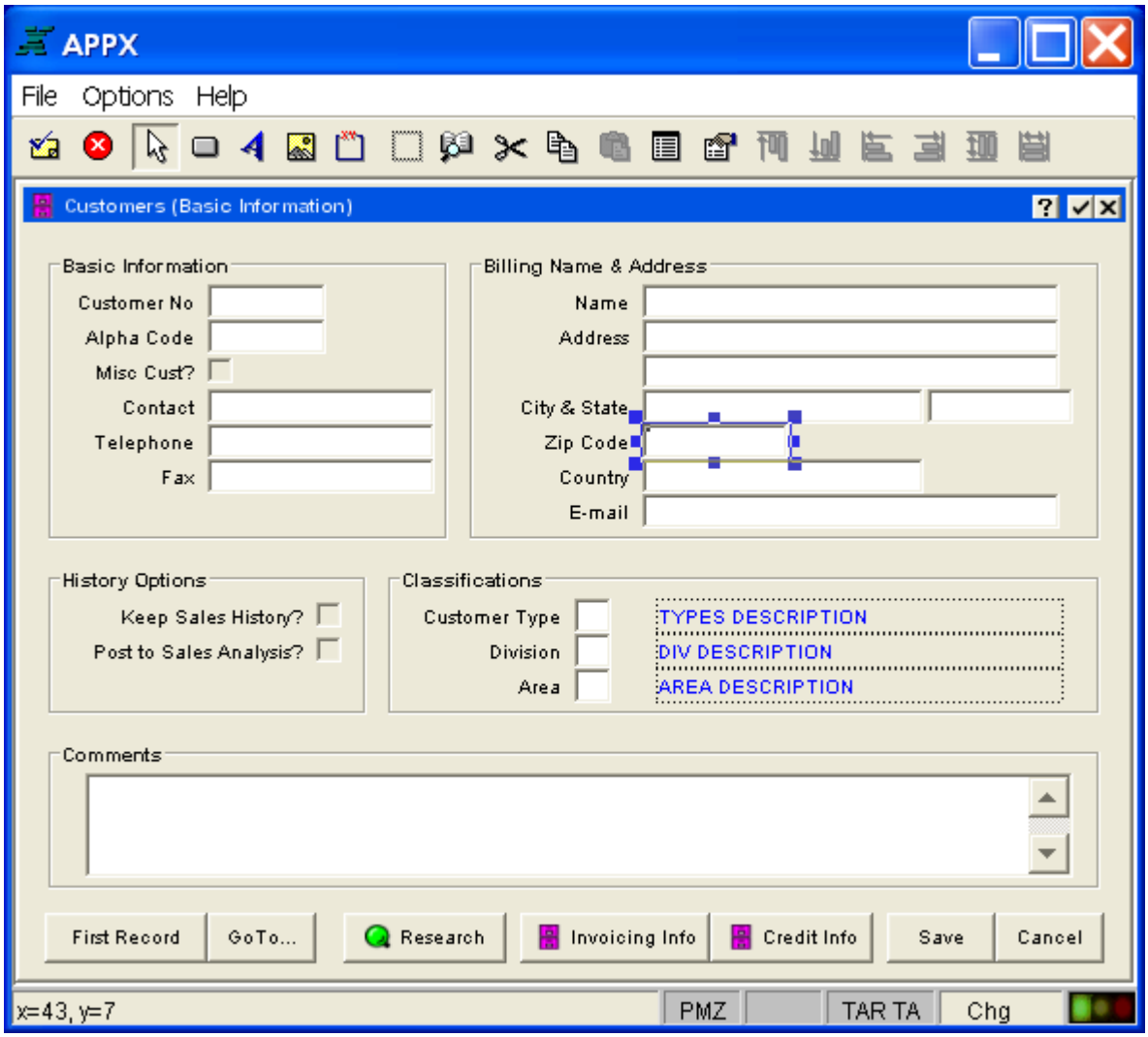


Figure 3-15-6. Using the Data Palette to Paint Objects on an Image

The selected data object will be painted on the image using its default attributes. If you want to change those attributes, simply double click on the object or select the object and then select "Edit Object Properties" from the toolbar. To remove it from the image, select the object and then select "Cut Selection" on the toolbar.

Data Object Specifications

If you are editing an existing data object or adding a new data object, as described in Editing or Defining Data Objects for Images, APPX presents a number of specification screens that establish various operational and display attributes, as well as data lookup definitions.

The item screen contains specifications that relate to the appearance of a data dictionary field on an image. Such specifications are described in the Image *Item Specifications* section in *Chapter 3-14: Using the Image Editor*. The GUI ATTRIBUTES option on the *Item Specifications* overlay contains fields that vary

depending on the type of field (alphanumeric, numeric, date, or logic) and on the type of process. These fields are used to define graphical attributes such as widgets, font characteristics, and colors. For details, see the *Defining Graphical Attributes for Image Fields* section.

Painting Buttons, Labels, Pictures, and Boxes or Lines

You can add buttons, labels, pictures, lines and boxes to your image by using the GUI features of the Image Editor. To learn more about using ILF statements to control the display of pictures, see the PICTURE statement in *Chapter 4-6: ILF Keyword Reference*.

To add buttons, labels, pictures, lines or boxes to an image, you follow the same basic steps:

1. **1) Select the icon** from the Image Editor tool bar that corresponds to the graphic you want to paint on the image.
   To paint a button
   To paint a label
   To paint a picture
   To paint a box (or a line)
2. **2) Select the region** in which you want to place the graphic. Using the mouse, position the pointer to the upper leftmost corner of the intended region. Click the left mouse button and hold, and start dragging the pointer to the lower rightmost corner of the region. This should create the outline of a box on the image, as shown in *Figure 3-15-7*.

   Release the mouse button when the region is outlined approximately where you want the graphic to appear, as shown in *Figure 3-15-8*. You can always make fine adjustments later.

   When defining the region for lines or boxes, the end of the region must be the same row for lines and a subsequent row for boxes. For boxes, you cannot begin in column 1 or end in the last column of your image.
3. **3) Define the graphic** by filling in the overlay that appears. The overlays are different depending on the graphic you are adding. See the appropriate topic (Painting Buttons, Painting Labels, Painting Pictures, or Painting Boxes or Lines) for specifics on defining a particular graphic.

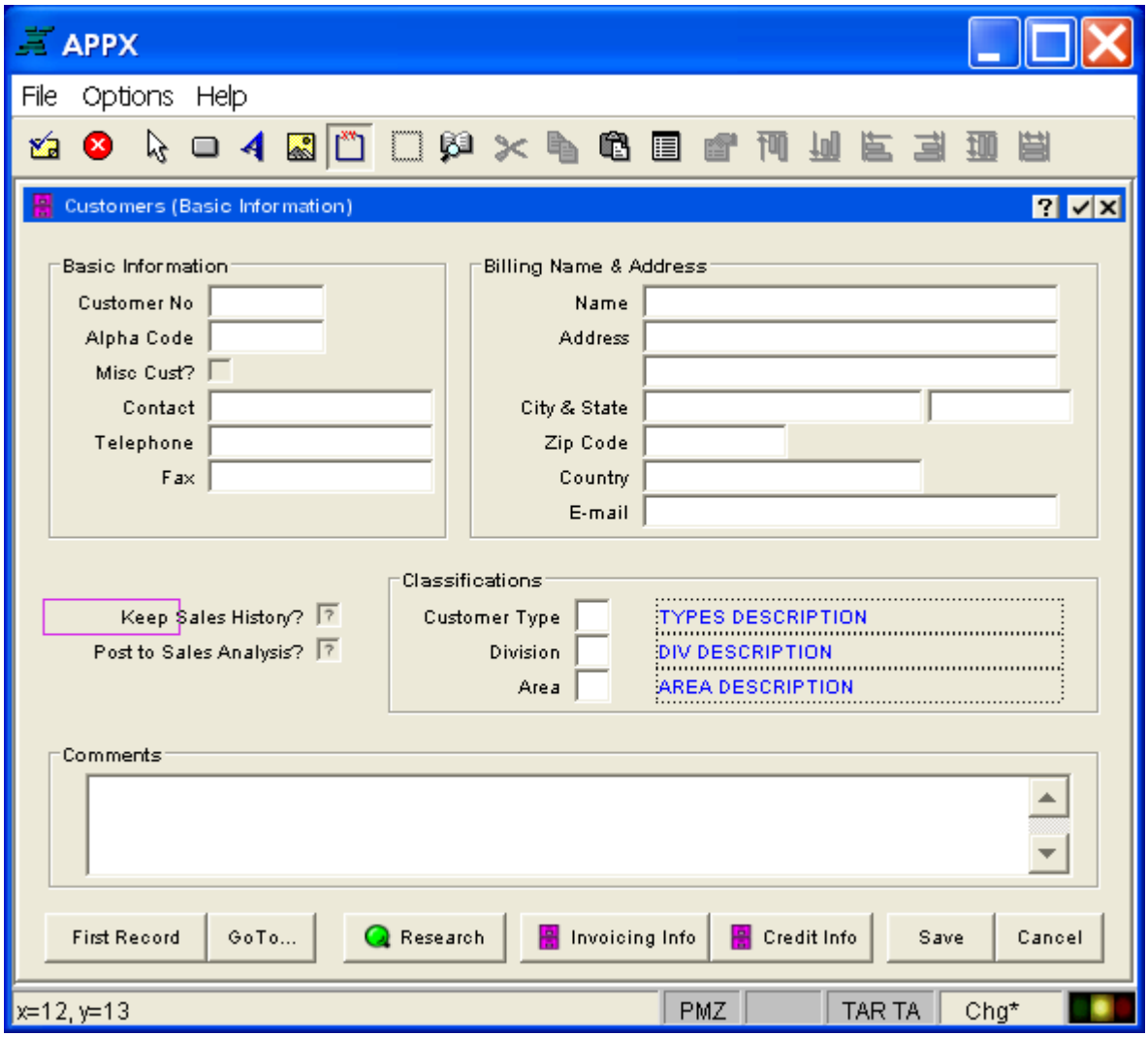


Figure 3-15-7. Selecting a Graphics Region (Part 1)

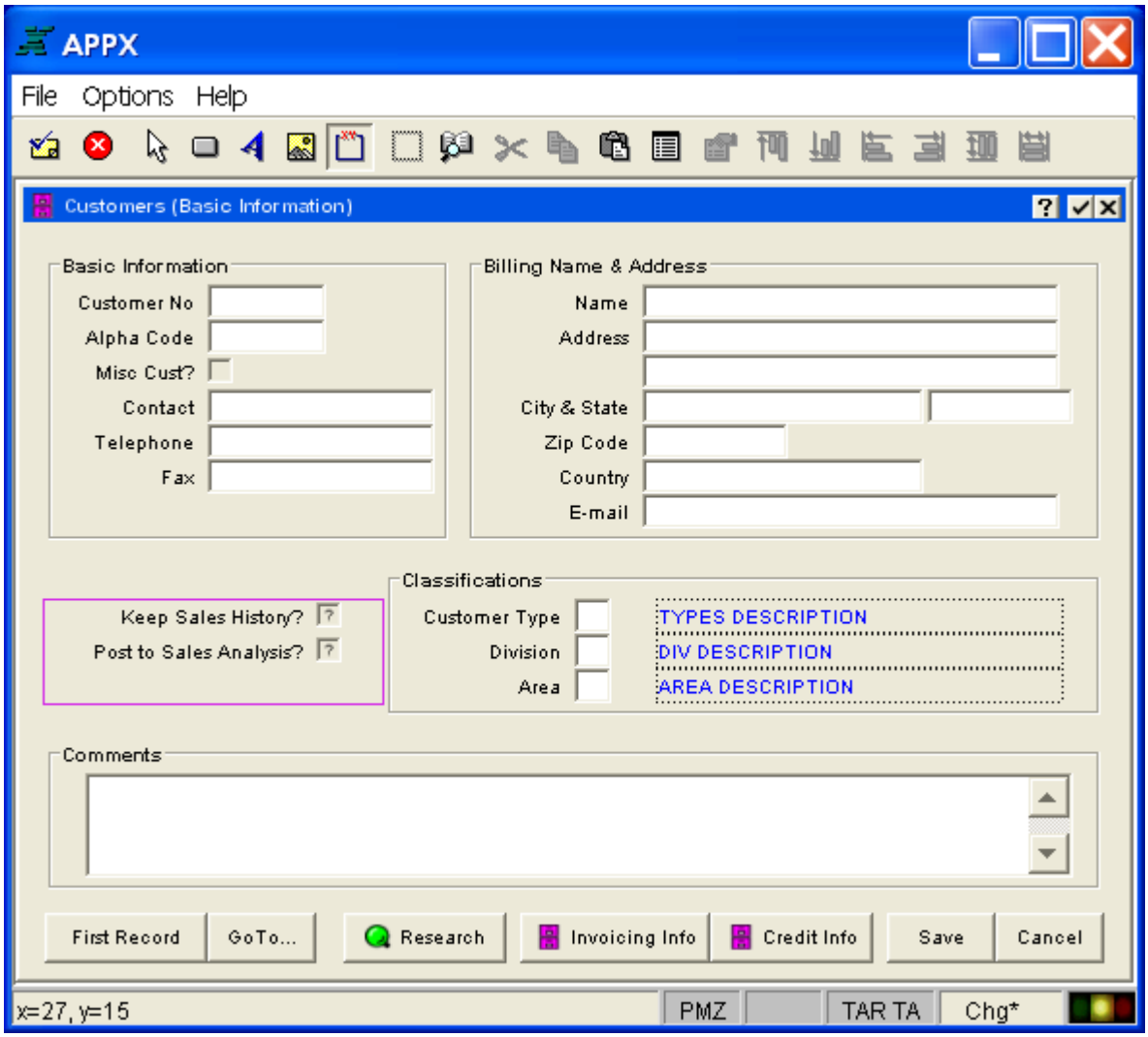


Figure 3-15-8. Selecting a Graphics Region (Part 2)

Painting Buttons

A button is used to provide users with a simple means of initiating an APPX process, function, or option. For example, the button labeled 'Research' shown on the bottom of the screen in *Figure 3-15-6* allows users to quickly reference another process to perform Accounts Receivable research. You can add buttons to any menu or input process to enhance the functionality of the process.

After selecting the region in which you want to paint the button, as described in Painting Buttons, Labels, Pictures, and Boxes, the *Button Properties* overlay shown in *Figure 3-15-9* appears. The information entered in this example produced the RESEARCH button shown earlier in *Figure 3-15-6*. When the user selects this button, the process defined as Option 99 in the Optional Children for this image is executed.

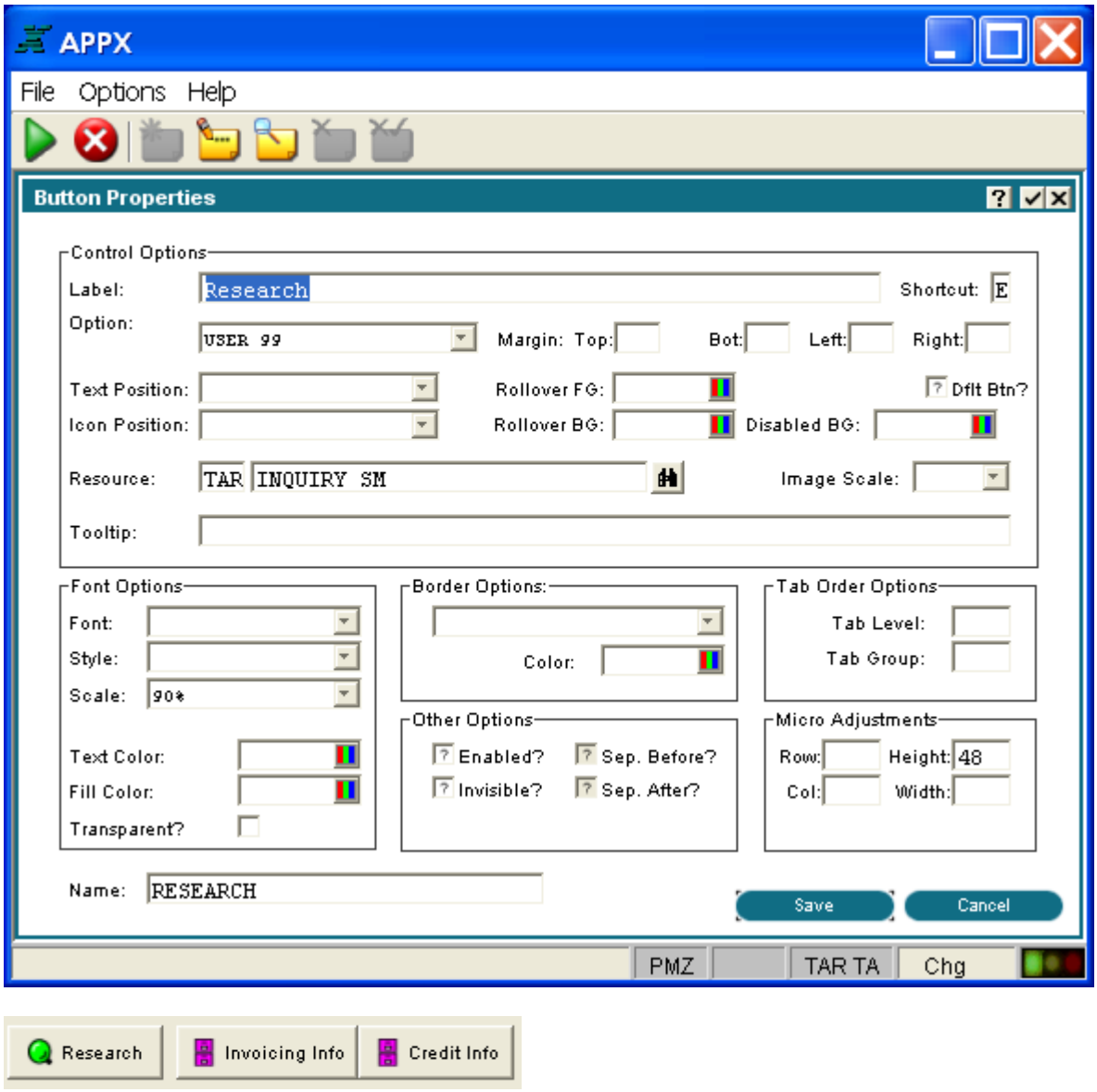


Figure 3-15-9. Button Properties Overlay

The *Button Properties* overlay contains several fields you can use to define the characteristics of your button:

Control Options

- ***Label*** marks the button with the option, function, or process name.
- ***Shortcut*** defines a single key that serves as a shortcut to this button's functionality.
- ***Option*** selects a pre-defined APPX option or an option specific to the user or process. This option corresponds to a specific process defined in the Optional Child Processes for this image.
- ***Margin (Top, Bottom, Left, Right) sets*** extra margin space between the contents of the button and the border of the button. This value is in pixels and must be less than or equal to 255.

- ***Text Position*** indicates where the text should appear inside the button. Click the list button for a drop-down list of selections that include CENTER, TOP, RIGHT, BOTTOM, LEFT, UPPER RIGHT, UPPER LEFT, LOWER RIGHT and LOWER LEFT.
- ***Icon Position*** indicates where the defined icon (if any) should be placed in relation to the text inside the button. Click the list button for a drop-down list of selections that include BEHIND TEXT, ABOVE TEXT, TRAILING TEXT, BELOW TEXT, LEADING TEXT, ABOVE & TRAILING, ABOVE & LEADING, BELOW & TRAILING, BELOW & LEADING and EXPAND TO FILL.
- ***Rollover FG*** indicates the foreground color of this button when the mouse pointer is positioned over the button. Click the color icon to begin and see the *Making Color Selections* section for further instructions.
- ***Rollover BG*** indicates the background color of this button when the mouse pointer is positioned over the button. Click the color icon to begin and see the *Making Color Selections* section for further instructions.
- ***Disabled BG*** indicates the background color of this button when it has been disabled. Click the color icon to begin and see the *Making Color Selections* section for further instructions.
- ***Dflt Btn?*** indicates whether or not this button is the default button.
- ***Resource specifies*** the application and name of a resource defined in Named Resources which links to one or more graphics files. Different graphics can apply depending upon resource state (disabled, enabled, rollover, pressed, etc).
- *Image Scale* defines the size of the graphic as a percentage of the runtime font size. Click the list button to make your choice.
- ***Tooltip*** defines one line of help text that is displayed after the mouse pointer has been positioned over the button for a few seconds.

Font Options

Font

Selects the font (Arial, Courier, etc.) the button's label is to be displayed with. Click the list button to make your choice. If left blank, the default font determined by your system administrator will be used.

Style

Selects the label's font style (bold, italic, etc.). Click the list button to make your choice. If left blank, the regular font style is used.

Scale

Defines the size of the label's font as a percentage of the runtime font size. Click the list button to make your choice.

Text Color

Defines the label's font color. Click the color icon to begin and see the *Making Color Selections* section for further instructions.

Fill Color

Defines the button's fill color. Click the color icon to begin and see the *Making Color Selections* section for further instructions.

Transparent?

Indicates whether or not the background of the button should be opaque or transparent.

Border Options

Type

Defines the type of border that will surround the button. Click the list button to make your choice.

Color defines

The color of the border. Click the color icon to begin and see the *Making Color Selections* section for further instructions.

Tab Order Options

Tab Level

Defines the physical pass over the whole screen with respect to tab order. If left blank, this the button is considered to be in tab level 0. APPX will tab from top-left to bottom-right for all data objects in tab level 0. Then it repeats the sequence for all button objects in tab level 0. The cycle is repeated as necessary for each successive tab level. This behavior can be altered by the specification of Tab Groups.

Tab Group

Marks related fields and buttons so that they are grouped together as part of a tab order. When making a pass over the screen for a particular "Tab Level", if a field is encountered that has a non-null "Tab Group", then the tab order proceeds to all the fields and buttons that are part of that Tab Level/Tap Group. After reaching the last field or button in the group, the order will continue where it left off prior to entering the group. Note that for "Tab Groups", fields and buttons are consolidated into the same top-left to bottom-right pass.

Other Options

Enabled

Indicates whether or not the button is initially enabled for use when this image is first presented to the user. If "no" then the button must be enabled via ILF code in order for the user to be able to select it. Click the checkbox to toggle between "yes" and "no" values.

Invisible

Indicates whether or not the button is initially invisible when this image is first presented to the user. If "yes" then the button must be made visible via ILF code in order for the user to be able to select it. Click the checkbox to toggle between "yes" and "no" values.

Sep. Before?
Is not a valid option when defining an button and is disabled.

Sep. After?
Is not a valid option when defining an button and is disabled.

Micro Adjustments

Row, Col, Height, and Width
Make slight adjustments to the size and position of the button. They are entered as a percentage of an image cell (1 row x 1 column) and must be less than or equal to 100.

Finally,

Name
Allows you to "name" the button so it can be controlled (enabled, disabled, etc.) via ILF statements.

Access to the GUI attributes at runtime with ILF code is done via the --- WIDGET file. The WIDGET records from the process are copied into this file and are available in the *Pre-Display* event point. WIDGET NAME is a key to this file. You can read and rewrite records in this file to affect the GUI attributes at runtime.

When you have finished defining the button you have two options available to you:

- Select the SAVE option on the bottom right corner of the screen to save the definition and return to the Image Editor.
- Or, you can select the CANCEL option to discard the new definition or changes to an existing definition, and then return to the Image Editor.

Painting Labels

A label is used to define image text with GUI attributes that enhance the appearance of your image. Labels can be defined for descriptive text beside data objects. Label can also be used for a variety of other situations such as column headings above a scrolling region or other text which is to appear on the image.

The *Label Properties* overlay shown in *Figure 3-15-10* appears after you have selected the region on the image in which you want the label to appear, as described in Painting Buttons, Labels, Pictures, and Boxes.

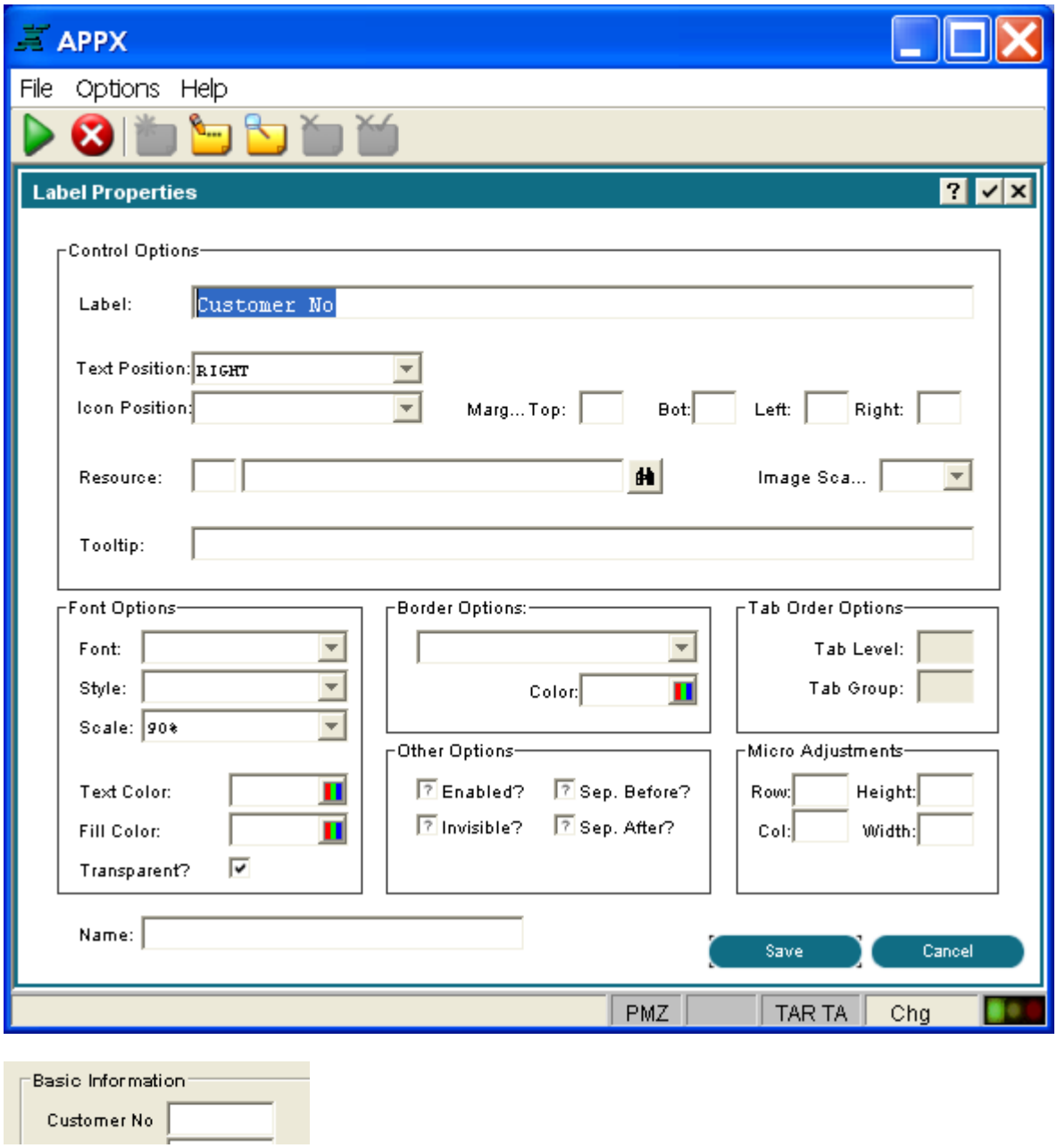


Figure 3-15-10. Label Properties Overlay

The Label Properties overlay contains several fields which are used to define the characteristics of a label:

Control Options

- ***Label*** to define the label's text.
- ***Text Position*** indicates where the label should appear inside the selected region. Click the list button for a drop-down list of selections that include CENTER, TOP, RIGHT, BOTTOM, LEFT, UPPER RIGHT, UPPER LEFT, LOWER RIGHT and LOWER LEFT.

- ***Icon Position*** indicates where the defined icon (if any) should be placed in relation to the label inside the selected region. Click the list button for a drop-down list of selections that include BEHIND TEXT, ABOVE TEXT, TRAILING TEXT, BELOW TEXT, LEADING TEXT, ABOVE & TRAILING, ABOVE & LEADING, BELOW & TRAILING, BELOW & LEADING and EXPAND TO FILL.
- ***Margin (Top, Bottom, Left, Right) sets*** extra margin space between the label text and the border of the region. This value is in pixels and must be less than or equal to 255.
- ***Resource specifies*** the application and name of a resource defined in Named Resources which links to one or more graphics files. Different graphics can apply depending upon resource state (disabled, enabled, rollover, pressed, etc).
- *Image Scale* defines the size of the graphic as a percentage of the runtime font size. Click the list button to make your choice.
- ***Tooltip*** defines one line of help text that is displayed after the mouse pointer has been positioned over the label for a few seconds. Tooltip does not apply to labels on output and inquiry images.

Font Options

Font

Selects the font (Arial, Courier, etc.) the label is to be displayed with. Click the list button to make your choice. If left blank, the default font determined by your system administrator will be used.

Style

Selects the label's font style (bold, italic, etc.). Click the list button to make your choice. If left blank, the regular font style is used.

Scale

Defines the size of the label's font as a percentage of the runtime font size. Click the list button to make your choice.

Text Color

Defines the label's font color. Click the color icon to begin and see the *Making Color Selections* section for further instructions.

Fill Color

Defines the region's fill color. Click the color icon to begin and see the *Making Color Selections* section for further instructions.

Transparent?

Indicates whether or not the background of the region should be opaque or transparent.

Border Options

Type

Defines the type of border that will surround the region. Click the list button to make your choice.

Color defines

The color of the border. Click the color icon to begin and see the *Making Color Selections* section for further instructions.

Tab Order Options

Tab Level

Is not a valid option when defining a label and is disabled.

Tab Group

Is not a valid option when defining a label and is disabled.

Other Options

Enabled

Indicates whether or not the label is initially enabled for use when this image is first presented to the user. If "no", then the label must be enabled via ILF code in order for the enabled state to apply. Click the checkbox to toggle between "yes" and "no" values.

Invisible

Indicates whether or not the label is initially invisible when this image is first presented to the user. If "yes" then the label must be made visible via ILF code in order for the user to see it. Click the checkbox to toggle between "yes" and "no" values.

Sep. Before?

Determines whether or not a vertical line precedes the label.

Sep. After?

Determines whether or not a vertical line follows the label.

Micro Adjustments

Row, Col, Height, and Width

Make slight adjustments to the size and position of the label. They are entered as a percentage of an image cell (1 row x 1 column) and must be less than or equal to 100.

Finally,

Name
Allows you to “name” the label so it can be controlled (enabled, disabled, etc.) via ILF statements.

Access to the GUI attributes at runtime with ILF code is done via the --- WIDGET file. The WIDGET records from the process are copied into this file and are available in the *Pre-Display* event point. WIDGET NAME is a key to this file. You can read and rewrite records in this file to affect the GUI attributes at runtime.

When you have finished defining the label you have two options available to you:

- Select the SAVE option on the bottom right corner of the screen to save the definition and return to the Image Editor.
- Or, you can select the CANCEL option to discard the new definition or changes to an existing definition, and then return to the Image Editor.

Painting Pictures

A picture object is used to add static graphic designs to a menu, input, output, or inquiry process, or to provide visual detail regarding, for example, a place, a product, or a person. To do so, you must first know the name and location (folder name) of the graphics file you want to add to the image.

An example input process with a picture on the image is shown in *Figure 3-15-11*. The overlay shown in *Figure 3-15-12 was used to define the picture for this image. The Picture Properties overlay* appears after you select your picture region, as described in Painting Buttons, Labels, Pictures, and Boxes. Notice that a Resource Name was used to identify the picture which is to be displayed. Picture objects are static and do not change at runtime. Picture objects can be used to display various types of static picture objects such as the corporate logo, a picture of the company's headquarters building, or a picture of the company's founding father.

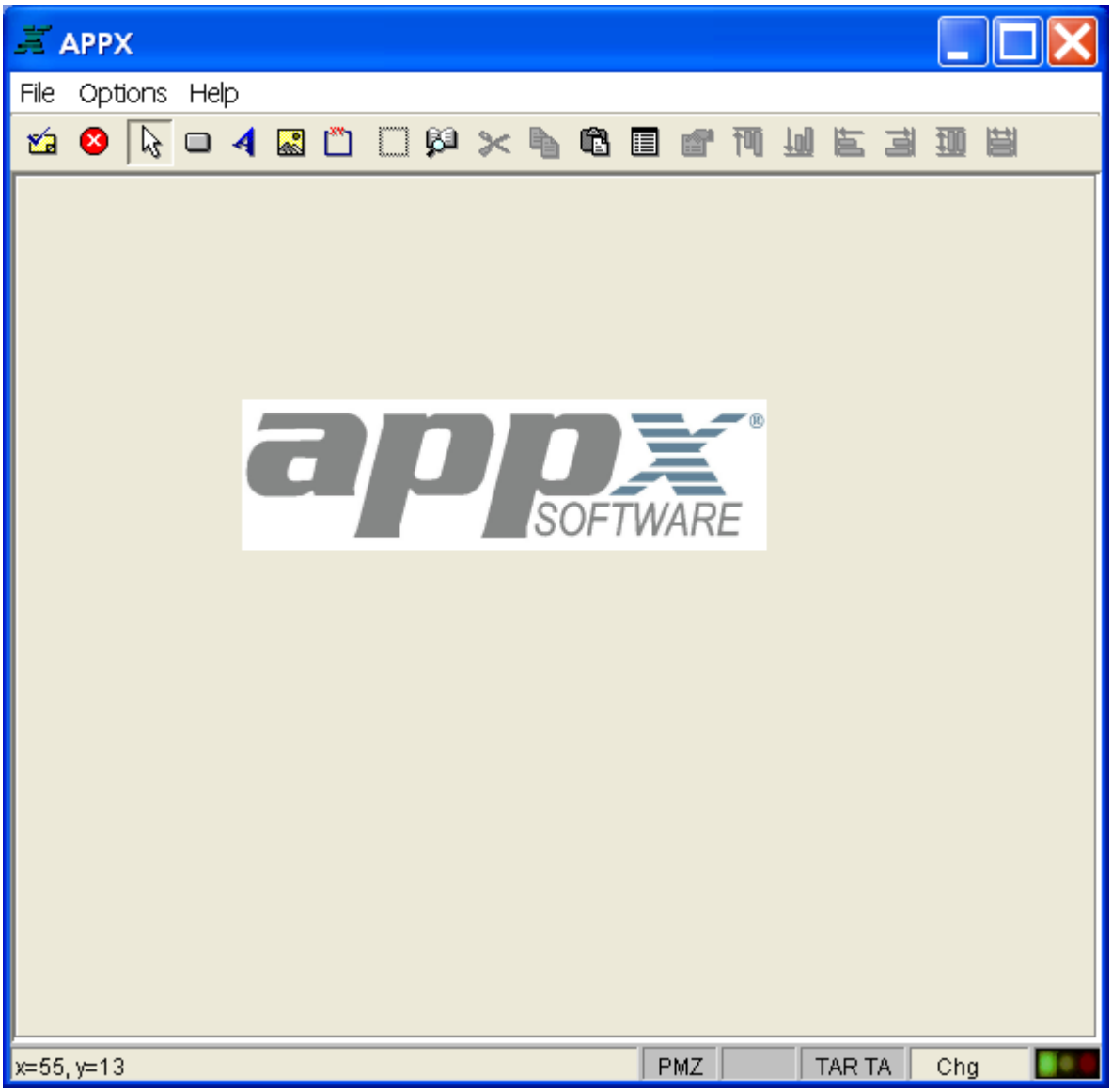


Figure 3-15-11. An Input Image Containing a Picture Icon

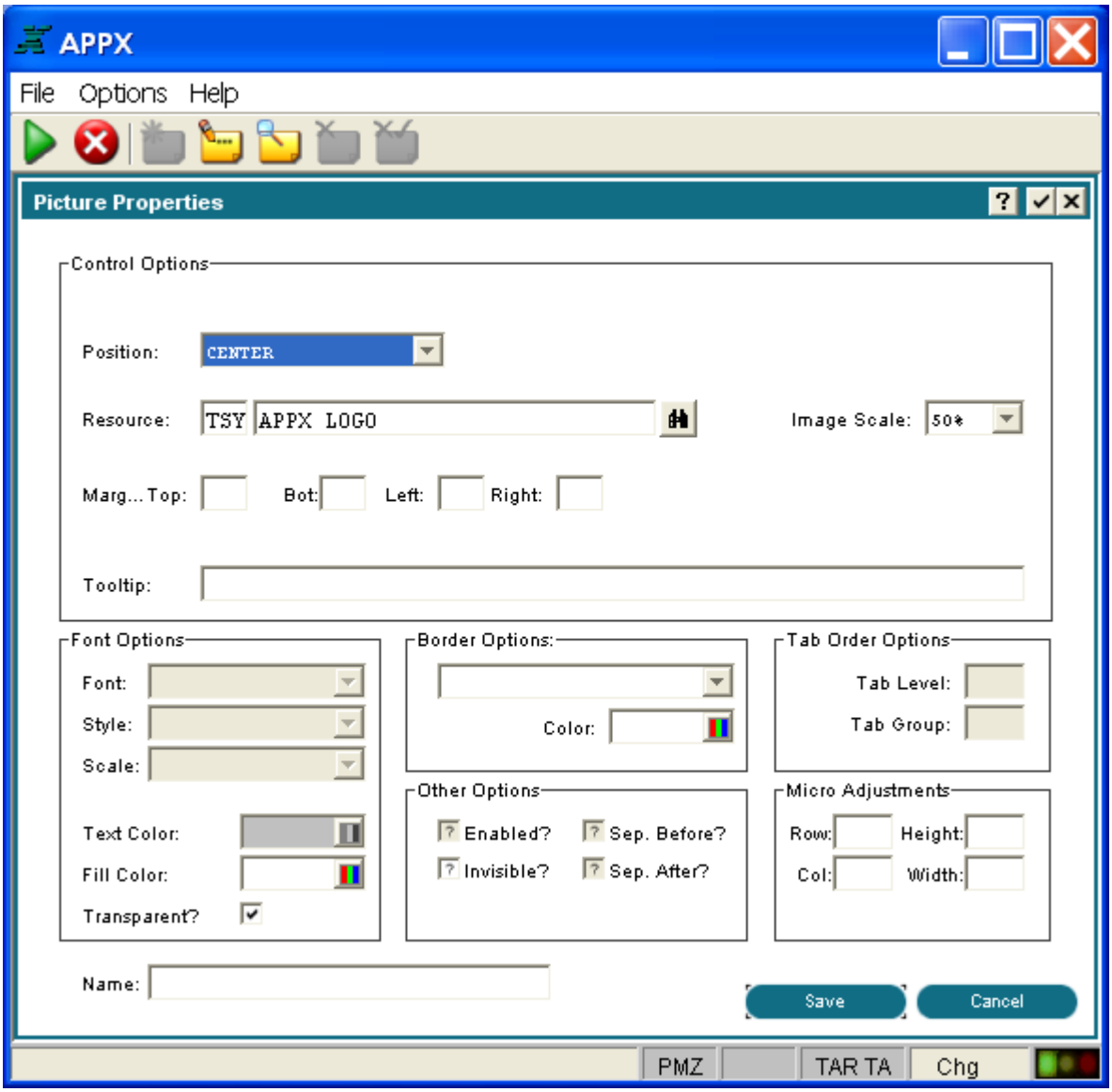


Figure 3-15-12. Pictures Properties Overlay

The Picture Properties overlay contains several fields that are used to define the characteristics of a picture.

Control Options

- ***Position*** indicates where the picture should appear inside the selected region. Click the list button for a drop-down list of selections that include CENTER, TOP, RIGHT, BOTTOM, LEFT, UPPER RIGHT, UPPER LEFT, LOWER RIGHT and LOWER LEFT.
- ***Resource specifies*** the application and name of a resource defined in Named Resources which links to one or more graphics files. Different graphics can apply depending upon resource state (disabled, enabled, rollover, pressed, etc).
- *Image Scale* defines the size of this appearance of the graphic as a percentage of the normal graphic. Click the list button to make your choice.

- ***Margin (Top, Bottom, Left, Right) sets*** extra margin space between the picture and the border of the region. This value is in pixels and must be less than or equal to 255.
- ***Tooltip*** defines one line of help text that is displayed after the mouse pointer has been positioned over the picture for a few seconds. Tooltip does not apply to pictures on output and inquiry images.

Font Options

Font
Is not a valid option when defining a picture and is disabled.

Style
Is not a valid option when defining a picture and is disabled.

Scale
Is not a valid option when defining a picture and is disabled.

Text Color
Is not a valid option when defining a picture and is disabled.

Fill Color
Defines the region's fill color. Click the color icon to begin and see the *Making Color Selections* section for further instructions.

Transparent?
Indicates whether or not the background of the region should be opaque or transparent.

Border Options

Type
Defines the type of border that will surround the region. Click the list button to make your choice.

Color defines
The color of the border. Click the color icon to begin and see the *Making Color Selections* section for further instructions.

Tab Order Options

Tab Level
Is not a valid option when defining a picture and is disabled.

Tab Group
Is not a valid option when defining a picture and is disabled.

Other Options

Enabled

Is not a valid option when defining a picture and is disabled.

Invisible

Indicates whether or not the picture is initially invisible when this image is first presented to the user. If "yes" then the picture must be made visible via ILF code in order for the user to see it. Click the checkbox to toggle between "yes" and "no" values.

Sep. Before?

Is not a valid option when defining a picture and is disabled.

Sep. After?

Is not a valid option when defining a picture and is disabled.

Micro Adjustments

Row, Col, Height, and Width

Make slight adjustments to the size and position of the picture. They are entered as a percentage of an image cell (1 row x 1 column) and must be less than or equal to 100.

Finally,

Name

Allows you to “name” the picture so it can be controlled (enabled, disabled, etc.) via ILF statements.

Access to the GUI attributes at runtime with ILF code is done via the --- WIDGET file. The WIDGET records from the process are copied into this file and are available in the *Pre-Display* event point. WIDGET NAME is a key to this file. You can read and rewrite records in this file to affect the GUI attributes at runtime.

When you have finished defining the picture you have two options available to you:

- Select the SAVE option on the bottom right corner of the screen to save the definition and return to the Image Editor.
- Or, you can select the CANCEL option to discard the new definition or changes to an existing definition, and then return to the Image Editor.

Painting Boxes or Lines

After selecting the region in which you want your box or line to appear, as described in Painting Buttons, Labels, Pictures, and Boxes, the *Box Properties* overlay shown in *Figure 3-15-13A* appears. This example defines the "History Options" box shown in *Figure 3-15-13B*.

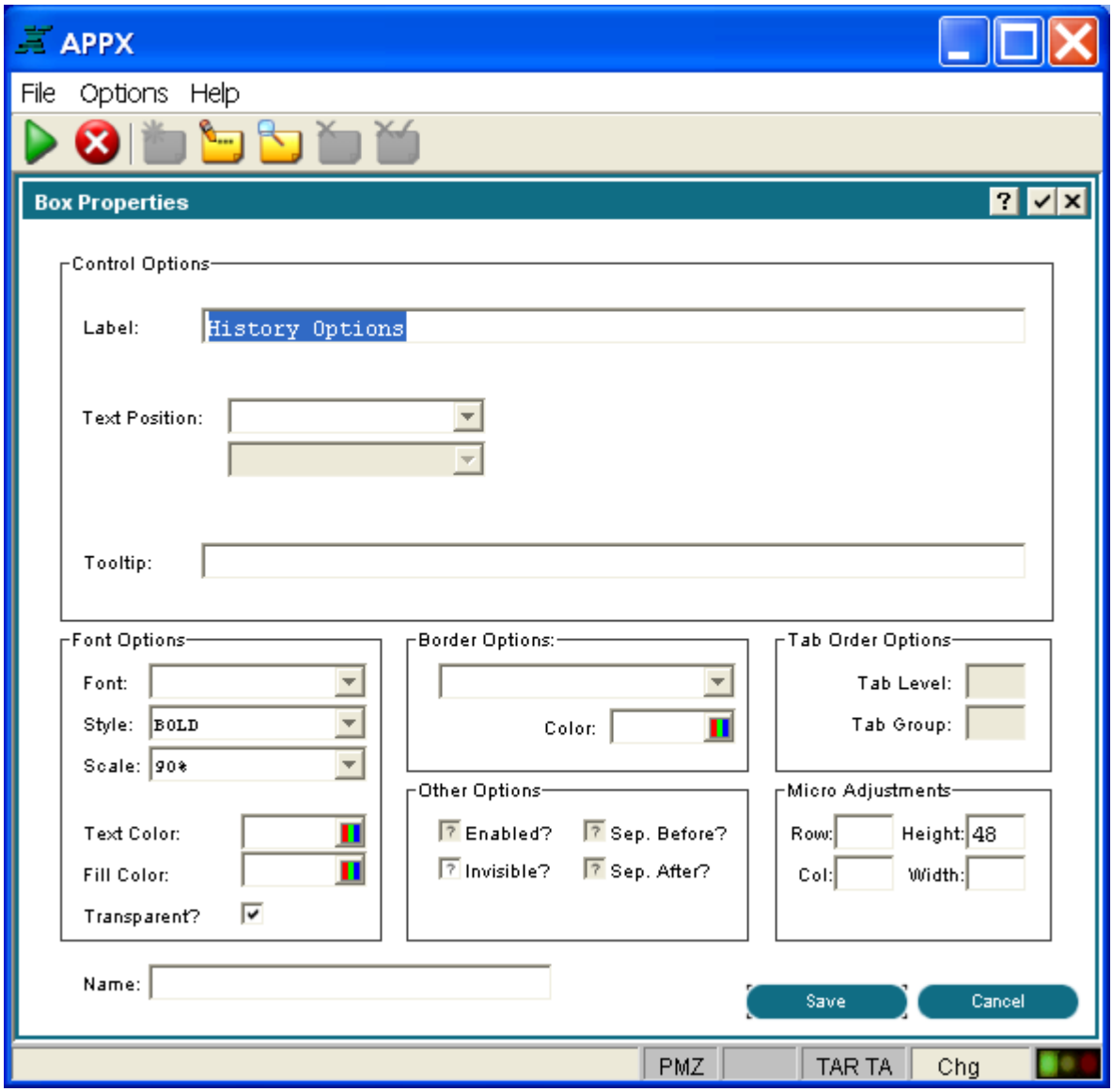


Figure 3-15-13A. Box Properties Overlay for History Options Box

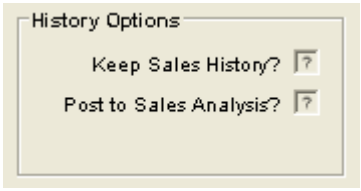


Figure 3-15-13B. History Options Box

Control Options

Label

To define any text to appear as a box title.

Text Position

Indicates where the text should appear inside the box border. Click the list button for a drop-down list of selections that include CENTER, TOP, RIGHT, BOTTOM, LEFT, UPPER RIGHT, UPPER LEFT, LOWER RIGHT and LOWER LEFT.

Tooltip

Defines one line of help text that is displayed after the mouse pointer has been positioned over the box for a few seconds. Tooltip does not apply to boxes on output and inquiry images.

Font Options

Font

Selects the font (Arial, Courier, etc.) the box's label is to be displayed with. Click the list button to make your choice. If left blank, the default font determined by your system administrator will be used.

Style

Selects the label's font style (bold, italic, etc.). Click the list button to make your choice. If left blank, the regular font style is used.

Scale

Defines the size of the label's font as a percentage of the runtime font size. Click the list button to make your choice.

Text Color

Defines the label's font color. Click the color icon to begin and see the *Making Color Selections* section for further instructions.

Fill Color

Defines the region's fill color. Click the color icon to begin and see the *Making Color Selections* section for further instructions.

Transparent?

Indicates whether or not the background of the region should be opaque or transparent.

Border Options

Type

Defines the type of border for the box. Click the list button to make your choice.

Color defines

The color of the border. Click the color icon to begin and see the *Making Color Selections* section for further instructions.

Tab Order Options

Tab Level

Is not a valid option when defining a box and is disabled.

Tab Group

Is not a valid option when defining a box and is disabled.

Other Options

Enabled

Is not a valid option when defining a box and is disabled.

Invisible

Indicates whether or not the box is initially invisible when this image is first presented to the user. If "yes" then the box must be made visible via ILF code in order for the user to see it. Click the checkbox to toggle between "yes" and "no" values.

Sep. Before?

Is not a valid option when defining a box and is disabled.

Sep. After?

Is not a valid option when defining a box and is disabled.

Micro Adjustments

Row, Col, Height, and Width

Make slight adjustments to the size and position of the box. They are entered as a percentage of an image cell (1 row x 1 column) and must be less than or equal to 100.

Finally,

Name

Allows you to “name” the box so it can be controlled (enabled, disabled, etc.) via ILF statements.

Access to the GUI attributes at runtime with ILF code is done via the --- WIDGET file. The WIDGET records from the process are copied into this file and are available in the *Pre-Display* event point. WIDGET NAME is a key to this file. You can read and rewrite records in this file to affect the GUI attributes at runtime.

When you have finished defining the box you have two options available to you:

- Select the SAVE option on the bottom right corner of the screen to save the definition and return to the Image Editor.
- Or, you can select the CANCEL option to discard the new definition or changes to an existing definition, and then return to the Image Editor.

Changing and Deleting Graphical Items

Buttons, labels, pictures, boxes, and lines that already exist on an image can be changed or deleted.

- To **change** a button, label, picture, box, or line, double click on it to select it and to display the corresponding properties overlay. Make your changes and remember to select SAVE when finished.
- To **delete** the item, click it once to select it, then click the CUT SELECTION icon on the Image Editor toolbar.

Use of Graphical Attributes

Graphical attributes define the way menus, input, output, and inquiry processes, options, and fields *appear, print,* and/or *operate.* They are recognized only when running the GUI-based version of APPX (also know as the APPX Client). While these attributes may be defined by the designer while using either the GUI or character-based version of APPX, they are ignored when users are running APPX in character mode and will not affect the display of character-based images.

Types of graphical attributes include the following:

Wallpaper
Consisting of a graphic image or a solid color background.

Borders
To offset a frame.

Titlebars
That can include graphics such as a corporate logo.

Buttons and Icons
To provide graphical representation of available tasks and processes.

Widgets
Such as checkboxes, radio buttons, slide bars and date selection calendars.

Menu types
Such as pull-down menus, pop-up menus, and right-click context menus.

Graphical attributes can be defined within these APPX components:

Menu Processes
For adding graphical features to a menu image.

Input, Output, and Inquiry Processes
For adding graphical elements to a specific image, or if defined as global, all images within the frame.

Optional Children

Of Menu Processes and Input Process images. At this level, graphical attributes are used to display lower level processes in a manner other than traditional APPX option formats.

Image Fields

For adding graphical elements to specific items (fields) on an image.

The Defining Graphical Attributes for Images section discusses the options available for menu, input, output, and inquiry processes through the use of Window Properties from the Image Editor Right-Click Menu.

Another section, *Defining Graphical Attributes for Optional Children,* explains how to design the way optional child processes appear and are selected on your menu or input process image. You can use the default traditional button approach, or define pull-down and right-click menus.

Finally, the Defining Graphical Attributes for Image Fields section covers the various attributes that can be attached to data objects.

Defining Graphical Attributes for Images

Graphical attributes for images can be defined by selecting the WINDOW PROPERTIES option from the Image Editor Right-Click Menu. In an input, output, or inquiry process, these attributes can be defined at the frame or the image level by selecting GLOBAL WINDOW PROPERTIES or WINDOW PROPERTIES, respectively, from the subsequent overlay. If the attributes are defined globally, the defined elements will apply to all images within the current frame. If the frame consists of multiple images, each with unique graphical attribute requirements, then the graphical attributes must be defined for each image at the image level. Should a frame and an image within the frame each have graphical attributes defined, the graphical attributes defined at the image level will override those defined at the frame level.

Figure 3-15-14 shows a menu with a wallpaper background and pull-down menus. The wallpaper was defined by selecting the WINDOW PROPERTIES option on the Right-Click Menu, shown in *Figure 3-15-15, from the Accounts Receivable menu image.* Selecting this option displays the *Window Properties* overlay shown in *Figure 3-15-16.* The attributes defined in this example produced the wallpaper shown in *Figure 3-15-14.* Note that there is no title bar defined for this menu and that the title bar that does appear is the standard APPX session title bar.

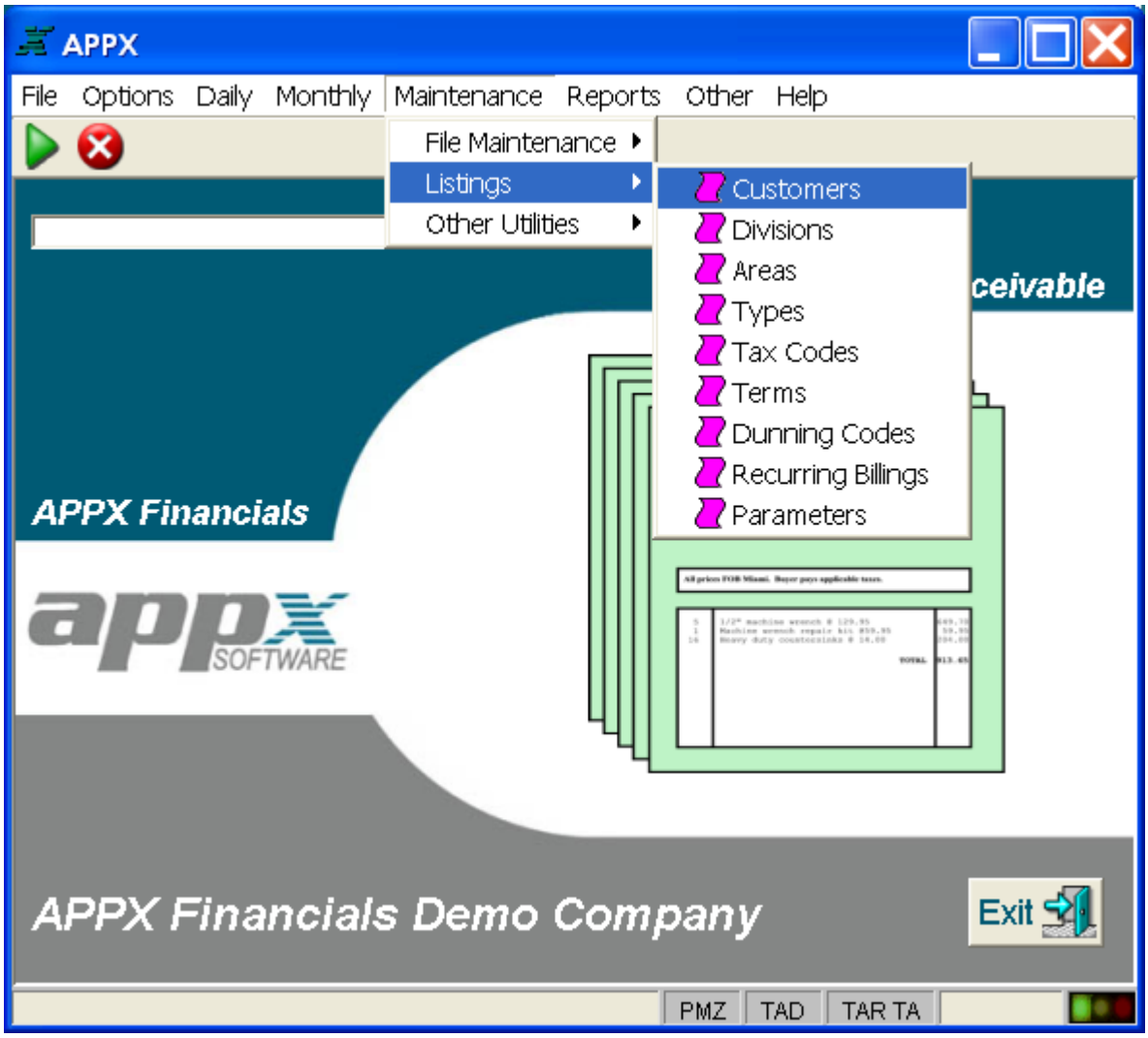


Figure 3-15-14. Accounts Receivable Menu Defined with Graphical Attributes

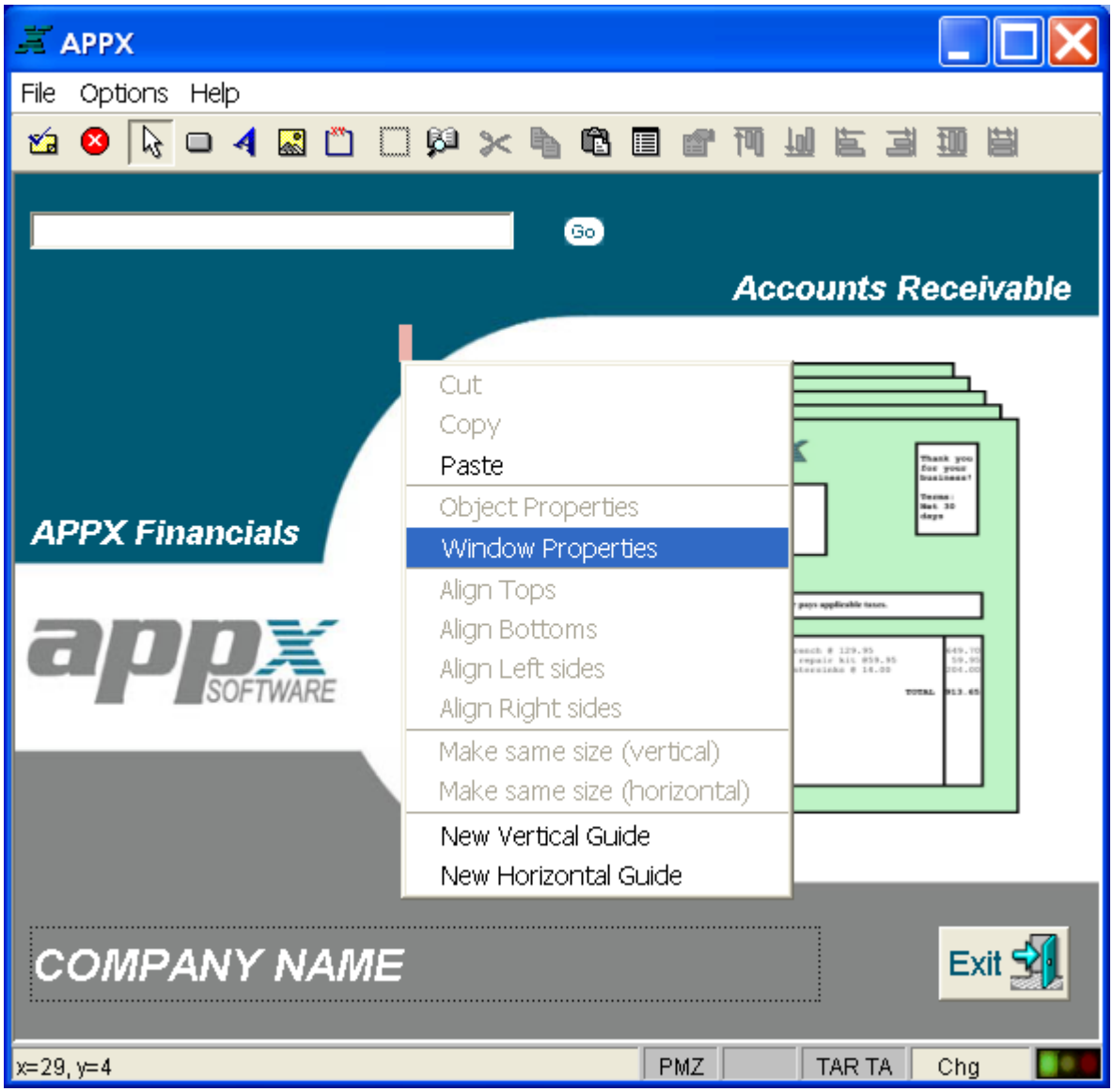


Figure 3-15-15. Right-Click Menu Overlay from Accounts Receivable Menu Image

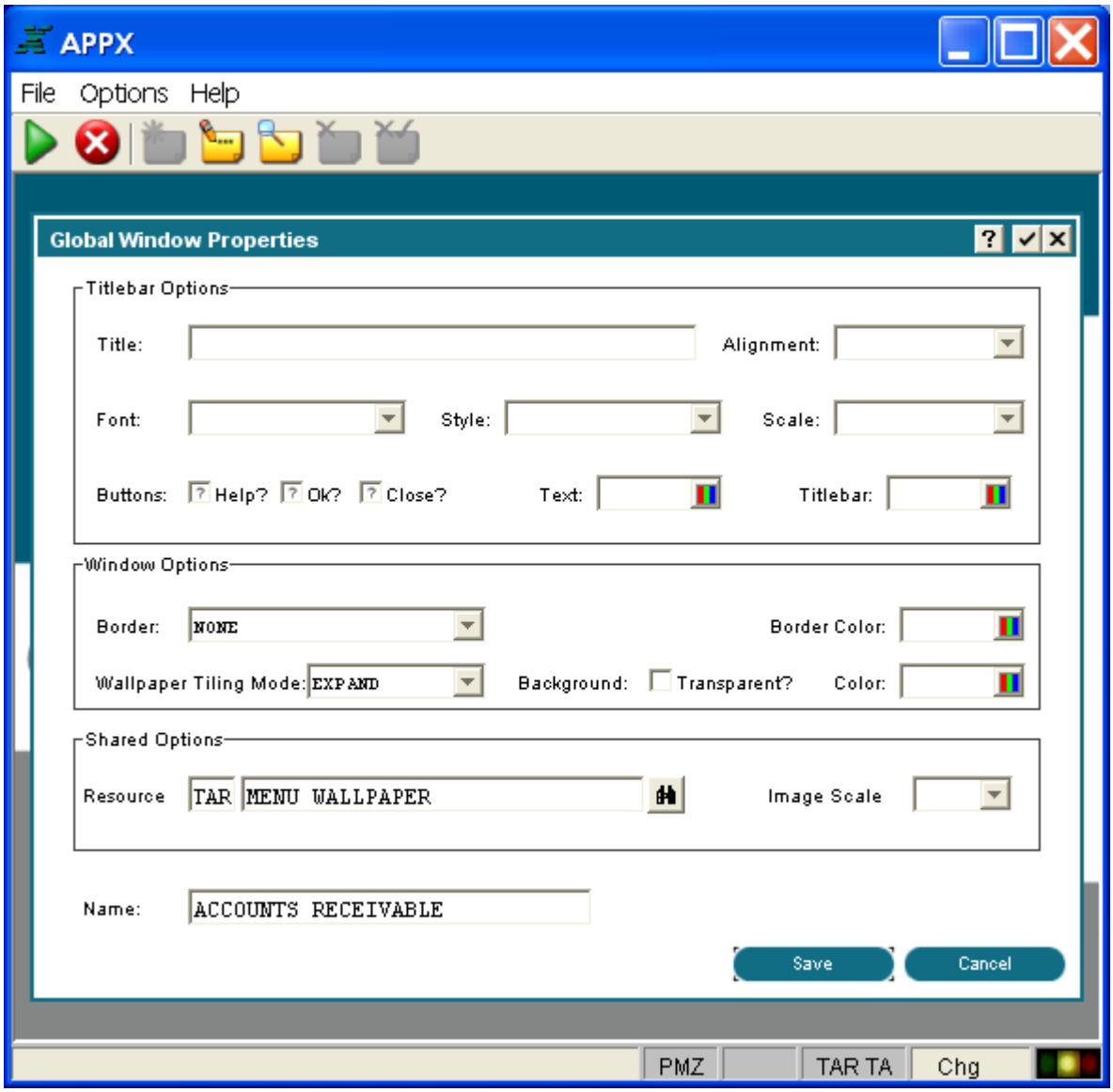


Figure 3-15-16. Window Properties Overlay Specifying Wallpaper

Figure 3-15-17 provides an example of an input with a title bar. The *GUI Attributes* overlay shown in *Figure 3-15-18* details how this title bar was defined. This overlay can be used to define either wallpaper or a title bar, or both.

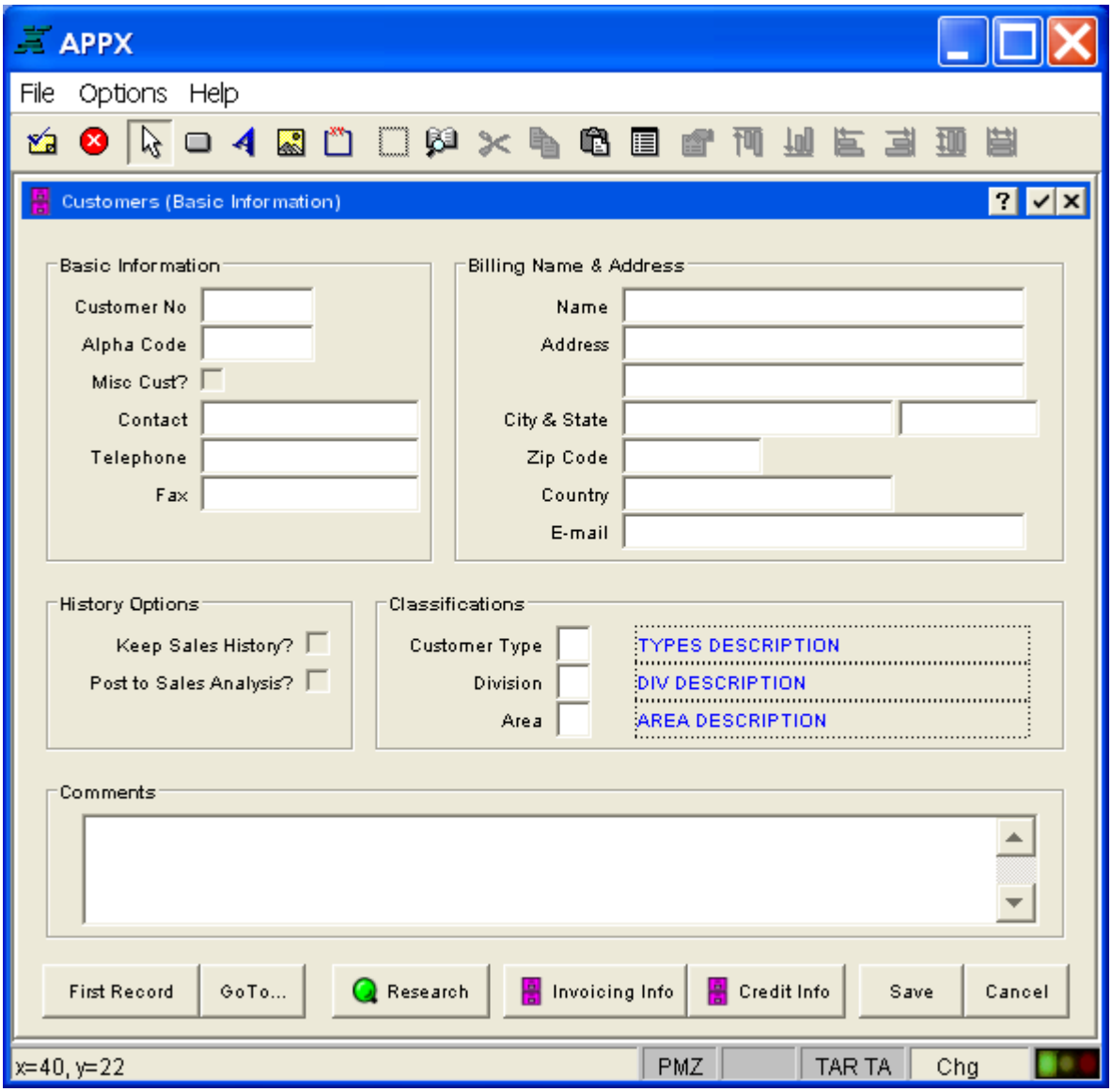


Figure 3-15-17. Input Image with a Title Bar

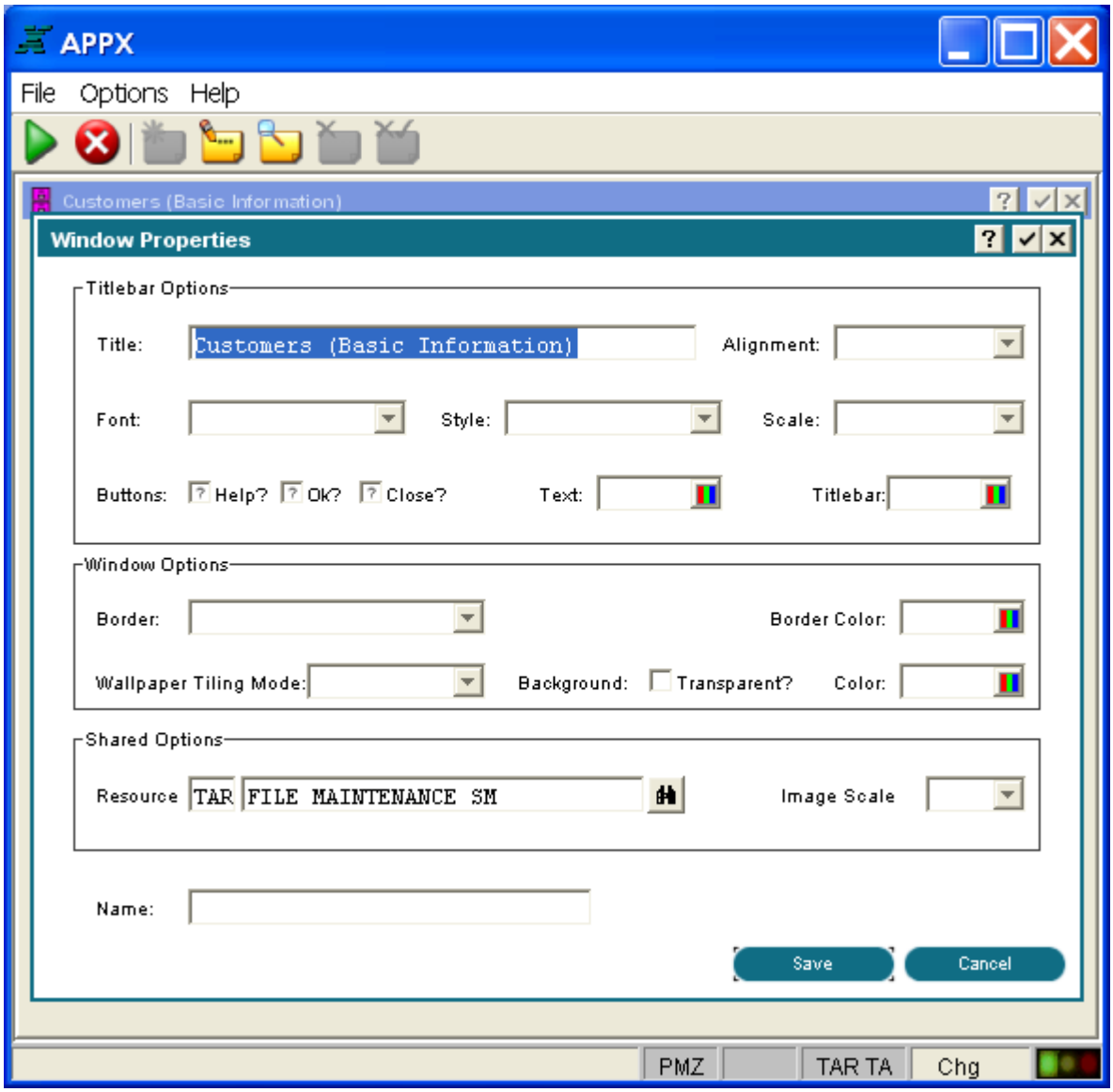


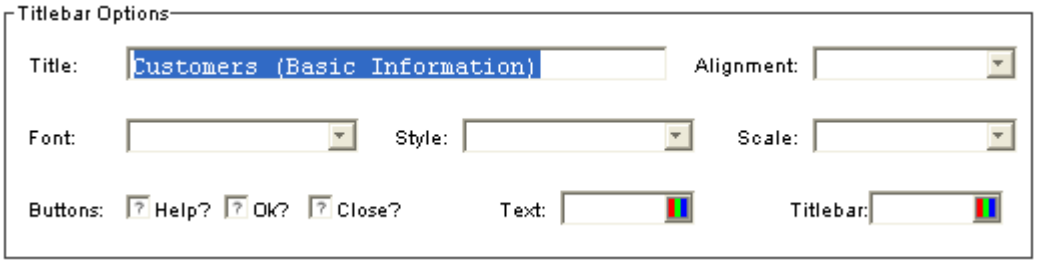
Figure 3-15-18. Window Properties Overlay Specifying Title Bar

The overlays shown in *Figure 3-15-16* and *3-15-18* contain many options that can be utilized when designing your image appearance including:

- ***Titlebar Options - See the Titlebar Options section for details. These options only apply to menus and inputs.***
- ***Window Options - See the Window Options section for details.***
- *Shared* ***Options - See the Shared Options and Name section for details.***
- *Name* ***- See the Shared Options and Name section for details.***

These options are defined in the following sections. When you have finished defining the graphical attributes for this image, you must save or cancel the attributes by selecting either the SAVE or CANCEL options located in the bottom right of the screen.

Title Bar Options



Titlebar Options from Figure 3-15-18

Titlebar options only apply to menu and input images and contain the following items:

Title

Is the menu or input process title that is displayed in the titlebar.

Alignment

Indicates whether the title should be left or right justified, or centered within the title bar. Click the list button to select your choice.

Font

Identifies the font (Courier, Arial, etc.) to be used for the title. Click the list button to select your choice. If left blank, the system default determined by your system administrator is used.

Style

Identifies the font style (bold, italic, etc.) to be used for the title. Click the list button to select your choice. If left blank, a regular font style is used.

Scale

Indicates a percentage by which to scale the default size of this font (50%, 150%, etc.). Click the list button to select your choice. If left blank, a default of 100% is used.

Show Help Button?

Specifies whether the HELP button will be shown in the right end of the titlebar.

Show OK Button?

Specifies whether the ENTER button will be shown in the right end of the titlebar.

Show Close Button?

Specifies whether the END button will be shown in the right end of the titlebar.

Text Color

Indicates the color of the title bar text. You can enter a specific color name or click the COLOR SELECTION icon to make the color selection. See the Making Color Selections section at the end of this chapter for more details.

Titlebar Color

Indicates the color of the title bar background. You can enter a specific color name or click the COLOR SELECTION icon to make the color selection. See the Making Color Selections section at the end of this chapter for more details.

Window Options

Window Options from Figure 3-15-16

Window Options apply to all images and contain the following items:

Border

Identifies the border style for the frame or image. Click the list button to view a list of choices.

Border Color

Indicates the color of the border. You can enter a specific color name or click the COLOR SELECTION icon to make the color selection. See the *Making Color Selections* section for more details.

Wallpaper Tiling Mode

Indicates where on the screen the wallpaper should be placed when it overlays another image. Click the list button to select from a list of choices that include:

TILE to tile the wallpaper graphic to fill the menu or input image.

EXPAND to expand or shrink the wallpaper graphic to fit the menu or input image.

CENTER to center the wallpaper graphic within the menu or input image.

NORTH, EAST, SOUTH, WEST, NORTHEAST, NORTHWEST, SOUTHEAST, or **SOUTHWEST** position the wallpaper graphic at the specified compass point on the menu or input image.

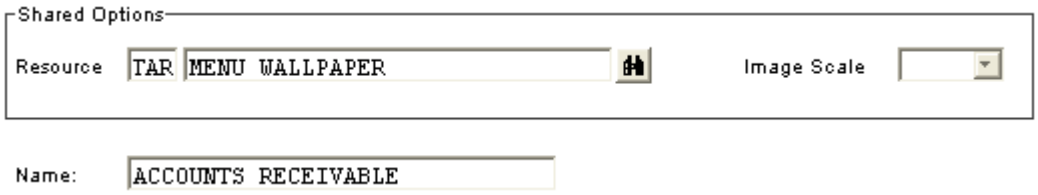
Background Transparent?

Indicates whether or not the background should be transparent when displayed. Click the checkbox to toggle between "yes" and "no" .

Background Color

Indicates the color of the background and is only applicable if you do not define a wallpaper in the *Resource* field. If you define both a wallpaper and a background color, the wallpaper will hide the background color selected. The wallpaper overlays the background color so, if the wallpaper does not totally fill the image, then the background color will appear around the wallpaper. You can enter a specific color name or click the COLOR SELECTION icon to make the color selection. See the *Making Color Selections* section for more details.

Shared Options and Name



Shared Options and Name from Figure 3-15-16

Shared Options and Name apply to all images and contain the following items:

Name

Is the name assigned to this GUI Attribute. The name is used for controlling the GUI object with ILF code so that, for example, certain image selections could be disabled by user id.

Access to the GUI attributes at runtime with ILF code is done via the --- WIDGET file. The WIDGET records from the process are copied into this file and are available in the *Pre-Display* event point. WIDGET NAME is a key to this file. You can read and rewrite records in this file to affect the GUI attributes at runtime.

Resource specifies

The application and name of a resource defined in Named Resources which links to one or more graphics files. Different graphics can apply depending upon resource state (disabled, enabled, rollover, pressed, etc).

Defining Graphical Attributes for Optional Children

Because *Optional Children* can appear in one or more places (pulldown menu, right-click context menu, etc.), graphical attributes can be defined to control how and where available optional child processes are listed on menu or input images. The GUI Attributes overlay for Optional Child Processes is accessed

from the Optional Children overlay shown in *Figure 3-15-19*. Graphical Attributes must be defined individually for each Optional Child process you define. They cannot be defined globally for all optional processes defined for a process.

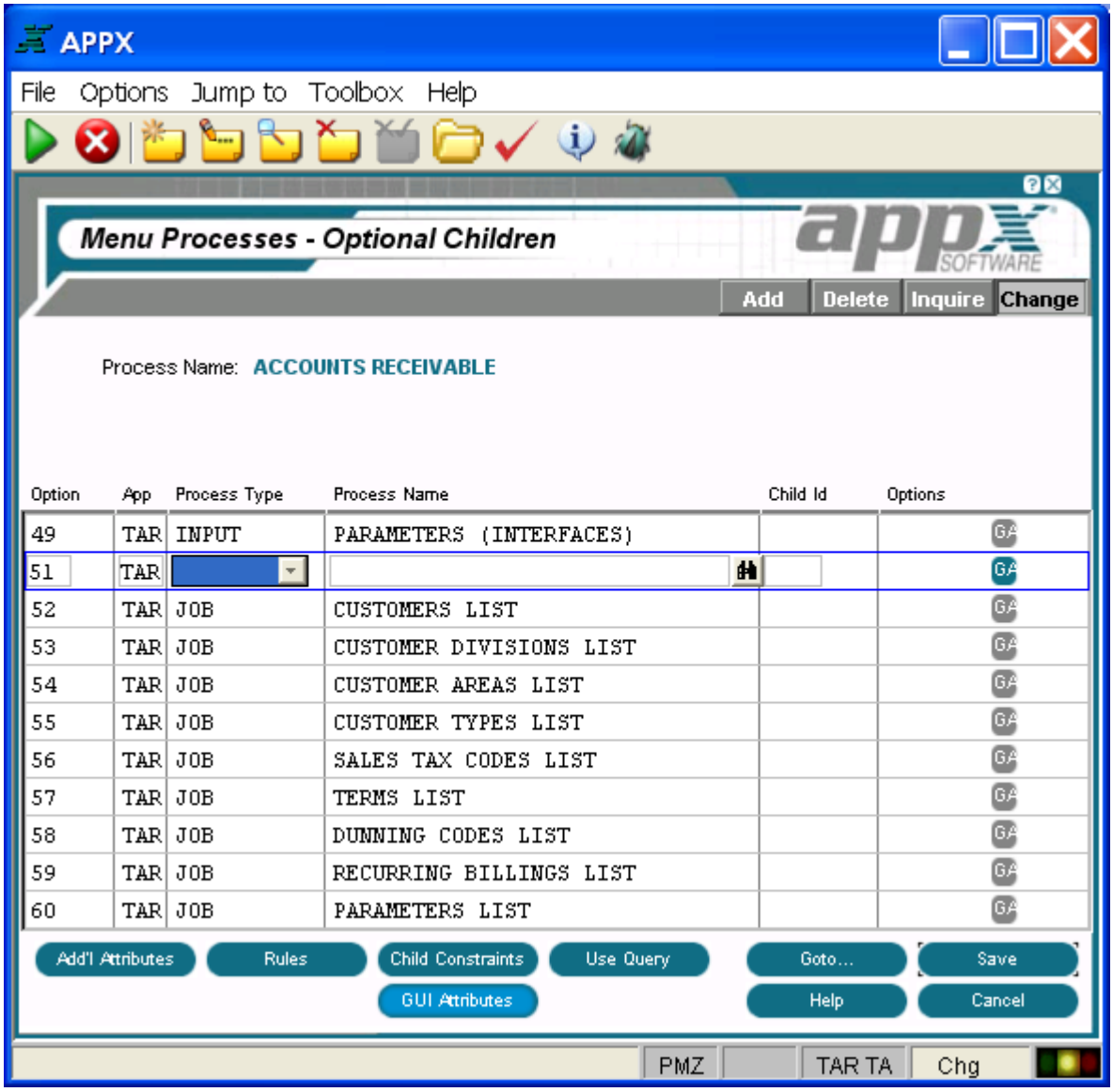


Figure 3-15-19. Optional Children Overlay for a Menu Process (from Figure 3-15-14)

When selecting the GUI ATTRIBUTES option from either a menu process or input process image, the Graphical Attributes overlay shown in *Figure 3-15-20* appears.

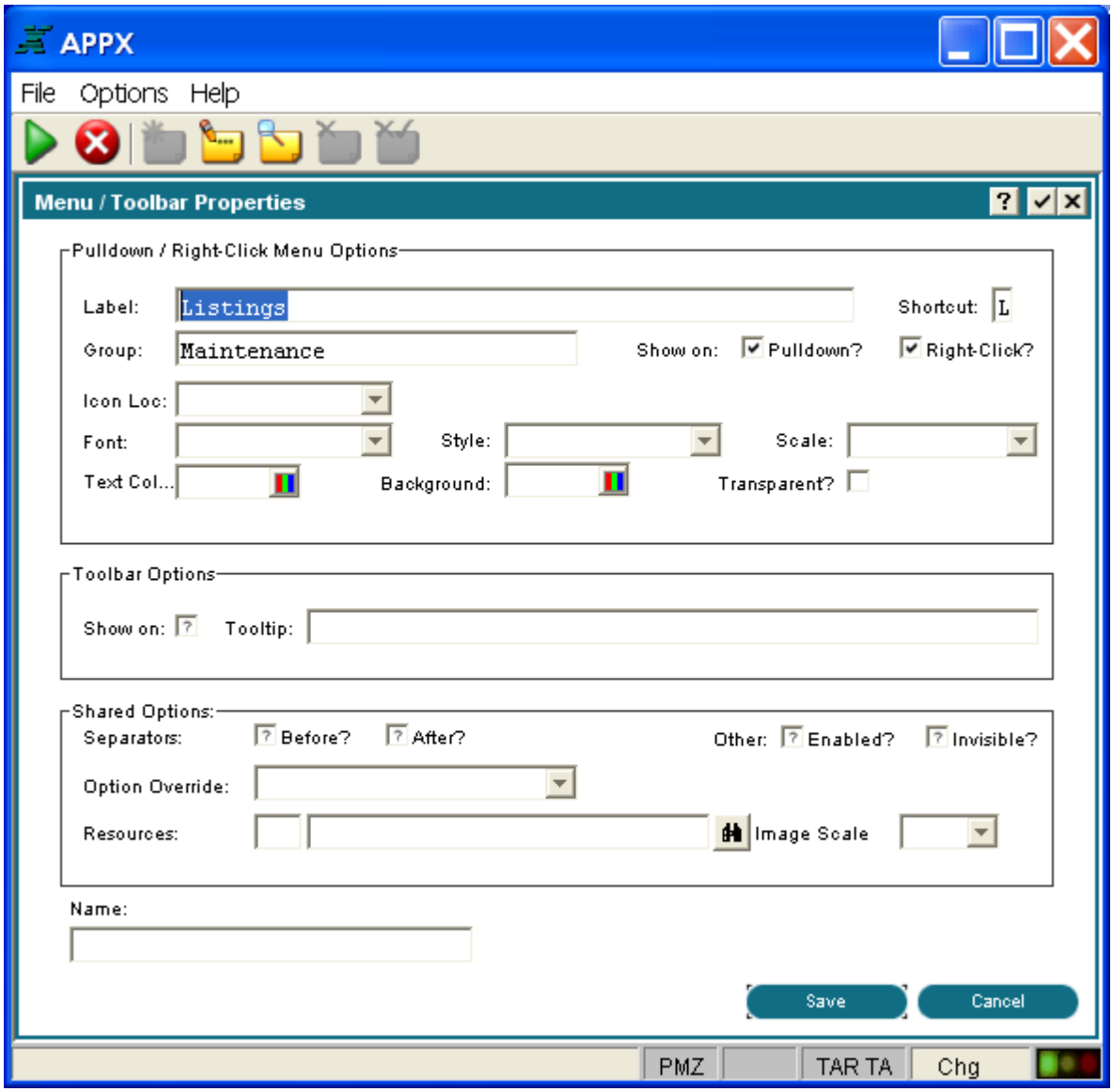


Figure 3-15-20. Optional Child Graphical Attributes Overlay (Figure 3-15-19 Option 51)

The Optional Child Graphical Attributes overlay contains the following items:

Pulldown/Right-Click Menu Options

Label

Is the descriptive text which appears on the menu or input image. If left blank, the descriptive name of the process invoked is used.

Shortcut

Is a single key on the keyboard that when pressed in conjunction with the ALT key, invokes this optional child process. The first occurrence of this key in the *Label* text will be underlined to tell the user what the *shortcut* key is. For example, if the *Label* is *Customers* and the *Shortcut* is *C*, then the *Label* will display as **Customers**.

Group

Refers to either the top-level name which appears on the pull-down menu or the *Label* of a previously defined child so that a cascading menu can be defined. In the example shown in *Figure 3-15-14*, all three items (*File Maintenance, Listings, Other Utilities)* on the *Maintenance* pull-down have a group designation of *Maintenance* as shown in *Figure 3-15-20.* Then, all ten of the items listed on the *Listings* pull-down menu have been assigned to the group *Listings (*as shown in *Figure 3-15-21*). Because *Maintenance* is the first level, there is no group designation defined as shown in *Figure 3-15-22*. Grouping these different menus in a multi-tiered fashion creates the cascade effect shown in the example. Items appear sequentially across the title bar from left to right and within the pull-down menus according to the *Optional Process Number* you assign to each process.

Show on Pull Down Menu?

Indicates if this optional child process should be displayed as a pull down menu. Click the checkbox to toggle between "yes" and "no" .

Show on Right-Click Menu?

Indicates if this optional child process should be displayed as a right-click, context menu. Click the checkbox to toggle between "yes" and "no" .

Icon Location

Indicates where the icon should be placed in relation to the *Label*. Click the list button to select from either LEADING or TRAILING.

Font

Identifies the font (Courier, Arial, etc.) to be used for the title. Click the list button to select your choice. If left blank, the system default determined by your system administrator is used.

Style

Identifies the font style (bold, italic, etc.) to be used for the title. Click the list button to select your choice. If left blank, a regular font style is used.

Scale

Indicates a percentage by which to scale the default size of this font (50%, 150%, etc.). Click the list button to select your choice. If left blank, a default of 100% is used.

Text Color

Indicates the color of the label text. You can enter a specific color name or click the COLOR SELECTION icon to make the color selection. See the *Making Color Selections* section for more details.

Background Color

Indicates the color of the label background. You can enter a specific color name or click the COLOR SELECTION icon to make the color selection. See the *Making Color Selections* section for more details.

Transparent?

Indicates whether or not the background should be opaque or transparent.

Toolbar Options

Show On

Indicates if this optional child will appear graphically on the tool bar. Note that if you define a graphical representation for this item (an icon, for example), then a resource with an enabled state icon must be specified in order for this child to appear on the tool bar.

Tooltip

Is 60 characters of help text that you can enter. The tooltip is displayed to users as a "hover over" when the mouse pointer is passed over the respective icon.

Shared Options

Separator Before?

Indicates if a separator line should be placed before this optional process label to better segregate labels into logical groups. Click the checkbox to toggle between "yes" ☑ and "no" ☐.

Separator After?

Indicates if a separator line should be placed after this optional process label to segregate labels into logical groups. Click the checkbox to toggle between "yes" ☑ and "no" ☐.

Enabled?

Indicates if this icon or button should initially be enabled and available for use by the user. The icon or button can also be enabled or disabled through the use of ILF code that references the *Name* field. Click the checkbox to toggle between "yes" ☑ and "no" ☐.

Invisible?

Indicates whether this icon or button should initially be invisible to the user. The icon or button can also be made visible or invisible through the use of ILF code that references the *Name* field. Click the checkbox to toggle between "yes" ☑ and "no" ☐.

- *Option Override?* allows the current option to be overridden at runtime with the option specified here. Click the list ▼ button to make your choice.

Resource specifies

The application and name of a resource defined in Named Resources which links to one or more graphics files. Different graphics can apply depending upon resource state (disabled, enabled, rollover, pressed, etc).

Image Scale

Defines the size of the graphic as a percentage of the runtime font size. Click the list button to make your choice.

Finally,

Name

Is the name assigned to this GUI Attribute. The name is used for controlling the GUI object with ILF code so that, for example, certain selections could be disabled by user id.

Access to the GUI attributes at runtime with ILF code is done via the --- WIDGET file. The WIDGET records from the process are copied into this file and are available in the *Pre-Display* event point. WIDGET NAME is a key to this file. You can read and rewrite records in this file to affect the GUI attributes at runtime for the current frame only.

When you have finished defining the graphical attributes for this image, you must save or cancel the attributes by selecting either the SAVE or CANCEL options located in the bottom right of the screen.

Other examples of pulldown menu specifications are provided in the figures below:

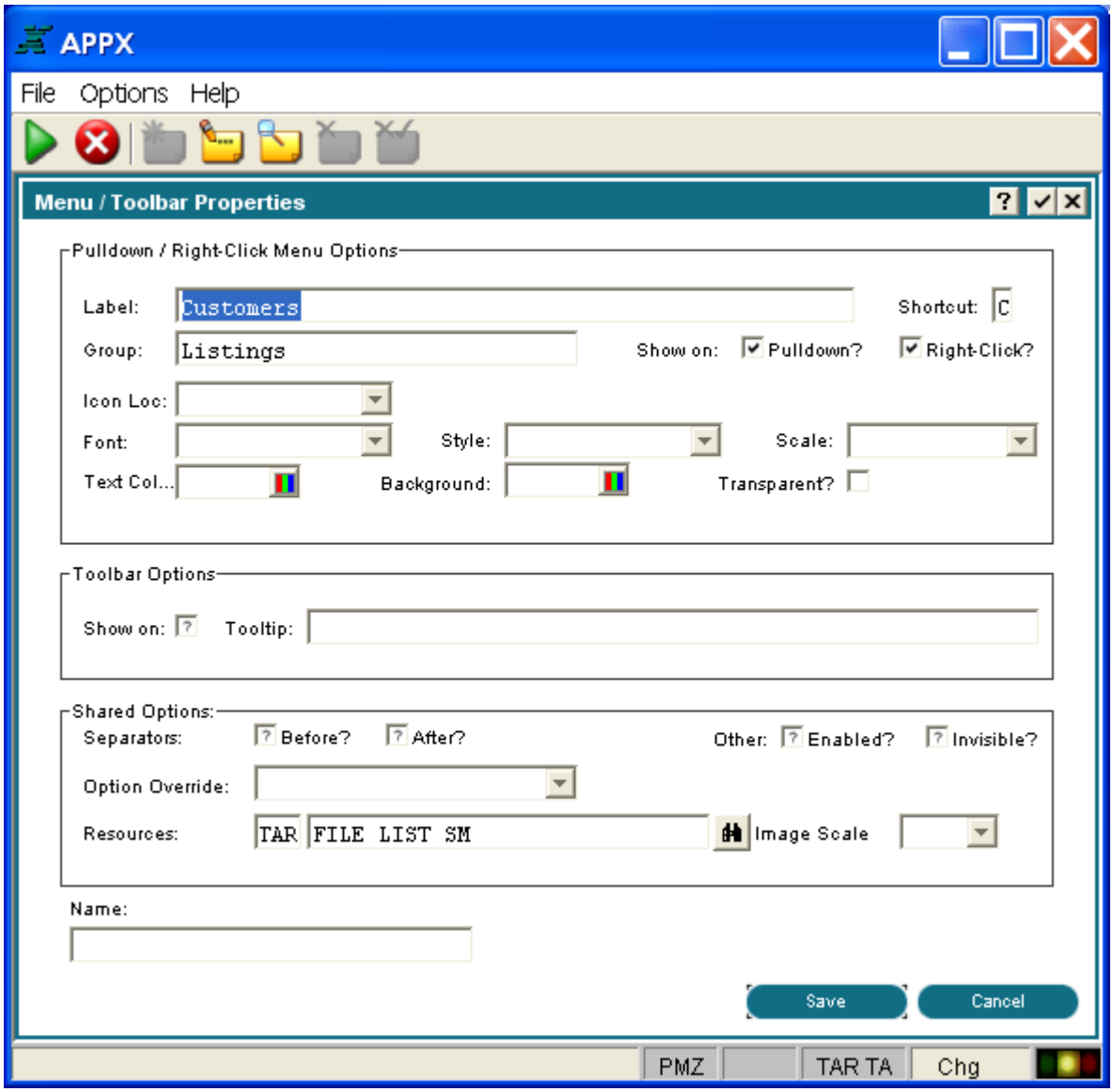


Figure 3-15-21. Final Level of a Cascading Pull-Down Menu (Figure 3-15-19 Option 52)

First Level of a Cascading Pull-Down menu

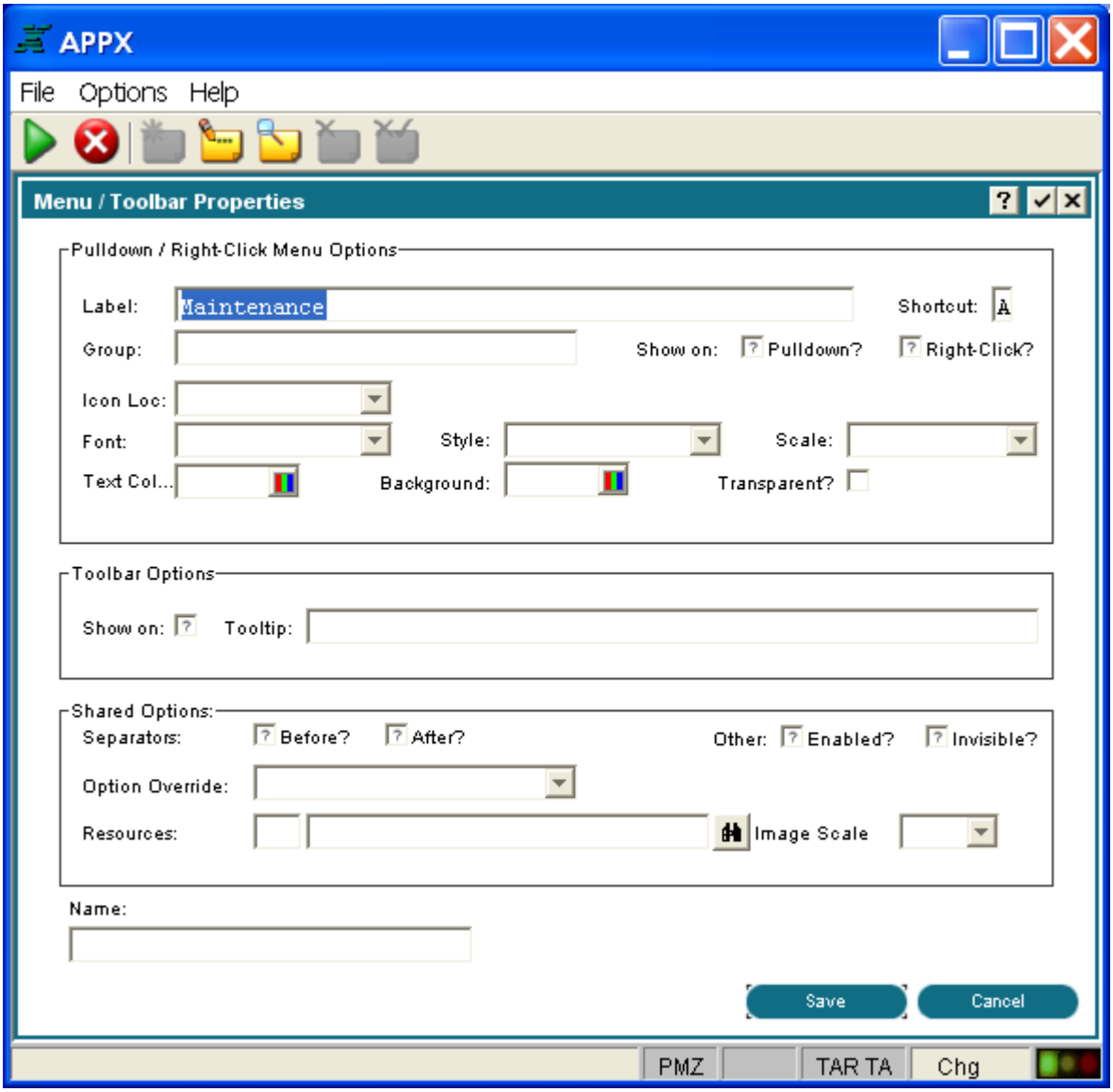


Figure 3-15-22

Defining Graphical Attributes for Image Fields

Image fields can also be displayed with a variety of graphical attributes. You can use a variety of *Widgets* to represent certain field types. You can also display the contents of a field using different fonts, font styles, colors, and other attributes. The Graphical Attributes overlay for image fields is accessed from the *Item Specifications* overlay in the Image Editor.

From the Item Specifications overlay, you select the GUI ATTRIBUTES option to display the *GUI Attributes* overlay used to define the GUI Attributes for this field. There are five different overlays, one for each major field type:

- **Alpha Items** (see *Figure 3-15-26*)
- **Date Items** (see *Figure 3-15-27*)
- **Logic Items** (see *Figure 3-15-28*)
- **Numeric Items** (see *Figure 3-15-29*)
- **Text Items** (see *Figure 3-15-30*)

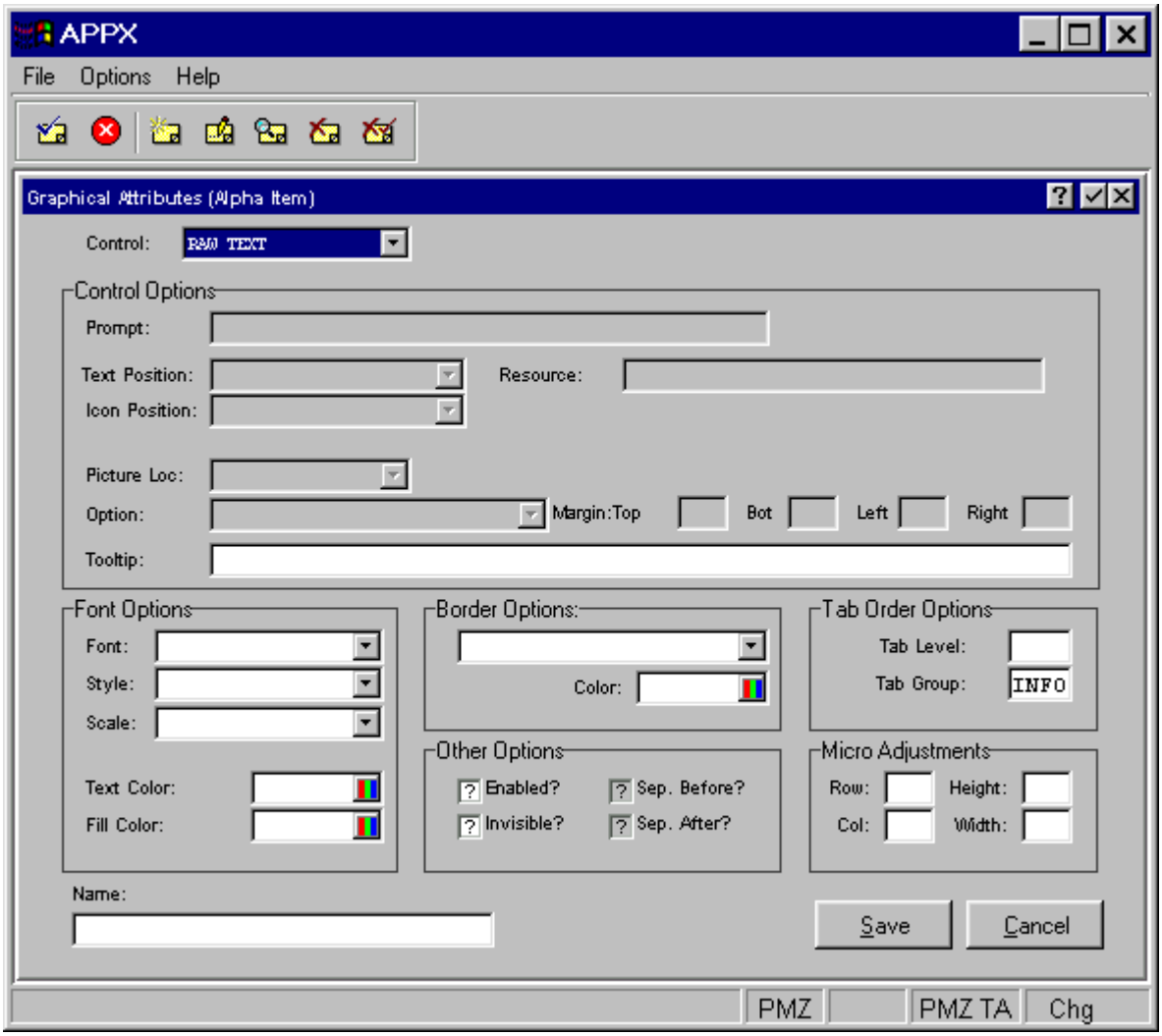


Figure 3-15-26. Graphical Attributes Overlay For Alpha Items

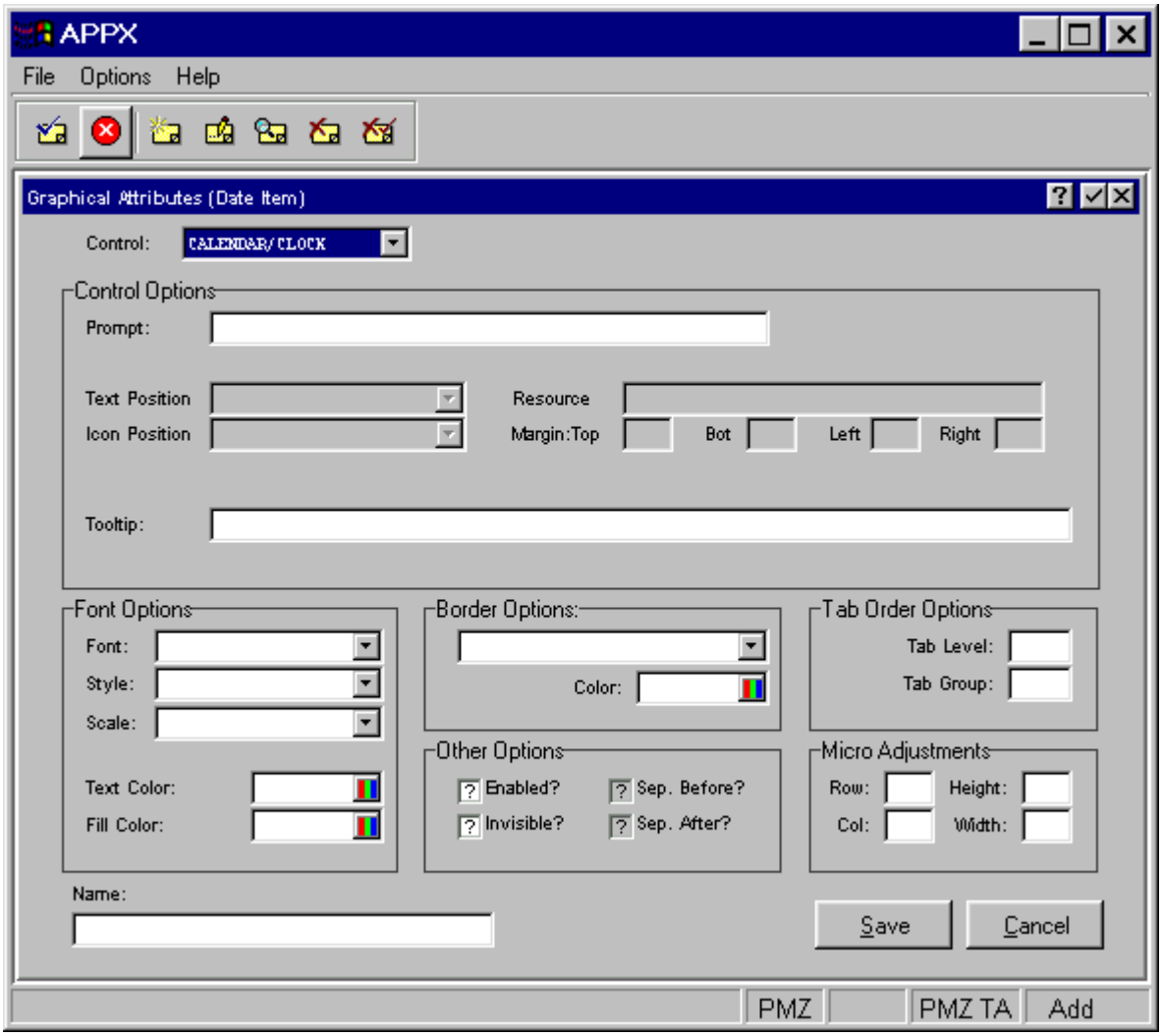


Figure 3-15-27. Graphical Attributes Overlay For Date Items

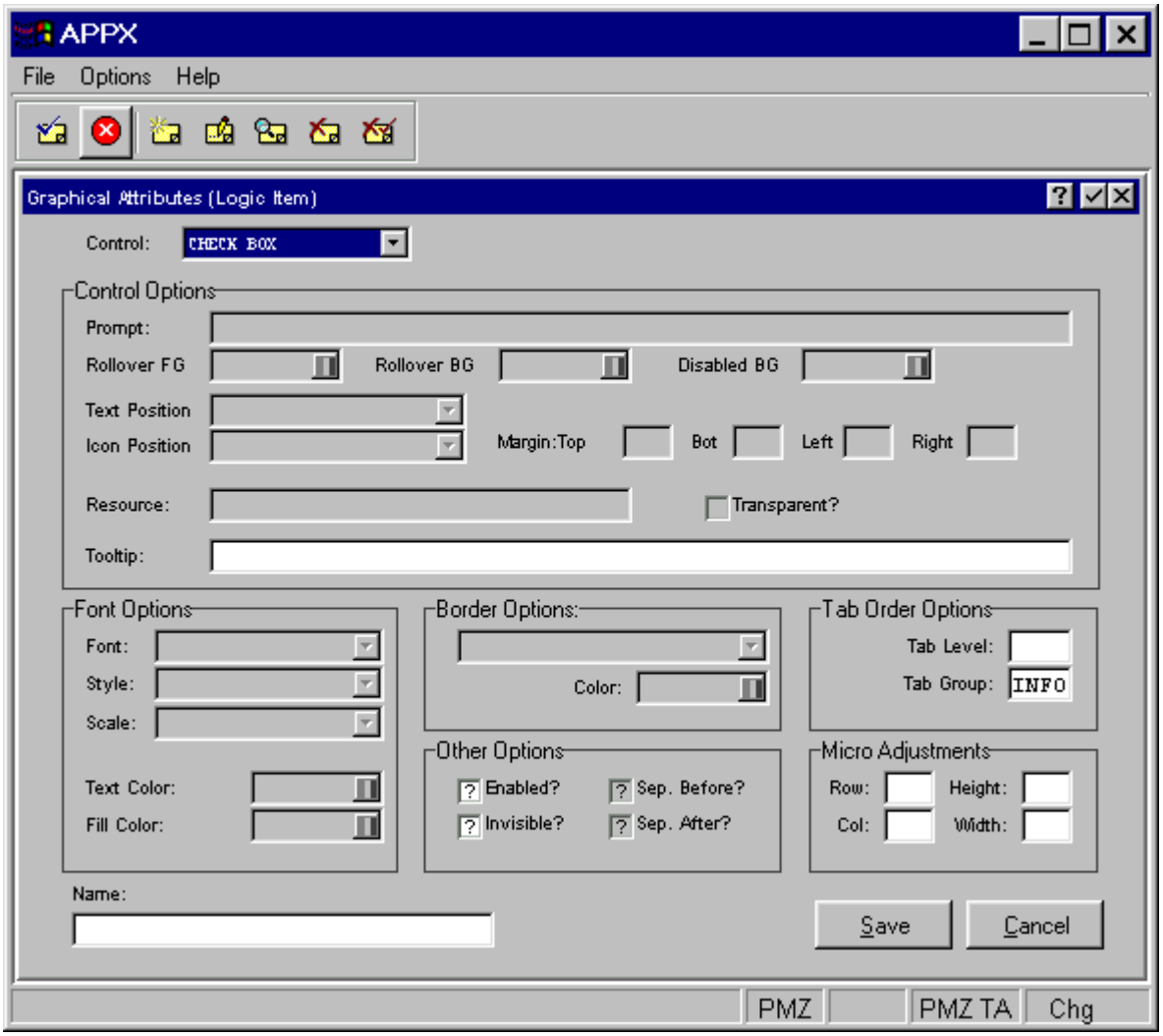


Figure 3-15-28. Graphical Attributes Overlay For Logic Items

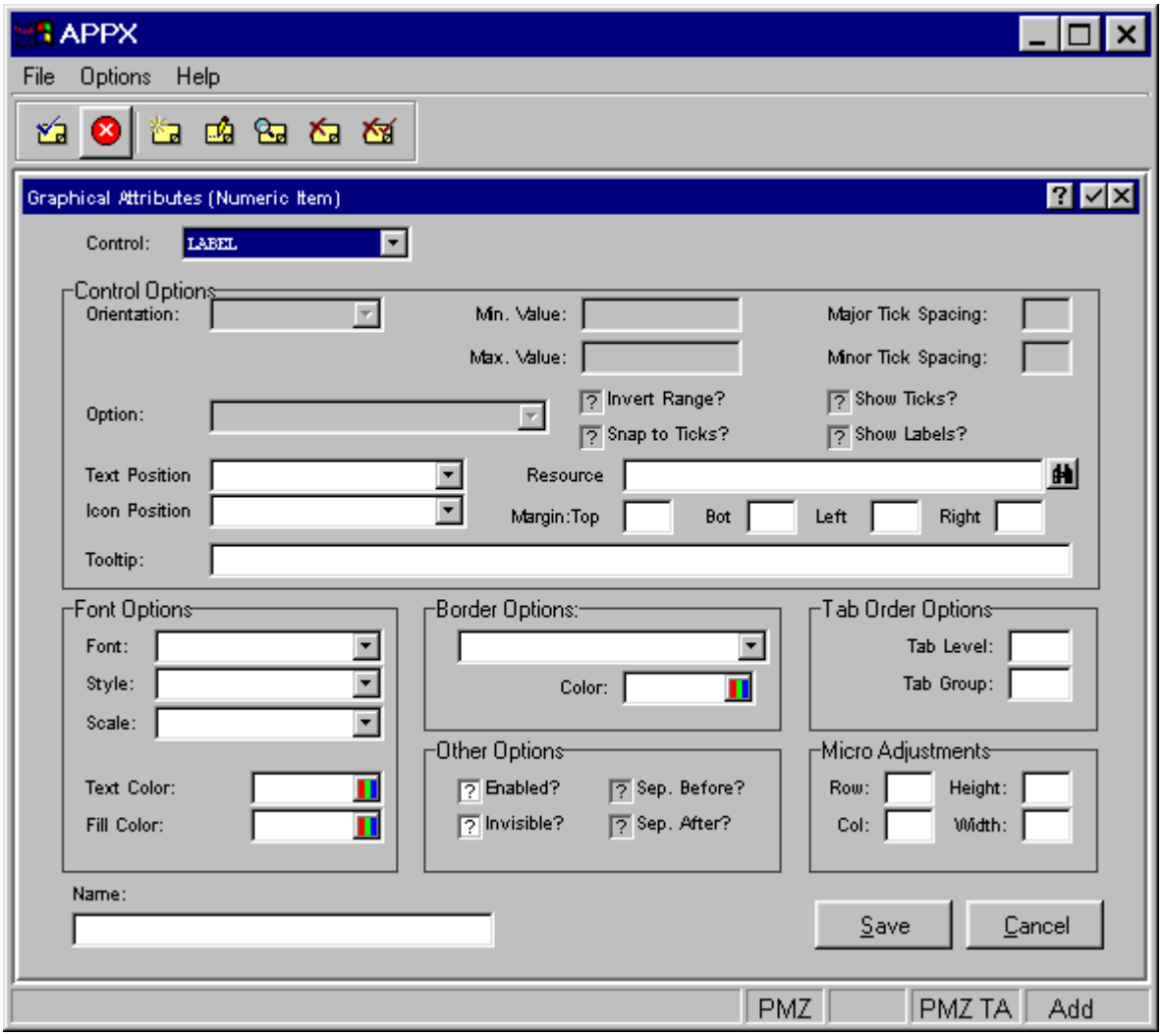


Figure 3-15-29. Graphical Attributes Overlay For Numeric Items

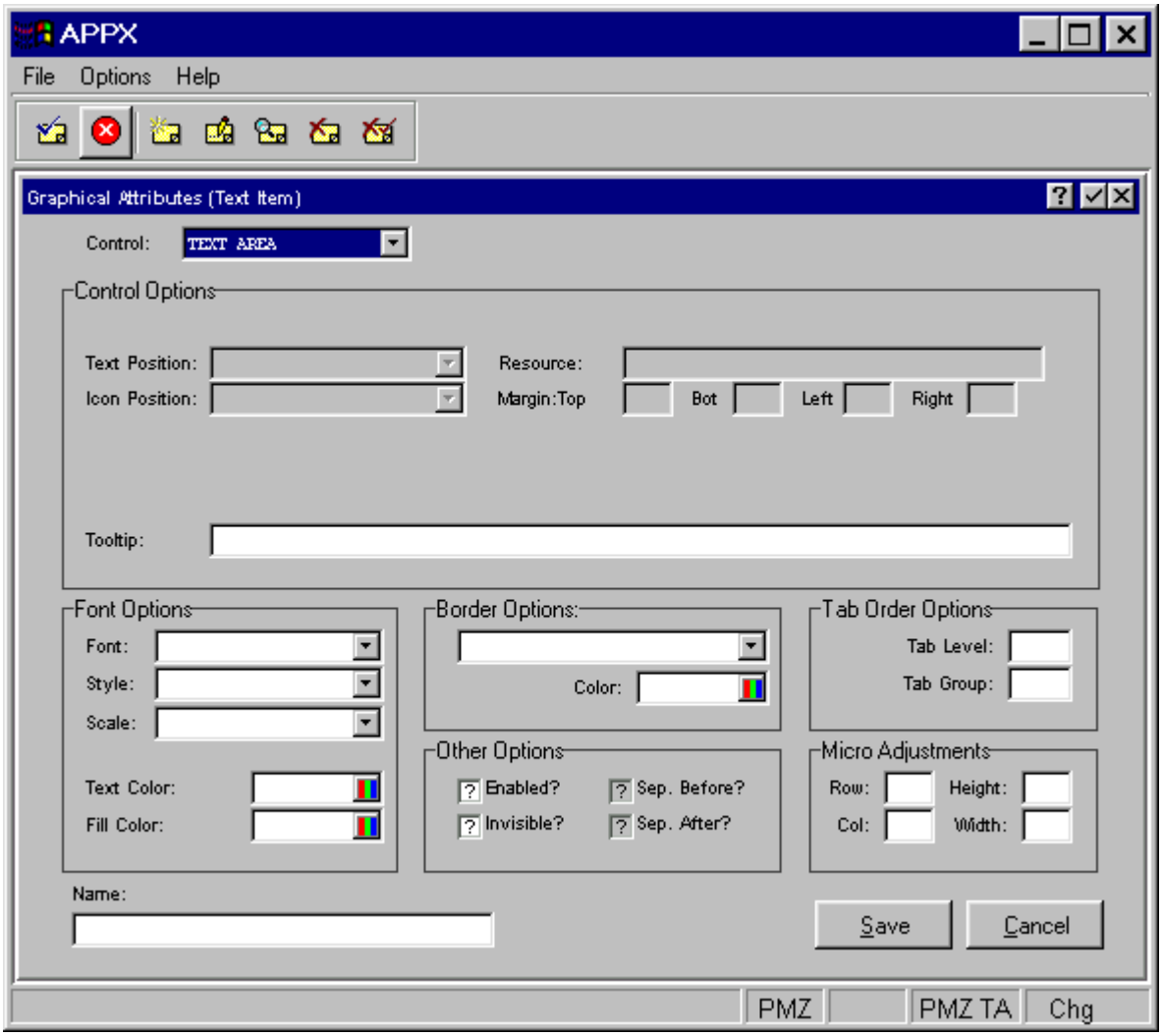


Figure 3-15-30. Graphical Attributes Overlay For Text Items

Some or all of these fields can be defined on the overlay depending on the type of field you are defining attributes for:

Control

Indicates the graphical representation of this field. Choices vary by field type but can include:

- **Raw Text** to indicate no graphical representation.
  Calendar/Clock to allow the user to select a date from a calendar.
  Check Box to display logic fields as a checkbox that toggles from "yes" ☑ to "no" ☐ and, optionally, to "unknown" ? (if entry is not required).
  Toggle Button to display logic fields as a button that toggles from "yes" ☐ to "no" ☐ ("unknown" is treated as a "no").
  HTML Viewer to display text fields in a scrolling edit box.
  RTF Viewer to display text fields in a scrolling edit box.

Control Options

Are described in the Image Field Control Options section.

Font Options

Are described in the Image Field Font Options section.

Border Options

Are described in the Image Field Border Options section.

Other Options

Are described in the Image Field Other Options section.

Tab Order Options

Are described in the Image Field Tab Order Options section.

- *Micro Adjustments* -- Row, Col, Height and Width are entered as a percentage of an image cell (1 row x 1 column) and must be less than or equal to 100. These fields can be used to make slight adjustments to the size and position of the data object.

Name

Is the name assigned to the GUI Attributes for this item and can be used to access the attributes at runtime as described below. You can then control the GUI object with ILF code so that, for example, certain selections could be disabled by user ID

Access to the GUI attributes at runtime with ILF code is done via the --- WIDGET file. The WIDGET records from the process are copied into this file and are available in the *Pre-Display* event point. WIDGET NAME is a key to this file and can be set to the value specified in *Name*. You can read and rewrite records in this file to affect the GUI attributes at runtime.

Image Field Control Options

> Note that available options differ depending on the control type selected.

Prompt

Indicates the text to appear as the label appropriate for this design control type. For example, at the frame level this would be titlebar text. At the button level, it would be button label text. For an optional child, it is the text that appears on the menu but if left blank, the processes' descriptive name is used.

Text Position

Indicates where the text should appear inside the button. Click the list button for a drop-down list of selections that include: CENTER, TOP, RIGHT, BOTTOM, LEFT, UPPER RIGHT, UPPER LEFT, LOWER RIGHT and LOWER LEFT.

R

Esource specifies the application and name of a resource defined in Named Resources which links to one or more graphics files. Different graphics can apply depending upon resource state (disabled, enabled, rollover, pressed, etc). Select the SCAN icon for a list of available resources.

Icon Position

Indicates where the defined icon (if any) should be placed in relation to the field contents. Click the list button for a drop-down list of selections that include BEHIND TEXT, ABOVE TEXT, TRAILING TEXT, BELOW TEXT, LEADING TEXT, ABOVE & TRAILING, ABOVE & LEADING, BELOW & TRAILING, BELOW & LEADING, EXPAND TO FILL, and EXPAND PROP'L.

Picture Loc

Indicates the location of the graphic to be used. Click the list button for a list of available locations that include: CLIENT URL, CLIENT FILE, DIRECT, SERVER URL, SERVER FILE, and DESIGN FILE.

Transparent Background?

Indicates whether or not this field should have a transparent background. Select "yes" or "no" .

Option

Indicates the option number, if any, to be sent to the server by the client whenever the contents of this field are changed. This is useful in situations where you want other changes made to the screen based on the contents of this field. Click the list button for a list of available options.

Tooltip

Is 60 characters of help text that you can enter. The tooltip is displayed to users as a "hover over" when the mouse pointer is passed over the respective icon.

Image Field Font Options

Font

Identifies the font (Courier, Arial, etc.) to be used for the label. Click the list button to select your choice.

Style

Identifies the font style (bold, italic, etc.) to be used for the label. Click the list button to select your choice.

Scale

Indicates a percentage by which to scale the default size of this font (50%, 150%, etc.). Click the list button to select your choice.

Text Color

Indicates the color of the label text. You can enter a specific color name or click the COLOR SELECTION icon to make the color selection. See the *Making Color Selections* section for more details.

Background Color

Indicates the fill color of the label background. You can enter a specific color name or click the COLOR SELECTION icon to make the color selection. See the *Making Color Selections* section for more details.

Image Field Border Options

Border Style

Indicates the style for the border around an object (Simple Line, Simple Line Wider, Etched Sunken, etc.). Click the list button to select your choice.

Color

Indicates the color of the default border. You can enter a specific color name or click the COLOR SELECTION icon to make the color selection. See the *Making Color Selections* section for more details.

Image Field Other Options

Enabled?

Indicates if this optional icon should initially be enabled and available for use by the user. The icon can also be enabled or disabled through the use of ILF code that references the *Name* field. Click the checkbox to toggle between "yes" and "no" .

Invisible?

Indicates whether this optional icon should initially be invisible to the user. The icon can also be made visible or invisible through the use of ILF code that references the *Name* field. Click the checkbox to toggle between "yes" and "no" .

Separator Before?

Indicates, for a data object on a scrolling image, if a separator line should be placed before this object to better segregate objects into logical groups. Click the checkbox to toggle between "yes" and "no" .

Separator After?

Indicates, for a data object on a scrolling image, if a separator line should be placed after this object to better segregate objects into logical groups. Click the checkbox to toggle between "yes" ☑ and "no" ☐.

Image Field Tab Order Options

Tab Level

Defines the physical pass over the whole screen with respect to tab order. If left blank, the data object is considered to be in tab level 0. APPX will tab from top-left to bottom-right for all data objects in tab level 0. Then it repeats the sequence for all button objects in tab level 0. The cycle is repeated as necessary for each successive tab level. This behavior can be altered by the specification of Tab Groups.

Tab Group

Is used to mark related data objects and buttons so that they are grouped together as part of a tab order. When making a pass over the screen for a particular "Tab Level", if a data object is encountered that has a non-null "Tab Group", then the tab order proceeds to all the data objects and buttons that are part of that Tab Level/Tap Group. After reaching the last data object or button in the group, the order will continue where it left off prior to encountering the group. Note that for "Tab Groups", data objects and buttons are consolidated into the same top-left to bottom-right pass.

Making Color Selections

You can make color selections for graphical attributes by using one of three methods:

Swatch

To select a specific color from a collection of color swatches (see *Figure 3-15-31)*. Click on the desired swatch to select the color.

HSB

To define a color based on a combination of hue, saturation and brightness (see *Figure 3-15-32).* To select each of the three color components, click the appropriate radio button and use the slider to increase or decrease the component's factor.

Hue

Is the actual color on a scale that starts and ends at red (0 and 359). Yellow is 60, green is 120, etc.

Saturation

Is the purity of the color, measured in percent from 0 to 100. At 0% saturation, hue is meaningless.

Brightness

Is measured in percent from black (0) to white (100). At 0% brightness, both hue and saturation are meaningless.

RGB

To mix a color by selecting the quantity of red, blue and green (see *Figure 3-15-33)* from the three respective sliding scales that range from 0 to 255.

To use one of the three methods, click on the appropriate tab (SWATCHES, HSB OR RGB) at the top of the *Select Color* overlay. The swatch method is the default method.

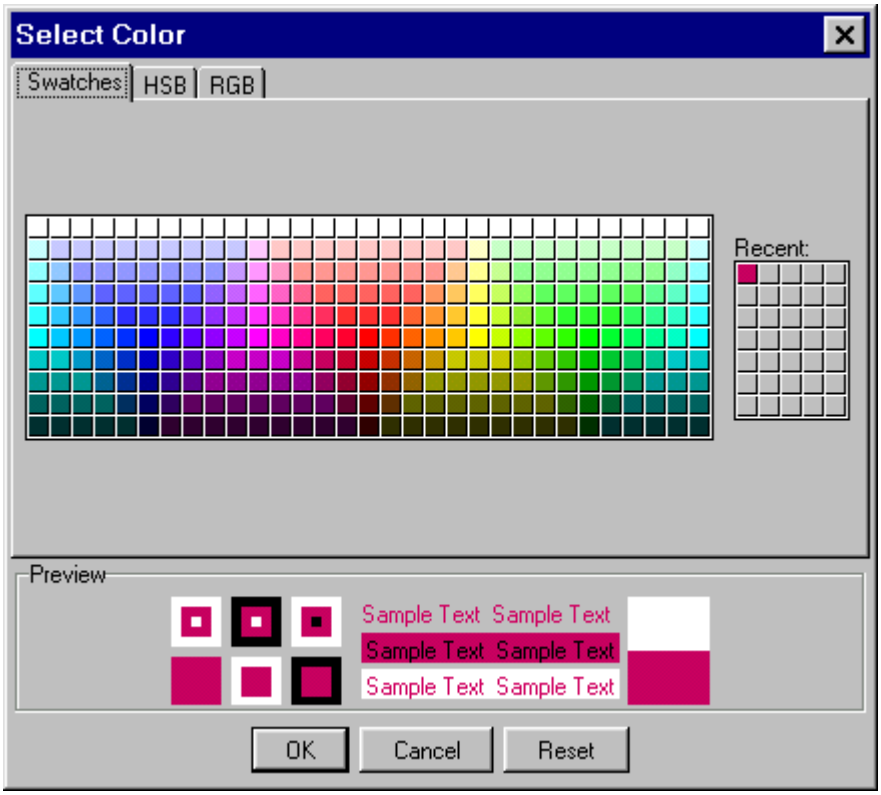


Figure 3-15-31. Swatch Method of Color Selection

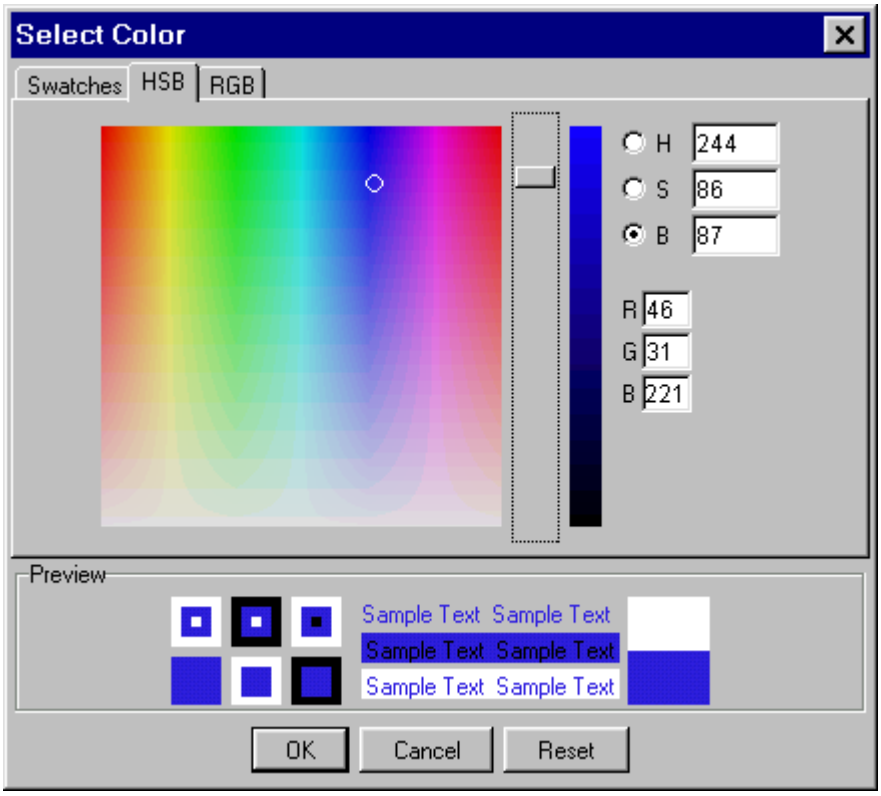


Figure 3-15-32. Hue, Saturation and Brightness (HSB) Method of Color Selection

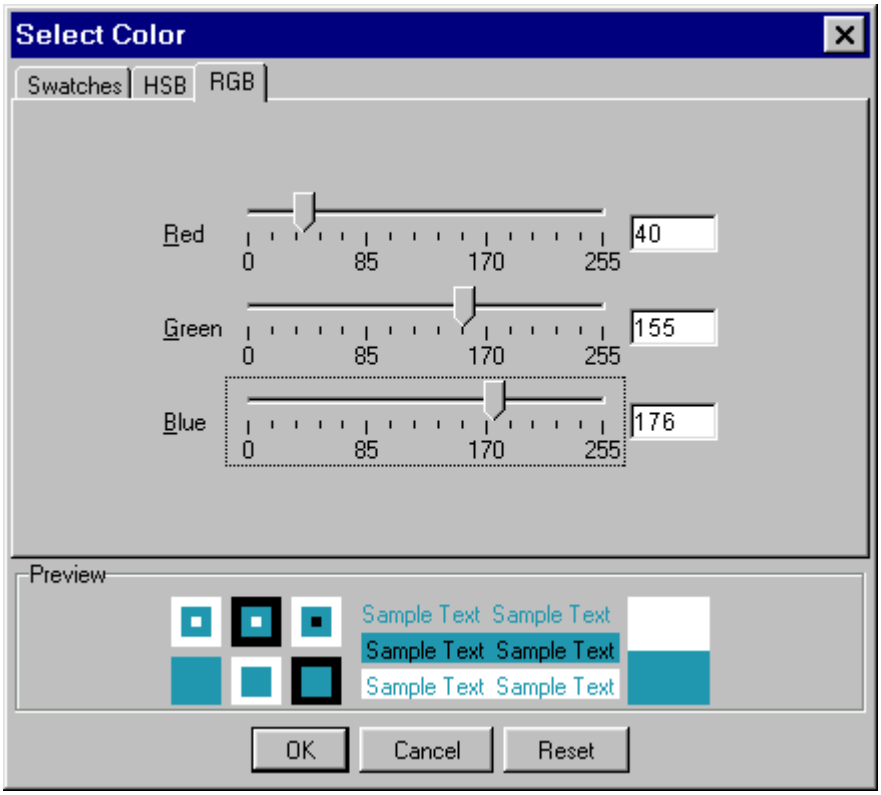


Figure 3-15-33. Red, Green, and Blue (RGB) Method of Color Selection

After selecting the desired color, each of the three color selection method overlays provide three options:

- OK to apply the selected color.
- CANCEL to cancel the color selection process and return to the *Additional Attributes* overlay, restoring the original color selection.
- RESET to restore the original color selection and make a different selection using the same or alternate color selection method.

CHAPTER 3-16

Using Event Points

Event Points

APPX *event points* provide opportunities to insert procedural code, or *statements,* from the *Integrated Language Facility (ILF)* into the execution sequence of a process. They are entered through a specialized editor that validates syntax at entry time. Refer to *Chapters 4-1* thru *4-7* of this manual for definitive information on APPX's integrated procedural language.

Event points permit you to respond effectively to unusual or unique application requirements that are not addressed by APPX specifications. Each event point can contain a sequence of statements that execute at a specific, predetermined point within an APPX process. Commonly used groups of statements can be defined once in a subroutine process, and then accessed in an event point or executed as an automatic or optional child.

As indicated in Chapter 3-1, event points operate at four levels: process, frame, image, and child invocation. In addition, each type of process varies slightly in the specific event points it includes at each level. Table 3-1-3 summarizes the event points that execute for each process type.

The following sections define each event point, including the chronology according to which it executes, in general terms. You should note, however, that event point execution can be affected by a number of factors, including the process type, the mode and interactive phase, the existence of ancestor and descendent processes, and potential interdependencies between statements in other event points.

Because of the variety of ways you can use event points, it is important to step through the execution sequence of your process using the ILF debugger, verifying the placement of ILF routines. Indications that you may have used the wrong event point to perform a desired action include actions that are never performed, conditional actions that are performed in every instance, and an action performed at any time other than that intended.

Process-Level Event Points

Process-level event points include *Start of Process/Subroutine/Status, Set Run-Time Defaults, Post PCF Read, Pre-PCF Update, Post PCF Update, Start of Query Execution, Establish PCF Range, Pre-User Selection, Post User Selection, End of Query Execution,* and *End of Process*. File-level event points will also be executed for the process control file, and any file that is accessed with statements at the process level, including the PCF.

Start of Process

The *Start of Process* event point (also referred to as the *Subroutine* event point in a subroutine process or *Status* event point in a status process) executes once for a process, each time the process invokes. This means that, for a child process, the *Start of Process* event point for the child executes once for every record in the parent process. For example, in an order entry application in which an order input process invokes a child line item input process, the *Start of Process* event point in the line item process executes once for each order.

Start of Process executes before APPX determines the mode. As a result, you can include statements to modify the mode default by setting the value in the predefined field DEFAULT MODE.

All processes include the *Start of Process* event point. For a query process, *Start of Process* executes as a part of query setup (not query execution).

Start of Process is often used to:

- Read information from related files (a parameters or descriptions file, for example) to make the information available for the rest of a process.
- Verify the existence of required applications and files, and inform the user and cancel the execution if required data or files are not available. For example, if an accounts payable invoice post process updates balances in the general ledger application, this event point would check for required general ledger files.
- Perform actions that would be redundant if performed anywhere else.

Set Run-Time Defaults

The *Set Run-Time Defaults* event point executes for input processes only. It executes before *Start of Frame* for the first frame of a process in the data addition or key entry interactive phases. In the data addition interactive phase, it executes before a record is entered. In the data deletion, data inquiry, or data modification interactive phases, it executes before a key is entered on the first image.

Immediately before executing this event point, APPX restores the default record into the current record area that is in memory. (The default record is automatically maintained for any fields that are painted on any image including, but not limited to, fields in the process PCF.)

Set Run-Time Defaults is used when you wish to supply a user with a particular value, or in some other fashion to anticipate a user's requirements. It can be used to:

- Explicitly modify values in the default record, in data addition interactive phase. For example, you could use this event point to set a transaction date equal to the current date, establish a transaction number based upon a parameterized value, or supply report setup values in input processes that precede output processes. While APPX can automatically establish a static default value based on data dictionary specifications, defaults are often dynamic, based upon variable information that is

maintained elsewhere in the system. It is for these dynamic defaults that the *Set Run-Time Defaults* event point is used.

- Establish values for items that are used later in a process. For example, in a transaction entry input function, you might use this event point to set control group or batch information such as the operator ID or the batch number.
- Default portions of a key field before a user enters the key in delete, inquire, or change modes during key entry interactive phase.

Post PCF Read

The *Post PCF Read* event point executes immediately after APPX reads or rereads a record from the process control file.

APPX reads PCF records after the key entry interactive phase completes and APPX switches to the data modification, data deletion, or data inquiry interactive phase (depending upon the selected mode). It is not executed in the data addition interactive phase.

APPX rereads a record when a user reverse-navigates, using the *Previous Image* option from an automatic child process, back to a parent process (for example, moving from order line items back to the order header). It also rereads a record when it executes an additional frame for a process after executing an automatic child process (for example, entering order "footer" information after processing the final line item record for an order).

By implication, execution of this event point is dependent upon the existence of a PCF. If there is no PCF, or if the PCF is a "pseudo" file, this event point is not executed. It is found in input, output, and inquiry processes only.

Post PCF Read can be used to:

- Save the values of fields before modification by a user. You can then compare the saved values with any new values during *Pre-PCF Update* or *Post PCF Update*, and perform any necessary processing as a result. For example, you might save the quantity on a line item and, in *Post PCF Update*, compare the current quantity with the original quantity. If the quantity changes, you might execute statements to update committed inventory.

Pre-PCF Update

The *Pre-PCF Update* event point executes when a record is changed, added, or deleted. As a result, its execution is dependent upon the existence of a PCF. If no PCF file is designated, this event point does not execute. It also does not execute in inquire mode, or in change mode if the record is not modified in any manner by the user or event point code.

Pre-PCF Update is found exclusively in input processes, and executes only in the data addition, data modification, and data deletion interactive phases. It executes immediately before the PCF record is saved, and is followed by *Post PCF Update* immediately after the record is saved.

Because the event point only executes when a record is being saved or deleted, you can use this event point to perform actions on the PCF record you are saving. It is generally restricted to performing actions in the PCF record itself because the operating system has a hold on the PCF record. Normally, you should separate manipulation of PCF records from manipulation of records in other files. Use this event point to perform any required manipulation of the PCF record. Use *Post PCF Update* to manipulate records in other files.

Post PCF Update

The *Post PCF Update* event point is triggered by a change to a record. As a result, its execution is dependent upon the existence of a PCF. If the PCF is not stored, APPX still senses the change and executes the event point.

Post PCF Update is found exclusively in input processes, and executes only in the data addition, data modification, and data deletion interactive phases. It executes immediately after the PCF record is saved.

Post PCF Update is generally used to perform manipulation of records in files other than the PCF. Remember that *Pre-PCF Update* is used to perform actions on the PCF when the operating system has a hold on the PCF record. By the time APPX executes *Post PCF Update*, the system has saved and relinquished control of the PCF record. If you wish to manipulate a PCF record in this event point, you must reread and hold it.

Because the event point only executes when a PCF record has just been saved, you can use this event point to perform actions on non-PCF records that depend upon the PCF record you saved. Examples include resetting a journal printed flag to *N* whenever a user changes transaction information related to the journal, and updating a committed quantity field that is in an inventory application that is based upon changes to a sales order in an order entry application.

Query Process Event Points

The following five event points execute only during query execution. They execute at the same time as the first process that uses the queried records. They do not, however, share record areas or record pointers with the process that uses the queried records.

Start of Query Execution

The *Start of Query Execution* event point is to query execution what the *Start of Process* event point is to query setup. It executes once, at the start of query execution.

Immediately before this event point, APPX restores the copy of all record areas in memory that was saved immediately subsequent to executing the *End of Process* event point in query setup (see *End of Process* below). However, positions within files are not restored. For example, if you performed a READ on customer 10 before the *End of Process*, a READNEXT in *Start of Query Execution* accesses customer 1, not customer 11. To access customer 11 with READNEXT, you must first READ customer 10 in *Start of Query Execution*.

Establish PCF Range

This event point can be used to constrain the number of records that pass through a query. It is executed once, immediately after the *Start of Query Execution* event point. You can set range limitations either on the primary key or on alternate keys of the process control file, depending on the access path you choose. You can constrain the full key or a partial key. If this event point contains no statements, APPX will construct a range based upon the child constraints specified in the query's parent process. If no child constraints are specified in the parent, APPX uses the designer and end user selection criteria to establish the PCF range.

If this event point contains statements, even just comments, APPX assumes the designer wants control over the constraints and does not attempt to create its own constraints or use any child constraints specified in the parent. For example, if code exists in the event point to constrain records but this code is bypassed or disabled, no records will be selected.

If statements are entered in this event point, at minimum the range should be set by BEG AT and END AT statements, which should be followed by a BEG READ statement, then a GOSUB --- PROCESS QUERY RECORD statement that passes the record to the query, then an END READ statement. The predefined subroutine PROCESS QUERY RECORD calls the *Pre-User Selection* event point, evaluates the designer and end user selection criteria, and then calls the *Post User Selection* event point.

You may assign multiple constraints in this event point, and gather data from multiple files to establish your constraints. If the constraints overlap and produce a duplicate record situation, APPX will sense that and retain only one copy of the duplicate.

Pre-User Selection

The *Pre-User Selection* event point executes after a PCF record is read and before APPX applies the selection criteria. It executes once for every record in the PCF that satisfies the constraints of the *Establish PCF Range* event point, or whatever APPX constructs in the absence of statements in that event point.

This event point is used to select records that cannot be included or excluded through the normal record selection specifications; their selection may be dependent upon a highly complex combination of selection criteria, for example. *Pre-User Selection* routines become part of the selection criteria for the query.

The INCLUDE RECORD predefined field determines whether or not a record is selected for the query. INCLUDE RECORD is reset to blank immediately before the *Pre-User Selection* event point executes. To exclude the record, regardless of the selection criteria, set INCLUDE RECORD to *0* or *N* (no) in the event point. To include the record, regardless of the selection criteria, set INCLUDE RECORD to *1* or *Y* (yes) in the event point. If INCLUDE RECORD is left blank, the record is included or excluded based on the selection criteria.

MENU, INPUT, OUTPUT, INQUIRY, JOB, STATUS, UPDATE, and QUERY statements should not be entered in the *Pre-User Selection* event point. Additionally, display control statements such as DISPLAY, BLINK, NO INPUT, OK INPUT, etc. should not be entered here. APPX will ignore all such statements at runtime.

Be aware that performance can be significantly affected based on the location of event point statements. Remember that statements in *Pre-User Selection* are executed with every PCF record accessed while those in *Post User Selection* are only executed after a record has been selected. For example, if statements are required to read auxiliary files and access fields evaluated in the *selection* of records, place the statements here. However, if the statements access files to get field values used to determine the *sort pattern* for selected records, place the statements in *Post User Selection.*

Post User Selection

The *Post User Selection* event point executes after a PCF record is read and satisfies the constraints imposed by the *Establish PCF Range* event point, and after APPX applies the selection criteria, and only if the record satisfies all selection criteria. This means that this event point only executes for a specific subset of records in the PCF.

This event point can be used to read other files to make values available for sorts, because it only executes for records that satisfy the selection criteria. For example, you might want to sort information in a transaction file that contains a customer number, based upon a customer alpha field that is

maintained in a customer file. If you place the routine to retrieve the customer information in this event point, APPX only retrieves the information for transaction records that satisfy the selection criteria, and not for every transaction record in the file.

APPX cannot, however, guarantee that the sort will complete by the time the process that uses the queried records begins. Therefore, you should not use this or any query event point for performing file updates or accumulating file totals.

MENU, INPUT, OUTPUT, INQUIRY, JOB, STATUS, UPDATE, and QUERY statements should not be entered in the *Post User Selection* event point, and if present they will be ignored at runtime. This is also true for display control statements such as DISPLAY, BLINK, BLANK, NO INPUT, and so forth.

End of Query Execution

The *End of Query Execution* event point is to query execution what the *End of Process* event point is to query setup. It executes once, at the end of query execution.

End of Process

End of Process statements execute once, immediately before the data files close and the process ends. All processes, with the exception of subroutine and status processes, include this event point. If a process is cancelled for any reason, APPX executes *End of Process* before canceling. The predefined field OPTION can be interrogated to determine if the process is ending normally or if it was cancelled.

For a query process, this event point executes as a part of query setup. Query processes are unique in that the *End of Process* ends before the process completes execution; the query itself has not yet executed. This means that *End of Process* executes before the query event points *Start of Query Execution, Establish PCF Range, Pre-User Selection, Post User Selection,* and *End of Query Execution*. Immediately following *End of Process* for a query process (completion of query setup), APPX automatically saves a copy of the contents of all record areas in memory. Query execution restores the copy of memory immediately before the *Start of Query Execution* event point.

End of Process is often used to:

- Determine how a process ends (END, CANCEL, or PREV IMAGE) by interrogating the predefined field OPTION, and then perform any required action.
- Perform processing that is dependent upon process completion, such as storing a control or hash total, setting a report printed flag for an output process, or checking for errors or warnings to advise the user of an error log.
- Perform any final updating of files.
- Write data to a file that is read in the *Start of Process* event point of the next process in the family.

Frame-Level Event Points

Frame-level event points include *Start of Frame*, *Select Image*, *Global Pre-Display*, *Global Option Intercept*, and *End of Frame*. File-level event points will also be executed for any file that is accessed with statements at the frame level, including the PCF.

Start of Frame

The *Start of Frame* event point executes once for a frame, each time the frame invokes.

Select Image

The *Select Image* event point executes for input, inquiry, and output processes only. For a repeating frame, this event point executes for each repetition. Immediately after it executes, APPX initializes the value in the OPTION predefined field and determines which alternate image to execute by interrogating the predefined field ALTERNATE IMAGE NUMBER.

Select Image is generally used to modify the value in the ALTERNATE IMAGE NUMBER predefined field to identify a specific image or bypass an image (for repeating and optional frames).

Select Image executes before the image-level event points *Pre-Display* and *Pre-Display/Verify*. As a result, it can be used for routines that should execute regardless of which alternate image is selected, so you do not have to repeat such a routine for every image in the frame.

Global Pre-Display

The *Global Pre-Display* event point executes immediately before the actual appearance of the selected alternate image to a user. It executes only for input and output processes, and only if the frame includes an image. If the frame is a repeating frame, this event point executes once for each repetition. It also executes once for each entry level (up to nine) in the data addition interactive phase.

Global Pre-Display executes after the image-level event points *Pre-Display* and *Pre-Display/Verify*.

After this event point executes, APPX reinterrogates the OPTION predefined field to determine whether or not the content of the field was set with statements. If so, APPX responds as if the user had selected this option.

Global Option Intercept

The *Global Option Intercept* event point executes immediately after the image-level event point *Option Intercept*. As a result, it can be used for routines that should execute regardless of which alternate image is selected, so you do not have to repeat such a routine for every image in the frame. *Global Option Intercept* executes only for input processes, and the *Key Entry* frame of inquiry processes, and only if the frame includes an image.

Immediately following this event point, APPX interrogates the value in the OPTION predefined field to determine which option the user selected. The OPTION predefined field can be modified by statements in this event point. You can choose to include ERROR statements in the event point to prohibit the selection of a specific option, but ERROR statements in this event point are ignored in inquire mode.

End of Frame

The *End of Frame* event point executes once for a frame, immediately after the final execution of the image-level event points, before the invocation of any automatic child processes and before the frame ends. It executes for input, output and inquiry processes only.

End of Frame is often used to determine how a frame ends (END, CANCEL, PREVIOUS IMAGE, RETURN, NEXT RECORD, or ACKNOWLEDGE DELETE, for example) by interrogating the predefined field OPTION, and then performing any required action.

Immediately after *End of Frame* completes, APPX interrogates the INCLUDE CHILDREN predefined field to determine whether or not to skip the automatic children that are associated with the frame. If the INCLUDE CHILDREN predefined field has been preset with a value of *N* or *0* (no), then the associated automatic children are not executed. This field is defaulted to *Y* or *1* (yes) and these values are retained unless specifically changed by the designer.

Effect of Optional Frames on Event Point Chronology

If the ALTERNATE IMAGE NUMBER predefined field is set to *0* before *Select Image* completes, APPX immediately executes *End of Frame*. It does not execute the *Global Pre-Display* or *Global Option Intercept* event points, or any of the image-related event points.

Effect of Repeating Frames on Event Point Chronology

If the ALTERNATE IMAGE NUMBER predefined field is not set to *0* before *Select Image* completes, APPX executes a repeating frame in the normal (non-repeating) sequence. However, instead of executing *End of Frame,* APPX re-executes *Select Image*. In each repetition, a different alternate image can be selected.

This cycle repeats until the ALTERNATE IMAGE NUMBER predefined field is set to *0* before *Select Image* completes, in which case, APPX immediately executes *End of Frame*; it does not execute the *Global Pre-Display* or *Global Option Intercept* event points, or any of the image-related event points.

Effect of Scrolling Frames on Event Point Chronology

In change, delete, or inquire modes, APPX attempts to fill a frame with images before displaying it to a user. (In add mode, APPX adds a single image at a time.) As a result, APPX does not display a screen after executing the *Global Pre-Display* event point, but instead interrupts the normal sequence and executes *End of Frame*. It then reads the next record, and repeats the same sequence. This process continues until the frame is completely filled or until APPX runs out of records. During this process, the interactive phase is set to data scroll.

When the frame is filled, APPX rereads the first record on the screen and executes all of the event points for this first record.

Image-Level Event Points

Image-level event points include *Pre-Display, Pre-Display/Verify, Option Intercept,* and *Verify*. File-level event points will also be executed for any file that is accessed with statements at the image level, including the PCF.

For a very simple input process in change mode, the order of frame- and image-level event points before displaying an image is: frame-level *Select Image*, image-level *Pre-Display/Verify*, image-level *Pre-Display*, and frame-level *Global Pre-Display*. After an image displays, the order is: image-level *Option Intercept*, frame-level *Global Option Intercept*, image-level *Verify*, and image-level *Pre-Display/Verify*. The sequence of event point execution changes depending on the mode or interactive phase, and is easily complicated by the addition of multiple frames or optional or automatic child processes.

Image-related statements, such as BLINK and NO INPUT, can only execute in image-level event points. They are ignored in all other event points.

Pre-Display

In change, delete, and inquire modes the *Pre-Display* event point executes once, after the *Pre-Display/Verify* event point and before *Global Pre-Display*. During data addition *Pre-Display/Verify* is not executed prior to displaying the image. Instead, *Pre-Display* is executed once for each entry level (up to 9 times).

Because *Pre-Display* is an image-level event point, it is used for image-specific routines. Recall that the frame-level event points *Select Image* and *Global Pre-Display* can be used for routines that apply to all alternate images.

Pre-Display is found only in menu, input, inquiry, and output processes. It is often used to:

- Prohibit a user from entering a field by using a NO INPUT statement.
- Set the value of the predefined field OPTION to perform automatic operations, such as automatically ending an input process.
- Bypass a record. For example, in an accounts payable invoice maintenance process, you might choose to display only those invoice records that were created in accounts payable, bypassing those that were entered through a purchase orders application. You can do so by conditionally executing the NEXT RECORD option in this event point.
- Establish various control totals or other computed values for display on an input process image.
- Display fields with DISPLAY statements.

Pre-Display/Verify

The *Pre-Display/Verify* event point executes before *Pre-Display* and again after *Verify*, except during data addition, when it is not executed prior to displaying the image. After displaying the image, *Pre-Display/Verify* executes the same number of times as *Verify* because it always executes after *Verify,* and never unless *Verify* has executed.

The primary reason *Pre-Display/Verify* executes twice is for data that is retrieved from files, and then manipulated and displayed to the screen. It does not execute before *Pre-Display* during data addition because the data has not yet been entered and stored.

This event point is found only in input processes and the *Key Entry* image of inquiry processes. It is often used for:

- Validating the interrelationships between fields on the screen that a user entered. While APPX automatically validates individual items of information that are stored in a database, it is up to the designer to validate their interrelationships.
- Building keys and accessing information that cannot be accessed automatically with a data lookup. For example, you may have to perform calculations or look up data to determine the key of the record to be read.
- Flagging error or warning conditions using ERROR, WARNING, and BLINK statements.
- Performing computations and accumulations.

Option Intercept

The *Option Intercept* event point executes after an image is presented and the user selects an option (including ENTER). It is followed by the frame-level event point *Global Option Intercept.*

Because *Option Intercept* is an image-level event point, it is used for image-specific routines. Recall that the frame-level event point *Global Option Intercept* can be used for routines that apply to all alternate images.

Option Intercept is found only in menu and input processes. It is used, for example, to:

- Interrogate the predefined field OPTION to determine which option the user selected. OPTION can be modified during this event point. You can also include ERROR statements to prohibit the selection of a specific option, but ERROR statements are ignored in inquire mode.
- Create auxiliary records. For example, you might include statements to create invoice records from information that is stored on a purchase order record when the user selects a specific option key.

Verify

The *Verify* event point executes after *Global Option Intercept* in the data addition or data modification interactive phases. It does not execute in the key entry interactive phase, because records are not changed in this phase. This event point is found only in input processes and the *Key Entry* image of inquiry processes.

APPX performs automatic data lookups immediately before executing the *Verify* event point.

Child-Level Event Points

Child-level event points include both *Pre-Invocation* and *Post Invocation*. They are found in menu, job, input, output, update, and inquiry processes. File-level event points will also be executed for any file that is accessed with statements at the child level, including the PCF.

Pre-Invocation

Pre-Invocation event points execute once for a parent process, immediately before a child is invoked.

This event point is most useful for executing PASS statements to send information to a child process. (A corresponding RECEIVE statement can be placed anywhere in the child process, but is generally located in *Start of Process*.)

Post Invocation

Post Invocation event points execute once for a parent process, immediately after a child completes execution. The predefined field OPTION can be interrogated to determine if the child ended normally or was cancelled.

The predefined field INCLUDE CHILDREN can be used to control whether or not one or more child processes execute. To skip execution, INCLUDE CHILDREN must be set to *0* or *N* (no). Its effect depends on the event point it is set in, as follows:

- If set to *0* or *N* in *End of Frame,* all child processes are skipped. (Recall that *End of Frame* executes before any children are invoked.)
- If set to *0* or *N* in *Pre-Invocation,* that single child process is skipped.
- If set to *0* or *N* in *Post Invocation,* all remaining child processes are skipped.

Using the Predefined Field NEXT CHILD ID

The predefined field NEXT CHILD ID can be used to alter the sequence of child process invocation. Normally, child processes are invoked in sequence number order. However, if NEXT CHILD ID is set, the first child process with a CHILD ID that matches the value in NEXT CHILD ID is invoked.

NEXT CHILD ID is also used to manage forward and backward navigation through the images in input processes. As a user moves "forward" through the entry screens, NEXT CHILD ID is set by APPX to +. If a user moves "backward" through the entry screens by selecting *Previous Activity*, NEXT CHILD ID is set by APPX to –, in which case APPX navigates backward through the child processes.

If this field is set in the *End of Frame* event point on a frame that has automatic child processes, the first child process defined with a CHILD ID that matches NEXT CHILD ID will be invoked. Once that child is completed, processing continues normally.

If this field is set in the *Pre-Invocation* event point, processing will immediately skip to the first child encountered that has a matching CHILD ID.

If this field is set in the *Post Invocation* event point, the next child process that is invoked will be the first child process encountered where CHILD ID matches NEXT CHILD ID.

File Level Event Points

File-level event points are executed every time a specific action is performed on the designated file. These event points are accessed by the use of the Rules Option on the Files/Fields screen from the Database Management Menu in Application Design. They are used to perform standard operations in lieu of duplicating identical rules in certain event points of multiple processes or repeatedly calling the same subroutine. File-level event points are executed by the File Manager module of APPX and are independent of any other event points. The only data shared with the calling process is the record buffer. PASS and RECEIVE statements are ignored in file-level event points. However, work fields set in one file-level event point can be interrogated in another file-level event point (for the same file), without being corrupted by any intervening process or by the file-level event points for another file.

File-level event points include the following as described in the corresponding sections:

- File Open
- Post Read
- Pre Update
- Post Update
- File Close

File Open

The *File Open* file-level event point is executed every time the designated file is opened by an APPX process. It is the equivalent of adding such statements to the *Start of Process* event point of each and every process accessing the designated file (except for how data is shared).

See the Files Rules Option and File-Level Event Points sections for more information.

Post Read

The *Post Read* file-level event point is executed immediately after a record is read in the designated file in any manner under APPX control. It is the equivalent of adding such statements to the *Post PCF Read* event point of each and every process whose PCF is the designated file (except for how data is shared). In addition, *Post Read* code is executed immediately following the execution of any read statement (READ, READNEXT, or BEG READ) affecting the designated file.

See the Files Rules Option and File-Level Event Points sections for more information.

Pre-Update

The *Pre Update* file-level event point is executed immediately prior to updating (adding, modifying, or deleting) a record in the designated file in any manner under APPX control. It is the equivalent of adding such statements to the *Pre-PCF Update* event point of each and every process whose PCF is the designated file (except for how data is shared). In addition, *Pre Update* code is executed immediately prior to the execution of any update statement (WRITE, REWRITE, or DELETE) affecting the designated file.

See the Files Rules Option and File-Level Event Points sections for more information.

Post Update

The *Post Update* file-level event point is executed immediately after updating (adding, modifying, or deleting) a record in the designated file in any manner under APPX control. It is the equivalent of adding statements to the *Post PCF Update* event point of each and every process whose PCF is the designated file (except for how data is shared). In addition, *Post Update* code is executed immediately after the execution of any update statement (WRITE, REWRITE, or DELETE) affecting the designated file.

See the Files Rules Option and File-Level Event Points sections for more information.

File Close

The *File Close* file-level event point is invoked every time the designated file is closed by an APPX process. It is the equivalent of adding such statements to the *End of Process* event point of each and every process accessing the designated file (except for how data is shared).

See the Files Rules Option and File-Level Event Points sections for more information.

CHAPTER 4-1

ILF Integration

Using ILF Routines

An application consists of a number of related components, including the data dictionary specifications and various processes, such as menus, inputs, and outputs. You can develop an entire application in APPX using the specification method, traversing through the APPX application design menus and providing responses to predefined questions and choices. You support these specifications with appropriate screen and report image definitions.

Complex applications may require more specific control over the editing or processing criteria. For example, you may want to check the relationship between two different field values for consistency. In an order processing application, you could use ILF (Integrated Language Facility) statements to verify that the cost of a new order cannot exceed a customer's credit limit. If the order value exceeds the credit limit, you can display an error message and prevent any subsequent processing.

You implement this type of edit checking by including an ILF routine within the context of the input process that checks a user's entries and displays an appropriate message. This is a simple example where a few lines of ILF code achieve the desired results. In many applications, you may want to perform significantly more complex condition testing and logic operations.

You also use the ILF as the link between APPX and program routines written in other languages, such as COBOL and "C," and to invoke other programs like word processing or command editors. To execute a non-APPX program, you invoke either a CALL or RUN statement (each of which is described in detail in *Chapter 4-6: ILF Keyword Reference*).

An Example of a Statement

Like traditional programming languages, the ILF has its own set of rules and syntax that you follow when developing routines. For example, the following set of statements sets a work field value to the constant 10 and checks to see if a second field is equal to or greater than 10. Based on the result, the routine either calculates a revised total or ends without further action.

```
      SET      TAP WORK COUNT                =      10
      IF       1EX MONTH                     GE     10
T     COMPUTE  1EX WORK TOTAL AMOUNT         +  TAP WORK AMOUNT
F     END
```

This statement series contains four different keywords (SET, IF, COMPUTE, and END). Although each statement has its own format, there are common patterns among all of them. For example, the keyword always starts in the same character position and is no longer than eight characters long. Many keywords require you to specify a field name, and some (such as IF) require a second field name or a

constant to identify the two things that are being compared. The *T* and *F* at the beginning of the COMPUTE and END statements provide a condition-testing facility that allows you to dynamically control whether or not a statement executes at runtime.

The Components of a Statement

The previous example demonstrates a few of the statements. The following are some common items of information that you specify when you define a statement:

- ***True/false indicators***
- ***A keyword*** (SET, COMPUTE, IF, etc.)
- ***Field names*** (always associated with a specific application and, if appropriate, occurrence and appearance numbers)
- ***File names***
- ***Constant values***
- ***Relationals*** (such as *EQ* and *GT*)
- ***Arithmetic operators*** (such as + and -)

The Statements Included in the ILF

Each ILF statement is identified by a unique keyword (such as IF, READ, and SET) that indicates to the system what actions to perform. The ILF statements can be grouped into seven functional categories as follows:

Field-related statements

Modify and compute values for fields within an application.

Date/time statements

Calculate and compare dates and elapsed time periods.

Control statements

Allow you to exercise control over the processing flow in an ILF routine, and to invoke non-APPX programs.

Condition-testing statements

Include IF, AND, and OR statements that perform logic tests.

Image-related statements

Allow you to override the “visual” aspects of a screen or report, such as whether an item should be highlighted or allow user input.

File-related statements

Control file management operations, such as creation, deletion, and updates.

Text-related statements

Allow you to provide free-form text messages to users or to insert comments within a routine.

Table 4-1-1 below contains a list of the keywords in their respective categories. Each of these statements is described in detail in *Chapter 4-6: ILF Keyword Reference*.

Table 4-1-1. Statement Keywords by Category

FIELD	DATE/TIME	CONTROL	CONDITION	IMAGE	FILE	TEXT
APPEND	DATE ADD	BEG LOOP	AND	AT FIELD	BEG AT	* (Comment)
CALC	DATE BTW	CALL	IF	BLANK	BEG READ	** (Comment)
CNV BIN	SET DATE	COPY	OR	BLINK	CLOSE	CANCEL
CNV PORT	SET DAY	END		BOLD	COMMIT	ERROR
CNV TEXT	SET MNTH	END LOOP		BRIGHT	CREATE	MESSAGE
COMPUTE		END PROC		COLOR	DELETE	WARNING
IF DIFF		EXIT		DIM	END AT	
LOCAL		GOSUB		DISABLE	END READ	
PASS		GOTO		DISPLAY	IF EXIST	
RECEIVE		INPUT		ENABLE	MARK TRX	
RESTORE		INQUIRY		FONT	OPEN	
SCAN		JOB		INVERSE	READ	
SET		LABEL		NO BOLD	READNEXT	
SET TEMP		LOGOUT		NO INPUT	RELEASE	
SET TOT		MENU		NO ULINE	REWRITE	
STATIC		OUTPUT		NORMAL	ROLLBACK	
STORE		OVERFLOW		OK INPUT	SAVEPNT	
TOTAL		QUERY		PICTURE	SCRATCH	
VERIFY		RETURN		POSITION	WRITE	
		RUN		REFRESH		
		STATUS		ULINE		
		SUBR				
		TIMEOUT				
		TRAP				
		UPDATE				
		USE CC				

When to Invoke ILF Routines: Event Points

One important consideration when developing an ILF routine is to designate the exact point during the process when the routine will execute. You control the timing of when ILF routines execute by using appropriate *event points*. An event point is a predefined chronological point during the execution of a process when an ILF routine can be executed.

There are several different types of processes (menus, inputs, etc.) and several process components (frames, images, etc.). Each process component has its own specific set of event points. You can develop and execute an ILF routine at any one of these predefined event points, depending on the unique requirements of your application.

As an example, any process that invokes a child process has both a *Pre-Invocation* and a *Post Invocation* event point. You can select either of these event points (as options when you are defining a child invocation) and develop an ILF routine. A pre-invocation ILF routine executes immediately before the child process is invoked, while a post invocation ILF routine executes immediately after the child process completes.

You may want the application to test certain conditions and, based upon the results, perform a specified set of actions at a precise point or points in time. It is the event points that accommodate this requirement to control the timing of executing an ILF routine.

Event Point Definition and Description

Event points are defined to correspond to the nature of a process or process component. For example, an image has a *Pre-Display* event point, while a frame has a corresponding *Start of Frame* event point. Input processes also have a *Post PCF Update* event point, although a menu process does not because a menu does not update a process control file. As a result, while each process type/component has a unique set of event points, there is a consistent pattern throughout the system.

Another item to note is that some event points may not occur in certain instances, depending on what you specified the process to do. For example, a frame may or may not have an image associated with it. If the frame produces an image, it has a *Pre-Display* event point; if no image is produced, the *Pre-Display* event point is not available. As a result, the applicable set of event points for a process is dependent upon the context in which you specify it.

Refer to *Chapter 3-16* for details on event point timing and usage.

The Steps in Using the ILF

There are three steps involved in defining and using ILF routines:

- Entering and editing individual ILF statements.

- Executing processes that include ILF routines, and resolving compilation and runtime errors.
- Testing and resolving logic errors in routines with the ILF debugger.

The ILF Editor

The ILF editor is an integrated facility within APPX that allows you to enter and modify ILF statements. You invoke the editor by selecting the appropriate event point option from the process component screen. The editor provides specialized entry options that allow you to develop and customize ILF routines. The editor checks each statement to ensure that you have followed the proper syntax rules, and prevents you from entering certain types of invalid data. The ILF editor is fully integrated with the data dictionary so all fields entered in statements are verified against the dictionary. Refer to Chapter 4-4 for details on how to use the ILF editor. An example of the ILF Editor is shown in *Figure 4-1-1*.

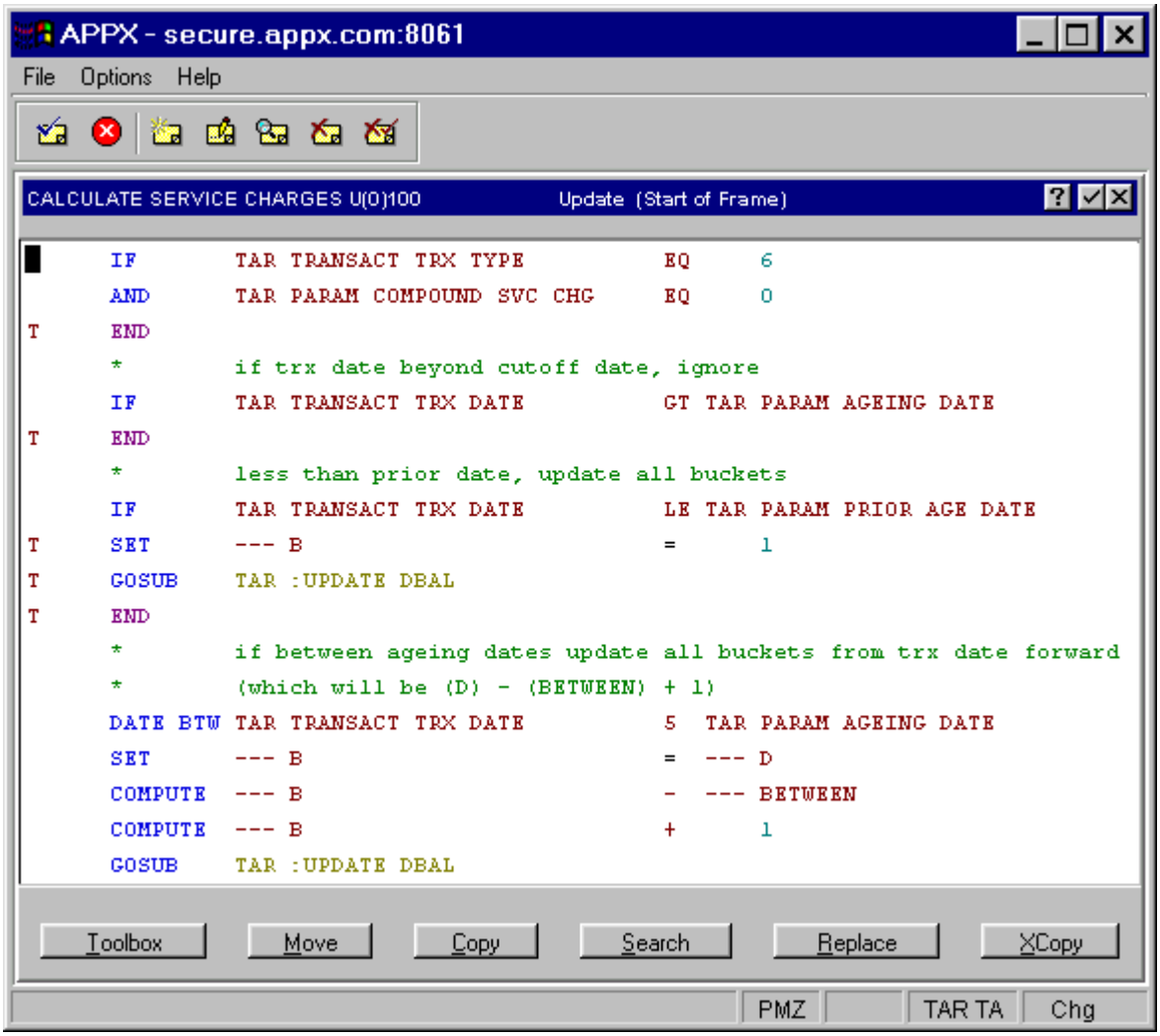


Figure 4-1-1. The ILF Editor

The Compilation Step

When you define an ILF routine, it becomes an integral part of the associated APPX process. To achieve this integration, the ILF routines are compiled along with all the other process specifications that you entered. The APPX process compiler is invoked automatically at process runtime. The compiler pulls together all the parts of the application, including any ILF routines that you defined. As a designer, you do not have to be concerned about the compilation step other than to understand that, if any errors are encountered, you will receive an appropriate compilation error message (refer to the next section).

The system automatically invokes compilation at runtime when it is required, either when a process is first developed or if changes were made. The system checks to see if there were any changes since the last compilation and, if there were, a recompilation is automatically initiated. Note that the verification and compilation is done when an end user executes a process, not when the designer applies changes or makes additions to the process specifications. As a result, the start-up time for an end user is longer when a compilation is required. Once compiled, subsequent start-up time is much faster because the system uses the compiled version (called an *executable module*).

You may also force a process to compile while using Application Design. Simply position your cursor on the selected process and select SYNTAX CHECK from the *Toolbox* menu (or simply select `OPTION 97`).

Resolving Compilation Errors

When a process compiles, some compiler-generated errors may result. The system notifies the user (or the designer, if the designer executed the process) of each of the errors encountered during compilation by displaying and highlighting the erroneous statements, and by providing an error message describing the nature of each error.

At the end of compilation, if any errors occurred, an error listing (labeled *Process Compilation Errors)* is displayed on your screen as shown in *Figure 4-1-2* below.

A Process Compilation Errors screen

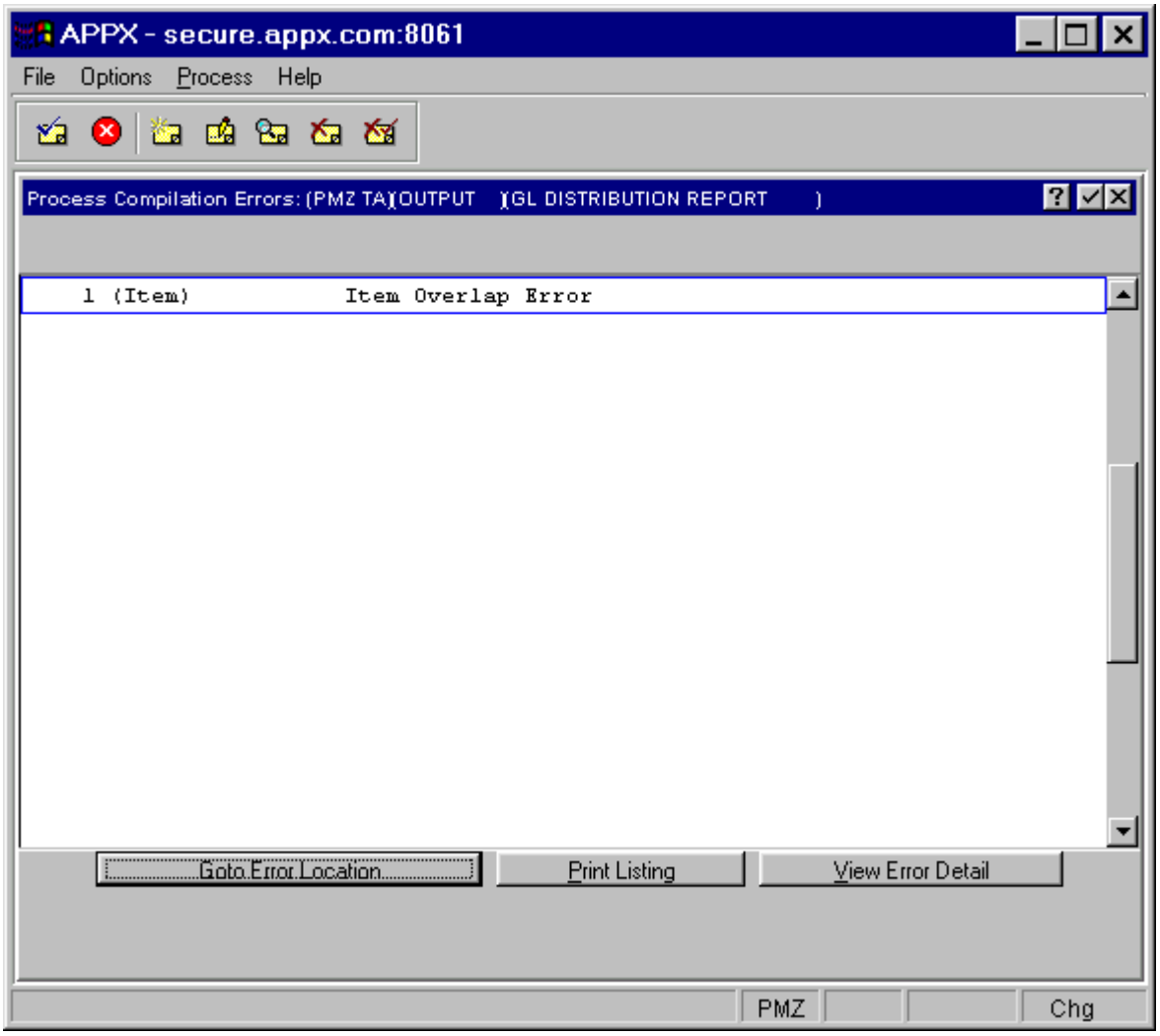


Figure 4-1-2

From this screen, there are three options available to you. Two of the options provide methods for correcting compilation errors:

- GOTO ERROR LOCATION. You can position the cursor to a specific error and select GOTO ERROR LOCATION to go directly to the ILF Editor and the offending item via a hypertext link. Make the appropriate modifications to the ILF code. When you are finished, you can force the process to compile again by selecting *END* until you return to the original error display. Repeat these steps until all errors have been corrected, then select *END* from the Process Compilations Error screen to start the recompile.
- PRINT LISTING. You can print the error listing by selecting the PRINT LISTING option. Using the printed listing, return to the process that the ILF routine is associated with and invoke the editor by selecting the appropriate event point option. Use the editor functions (refer to *Chapter 4-4*) to make appropriate modifications to the ILF routine and then re-execute the process. This is an iterative process that you may repeat as many times as required until compilation is successful.

The third option is:

- VIEW ERROR DETAIL. By positioning the cursor on a specific error and selecting this option, APPX provides you with more detail regarding the selected error. After viewing the details, select ENTER ☑ to return to the Process Compilation Errors Screen.

Resolving Runtime Errors

Even if the compilation step is successful, you may encounter errors that occur during the execution of a process. For example, you may have a calculation that includes a division step. If the divisor has a value of 0, the result is undefined and a processing error occurs. When a runtime error occurs, system processing terminates and an error message displays. There is also an error message indicating the reason for termination. Resolving the problem may require you to modify one or more statements in an ILF routine. If this is the case, you need to exit from the error screen display, invoke the editor, make the required changes, and re-execute the process to determine if the problem is corrected. It may be that the problem resulted from some condition in the system environment other than the ILF routine itself. If so, you need to resolve this condition and then re-execute the process to verify that the problem is corrected.

Using the ILF Debugger

Even after resolving any runtime errors, the routine you developed may still not produce the results that you expected. In this situation, the system does not provide any error or warning messages, so it is up to you to test the routine thoroughly to identify any flaws in the logic. To assist you in locating and resolving these errors, the ILF contains a debugging facility that you invoke within the routine in question. The debugger allows you to insert break points at specified lines within ILF routines so that, as you execute a process, you can follow the execution and watch what happens to certain field values. Refer to Chapter 4-5: The Debugger for details on using the ILF debugging facility.

CHAPTER 4-2

True/False Status Indicators

How APPX Uses T/F Status Indicators

As a part of the ILF (Integrated Language Facility), APPX incorporates a unique condition testing facility that allows you to perform complex logic checking operations. As the basis of this facility, the system maintains a set of five *internal true/false status indicators*. These indicators provide you with a method to control the logic flow within an ILF routine, depending on the result of conditions encountered at runtime.

ILF statements (except for those with keywords *, **, and LABEL) incorporate within their syntax the ability to enter values for comparison against the current status of the internal true/false status indicators. The first five positions of most statements are reserved for this purpose; hence, these five positions are known as *true/false execution conditions*. From left to right, they correspond to the five true/false status indicators, levels 1 through 5, respectively. Based on the result of the comparison between the execution conditions and the internal status indicators, APPX either executes or bypasses a statement.

For example, the following is an example of a SET statement used to modify the value of the field WORK ALREADY POSTED:

```
    TTFTF SET      TAR WORK ALREADY POSTED        =      1
```

As coded, this statement would only be executed if the current status of the five internal indicators is TTFTF, to agree with the execution condition values specified in the statement.

By the same token, several statements set the values of the true/false status indicators. As each of these statements operates, if it finds a *true* or *met* condition, it sets the value of the next true/false status indicator to *T*. If the statement encounters a *false* or *unmet* condition, it sets the next indicator to *F*. For example, consider the following statement:

```
          IF       TAR COUNT                      EQ     6
```

If the IF statement determines that the value of the field named COUNT is 6, the first true/false status indicator is set to *T*. If the value of the field is not equal to 6, the indicator is set to *F*.

Both the true/false status indicators and the true/false execution conditions will be discussed more completely further in this chapter.

Setting T/F Status Indicators

As part of learning how to specify true/false execution conditions when you design ILF routines, you need to understand how the system sets the internal true/false status indicators. APPX clears (sets to blanks) all five status indicators at the beginning of each event point. This allows each ILF routine to start with a "clean slate," and enables APPX to set the values based on statement evaluation at runtime. In addition, when you invoke a subroutine using the keyword GOSUB, APPX temporarily clears out the true/false status indicators at the start of the subroutine so they can be used in the GOSUB routine. When control is returned to the ILF routine that invoked the GOSUB (via the keyword RETURN), the system restores the indicators to the values that existed before the GOSUB was invoked.

Only certain statements have the capability to affect the value of internal true/false status indicators. See Table 4-2-1 for a complete list of these statements, and refer to the statement keyword in *Chapter 4-6: ILF Keyword Reference* for a description of how each one operates.

In general, here is how indicator-setting works:

- Only the statements included in Table 4-2-1 affect status indicator values, and each of these statements sets a status indicator when it executes.*
- The five true/false status indicators are always set from left to right and are referred to as levels 1 through 5, respectively (corresponding to the execution conditions specified in the first five positions of the statement).
- Each statement affecting true/false status indicators sets the next level (as designated by the execution conditions) to either *T* or *F*, depending on the success or failure of an operation*. The status indicator to be set corresponds to the leftmost execution condition which contains a blank value. In addition, any status indicators to the right of the level being set (i.e., with a higher level number) are automatically cleared. For example, an IF statement with blanks in the first five positions would set level 1 and clear levels 2 through 5.
- Whereas the first five positions in the statement format are reserved for specifying execution conditions, statements affecting true/false status indicators (other than GOSUB and RETURN) are limited to a maximum of four non-blank execution conditions. This is because there must be one blank value corresponding to the next status indicator to be set. Once a blank value is specified, any execution conditions to the right must also be blank.

*** Exceptions:**

- Statements with keywords GOSUB, RETURN, AND, and OR affect status indicator values but behave a little differently.
- As described above, GOSUB stores the current value of all status indicators and clears all indicator levels prior to invoking the subroutine. RETURN restores the original values before code following the GOSUB is resumed.
- AND and OR modify or preserve the current value of the next indicator level, rather than simply setting it, since this indicator value (as already set by a preceding statement) must be considered in

evaluating the AND and OR statements. Note that, for proper execution, AND and OR statements must be subject to the same execution conditions as the statement (normally an IF) which originally sets the shared indicator level.

- If execution conditions are specified on a COPY statement, behavior can get a little confusing. This is because, at runtime, the COPY statement's execution conditions become the leftmost execution conditions of every statement copied (that is, they are inserted in front of specified values). It is best to avoid this situation since you may forget about the conditions to be inserted at runtime and thus end up dropping some non-blank execution conditions or status indicators. Worse, instead of getting a runtime error, the routine may simply execute improperly. You can avoid potential grief by using a GOSUB or SUBR statement instead of a COPY if you need to specify execution conditions.

T/F Status Indicator Values

There are three possible values for each of the five internal true/false status indicators:

A blank

Signifies a *cleared* value that is ready to be set to a true or false value.

The letter T

Represents a *true* or *met* condition.

The letter F

Represents a *false* or *unmet* condition.

Statements That Affect T/F Status Indicators

There are a number of statements that affect the value of internal true/false status indicators. Of the seven functional categories of statements shown in *Table 4-1-1*, only the following affect these indicators:

- Two field-related statements set the next status indicator to *T* or *F*: RECEIVE (depending on whether or not there is a matching PASS statement) and VERIFY (depending on whether or not the VERIFY operation is successful).
- Two types of control statements:
- The ***process-type statements*** (CALL, INPUT, RUN, etc.) alter the flow of processing and invoke a specified process. The specified process is either invoked successfully (sets a value of *T*) or fails (sets a value of *F*)
- The ***GOSUB and RETURN statements*** store the current indicator values, clear the values for independent use within the GOSUB subroutine, and then restore the indicators to their original values for continued processing following the RETURN. In addition, when implemented, GOSUB will set the next status indicator to the value indicated in the RETURN statement, if any.

- All condition-testing statements (IF, AND, and OR) test the relationship between two values. The result is always either true or false, and the indicator is set or modified accordingly.
- Only one image-related statement sets the next status indicator to *T* or *F*: AT FIELD (depending on whether or not the cursor is positioned on the field or fields specified).
- Several file-related statements (CREATE, READ, WRITE, etc.) set a value of *T* if successful, or *F* if not.

All statements affecting T/F values are listed in alphabetical order in *Table 4-2-1* below:

Table 4-2-1. Statements That Affect Internal True/False Status Indicators

Statement	Condition
AND	If the AND statement is evaluated as false, APPX changes the current value of the next status indicator to F; otherwise, it preserves the current value. Refer to the Using AND and OR Statements section.
AT FIELD	If the cursor is positioned on the specified field or fields, APPX sets the value of the next status indicator to T, otherwise, to F. See the AT FIELD keyword for further options.
CALL	If the called routine is invoked successfully, APPX sets the next status indicator to T; otherwise, to F.
CNV PORT	If the conversion is successful, APPX sets the next status indicator to T; otherwise, to F.
CNV TEXT	If the conversion is successful, APPX sets the next status indicator to T; otherwise, to F.
CREATE	If the file creation executes successfully, APPX sets the next indicator to T; otherwise, to F.
GOSUB	When a GOSUB statement executes, APPX stores the current status indicators, then clears them to give the subroutine a clean slate for setting values as required. The indicators are restored to their original values for continued processing following the RETURN. In addition, when implemented, GOSUB will set the next status indicator to the value indicated in the RETURN statement, if any.
IF	If the IF statement is evaluated as true, APPX sets the next level to T; otherwise, to F.
IF DIFF	If the IF DIFF statement is evaluated as true, APPX sets the next level to T; otherwise, to F.
IF EXIST	If the specified file exists, APPX sets the next level to T; otherwise, to F.
INPUT	If the specified process completes successfully, APPX sets the value of the next status indicator to T; otherwise, to F.
INQUIRY	If the specified process completes successfully, APPX sets the value of the next status indicator to T; otherwise, to F.
JOB	If the specified process completes successfully, APPX sets the value of the next status indicator to T; otherwise, to F.
MENU	If the specified process completes successfully, APPX sets the value of the next status indicator to T; otherwise, to F.
OPEN	If the file is opened successfully, APPX sets the next level to T; otherwise, to F. An OPEN succeeds unless the file does not exist, the file was already open for exclusive use, or there is a file authorization failure.
OR	If the OR statement is evaluated as true, APPX changes the current value of the next status indicator to T; otherwise, it preserves the current value. Refer to the Using AND and OR Statements section.
OUTPUT	If the specified process completes successfully, APPX sets the value of the next status indicator to T; otherwise, to F.

Statement	Condition
QUERY	If the specified process completes successfully, APPX sets the value of the next status indicator to T; otherwise, to F.
READ	If the file READ operation is successful, APPX sets the next level to T; otherwise, to F. A READ is successful unless a specified record is not in the file.
READNEXT	If the READNEXT operation is successful, APPX sets the next level to T; otherwise, to F. A READNEXT is successful unless an end-of-file condition is encountered, or no record is found within the range specified by previous BEG AT and/or END AT statements.
RECEIVE	If the named field is passed explicitly via a PASS statement, APPX sets the next indicator to T; otherwise, to F.
RETURN	A RETURN statement has a special effect on true/false status indicators. You use a RETURN statement as the last step in a GOSUB routine to return control to the original point in an ILF routine where a GOSUB was invoked. When a RETURN statement executes, APPX automatically resets all five indicators to the values that existed just before the GOSUB operation was invoked. In addition, **when implemented**, GOSUB will set the next status indicator to the value indicated in the RETURN statement, if any.
REWRITE	If the REWRITE operation is successful, APPX sets the next level to T; otherwise, to F. A REWRITE is successful unless the record was not previously held for update, or it contains a duplicate key on a key path where duplicates are not allowed.
RUN	If the external program is invoked successfully, APPX sets the next status indicator to T; otherwise, to F.
SCRATCH	If APPX successfully scratches (deletes) a specified file or the file does not exist, APPX sets the next level to T; otherwise, to F.
STATUS	If the specified process completes successfully, APPX sets the value of the next status indicator to T; otherwise, to F.
SUBR	If the specified process completes successfully, APPX sets the value of the next status indicator to T; otherwise, to F.
UPDATE	If the specified process completes successfully, APPX sets the value of the next status indicator to T; otherwise, to F.
VERIFY	If the VERIFY operation is successful, APPX sets the next status indicator to T; otherwise, to F.
WRITE	If the record is written successfully, APPX sets the next status indicator to T; otherwise, to F.

Some Examples of T/F Usage

Consider the following series of statements as examples of how T/F status indicators are set and T/F execution conditions are evaluated.

```
          READ     TAR RECEIPT1              HOLD 0 FT 0 BY RECEIPT1 KEY
```

If the READ statement is successful in this example, the level 1 true/false status indicator is set to T; otherwise it is set to F.

```
      READ     TAR RECEIPT1               HOLD 0 FT 0 BY RECEIPT1 KEY
T     IF       TAR RECEIPT1 POST STARTED      EQ     1
```

Here we have added an IF statement following the READ with a T in the first character position. Thus, the IF statement will execute only if the level 1 internal true/false status indicator is T (meaning the READ statement was successful). Assuming the READ is successful, the IF statement executes and sets the next level of internal true/false status indicators (in this case, level 2). Note that the IF statement as coded is an example of a single statement which both checks an execution condition and, if executed, sets a status indicator.

Continuing with our example statements:

```
      READ     TAR RECEIPT1               HOLD 0 FT 0 BY RECEIPT1 KEY
T     IF       TAR RECEIPT1 POST STARTED      EQ     1
TT    SET      TAR WORK FLAG                  =      0
TF    END
```

Now we have added statements that act upon the result of the evaluation of the IF statement but if and only if the READ statement was successful. If the IF statement is true, the SET statement executes; otherwise, the END statement executes. However, if the READ statement was not successful, none of the statements following the READ are executed.

T/F Execution Condition Values

There are five possible values for each of the five true/false execution conditions specified in an ILF statement. (As explained previously, execution conditions are coded in the first five positions of most ILF statements and correspond, from left to right, to the internal true/false status indicators, levels 1 through 5.)

A blank

Signifies there is no condition. In other words, the current status of the corresponding internal T/F status indicator does not matter in order for the statement to execute. However, if the statement affects internal T/F status indicators (as indicated in Table 4-2-1), the position of the leftmost execution condition with a blank value designates the next status indicator to be set, while clearing all indicators with higher level numbers.

The letter T

Signifies that the current status of the corresponding internal T/F status indicator must be *True* in order for the statement to execute (i.e., a *true* or *met* condition).

The letter F

Signifies that the current status of the corresponding internal T/F status indicator must be *False* in order for the statement to execute (i.e., a *false* or *unmet* condition).

- ***The character "-"* (hyphen)** is similar to a *blank* in that it also signifies no condition,. However, in contrast to a *blank*, it preserves the current status of the corresponding internal T/F status indicator. For more information, see the Using Hyphens as Placeholders section.

The letter C

Or the character "*" (asterisk) (in the first position only) disables execution of the statement. APPX bypasses any statement that has a **C** or ***** in the first position. You must recode this as one of the four previous values in order for APPX to execute the statement.

Valid/Invalid Execution Conditions

The following are examples of valid and invalid true/false execution conditions as specified in the first five positions of a statement. The *next level* referred to below identifies the internal true/false status indicator to be set assuming the statement affects these indicators (as identified in Table 4-2-1).

Valid T/F Execution Conditions

`• • • • •`	`No execution conditions apply; next level is level 1.`
`T • • • •`	`One condition applies; next level is level 2.`
`T T F T •`	`Four conditions apply; next level is level 5.`
`F F F T T`	`Five conditions apply; next level is invalid.*`

*** Note: Five conditions are not valid on a statement affecting the internal T/F status indicators (except for GOSUB and RETURN).**

Invalid T/F Execution Conditions

`• T • • •`	`A non-blank execution condition cannot follow a blank.`
`T F • • T`	`A non-blank execution condition cannot follow a blank.`

Using Hyphens as Placeholders

There may be cases where you want to unconditionally execute a statement without checking one or more existing status indicator values, while preserving the state of that indicator level. To accomplish this action, you can use a “-” (hyphen) as a placeholder within the true/false execution conditions. For example:

```
        IF      TAR WORK FLAG                EQ     1
-       IF      TAR RECEIPT1 POST STARTED    EQ     1
```

In this example, the first IF statement executes unconditionally and sets the level 1 internal true/false status indicator to *T* or *F*. The second IF has a "-" in the first position of its execution conditions so it also executes unconditionally, setting the level 2 internal true/false status indicator (level 2 corresponds to the leftmost execution condition with a blank value). You can now execute additional statements conditionally based on the first two levels, which can be *TT, TF, FT*, or *FF*. This type of "truth table" logic can be extended to additional levels.

For instance, you can perform a three-statement series as shown in the following example, and then check for internal true/false status indicator values of *TTT, TTF, TFT, TFF, FTT, FTF, FFT*, or *FFF*.

```
        IF      TAR WORK FLAG                EQ     1
-       IF      TAR RECEIPT1 POST STARTED    EQ     1
--      IF      TAR POST ERRORS              GT     0
```

Simple T/F Examples for Comparison

To demonstrate the rules given so far, the following four examples each present a two-statement series using a READ followed by an IF.

```
        READ    TAR RECEIPT1              HOLD 0 FT 0 BY RECEIPT1 KEY
        IF      TAR RECEIPT1 POST STARTED    EQ     1
```

The results of the first example (with no conditions) are:

- The READ statement executes unconditionally.
- If the READ is successful, the level 1 true/false status indicator is set to *T*. If it is not successful, the level 1 indicator is set to *F*.
- The IF statement executes unconditionally. It does not test the result of the READ statement.
- If the IF condition is true, the level 1 true/false status indicator is set to *T*. If the IF condition is not true, the level 1 indicator is set to *F*. Note that APPX sets the same indicator here that the READ statement previously set, so the result of the READ operation is no longer available.
- Internal status indicator values possible at the end of this example are *T* or *F*.

```
        READ    TAR RECEIPT1              HOLD 0 FT 0 BY RECEIPT1 KEY
T       IF      TAR RECEIPT1 POST STARTED    EQ     1
```

The results of the second example (with a True condition on the IF) are:

- The READ statement executes unconditionally.

- If the READ is successful, the level 1 true/false status indicator is set to *T*; otherwise, level 1 is set to *F*.
- The IF statement executes only if the READ operation is successful (the level 1 indicator is *T*).
- Assuming the level 1 indicator is *T*, if the IF condition is true, the level 2 true/false status indicator is set to *T*; otherwise, level 2 is set to *F*. Note that APPX retains the first value as set by the READ statement, so it is still available for subsequent checking.
- Internal status indicator values possible at the end of this example are *F, TT,* or *TF*.

```
        READ     TAR RECEIPT1               HOLD 0 FT 0 BY RECEIPT1 KEY
F       IF       TAR RECEIPT1 POST STARTED      EQ     1
```

The results of the third example (with a False condition on the IF) are:

- The READ statement executes unconditionally.
- If the READ is successful, the level 1 true/false status indicator is set to *T*; otherwise, level 1 is set to *F*.
- The IF statement executes only if the READ operation is not successful (the level 1 indicator is *F*).
- Assuming the level 1 indicator is *F*, if the IF condition is true, the level 2 true/false status indicator is set to *T*; otherwise, level 2 is set to *F*.
- Internal status indicator values possible at the end of this example are *T, FT,* or *FF.*

```
        READ     TAR RECEIPT1               HOLD 0 FT 0 BY RECEIPT1 KEY
-       IF       TAR RECEIPT1 POST STARTED      EQ     1
```

The results of the fourth example (with a hyphen placeholder on the IF) are:

- The READ statement executes unconditionally.
- If the READ is successful, the level 1 true/false status indicator is set to *T*; otherwise, level 1 is set to *F.*
- The IF statement executes unconditionally, regardless of whether the READ operation is successful (the first indicator may be either *T* or *F*).
- If the IF condition is true, the level 2 true/false status indicator is set to *T*; otherwise, level 2 is set to *F*.
- Internal status indicator values possible at the end of this example are *TT, TF, FT,* or *FF.*

Using AND and OR Statements

You may frequently use an AND or an OR statement following an IF statement to further qualify a condition specified by the IF statement. AND places a further constraint (both or all of two or more conditions must be true), while OR lessens the constraint (either or any of two or more conditions must be true). Depending on whether the AND/OR statement is true or false, APPX either modifies or

preserves the value of the true/false status indicator just set by the preceding IF statement. With AND and OR statements, you effectively give APPX two or more opportunities to set (or modify) the same status indicator.

See the following sections demonstrating using AND after an IF statement and using OR after an IF statement.

Using AND After an IF Statement

Consider the following example using an AND statement after an IF. In this example, the two statements are really part of a single logical verification since your intent is to check if both of the two statements are true.

```
IF       TAR CUSTOMER BALANCE           GT     0
AND      TAR WORK ON ACCOUNT AMOUNT     NE     0
```

- When the IF statement executes, it automatically sets the level 1 true/false status indicator to either *T* or *F,* as appropriate.
- If the IF statement is false, setting level 1 to *F*, the AND statement is bypassed because the possibility that both conditions are true has already been eliminated.
- If the IF statement is true, the AND statement executes. If the AND statement is true, the level 1 indicator is preserved (left unchanged as *T*). If the AND statement is false, the level 1 indicator is changed to *F*.

Using OR After an IF Statement

The example below using an OR statement is very similar to the previous example using an AND, except that the OR verification checks to see if either of the two statements are true:

```
IF       TAR CUSTOMER BALANCE           GT     0
OR       TAR WORK ON ACCOUNT AMOUNT     NE     0
```

- When the IF statement executes, it automatically sets the level 1 true/false status indicator to either *T* or *F,* as appropriate.
- If the IF statement is true, setting level 1 to *T*, the OR statement is bypassed because the possibility that both statements are false has already been eliminated.
- If the IF statement is false, the OR statement executes. If the OR statement is false, the level 1 indicator is preserved (left unchanged as *F*). If the OR statement is true, the level 1 indicator is changed to *T*.

Adding Execution Conditions

Now consider the following modification to the example in Using AND After an IF Statement:

```
    T     IF       TAR CUSTOMER BALANCE           GT     0
    T     AND      TAR WORK ON ACCOUNT AMOUNT     NE     0
```

Notice that the first execution condition of each statement now has a *T* in it. Adding the *Ts* results in the following:

- The IF statement executes only if the level 1 true/false status indicator is *T.*
- If it executes, the IF statement sets the level 2 indicator to *T* or *F,* as appropriate.
- The AND statement executes only if the level 1 indicator is *T* and the level 2 indicator is *T*. (If the level 2 indicator is already *F*, there is no need to execute the AND statement.)
- The AND statement changes the level 2 indicator to *F* if the AND statement is false; otherwise, the level 2 indicator is preserved (left unchanged as *T*).

Potential Misuse of AND and OR

We can modify the previous example slightly to point out a potential misuse of an IF/AND statement combination:

```
          IF       TAR CUSTOMER BALANCE           GT     0
    T     AND      TAR WORK ON ACCOUNT AMOUNT     NE     0
```

If the IF statement is true, the IF statement sets the level 1 internal true/false status indicator to T and the AND statement executes. But the AND attempts to modify the level 2 internal true/false status indicator, which has not been set. This generates a runtime error. A similar problem would result for an IF/OR statement combination with mismatching execution conditions.

In general, AND and OR statements should be subject to the same execution conditions as the statement they qualify (that is, the statement which originally sets the internal status indicator).

AND and OR with Statements Other Than IF

You can use AND and OR statements to qualify another type of statement other than an IF statement, provided the statement affects the true/false status indicators (see Table 4-2-1). AND and OR statements behave in exactly the same manner as previously described when used after IF statements. For example:

```
          READ     TAR CUSTOMER               HOLD 0 FT 0 BY CUSTOMER NO
          AND      TAR CUSTOMER BALANCE           NE     0
    T     SET      --- INCLUDE RECORD             =      y
```

In this example, the SET statement executes only if both the READ statement is successful and the AND statement is true.

CHAPTER 4-3

Specifying Statements

Specifying Statements: overview

Every statement you specify within an ILF routine occupies a single line. For each statement, you enter a number of individual components, beginning with a keyword. Each statement type, identified by its unique keyword, has its own *syntax* (the name given to the predefined pattern of words and symbols) that determines what information you must provide. Table 4-1-1 provides a complete list of statement keywords by category, and *Chapter 4-6: ILF Keyword Reference* describes each statement in detail, including syntax requirements and available options. There are, however, a number of patterns that apply to several statement types. There are also rules that govern the use of many of the common statement components. This chapter describes many of these patterns so that you can apply them to individual statements as you develop ILF routines.

Remember, when entering or editing ILF statements, the F1 (Help) key provides context-sensitive help relating to the field at which you are positioned.

Patterns of Statement Keyword Use

With the previous definitions and categories in mind, there are several patterns that govern how different statement keywords operate. The following are some general principles to help you understand some of the differences in how certain statements require you to specify different components and types of data:

- The condition testing statements (IF, AND, and OR) require that you specify two fields to be compared. The two fields must be of the same data type (alpha-to-alpha, numeric-to-numeric, etc.). You are allowed to compare two numeric fields regardless of their storage format. For example, you can check to see if an integer field is greater than a packed decimal field. No intermediate data conversion is required. To compare two fields of different data types, first use a SET statement to convert one of the fields to a type compatible with the other field. You can also use the condition testing statements to compare a field to a constant.
- Several statements deal only with external data representations, such as BLINK, BRIGHT, and INVERSE. You use these keywords to affect how an item appears to an end user on the screen or in a report. These statements are known as *characteristic-type* statements.
- Several statements operate only on internal data representations, such as SET, COMPUTE, APPEND, and SET TEMP. You use them to compute or modify stored data values.
- Some statements deal specifically with numeric computations, and the destination field (the result of the computation) must be a numeric data type. COMPUTE and CALC are examples of this category of statements.

- Another category of statements, such as READ and READNEXT, operates at a file level. These statements require you to specify a file name.
- The text statements allow you to provide a message string that displays or prints information for a user under circumstances that you specify.

Rules for Statement Components

Each ILF statement consists of a number of words, symbols, phrases, and special characters that together represent a specific logical operation for the system to perform. Although each statement has a unique syntax, most of the individual statement components are derived from a common set of items that are used in several statement types. The following sections describe the most widely used statement components.

The T/F Execution Conditions

The first five positions of almost every statement type are reserved for true/false execution conditions. The following is a list of the only values allowed in this group and a brief description of what they mean. See section T/F Execution Condition Values in Chapter 4-2: True/False Status Indicators for a complete discussion.

Blank

Indicates no conditions are specified.

T

Indicates a true conditional check.

F

Indicates a false conditional check.

- - (hyphen) indicates an *either true or false* conditional check.

C

Or * indicates the presence of a temporary comment. APPX bypasses any statement that has a C or * in the first position. It is up to you to reset the indicator to one of the four values previously listed before APPX will execute the statement.

The following example demonstrates the true/false execution conditions:

```
    TTFF  IF       TAR RECEIPT1 POST STARTED      EQ     1
```

In this statement, the execution conditions *TTFF* requests APPX to match the corresponding values to the set of five internal true/false status indicators before executing the statement.

The Statement Keyword

The statement keyword is usually an action word (such as COMPUTE, SET, or READ). The *keyword* indicates to APPX which statement to invoke. When you begin to specify a statement after you enter the keyword, APPX provides you with a template to assist you in entering the remaining statement components. The keyword is always eight or fewer characters long and consists of letters and, in some cases, embedded spaces. The only exceptions to this pattern are the comment keywords, denoted by * and **. The keyword must be entered exactly as specified in the statement description provided in *Chapter 4-6: ILF Keyword Reference*. For example:

```
    T     BEG READ TAR CUSTOMER              HOLD 1 KEY IS  CUSTOMER NO
```

The statement in this example invokes the logic associated with the BEG READ keyword. Notice that there is an embedded space between the two parts of the keyword.

The Application ID

The *application ID* component identifies the application that a specified file or field is associated with. This allows you, for example, to compare the value of a field in one application to that of a field in a second application. All the statement types that allow you to specify files or fields require you to provide the corresponding application ID. Note that, when you refer to any of the predefined fields (PDFs) in a statement, you enter "---" (three hyphens) as the application ID. When you specify a constant (such as when adding the value 3 to a designated numeric field), you leave the application ID as spaces. The following is an example of an application ID reference:

```
    TFT   READ     TAR RECEIPT1               HOLD 0 FT 0 BY RECEIPT1 KEY
```

The component TAR refers to the application ID in which the RECEIPT1 file is defined.

When you are entering new ILF code with an application ID component, the application ID defaults to the current application but you can override it. APPX automatically changes the application ID to "---" if the referenced field (or file or process) cannot be found in the application specified but does exist in application 0LA, the runtime application. This lets you refer to predefined fields, files, or processes without having to change the application ID.

The File Name

The *file name* component identifies the name of a file you wish to operate on. You must specify the complete file name as defined in the APPX data dictionary. For example:

```
    T     READ     TAR RECEIPT1               HOLD 0 FT 0 BY RECEIPT1 KEY
```

The name RECEIPT1 refers to a file in the TAR application, which you want APPX to read.

The Field Component

Many statement types require you to specify either one or two *field* components, in which you enter a field name or value. Categories of field components are as follows:

- A ***defined field*** is one that is specified in the data dictionary of the current application, or in another APPX application. For example, ORDER1 CUSTOMER NO might be a defined field in the Order Processing (TOE) application.
- A ***work field***, like a defined field, is also specified in the data dictionary, but is not stored in a file.
- A ***predefined field*** specifies any one of APPX's predefined fields (PDFs). A complete list of the PDFs and a discussion of their nature and use are provided in Chapter 4-7: Predefined Fields and Processes.
- A ***constant*** includes any of four types of values, including numeric, alpha, date/time, or logic. Unlike some languages, you do not enclose a constant within quotation marks when you specify it in APPX. In addition to letters and numbers, constants may contain special characters. Refer to the discussion about specifying constant values later in this chapter.

Chapter 4-6: ILF Keyword Reference provides the syntax of each statement and also details the specific categories permitted for each field component. Unless a key field is specifically required, a defined field or work field can be used wherever the syntax calls for a field name. Predefined fields and constants can only be used where explicitly stated.

Partial-Record I/O

In order to increase performance associated with very wide RDBMS tables, you can specify either a file name or a field name when using any of these statements:

- READ
- READNEXT
- BEG READ and END READ
- WRITE
- REWRITE

If a file name is specified, APPX operates on the entire record. If a field name is designated, the I/O statement operates on the named field only. If the field is a group header, the I/O will be performed on all members of the group. Using this feature limits the amount of data transferred between APPX and an RDBMS. For example, if you know that a particular section of ILF code needs to update only one or two fields within a record, you can change the I/O statements to refer to those fields only.

Traditional method:

```
BEG READ TAR CUSTOMER               HOLD 1 KEY IS  CUSTOMER NO
COMPUTE  TAR CUSTOMER CREDIT LIMIT      *      1.05
REWRITE  TAR CUSTOMER
END READ TAR CUSTOMER
```

Using Partial-Record I/O:

```
BEG READ TAR CUSTOMER NO + CR LIMIT HOLD 1 KEY IS  CUSTOMER NO
COMPUTE  TAR CUSTOMER CREDIT LIMIT      *      1.05
REWRITE  TAR CUSTOMER CREDIT LIMIT
END READ TAR CUSTOMER NO + CR LIMIT
```

In the above example, the field CUSTOMER NO + CR LIMIT is a group header field containing synonyms of both the CUSTOMER NO and CUSTOMER CREDIT LIMIT fields.

When you WRITE a single field (or group field as shown above), APPX will write NULL values into the remainder of the record. When you READ (or READNEXT or BEG READ) a partial record, the remainder of the record will contain NULL values. If you REWRITE a partial record, the remainder of the record is not modified.

Other considerations:

- Even though the END READ statement allows you to specify a field name, this is for consistency purposes only and is not actually used other than to indirectly identify the file name so that the END READ can be associated with the appropriate corresponding BEG READ statement.
- REWRITE statements allow you to specify either a file or field name. If a field name is specified, only the named field or group of fields will be updated. All other values in the record being updated remain unchanged. The field name specified on the REWRITE statement does not need to be the same as the field name specified on the READ statement used to retrieve the record. The entire record can be read with hold by specifying a file name on the READ statement but just a single field can be updated by specifying that field name on the REWRITE statement. Or, you can READ a group of fields and REWRITE a subset of that group of fields.
- Partial-Record I/O only reads the field or group of fields specified. The key field specified on the READ statement is not read unless the key field is included in the group of fields specified on the READ statement. If the key field must be read, remember that a SYNONYM for the key field can easily be included in any group that you may define in the Data Dictionary. The object of Partial-Record I/O is to dramatically decrease the amount of data being retrieved and updated, therefore key fields that are not part of the READ statement are not retrieved.
- The READNEXT statement allows a field name to be entered (instead of a file name). However, that field name is not really used to restrict the set of fields retrieved from the RDBMS server. Because APPX is generating SQL statements as it executes, a “select <column-list> from <table> where <beg-at-constraints> and <end-at-constraints>” statement is generated when a BEG READ/END READ loop is executed. Executing that select statement causes the RDBMS to generate a record set that contains all of the selected columns for each selected record. The READNEXT statement (and the

END READ statement, as well) cannot change the record set, it can only retrieve the next row from that record set. Therefore, the field name supplied to a READNEXT or END READ statement is for consistency purposes only. You cannot use a READNEXT statement to change which columns are present in the record set.

The Label

Some of the control type statements allow you to identify the name of a specific point in an ILF routine where control passes. This name is called a *label*. You often use labels to cause a branching operation to be performed. For example:

```
    T     GOTO     :CALCTOTAL
```

The label :CALCTOTAL refers to the name of a section of the ILF routine to execute next. In this example, APPX passes processing control to the label :CALCTOTAL if the first internal true/false status indicator has a value of *T*.

> Note that the label in our example begins with a colon. This is because many programmers find a colon helps to make a label name stand out from ordinary field names but there is no requirement to use a colon.

The Occurrence Number

Many statements allow you to specify which *occurrence* of a multi-occurrence field you want to compare or operate on. For example:

```
    TT    IF       TGL COUNT                 001 EQ TGL COUNT                 002
```

The left-hand field in this example refers to the first occurrence of the field COUNT in the TGL application, and the right-hand field refers to the second occurrence of the field COUNT. You indicate if a field is to have more than one occurrence as a specification in the data dictionary.

Characteristic-Type Statements

Characteristic-type statements, such as BLANK, BLINK, BRIGHT, INVERSE,.and others in this category, operate on items displayed on an image, rather than on fields. You use these statements to affect how an item appears to an end-user on the screen or in a report. In other words, they affect the external data representation only, not the contents of the field itself. Because you can paint a single field more than once on an image, an extra parameter called the appearance number is required to uniquely

identify an item. Appearance numbers (starting at 1) are assigned to each occurrence of each field beginning at the top left-hand corner of an image, continuing across the row to the right and, then, to each subsequent row in the same manner. The following example demonstrates this pattern:

FIELD A (OCCURRENCE 1) (APPEARANCE 1)	FIELD A (OCCURRENCE 1) (APPEARANCE 2)
FIELD B (OCCURRENCE 1) (APPEARANCE 1)	FIELD A (OCCURRENCE 2) (APPEARANCE 1)
FIELD A (OCCURRENCE 2) (APPEARANCE 2)	FIELD A (OCCURRENCE 1) (APPEARANCE 3)

In addition, these statements are meaningful only in event points that are associated with an image (such as *Pre-Display* or *Verify*). Compiling a characteristic statement in a non-image event point is not an error; instead, the statement is ignored. This makes copied routines (using COPY or GOSUB statements) more flexible. A routine with a characteristic instruction may be copied into a non-image event point; therefore, the characteristic instructions verify at runtime that they are operating in an image-associated event point.

In general, the operation of a characteristic statement affects only the next generation of an image. In other words, the item characteristics are reset each time an image is generated. It is important to note that a characteristic statement has no effect until the specified item(s) become active. An item becomes active when the current entry level is greater than or equal to the entry level specified for that item.

Characteristic-Type Statement Keywords

BLANK, BLINK, BOLD, BRIGHT, COLOR, DIM, DISPLAY, FONT, INVERSE, NO BOLD, NO ULINE, NORMAL, PICTURE, ULINE

Example

In the following example, if the customer number entered on an order is not a valid customer, the customer name displays as blank.

```
        SET      TAR CUSTOMER NO               =  TAR ORDER CUSTOMER NO
        READ     TAR CUSTOMER              HOLD 0 FT 0 BY CUSTOMER NO
F       BLANK    TAR CUSTOMER NAME             (AT APPEARANCE #    )
```

The Appearance Number

Many of the characteristic-type statements allow you to specify which iteration, or *appearance number*, is to be affected by the statement. An appearance number is applicable only when a field appears more than once on an image. For example:

```
    T     INVERSE  TAR CUSTOMER NAME             (AT APPEARANCE # 002)
```

This example indicates that the system should make the CUSTOMER NAME field in the TAR application appear as inverse on the terminal for the second appearance only of this field on the image. The text, *AT APPEARANCE #*, is automatically provided by the ILF as part of this statement's format template.

The Arithmetic Operator

The *arithmetic operator* component includes the four basic arithmetic operator signs and two hash symbols:

- + (plus)
- - (minus)
- * (multiply by)
- / (divide by)
- ***P*** (hash-plus or -add)
- ***M*** (hash-minus or -subtract)

A hash total allows you to validate the accuracy of certain fields. For example, you can sum the numeric equivalent of a customer number field as a check on operator input.

The Relational Component

The *relational* component (or *relation*) includes several abbreviations that correspond to a series of relational operators that are used when comparing the values of two fields, as follows:

LT (less than)
Checks if the value of a first-named field is less than that of a second field.

GT (greater than)
Checks if the value of a first-named field is greater than a second field.

LE (less than or equal to)
Checks if the value of a first-named field is less than or equal to a second field.

GE (greater than or equal to)
Checks if the value of a first-named field is greater than or equal to a second field.

EQ (equal to)
Checks if the value of a first-named field is equal to that of a second field.

NE (not equal to)
Checks if the value of a first-named field is not equal to that of a second field.

IN (includes)
Checks if a specified character string is included within the value of a field. For example, you could check to see if the letter string *NOR* is included in the CUSTOMER NAME field.

EX (excludes)
Checks if a specified character string is excluded from the value of a field. For example, you could check to see if the letter string *NOR* does not exist in the CUSTOMER NAME field.

The Message Text

Message text is a free-form text area that allows you to display messages to a user. You can enter up to 64 characters in this area, including alphabetic, numeric, and special characters. If you specify leading spaces, they show accordingly in the displayed text.

The FAIL (or FT) Parameter

For many of the statements that set true/false status indicators, the statement format also offers a *fail action type* parameter. These statements include the process-type statements (such as INPUT, OUTPUT, STATUS, etc.) and certain file-related statements (such as CREATE, IF EXIST, READ, etc.) as shown in the example below.

```
          STATUS   TAR END OF MONTH STATUS              SUBPROCESS   END? Y  FAIL 3
or
          READ     TAR UNPAID                 HOLD 0 FT 2 BY UNPAID KEY
or
          SCRATCH  TAR TEMP               FAIL 0 CACHE? N
```

The *FAIL* (or *FT*) parameter controls what additional action to take if the statement returns a false condition, as follows:

- 0 = Do nothing,
- 1 = Issue a warning message,
- 2 = Issue an error message, or
- 3 = Cancel.

> Note that the system message generated according to a *FAIL* parameter is descriptive in nature and may not contain sufficient detail to locate the point of failure in the program. For instance, if an OPEN statement with *FAIL* = 2 were to fail, the message generated could simply be "Error - File Does Not Exist". Therefore, it might be desirable to code WARNING or ERROR messages on a false condition as an alternative or to provide supplemental information as shown in the example below.

```
          OPEN     TGL ACCTMS    SHARE? Y  FAIL 2 CACHE? N
    F     ERROR    Unexpected Error OPEN TGL ACCTMS (see error message above)
```

However, if a cancel is issued, either by use of *FAIL* = 3 or by a CANCEL statement, any supplemental warning/error messages will only appear if an output report or error log is also generated.

File-related statements with the *FAIL* parameter that return a false condition also set the STATUS CODE predefined field, a token field containing the reason for the I/O command failure. It can be used to obtain a descriptive error message from the system message file (this is the same message that would be generated by a FAIL parameter set to 1-3).

A Non-APPX Program Name

There are two ILF statements that can invoke a non-APPX program name. Use RUN to execute complete external programs. Use CALL to call external subroutines. Most operating systems support dynamic link libraries. If the subroutine you wish to call is part of a dynamic link library, you can invoke it directly in APPX by using the CALL statement. If a subroutine must be statically linked, you will have to create a custom APPX engine in order to invoke it. Contact your system administrator for details regarding this procedure.

When you invoke external programs and subroutines with the RUN and CALL statements, you specify the exact program name that you would enter if you were communicating directly with the native operating system on a command line. If you do not have enough spaces to specify the name completely, you can define a work field in the data dictionary and set its value to the complete program name. In turn, specify the work field name in the RUN or CALL statement to achieve the desired effect.

Another reason to consider using a work field definition is ease of maintenance. You may RUN or CALL the same program or routine in several places in your application environment. If for any reason you need to change the name of a program and had specified it directly in a RUN or CALL statement, you must change each statement where the name is specified. If you define a work field in the data dictionary, you only have one change to make if the program name changes.

Miscellaneous Operands

There are a few other miscellaneous operands that are unique to a specific statement or to a category of statements. For example:

```
          BEG LOOP A  = 001 TO 010  STEP 001
          *        (perform processing
          END LOOP A
```

Here, the BEG LOOP and END LOOP statements both have a loop index operand while the BEG LOOP statement also has operands for "from", "to", and "step" values.

Table 4-1-1 provides a complete list of statement keywords by category, and *Chapter 4-6: ILF Keyword Reference* describes each statement in detail, including any unique operands. The format template displayed after entering the keyword in the ILF editor also assists you in determining the type of entry that is required.

Using Predefined Fields (PDFs) in Statements

In the runtime application, APPX includes and maintains a number of fields called *predefined fields* or *PDFs*. These fields contain various "points of reference," such as who the current user is and the company address. Although the names of these fields are pre-specified, you can use them as you would any field that you defined in the data dictionary of your own applications. The primary difference between predefined fields and those you specify in the data dictionary is that APPX initializes the values of PDFs and automatically maintains these values at all times. As a designer, you can set or modify the values of certain PDFs with statements, and may thereby alter this pattern of automatic maintenance. Within the context of statements, you can examine the value of a PDF at any point in time, modify a PDF's value (if allowed), and display a PDF in a report or on a screen image.

There are five categories of predefined fields:

- ***Current status PDFs*** provide information about current values, such as the current user ID.
- ***Process control PDFs*** control what will happen next in the application.
- ***Terminal management PDFs*** provide information about such factors as current cursor position.
- ***Report management PDFs*** provide information about the disposition of reports, such as the print destination.
- ***General purpose PDFs*** are used as index work variables or work fields in ILF statements.

Each PDF has a number of attributes that uniquely define how it is initialized and maintained by APPX. A table of PDFs and a complete description of each can be found in Chapter 4-7: Predefined Fields and Processes.

PDFs and Field-Naming

In the APPX environment, you always specify field names within the context of an application ID. Therefore, you could specify CUSTOMER NAME as a field name within application TAR, and CUSTOMER NAME as a completely different field name within application TOE, if desired. Because PDFs by definition exist within the application ID (---), you are allowed to use the same name when you define fields or work fields in one of your own applications. Thus, the predefined field USER ID is different from a field named USER ID in the TGL application, for example.

The A through Z and AI through ZI PDFs

Within the overall context of PDFs, there are two sets of fields that have special uses as index work variables:

- ***The fields A through Z*** are a special set of 26 floating-point numeric fields reserved for use in ILF statements. You can use the fields *A* through *Z* as index work variables for any type of calculation. These fields can also be used as index work variables in BEG LOOP/END LOOP and SET TEMP statements and in place of occurrence numbers and/or appearance numbers wherever such numbers occur. When used in calculations, the number of digits to the right of the decimal varies to ensure the retention of 15 significant digits at all times. For example, if predefined field *A* contains a value of 10 and is divided by 3, *A* is set to 3.33333333333333.
- ***The fields AI through ZI*** are a special set of 26 integer fields. You can use the fields *AI* through *ZI* as index work variables for calculations which require only integers. These fields can also be used as index work variables in BEG LOOP/END LOOP and SET TEMP statements and in place of occurrence numbers and/or appearance numbers wherever such numbers occur. The range of values for these predefined fields is from –2,147,483,648 to +2,147,483,647, inclusive. Any calculations resulting in decimal positions are rounded up to the next higher integer. For example, if predefined field *AI* equals 10 and is divided by 4, the result (2.5) is rounded up to 3, and *AI* is set to a value of 3. If predefined field *AI* equals 9 and is divided by 4, the result (2.25) is rounded down to 2, and *AI* is set to a value of 2.

Specifying Constant Values

You can use constant values in certain ILF statements where a field name is allowed. You leave the application ID blank when specifying a constant value. Because there are different types of fields included within an APPX application, you need to understand exactly how to specify each data type as a constant value.

APPX ***stores or communicates data in any of three formats***, and automatically translates between these formats as appropriate. The three data representations are:

- The ***external or display format*** communicates data to an application user. This format includes characters such as $, -, and #, which often provide a more meaningful presentation to a user.
- The ***internal or stored format*** allows APPX to store and manage data in a specified (and usually optimum) manner. For example, you may choose to store a numeric money field in packed decimal format.
- The ***universal format*** allows you to reference a field within the context of ILF statements, regardless of how the data is stored or displayed.

Use the *universal format* to specify constants as fields within statements.

Universal Data Formats

Each of the four basic data types has its own universal format which is used for specifying constants within ILF statement and is described below:

- A ***numeric constant*** consists of digits, an optional decimal point, an optional trailing or leading minus sign, or a blank for a null entry. A plus sign is implied for positive numbers but is not allowed as a symbol. Currency signs, such as $, are not allowed.
- A ***date/time constant*** consists of a mandatory series of 16 digits (or hyphens, as described below), or a blank for a null entry. The 16 digits incorporate eight sets of two digits each that represent, from left to right, century, year, month, day, hour, minute, second, and hundreds of a second. To indicate that a particular part of a complete date/time format should be ignored in a comparison or operation, you can enter a pair of hyphens in the appropriate positions. For example, you would enter December 31st as "----1231--------".
- An ***alphanumeric constant*** consists of any combination of letters, numbers, and special characters. Leading, embedded, or trailing spaces are allowed. Leading and embedded spaces are treated as significant for comparison purposes.
- A ***logic constant*** is one that contains the answer to a "yes/no" question, such as, "Is this a domestic customer?" The valid values for a logic-type field within an ILF statement are:
- The **number 1** or the **letter Y** is equivalent to yes or affirmative or ☑.
- The **number 0** or the **letter N** is equivalent to no or negative or ☐.
- A **blank** (space) is equivalent to an unknown response or ?. It means the answer is not known or that the field value is not filled in yet.

CHAPTER 4-4

The ILF Editor

The Editor and Automatic Compilation

The ILF includes an integrated ILF editor that allows you to create, modify, and delete individual statements, sections of ILF routines, and entire routines. When you are finished with your work on an ILF routine, APPX automatically takes care of *compiling* the routine when an end user first executes the process from a runtime menu. You do not need to perform a separate step to compile a routine. This automatic compilation translates the source statements you enter into optimized internal instructions that APPX uses when a process executes.

The Features of the Editor

The ILF editor includes many features that allow you to manage statements and ILF routines. With the ILF editor, you can:

Create new ILF routines

By selecting the desired event point option from one of the process component definition screens.

- *Create stand-alone subroutine processes*.

Recall existing ILF routines

And modify them.

Edit (add, change, delete) statements

With a full-screen editing facility that provides entry format "templates" as a guide.

Copy existing routines

To use as a starting point for developing similar routines.

"Drill down"

To view or edit fields referenced by the current process. This is accomplished by selecting the referenced field, then selecting `OPTION` `94`. Upon exiting the referenced field, you are returned to the original process.

Invoking the ILF Editor

You invoke the ILF editor from within the context of process definition by selecting the appropriate *Rules* option displayed near the bottom of the screen. The editor automatically associates the statements you enter with the process/event point you selected. *Figure 4-4-1* is an example of a *Rules* overlay where you may choose to invoke the editor. Note that only the event points that currently contain ILF statements are highlighted.

The ILF routine you develop is "attached" to the appropriate process and event point within your application, so that it is invoked when an end user runs the process.

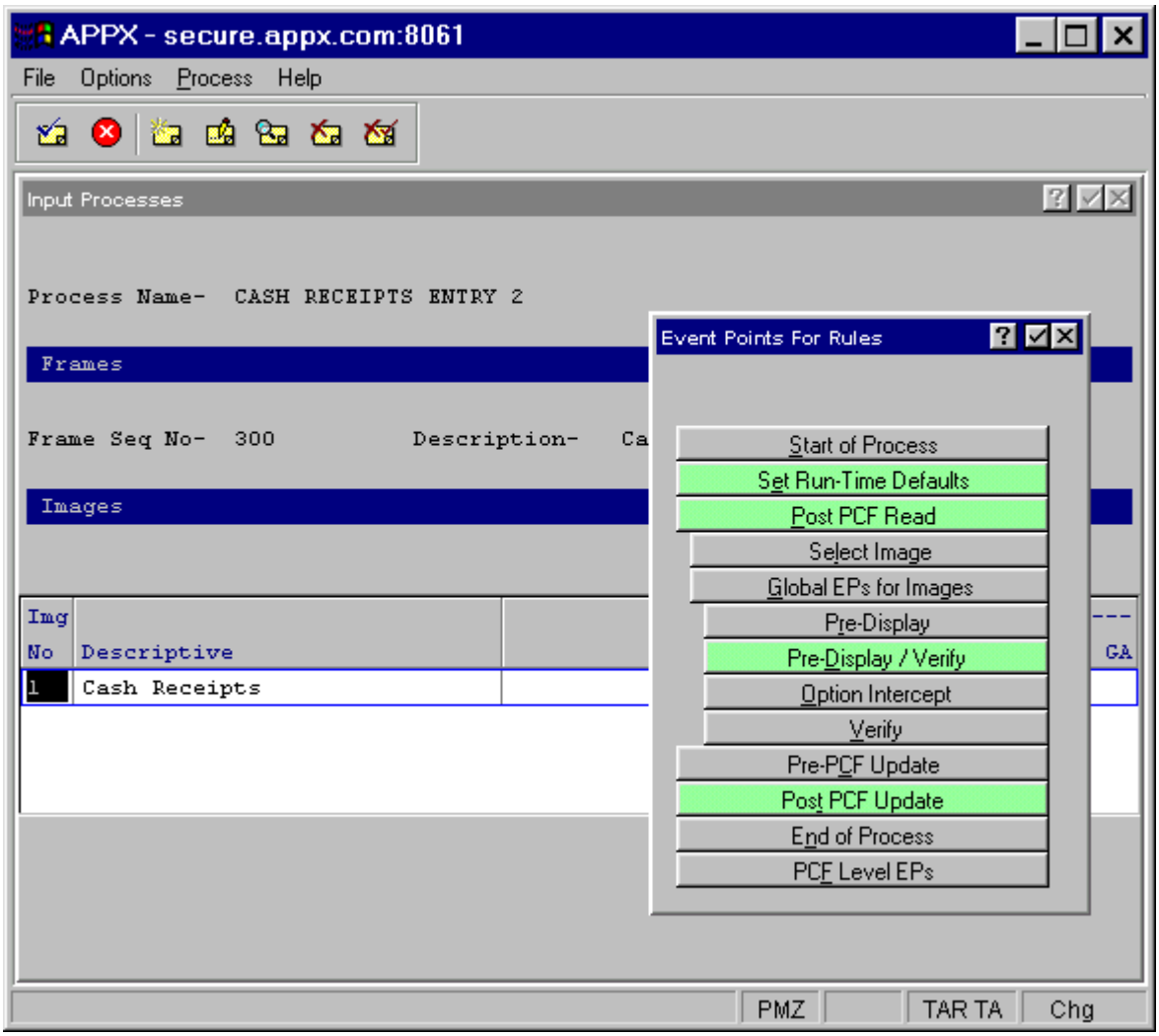


Figure 4-4-1. Invoking the ILF Editor

The Layout of the Editor Screen

An ILF Editor screen consists of three separate areas. Refer to the figure that highlights each area.

- The ***identification line*** identifies the process component and the event point that the ILF routine is associated with. After you invoke the editor, the identification line is filled in and the cursor is positioned in the statement editing area. The components are identified, starting with the process name and continuing (if applicable) with the frame number, image number, and child ID. In the center of the identification line, APPX displays the process type and, in parentheses, the event point. In *Figure 4-4-2*, the identification line is highlighted.

An ILF Editor screen

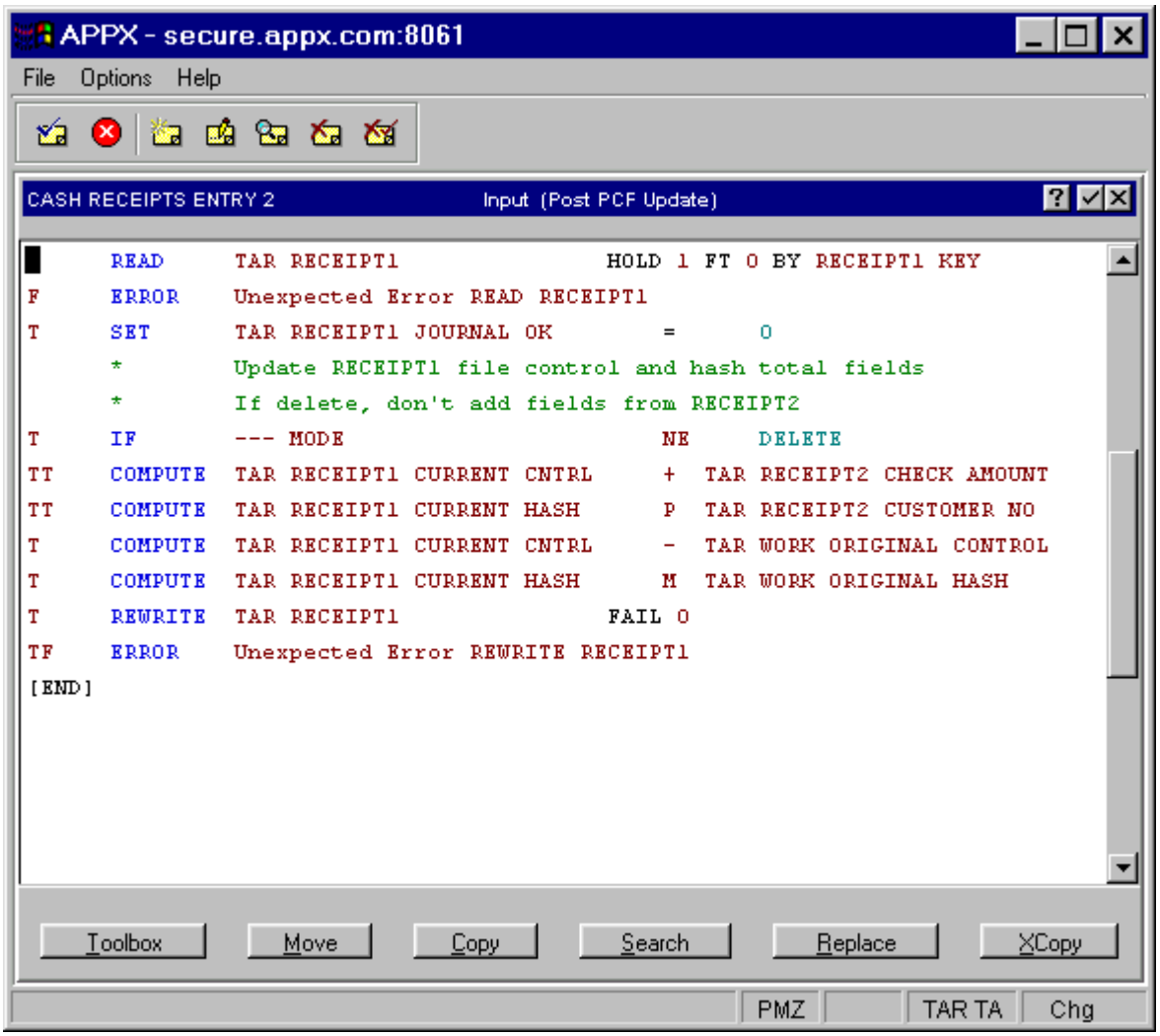


Figure 4-4-2

- The ***statement display and edit area***, shown in *Figure 4-4-3,* in an ILF Editor screen shows individual statements. This area is where statements display, and where you enter and modify them.

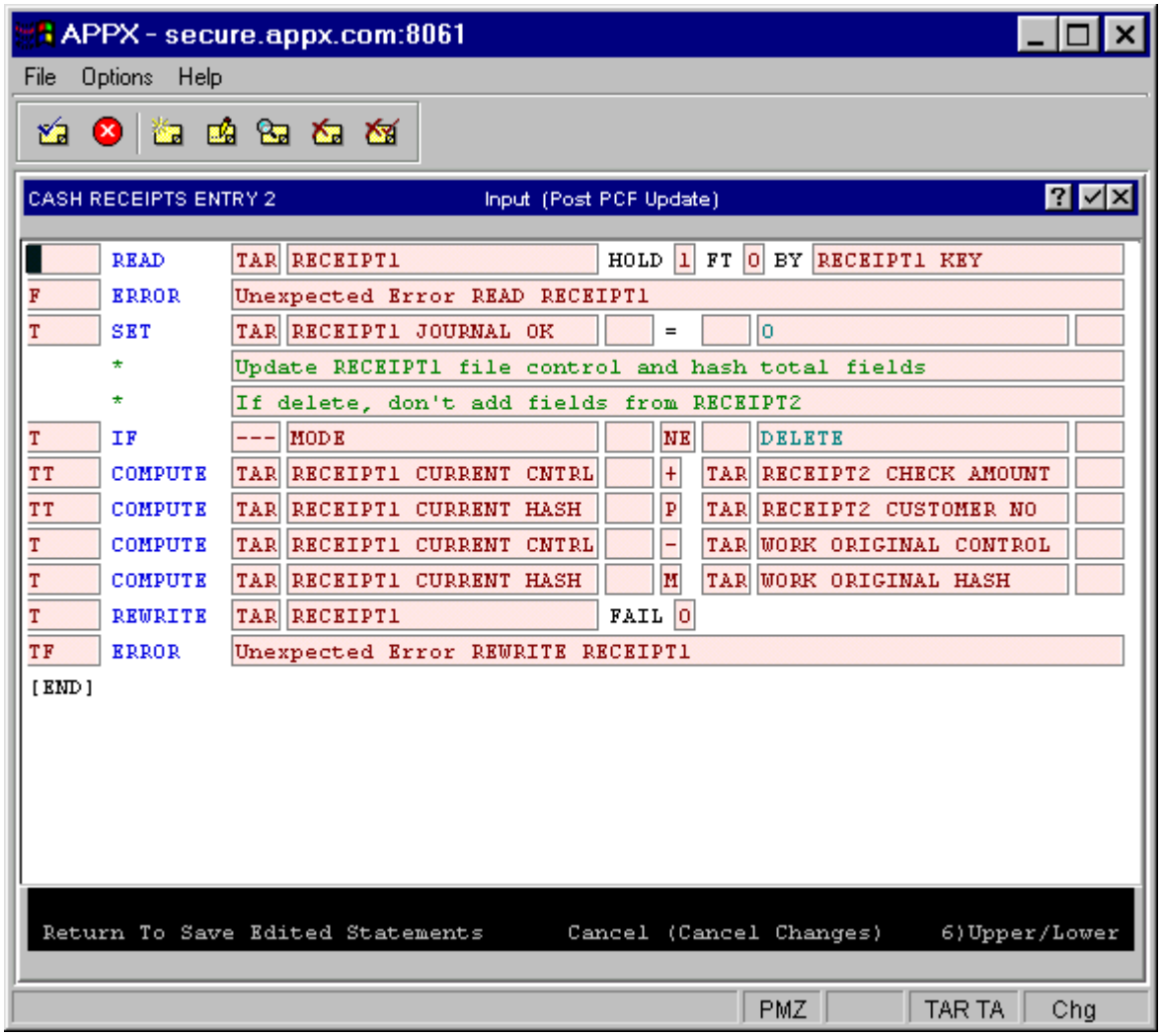


Figure 4-4-3. An ILF Editor Screen's Display and Edit Area

- The ***options*** found at the bottom of the ILF Editor screen can be selected to perform editing functions. The options are shown in *Figure 4-4-4*.

Options in an ILF Editor screen

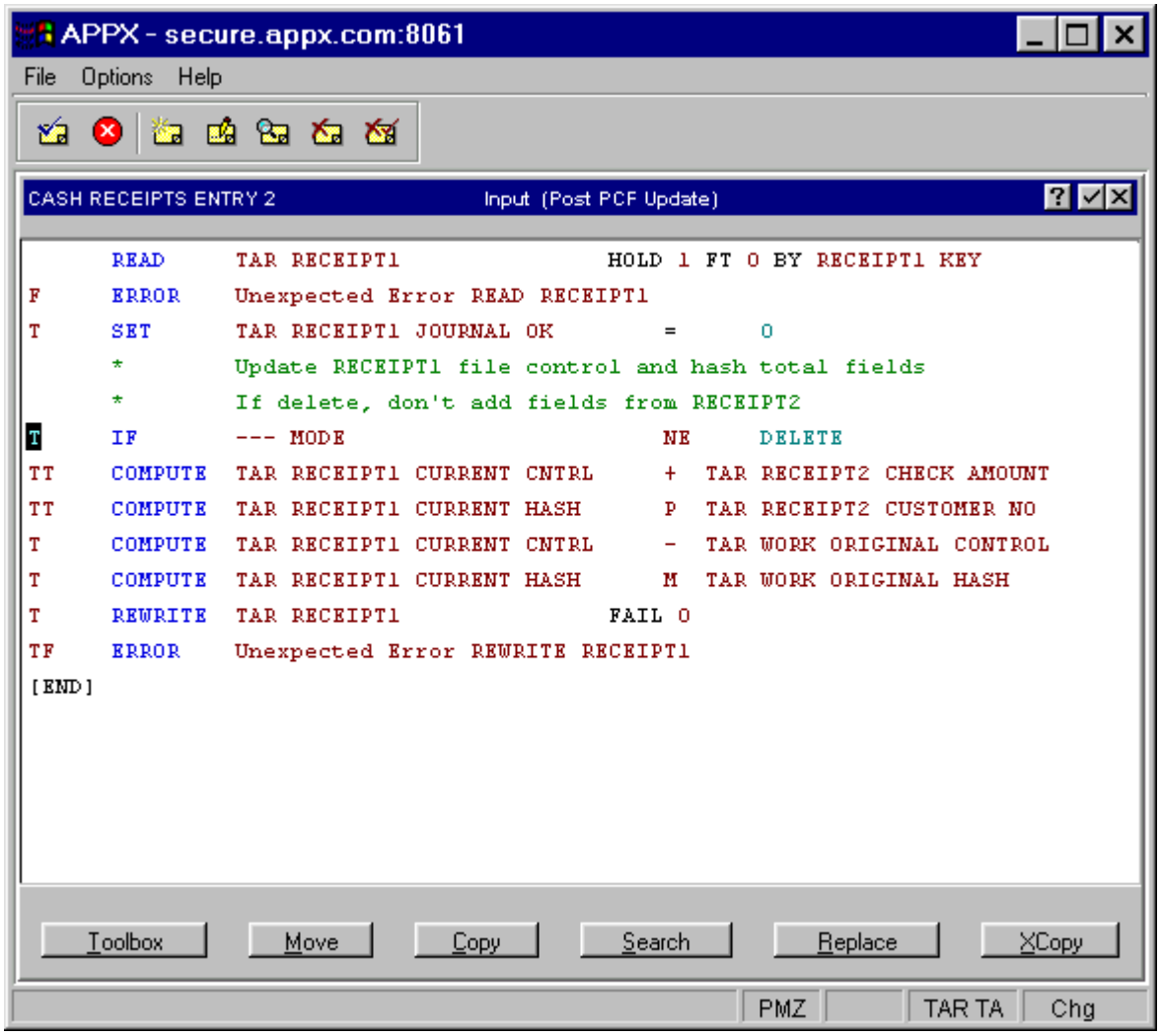


Figure 4-4-4

The Help Option

The HELP option is available on every editor screen. Press HELP (F1) to display a list of all options that are used in any of the editor functions. Select END or press ENTER to return to the screen that displayed when you selected HELP. The screen in *Figure 4-4-5* is the ILF Editor *Help* screen:

ILF Editor Help screen

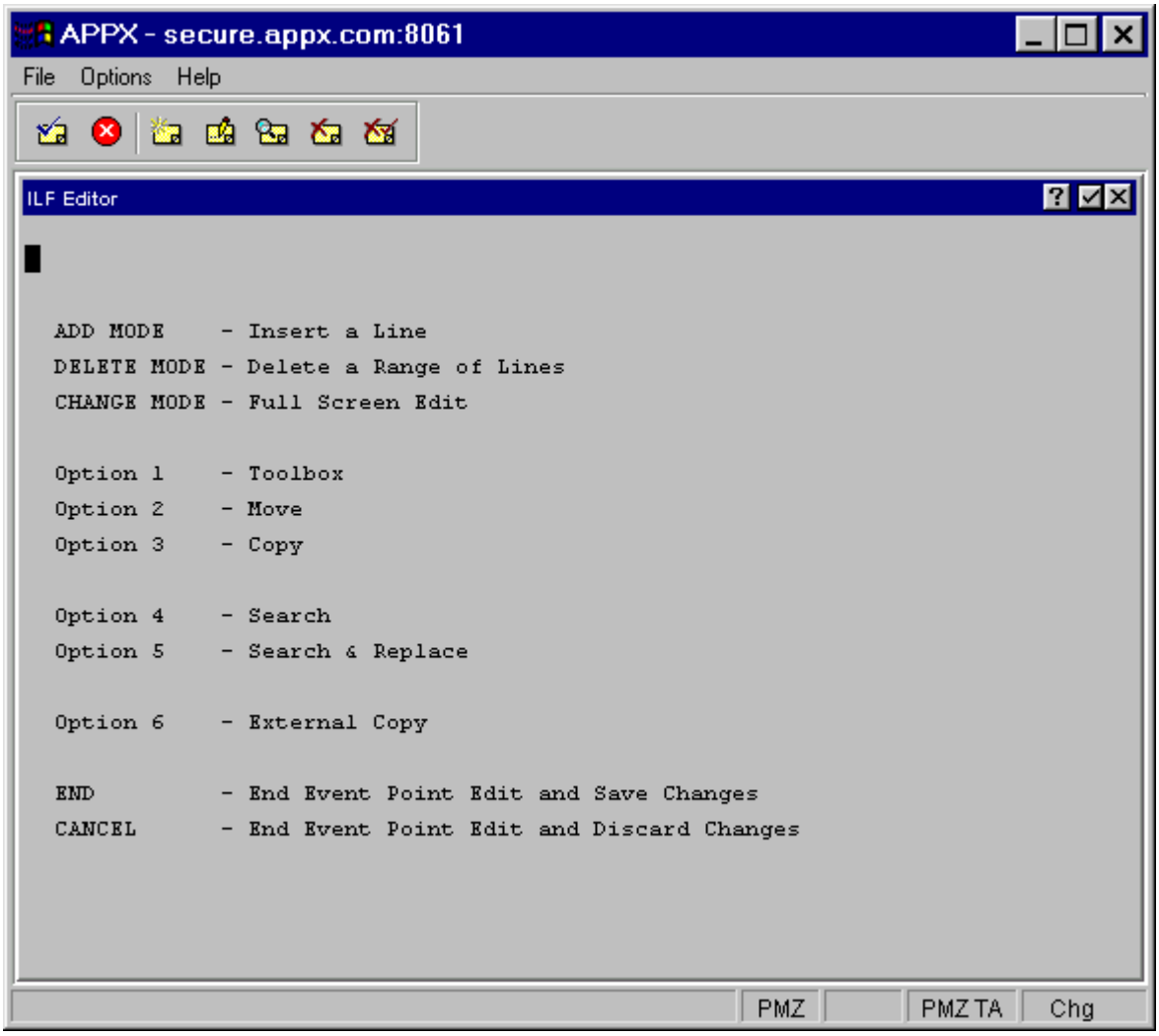


Figure 4-4-5

The Edit Modes and Options

When you enter the editor, you are initially placed in display mode. If you are reviewing or editing an existing routine, the first statements display on the screen (the exact number may vary, depending upon your terminal). If you are creating a new routine, all you see is [END] that indicates the end of the statements. To review existing statements, select from among the standard scrolling options to move around in an ILF routine. You can also use the arrow keys to position to a specific statement. If you want to perform any function other than displaying statements, you can select any of several other options. The following options are available (see corresponding sections for details):

Save and Exit

SAVE and EXIT

Select this icon from the tool bar to permanently save all the changes you make to an existing ILF routine, or to save a newly created routine.

Add Mode allows entry of new statements.

Delete Mode allows deletion of existing statements.

Change Mode allows editing of existing statements.

Toolbox allows quick access to the Database Management option.

Move provides for moving selected statements within the ILF routine.

Copy provides for duplicating selected statements within the ILF routine.

Search provides for searching the ILF routine for specific text.

Replace provides for replacing specific text within the ILF routine.

XCopy (external copy) provides for copying selected statements from another ILF routine.

Delete All Statements (Option 10) provides for deleting the entire ILF routine.

Restore All Deleted Statements (Option 11) provides for restoring.statements deleted by the last Option 10 assuming it was not followed by an End (Save and Exit).

Add Mode

Select this option from the tool bar when you want to enter statements in a new ILF routine, or when you want to insert additional statements in an existing routine. First, position the cursor anywhere on the statement *before which* you want to insert a new statement line and then select add mode. The screen that displays when you select this option differs from other editor screens: there is one blank line, the keyword area shows pseudo blanks, and the cursor is positioned on the first character of the keyword area. The editor is effectively reserving one line so you can enter a new statement. Now enter the desired keyword and then press ENTER. The screen at this point appears as follows:

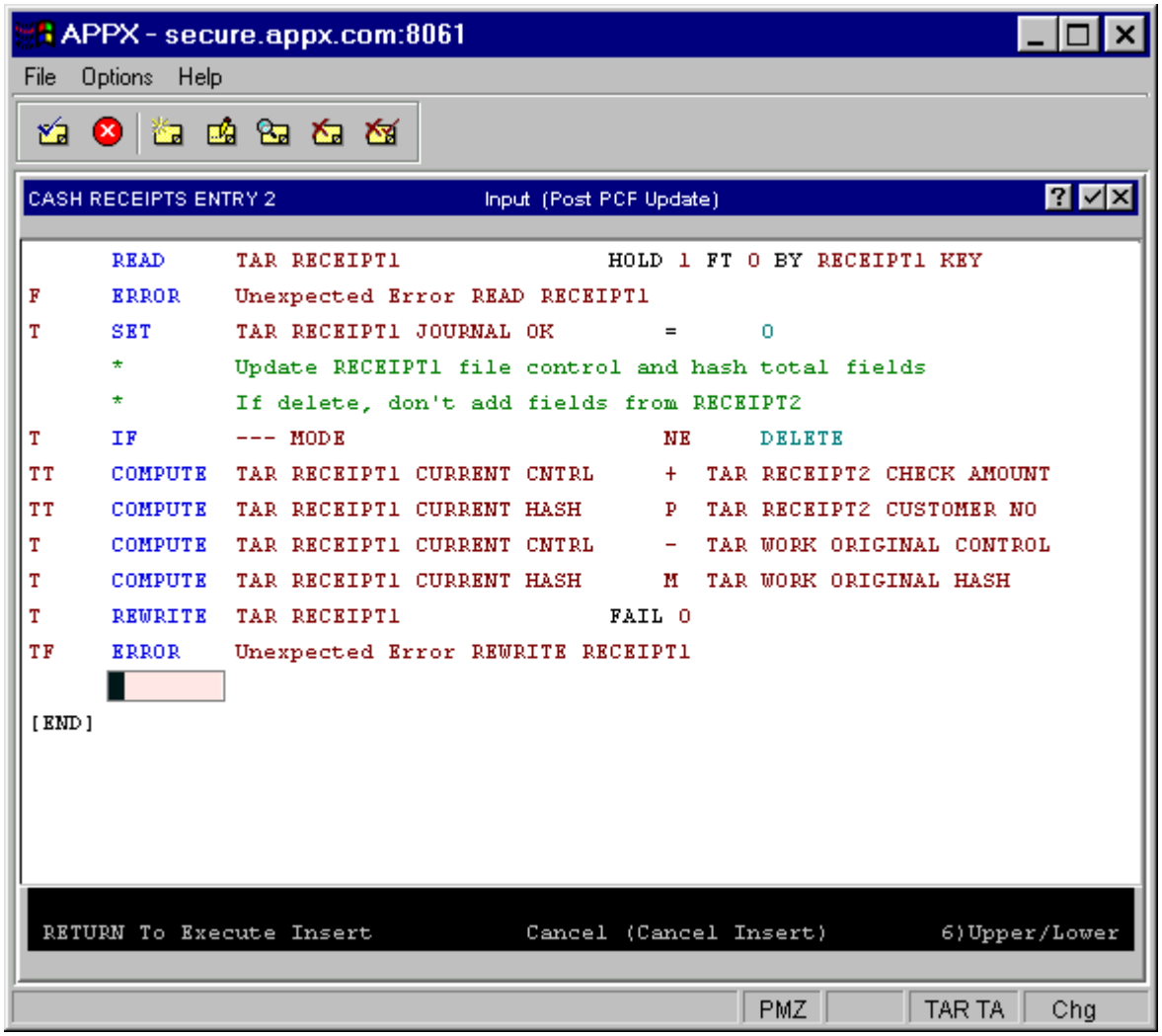


Figure 4-4-6. An ILF Editor Screen in Add Mode

After you press ENTER, the appropriate format "template" for the statement type you specified with the keyword displays. For example, the screen in *Figure 4-4-7* displays for a SET statement.

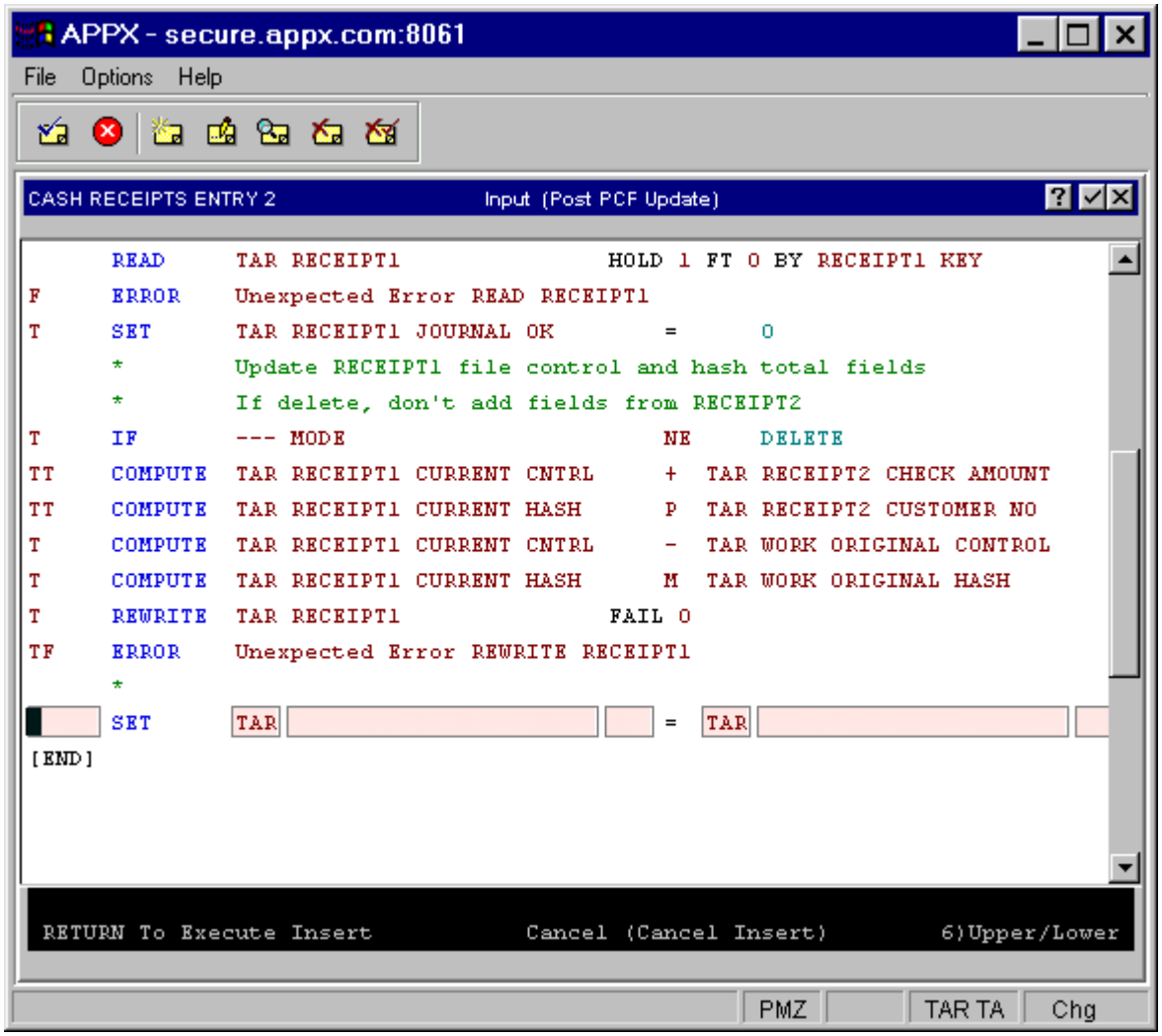


Figure 4-4-7. A SET Statement Format Template in Add Mode

The editor may display some default field values that you are likely to use, such as the application ID. You can override these defaults by entering the desired values or by entering spaces. When you enter a field name, APPX automatically searches for that name in the specified application's data dictionary. If it finds the specified field, it leaves the application ID as is. If it does not find the field, it next searches for the field in the runtime application's data dictionary (where all predefined fields are defined). If it finds the specified field, APPX automatically changes the application ID to "---" to signify the runtime application. If it does not find the field in either place, the editor provides an error message.

You now must enter all the statement operands that are appropriate for the keyword you specified. Table 4-1-1 provides a complete list of statement keywords by category, and *Chapter 4-6: ILF Keyword Reference* describes each statement in detail, including syntax requirements and available options. After you completely specify a statement, press ENTER to save the inserted statement in the existing routine. Then, APPX allows you to insert another statement by reserving a new line with pseudo blanks in the keyword area. The same cycle continues until you complete inserting all your statements. When you finish, leave the keyword blank and press ENTER to return to display mode.

The following options are available in add mode:

- **RETURN TO EXECUTE INSERT** Press ENTER when you want to insert the current statement.
- **CANCEL (CANCEL INSERT)** cancels the insert. Selecting this option from the tool bar returns you to display mode. Any statements you entered on the screen are lost.
- **UPPER/LOWER** allows you to enter upper- and lowercase letters. Some statements automatically default to upper/lowercase. Examples are CANCEL, ERROR, MESSAGE, and WARNING. Most other statements default to all uppercase. Sometimes you may want to specify message text that uses upper- and lowercase appropriately, without using the MESSAGE statement. You select this option to "toggle" between the uppercase-only and upper-and-lowercase modes.

Delete Mode

Select this icon from the tool bar when you want to delete a single statement or a group of contiguous statements within an ILF routine. Position your cursor anywhere on the first (or only) line you want to delete. When you select delete mode, two things happen:

You automatically set a "delete marker" on the line where the cursor is positioned, and

You enter into delete mode with its own set of options.

When you select delete mode, the screen in *Figure 4-4-8* displays.

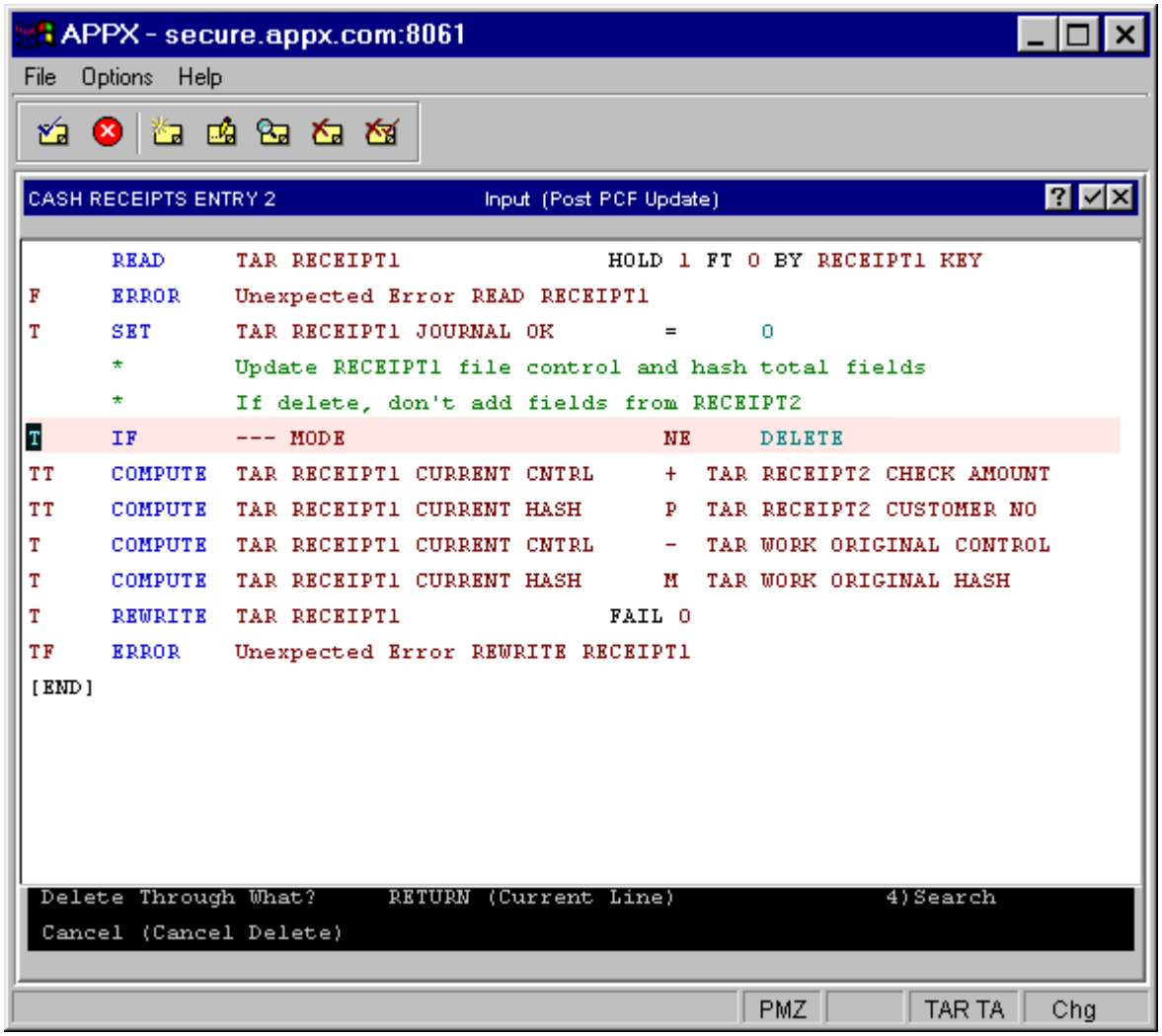


Figure 4-4-8. An ILF Editor Screen in Delete Mode

The question ***Delete Through What?*** is now on the left side of the option area. Each option is described as follows:

- RETURN **(CURRENT LINE)** By positioning the cursor (using *First, Last, Previous, Next, Down, Up,* the arrow keys, or SEARCH, described later in this chapter), you indicate to the system a second "delete marker." The second marker, together with the original marker that you set before you selected delete mode, specifies the range of statements you want to delete. The second marker can be above (before) or below (after) the original marker in an ILF routine. The cursor can be positioned anywhere on the statement line. To delete a single statement, do not move the cursor from its current position. When you press ENTER, all statements within the specified range (including the statements with markers) are deleted. Until you press ENTER, no deletion occurs; the ENTER button triggers the delete operation.
- **CANCEL (CANCEL DELETE)** cancels the deletion. Selecting this option from the tool bar returns you to display mode. No statements are deleted if you select this option.

- **SEARCH** allows you to search for a particular word or string of characters to locate a particular statement line. Refer to the description of the SEARCH option later in this chapter.

Change Mode

When you select this option, you are in full-screen editing mode. Up to an entire screen of statements displays, available for editing. In change mode, you can make modifications to any component of a statement except the keyword, which is protected from update. When you select change mode, the screen in *Figure 4-4-9* displays.

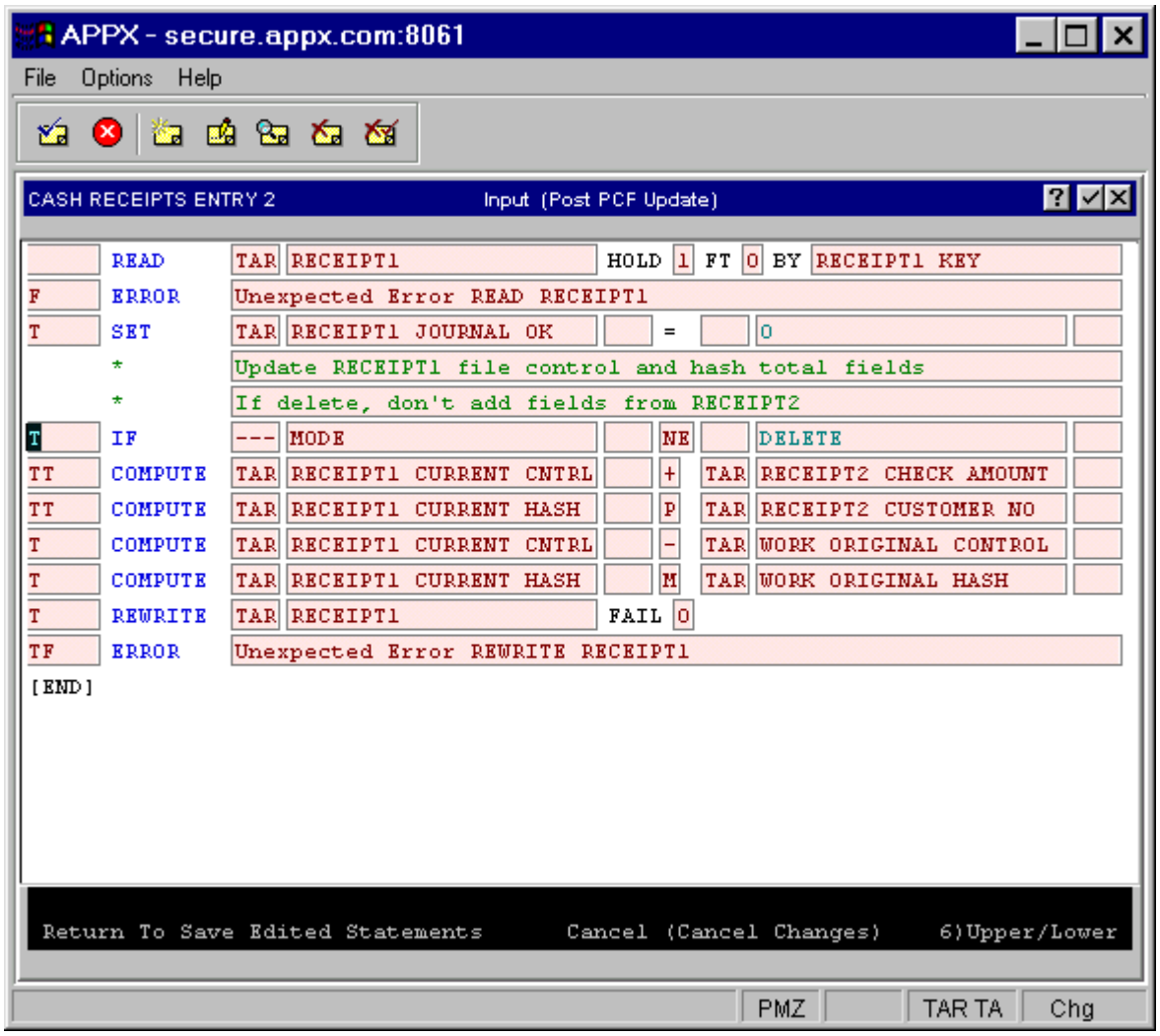


Figure 4-4-9. An ILF Editor Screen in Change Mode

The following describes the options presented in a change screen:

- **RETURN TO SAVE EDITED STATEMENTS** Press ENTER to save an entire screen of edited statements as modified.

- CANCEL (CANCEL CHANGES) cancels the modifications. Selecting this option from the tool bar returns you to display mode. Any changes that you just keyed on the screen are lost.
- UPPER/LOWER toggles between uppercase-only and upper-and-lowercase mode. APPX usually defaults to uppercase-only mode. Refer to the previous description under ADD MODE for a complete set of rules for using the UPPER/LOWER option.

Toolbox

The toolbox option allows you to have access to the DATABASE MANAGEMENT option with all of its facilities for maintenance of domains, files and fields, and work fields. *Figure 4-4-10* is an example of the ILF Editor Toolbox overlay.

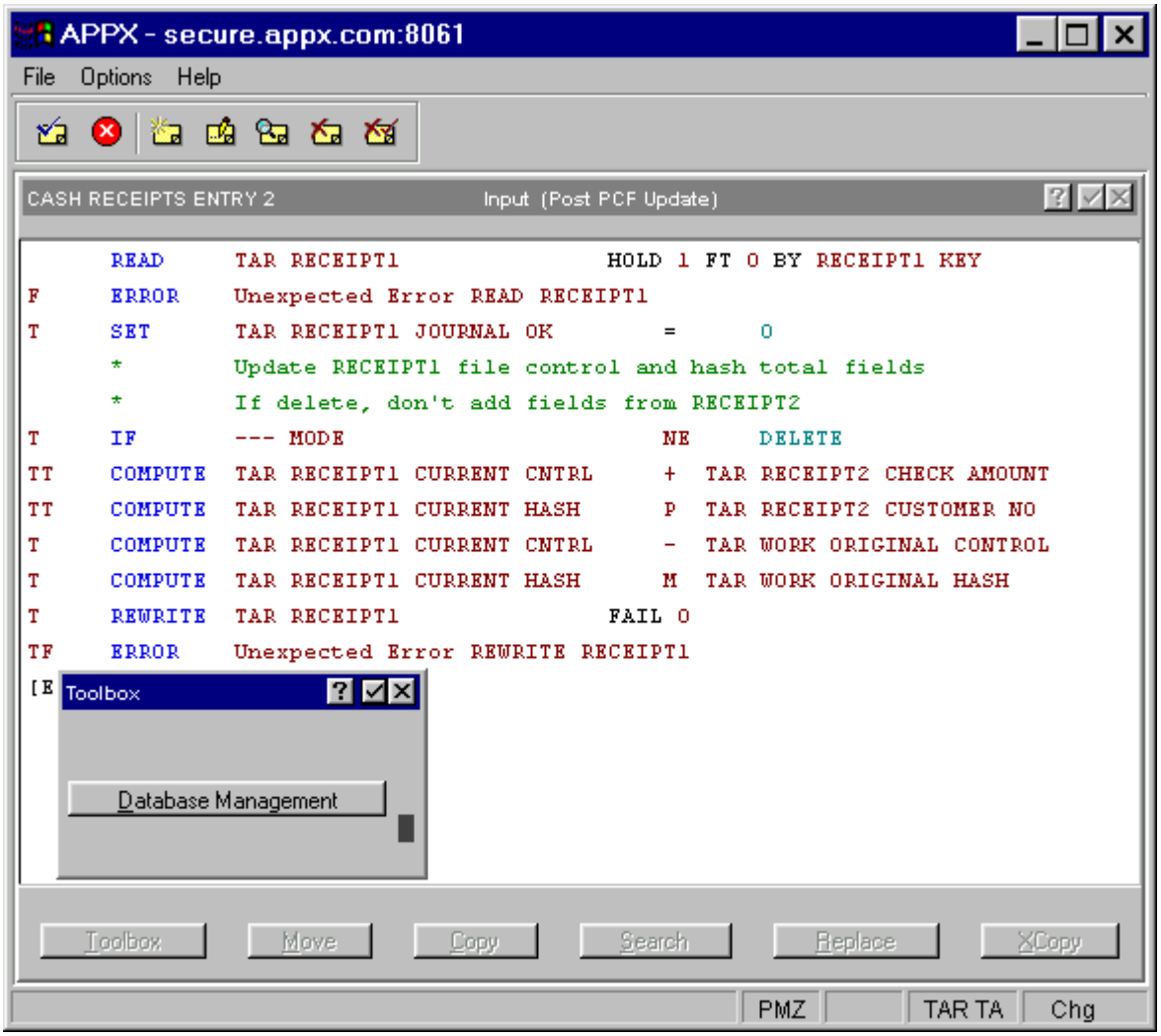


Figure 4-4-10. ILF Editor Toolbox Overlay

Move

Select this option when you want to move a single statement or a group of contiguous statements within an ILF routine. Position your cursor anywhere on the first (or only) line you want to move. When you select the MOVE option, two things happen:

You automatically set a "move marker"
On the line where the cursor is positioned, and

You enter into move mode
With its own set of options.

- When you select move mode, the screen in *Figure 4-4-11* displays.

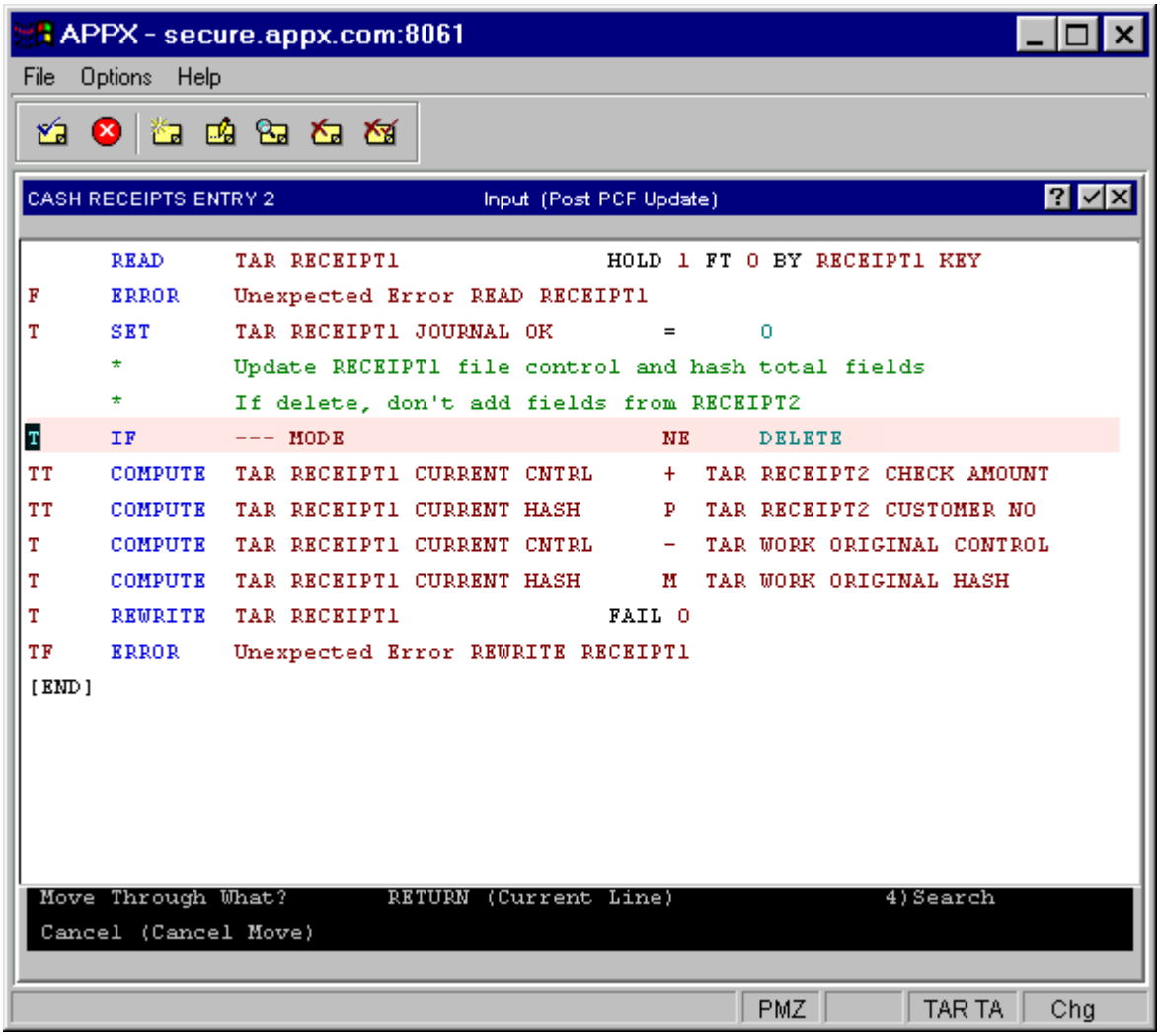


Figure 4-4-11. First Move Mode Screen in the ILF Editor

The question *Move Through What?* is now on the left side of the option area. Each option is described as follows:

- **RETURN (CURRENT LINE)** By positioning the cursor (using *First, Last, Previous, Next, Down, Up,* SEARCH, or the arrow keys), you indicate to the system a second "move marker." The second marker, together with the original marker that you set before you selected move mode, specifies the range of statements you want to move. The second marker can be above (before) or below (after) the original one in the ILF routine. The cursor can be positioned anywhere on the statement line. To move a single statement, do not move the cursor from its current position. When you press ENTER, the screen in *Figure 4-4-12* appears.

Second Move Mode screen

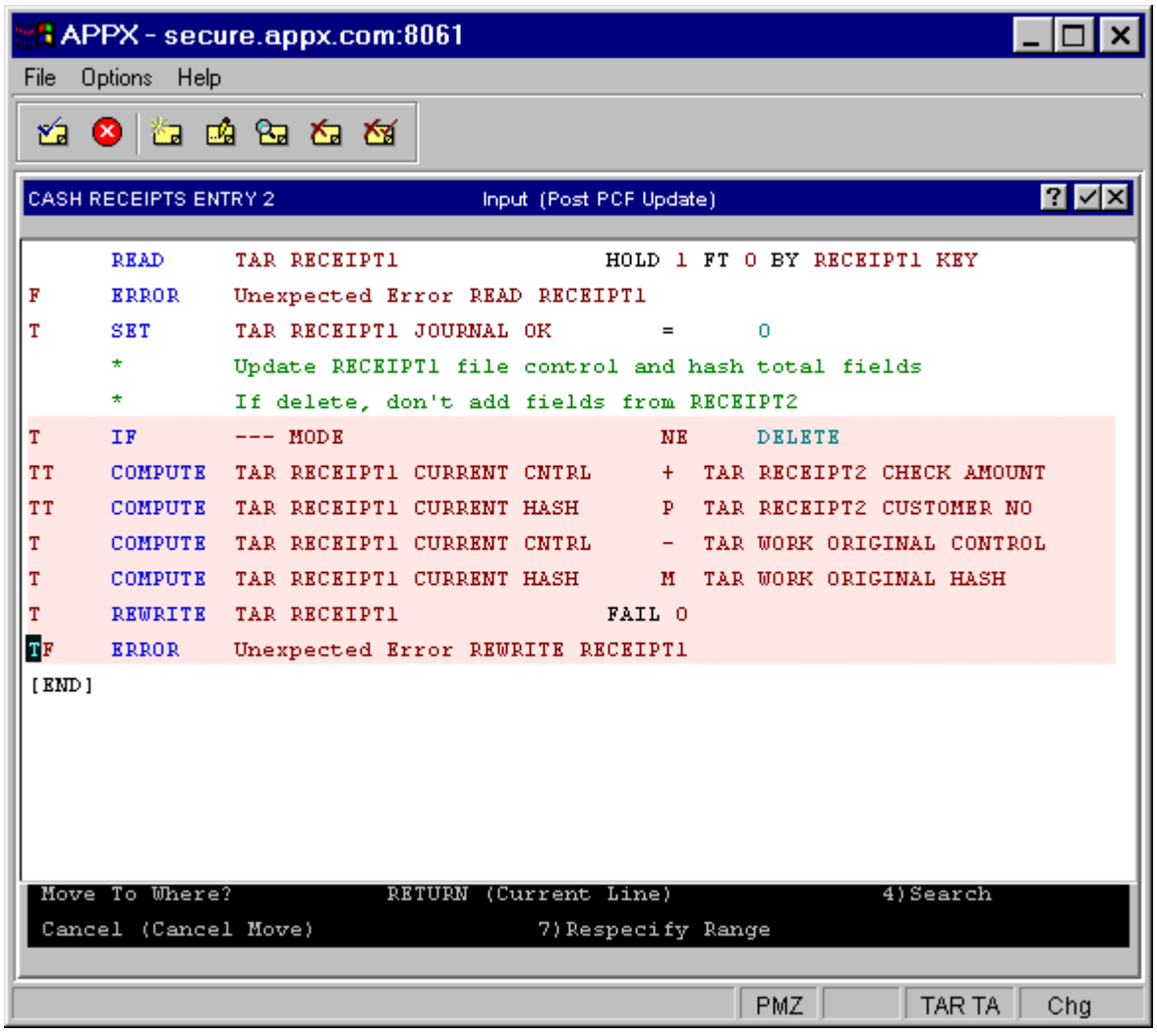


Figure 4-4-12

The question ***Move To Where?*** now appears on the left side of the option area. Using the appropriate options and arrow keys, you move the cursor onto the statement *before which* you want to move the range of statements you marked. Then, press ENTER to finish the move. Until you press ENTER, no move occurs. The following options are available on this screen:

- RETURN **(CURRENT LINE)** Press ENTER to finish the move operation.
- **CANCEL (CANCEL MOVE)** cancels the move operation. Selecting this option returns you to display mode. No statements are moved if you select this option.
- **SEARCH** allows you to search for a particular word or string of characters to locate a particular statement line. Refer to the description of the SEARCH option later in this chapter.
- **RESPECIFY RANGE** allows you to reset the second move marker.

Copy

This option operates in the same manner as the *Move* option described previously, except that the original statement or group of statements is left intact, and a duplicate copy of the statement(s) is inserted at a designated spot in a routine. There is both a ***Copy Through What?*** and a ***Copy To Where?*** screen where you specify the statement(s) you want to copy and the location to which you want to copy them. Refer to the instructions previously provided under *Move*, replacing the word *move* with *copy* wherever it occurs.

Search

Select this option when you want to search for a specific word or string of characters contained anywhere in an ILF routine. After selecting SEARCH, the screen in *Figure 4-4-13* appears.

The SEARCH screen presents a place on the bottom line to enter a word/string to be used in a search, and three options to choose from. The cursor is positioned on the first character of the search area at the start of a search operation.

You begin a search by entering the exact word or character string you want to find, up to 28 characters long. You enter the string after the *Search For* label at the bottom of the screen.

You may specify significant embedded spaces so that you can distinguish, for example, between a search for MAKE-UP from one for MAKE UP. You may enter a partial word such as ACC, which would find a match with both ACCOUNT and ACCESS, for example.

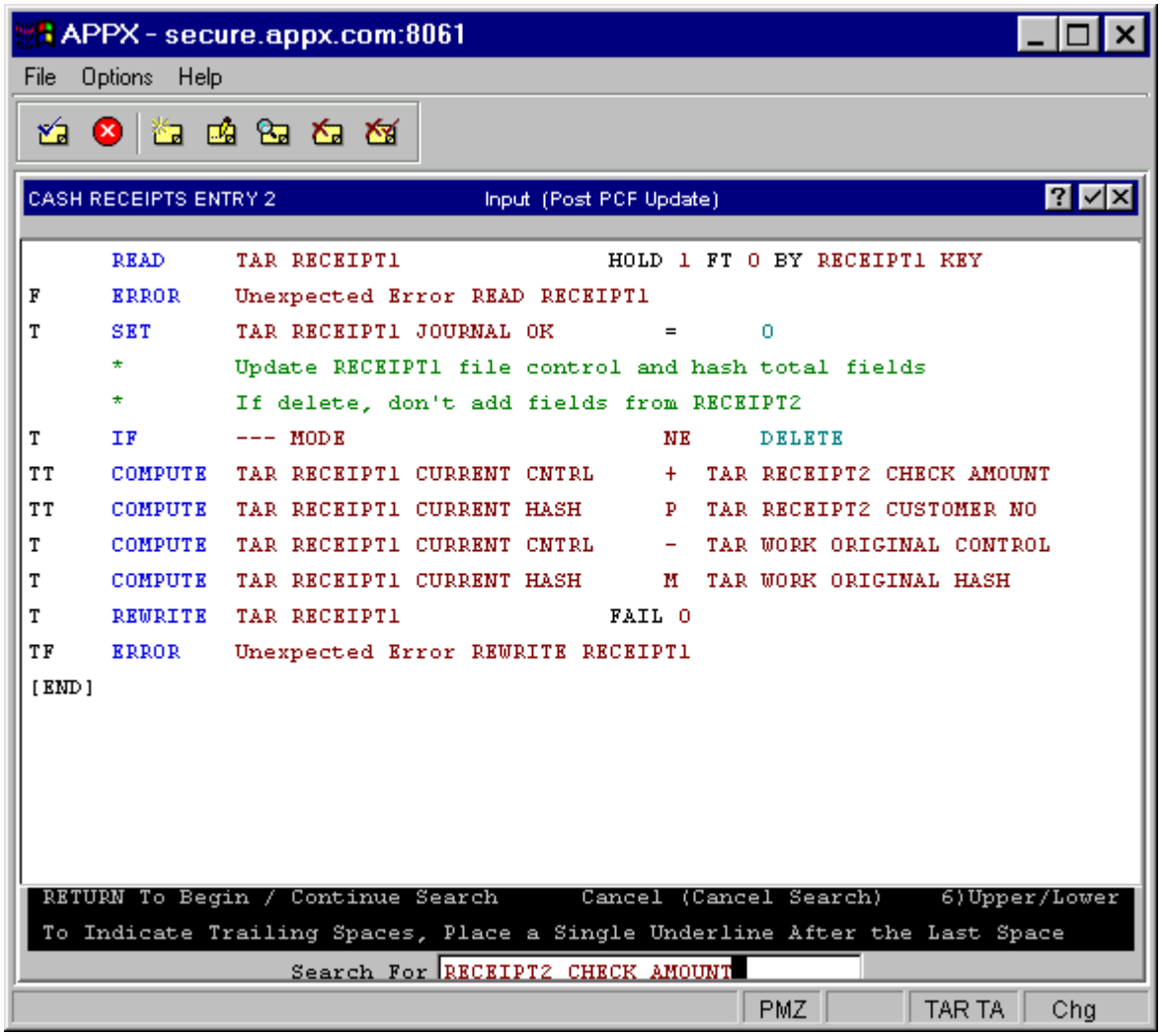


Figure 4-4-13. ILF Editor's Search Option

To indicate trailing spaces, enter a single underscore character after the final space you want to specify. For example, if you wanted to search for NAME with one space after the word, leave one space after NAME and enter an underscore after the space.

- Note also the use of the UPPER/LOWER option defined below to distinguish upper- and lowercase letters. The SEARCH operation is case-sensitive, so if you want to search for the string "Invalid Entry," you need to select UPPER/LOWER before entering the words "Invalid Entry." If you do not select the UPPER/LOWER option, your entry would be INVALID ENTRY, and you would not get a match on "Invalid Entry."

After entering the desired search text, you have the following three options to select from:

- **RETURN TO BEGIN / CONTINUE SEARCH** Press ENTER to initiate a search, and APPX highlights the first occurrence of the matching word/string. The cursor is positioned on the first character of the matched text, and the entire statement line is highlighted. Each time you press ENTER, you direct the system to locate the next occurrence of the matching word/string. If no match is found, either initially or in a subsequent search, a message to this effect displays.

- CANCEL (CANCEL SEARCH) terminates the search operation and returns you to display mode.
- UPPER/LOWER toggles between uppercase-only and upper-and-lowercase mode. See the previous description of this feature.
- The editor preserves the character string you enter until you leave the current event point. If you want to search for a different string, type over the previous text. If the previous string is longer than the new text you are entering, be sure to press the space bar enough times to clear out unwanted characters at the end of the string.

Replace

This option allows you to replace all occurrences of a specified word or character string with another string you designate. The REPLACE option appears very similar to the SEARCH option described previously. The primary difference is that you specify both the old value (which APPX searches for) and the new value that is to replace the old one. You control each replacement operation by pressing ENTER, and you can cancel at any time. The REPLACE option screen is shown in *Figure 4-4-14*.

Replace Option screen

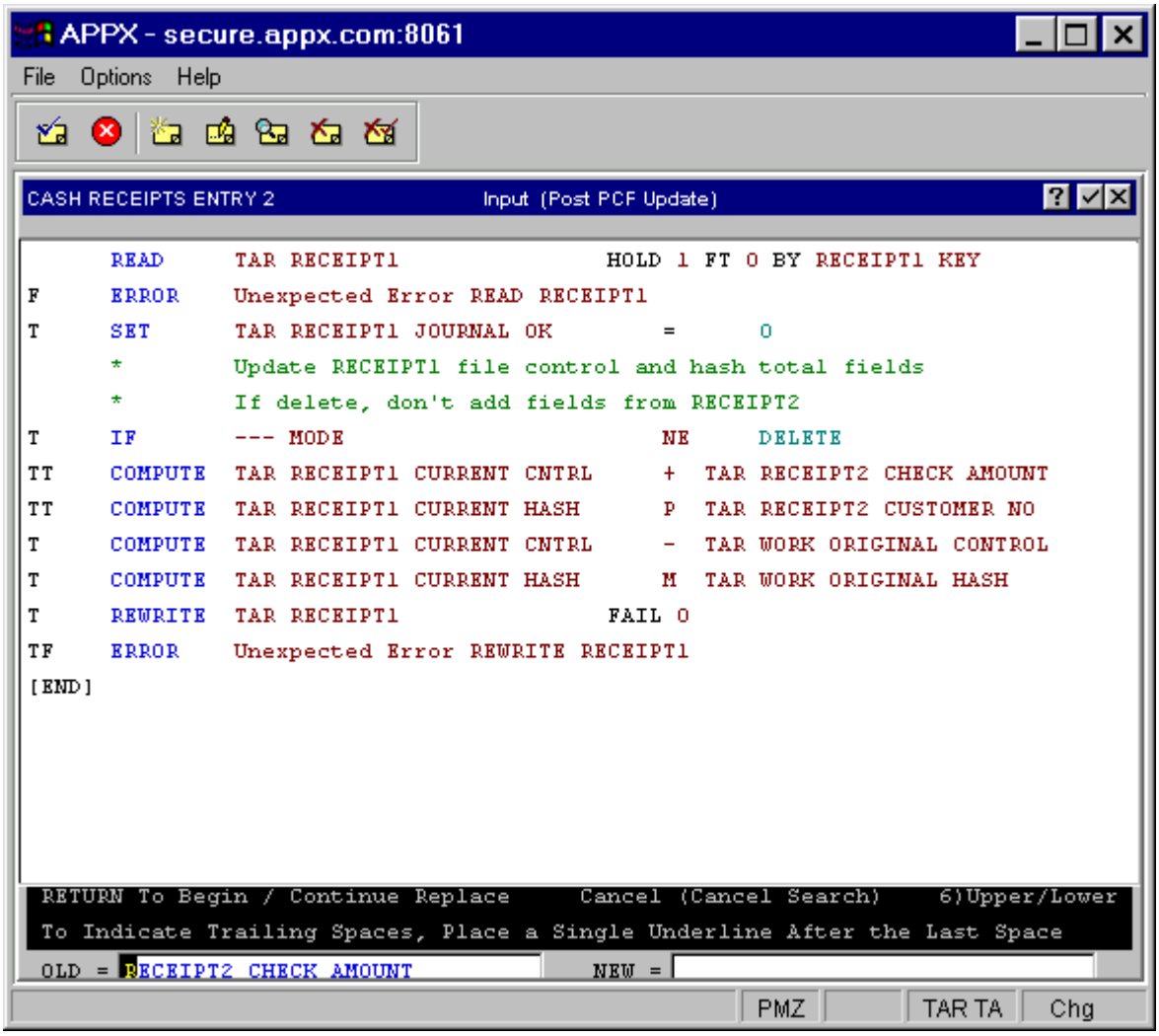


Figure 4-4-14

XCopy (external copy)

This option allows you to identify another existing ILF routine and copy some or all of the statements from that routine into the one you are currently working on. To use an existing routine as a "starting point" for a new routine, select the desired event point and, once on the editor screen, immediately select XCOPY and copy the desired statements. Then you can proceed to modify the copied routine as desired. For routines that already have existing statements, position the cursor on the line *before which* you want to move the external statements after you select the desired range of statements. The screen in *Figure 4-4-15* displays when you select the XCOPY option.

ILF Editor External Copy (Get Event Point) screen

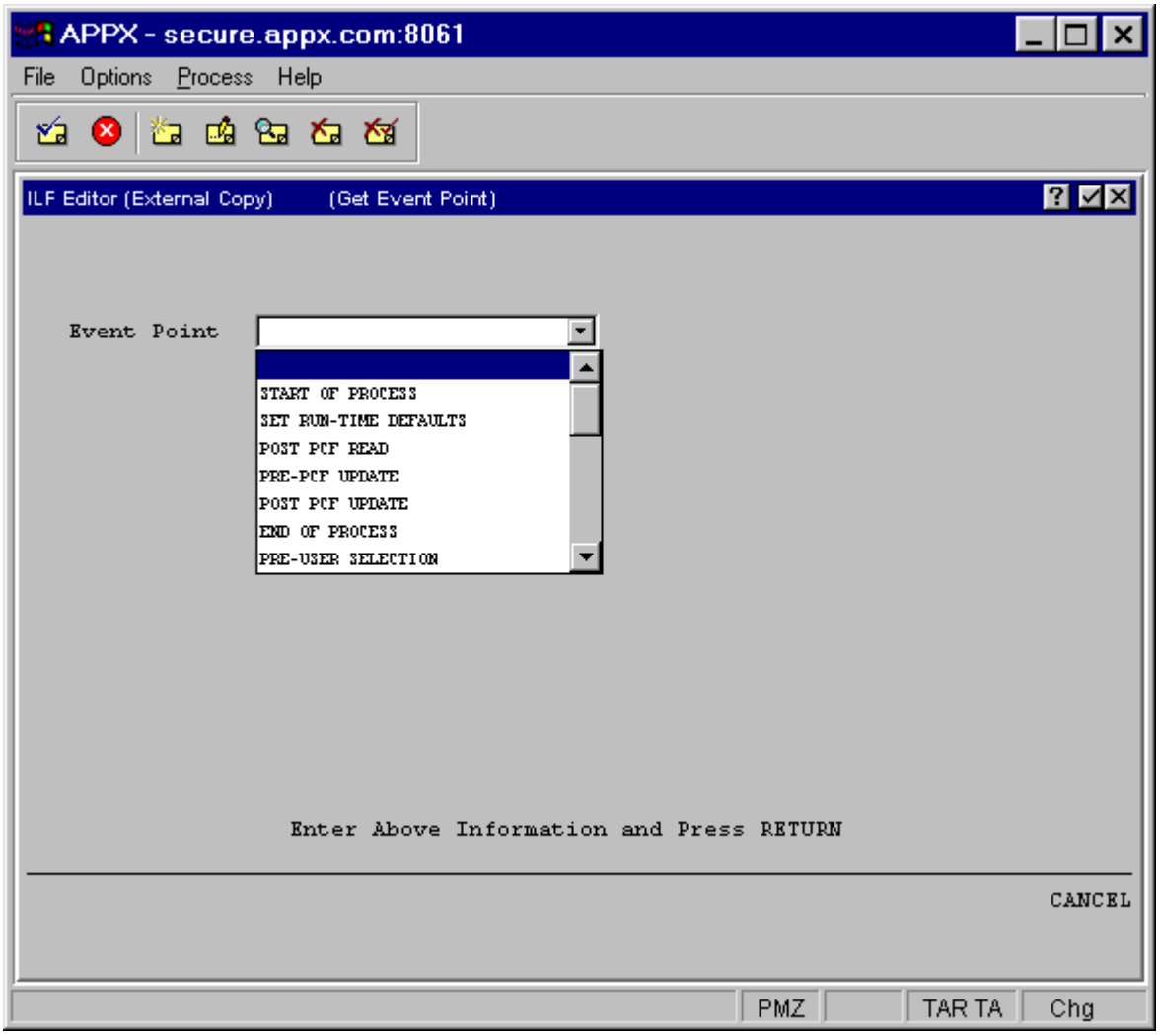


Figure 4-4-15

Enter the name of the event point from which you want to copy statements. The HELP option can tell you all of the possible event points. Depending on your entry, another screen will request additional information. On the screen shown in *Figure 4-4-16*, you are being asked to identify the process type and name for the *Pre-Display/Verify* event point.

ILF Editor External Copy (Process Level) screen

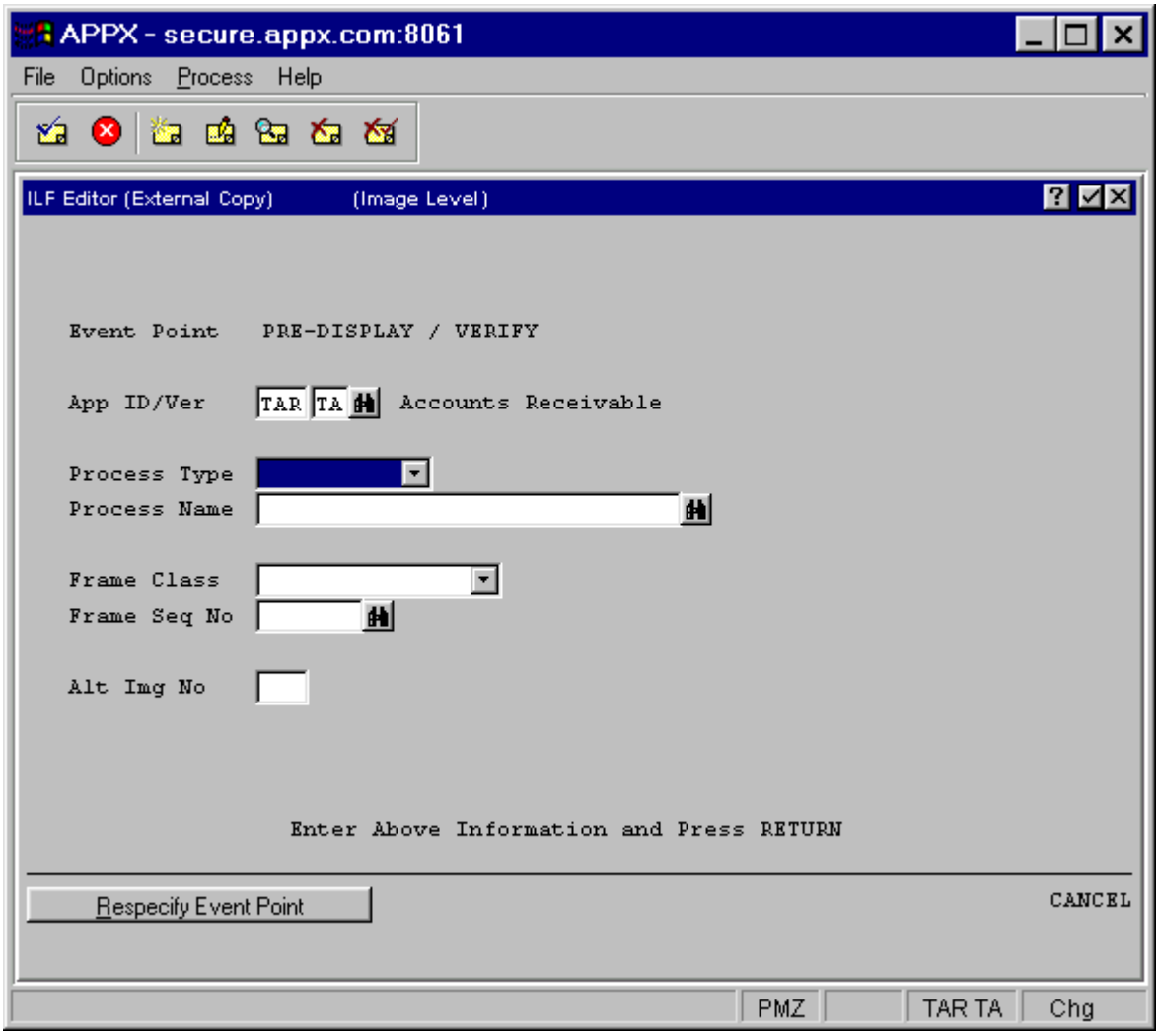


Figure 4-4-16

If you specify a frame, or image-level event point, you will also be asked for the frame and/or image number.

Delete All Statements (Option 10)

This option allows you to “erase the board” and delete all statements that you just entered for a new routine, allowing you to start all over again. You can also use this option to delete all statements from previously existing routines. To complete the deletion process for an existing routine, you must also select END after returning to the main editor screen to save the changes, as described previously. If you change your mind about deleting before selecting END, you can use the *Restore* option to restore statements to their original state as described below.

Restore All Deleted Statements (Option 11)

To restore statements deleted by using `OPTION 10`, simply select `OPTION 11`. This will "undo" the most recent *Delete All Statements* performed (provided that you did not `END` after executing `OPTION` 10).

Using Comments

There are two special keywords that you can use to enter comments in a routine: either a single or double asterisk (* or **). Refer to ILF keywords * (Comment) or ** (Comment) for details on how to use them.

In addition, during the course of debugging a routine, you may want to temporarily disable one or more valid statement keywords to see how a routine would execute without the statement(s) included. You can use the editor to accomplish this by entering a "C" or an "*" (for comment) in the first character position of each statement line. This area is normally reserved for a true/false execution condition but a "C" or an "*" overrides this normal use. When you execute the routine, the lines are bypassed as if they were comments. After appropriate testing, use the editor to reinstate the original true/false execution condition value, if desired, or delete the statements if you want to remove them permanently.

CHAPTER 4-5

The Debugger

Analyzing Logic Errors

As described in Chapter 4-1: ILF Integration, APPX automatically identifies errors encountered during statement entry, during the compilation process, and at runtime. However, ILF routines may still include the more subtle logic errors that can be identified and resolved only by thorough testing procedures. It is up to you to detect logic errors before you actually put an application into "live" production mode. The ILF provides you with a powerful tool to assist in this testing and debugging activity called the *ILF Debugger*.

The Features of the Debugger

The ILF debugger includes many features that allow you to test the logic of ILF routines. Within the debugger, you can:

Watch the effect

That a single statement or group of statements has on specific field values.

Execute a routine

One statement at a time.

- Dynamically ***set and clear break points*** to interrupt a routine at a point where you want to review and analyze it.
- Dynamically ***enter called subroutines*** and use the debugger in the called subroutines.

Modify field values

In accordance with known or desired expectations.

Invoking the Debugger

Invoking the ILF debugger involves two steps:

- Using the ILF editor, insert a TRAP statement in the routine you want to test, at the point where you want to enter the debugger. Save the modified ILF routine and exit the editor.
- Invoke the appropriate application menu using the *Test Database* option in Application Design. Select the option to run the process that includes the ILF routine you want to test. ***NOTE:*** *you* ***must*** *invoke the application from Application Design.* ***TRAP statements are not intercepted in the end user***

environment. Depending on the type of process it is, you may also be able to invoke it from the Toolbox during Process Maintenance.

When you execute the process that includes the ILF routine you want to test, APPX encounters the TRAP statement and recognizes that you want to invoke the debugger. At this point, the initial debugger screen displays as shown in *Figure 4-5-1* below.

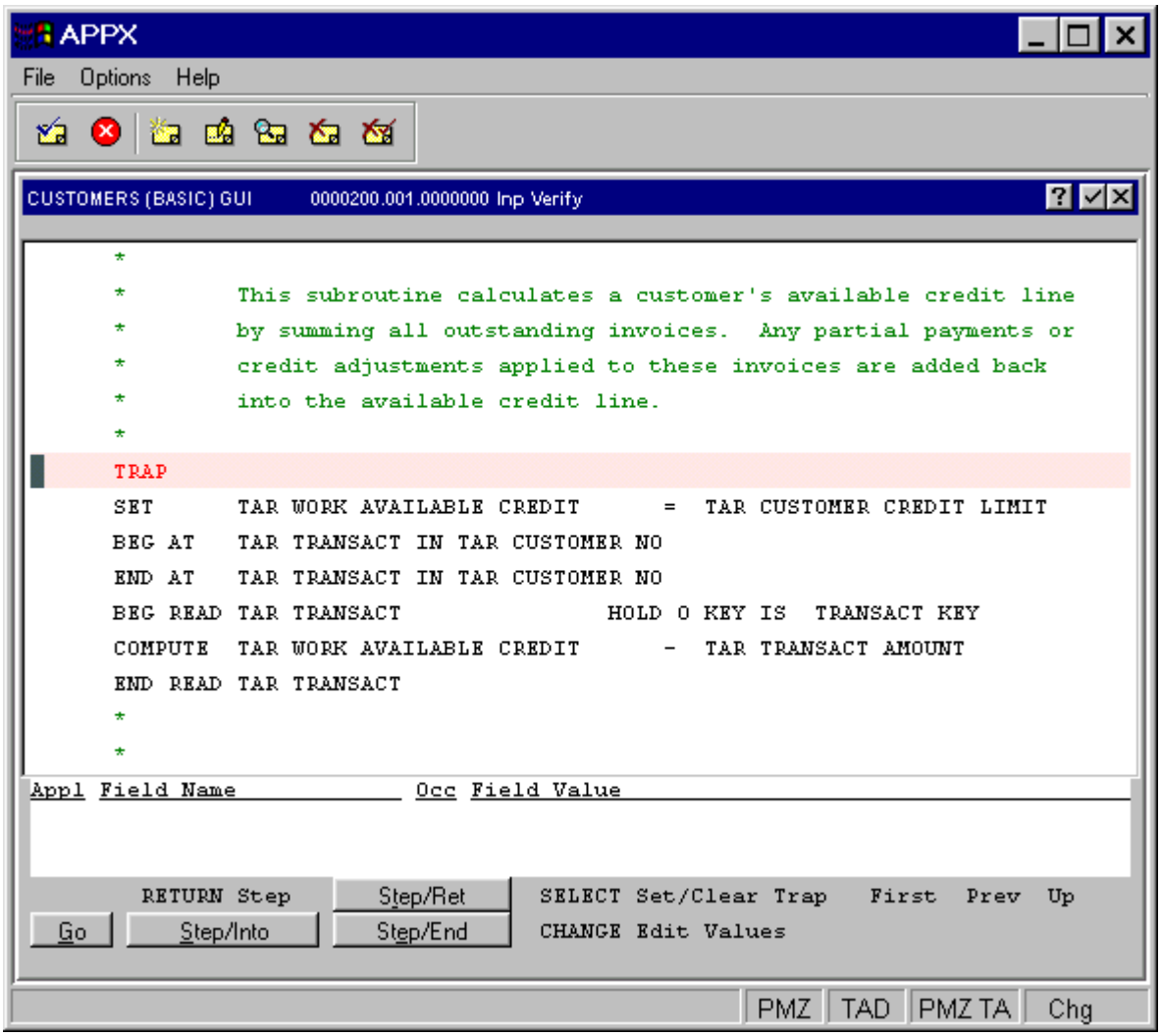


Figure 4-5-1. An Initial Debugger Screen at TRAP Statement

You are now in the debugger, and the cursor is positioned on the TRAP statement line. The TRAP keyword is brightened to indicate that the TRAP is currently active, and you are ready to perform any of the debugging functions.

On some computers, you are able to dynamically enter the debugger by pressing a terminal key, without having to add a TRAP statement as described above. Refer to the installation guide to see if this capability is available on your computer.

The Layout of the Debugger Screen

The debugger screen includes four separate areas:

- Key Information
- Display and Debugging Area
- Data Values Area
- Options

Key Information

The ***key information*** line (or title line) under the APPX toolbar in *Figure 4-5-2* below contains the following items, from left to right:

- Positions 1 through 5 display ***the current values of the internal true/false status indicators***. As you execute statements during the debugging procedure, the system dynamically updates these values as they are set and reset to help you evaluate whether statement logic is correct. Note that the line is left justified so if no true/false indicators are currently set, the next field (process name) appears first.
- The ***process name***, such as "Customers (Basic) GUI".
- The ***frame number, image number, and child sequence number***, such as "0000200.001.0000000".
- The ***process type*** abbreviated, such as “Inp” for input.
- The ***event point***, such as *Verify*.

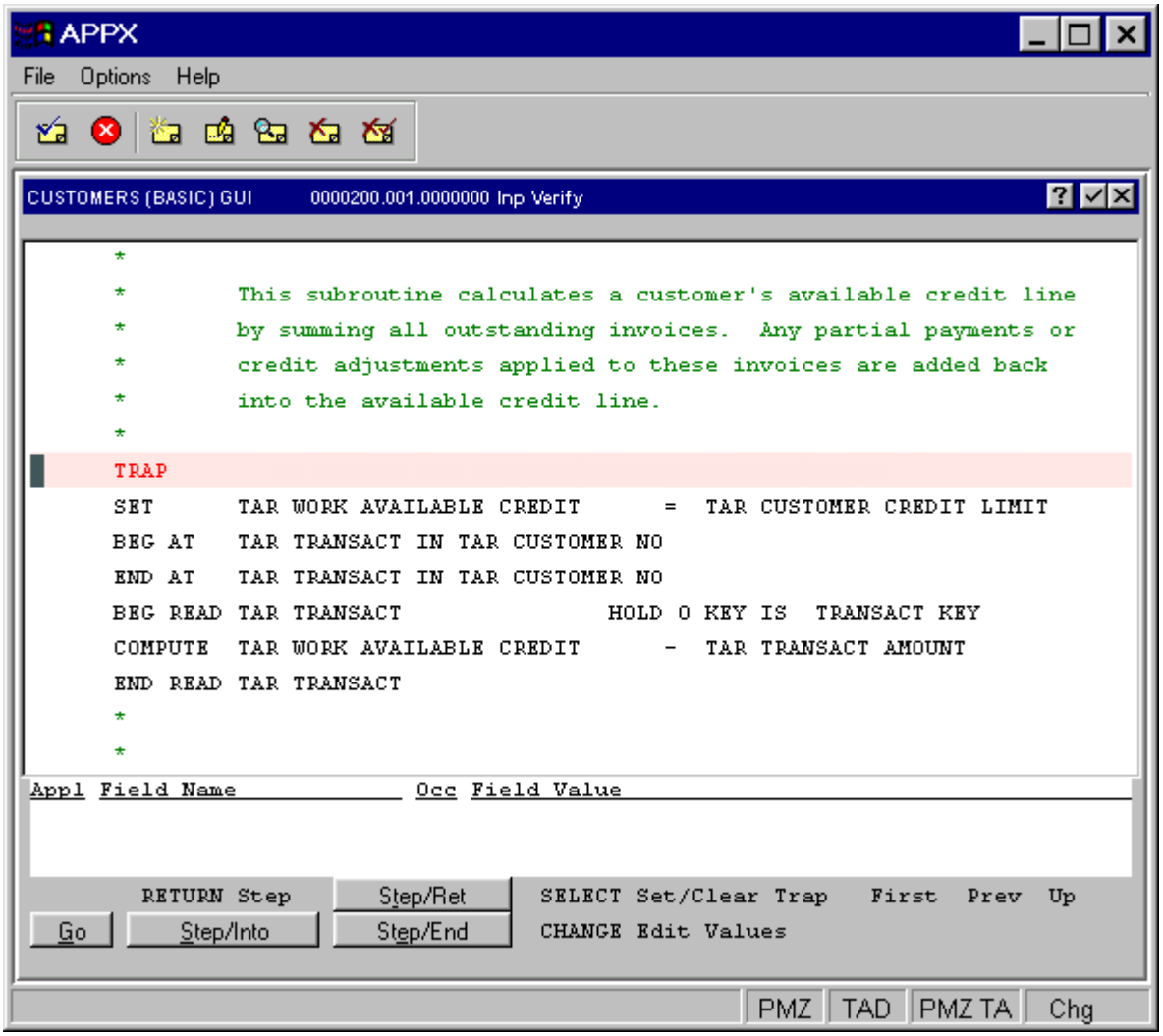


Figure 4-5-2. An Initial Debugger Screen With Key Information

Display and Debugging Area

The statement ***display and debugging area*** shows individual statements, and is where you specify which statements to include in the debugging operation. Note that the statement line where the cursor is positioned is highlighted; this is called the *current line*. This area is shown in *Figure 4-5-3* below:

Display and Debugging Area of a Debugger screen

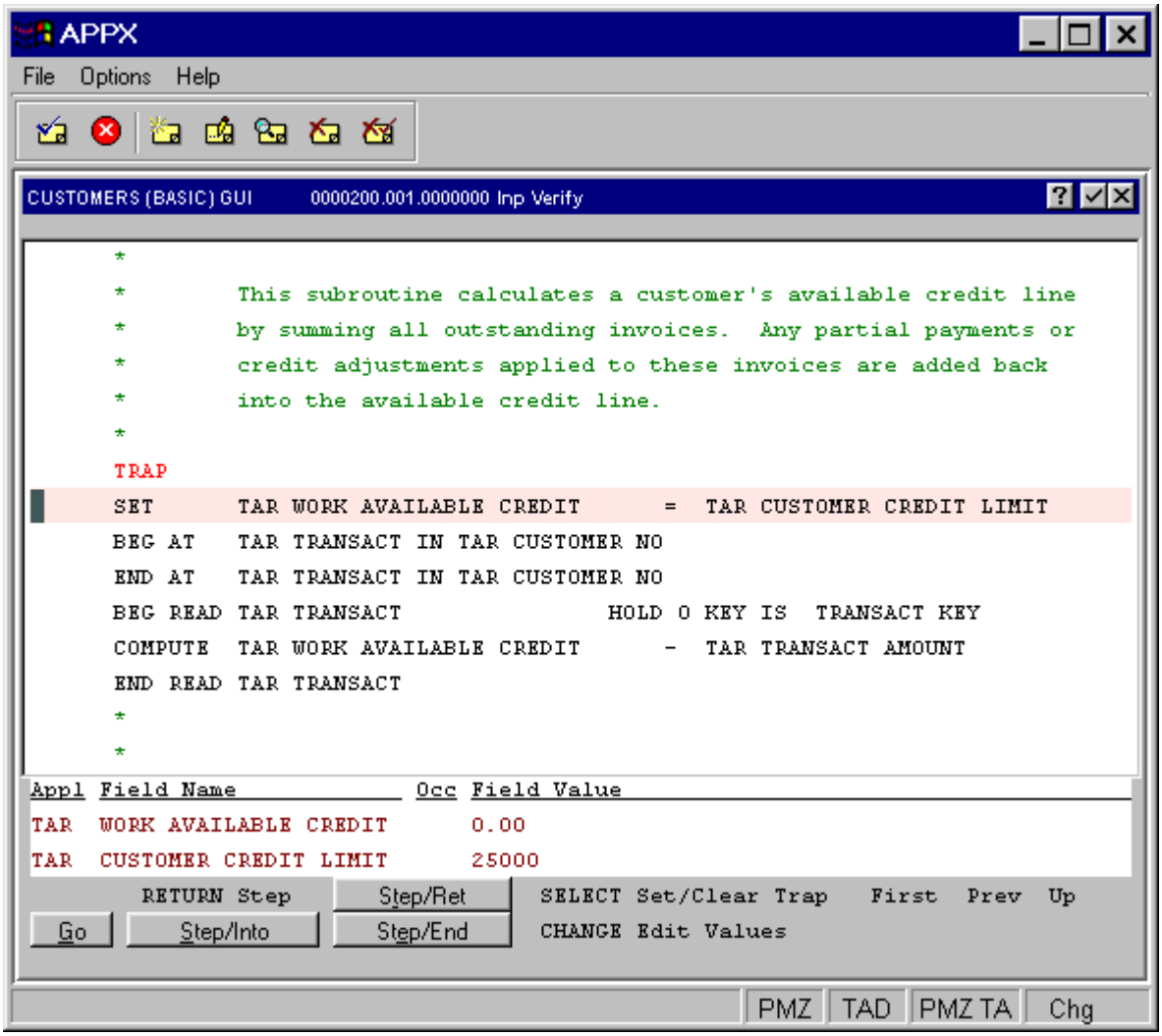


Figure 4-5-3

Data Values Area

The *data values area* displays the current values for any fields that are specified in the statement that is currently highlighted in the debugging area. The highlighted statement has not yet executed, so the values reflect the last statement executed. If the current statement has no specified fields, this area is blank.

Consider the following statement example:

```
COMPUTE  TAR WORK AVAILABLE CREDIT      -  TAR TRANSACT AMOUNT
```

The data area may look as follows, as shown in *Figure 4-5-4* below:

```
Appl Field Name            Occ  Field Value
TAR   WORK AVAILABLE CREDIT         25000.00
TAR   TRANSACT AMOUNT               957.45
```

Data Area of a Debugger screen

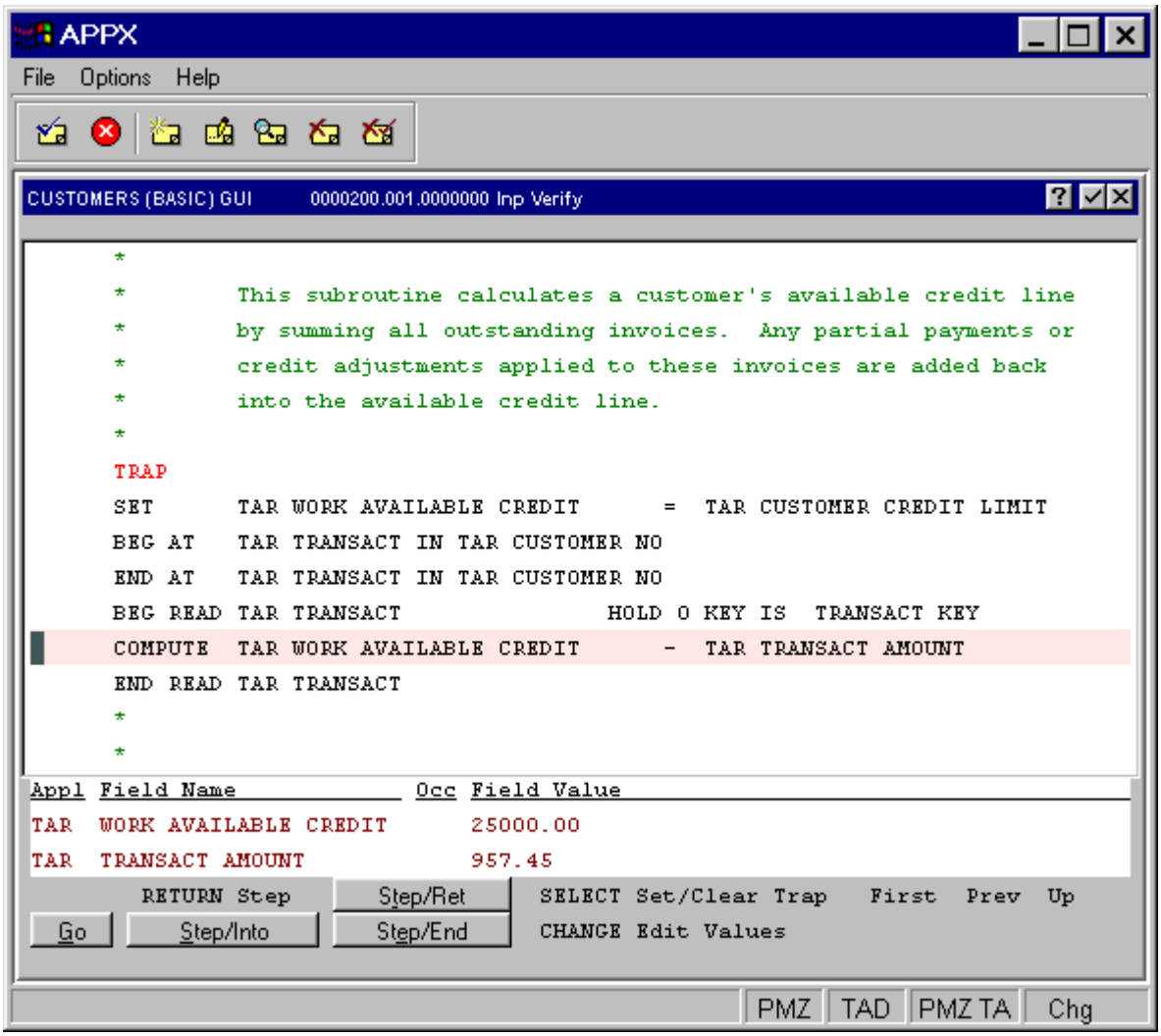


Figure 4-5-4

If the highlighted statement specifies only one field, the other field name and value will be blank. In the following example, only one field is referenced:

```
          DISPLAY  TAR WORK AVAILABLE CREDIT      (AT APPEARANCE #     )
```

The data area may look as follows:

```
Appl Field Name            Occ  Field Value
TAR   WORK AVAILABLE CREDIT         25000.00
(Line 2 is blank.)
```

For statements such as TRAP that have no field references at all, both field names and values are blank, as shown previously in *Figure 4-5-1*.

Options

Below the data area in a debugger screen are the ***options*** that allow you to perform the various debugging functions. These options are shown in *Figure 4-5-5*. The next section discusses available options and describes how to use them.

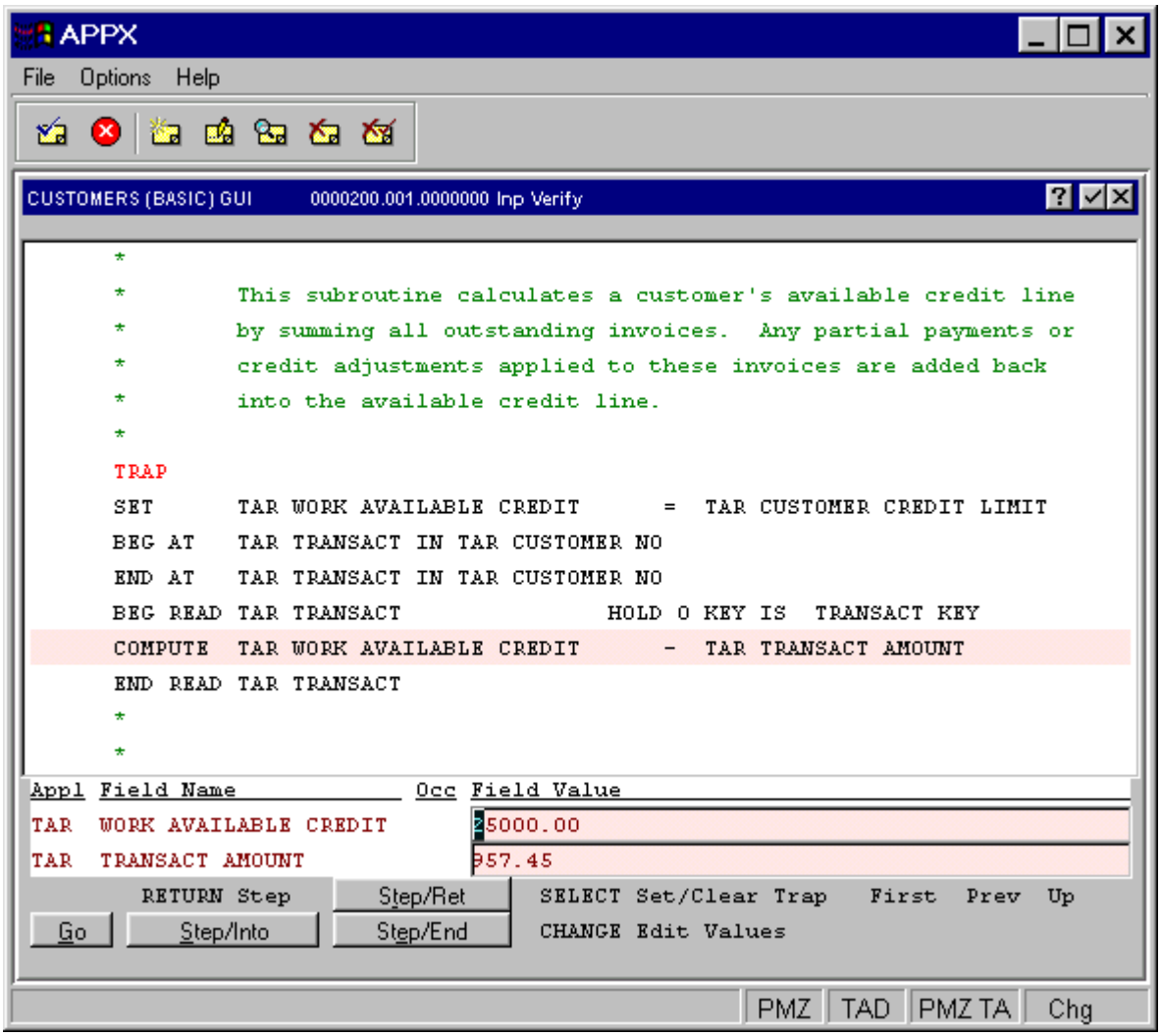


Figure 4-5-5. Options Area of a Debugger Screen (Change Values Selected)

The TRAP Statement

By inserting a TRAP statement in a routine, you are setting a break point that signals APPX to invoke the debugger when the process containing the routine executes. The placement of a TRAP statement is significant because, when the debugger is invoked, the first statement that displays on the debugger

screen is the first TRAP statement that executes during the normal execution of a process. You can optionally include additional TRAP statements at other points in a routine. If you insert more than a single TRAP, you are effectively setting a series of break points where processing is interrupted, one break point at a time, by repeatedly selecting the GO option. Each selection of the GO option moves you to the next break point.

Conditional Execution of the TRAP Statement

Like other statements, the TRAP keyword allows you to specify up to five true/false execution conditions. It is possible that, when a process executes, a TRAP statement is bypassed because of non-matching true/false status indicators. In this case, the TRAP does not interrupt processing, nor does it invoke the debugger. You are placed in the debugger only if a TRAP statement actually executes.

"Hard-Coded" versus Dynamic Trapping

If you insert only one TRAP statement in an event point, you have several alternatives upon reaching the debugger screen:

- By selecting GO, you can execute from the TRAP statement all the way through to the logical end of a process. To reach the logical end of a process, you may be taken through the invocation of one or more automatic children along the way. The only exception is that, if the routine involves a loop that causes the logic to re-execute the TRAP statement, processing is interrupted again at the TRAP until the loop ends.
- By pressing ENTER you can single-step through a routine, one statement at a time, beginning with the TRAP statement.
- You can set additional break points dynamically by positioning the cursor on any desired statement line and pressing SELECT (F3) to clear or set a TRAP. When you select GO, you execute from the TRAP statement up to (but not including) the next break point.
- By selecting STEP/END, you can step directly to the last statement in the current event point (and execute all the statements along the way). If, at this point, you press ENTER, you execute the last statement of the current event point and automatically step to the first statement of the next event point within the current process. In this way, you can debug more than one routine with only a single TRAP statement. If there are no more event points, you are taken to the normal end of the process.

You can use SELECT to "turn on" and "turn off" a trap on any line, including TRAP statements. If you turn off a TRAP statement, it is simply bypassed during execution. The debugger alerts you to the statements that are in *active trap* mode; these active lines are brightened. To reverse the existing active/inactive status, simply position the cursor anywhere on the desired statement and press SELECT (F3). If

a line was previously brightened, it reverts to normal color, and vice-versa. In this way, you control which lines have a TRAP set and, therefore, where break points occur when you execute a process by selecting the GO option.

The Options on the Debugger Screen

The options available on the debugger screen are listed below. Each option is described in detail in a following section of this manual.

- Go
- Return
- Step/Into
- Step/Ret
- Step/End
- [F3] Select (Clear or Set Trap)
- Add Mode
- Watch Points
- Inquire Mode
- Change Mode
- Other Options

Go

Select this option to execute a process from the point where the first break point is set, up to (but not including) the next break point or, if there is no other break point, through the logical completion of the process. The following conditions are applicable to the GO option:

- *Break point* means a statement line that is brightened: either a TRAP statement inserted into a routine that was not deactivated by pressing SELECT (F3), or any other statement that was activated dynamically by pressing SELECT (F3).
- Whether the break point is set by a TRAP statement or set dynamically, the system executes without interruption up to, but not including, the next break point.
- If no more break points are encountered, the process executes as if there were no specified break point. That is, a process executes through the normal termination point you specified in the application.

Return

This option allows you to "step through" the routine one statement at a time by executing the current line (which is highlighted). After you select RETURN (by pressing the ENTER key, or selecting either the ENTER icon on the APPX toolbar or the ENTER button on the right side of the title bar), three actions occur:

- The logic of the current statement executes.
- The data values change to reflect the result of executing the current statement.
- The cursor moves to the next logical line in the routine, and it becomes the new current line (and is highlighted).

One important thing to note about RETURN is that the next statement is the next executable statement, not necessarily the next sequential statement. For example, in the following routine:

```
        IF       TAR WORK AVAILABLE CREDIT      LT     0
   T    WARNING  Customer Has Exceeded Credit Limit!
        DISPLAY  TAR WORK AVAILABLE CREDIT
```

If the IF statement in this example encounters a false condition and sets the first true/false status indicator to *F*, the WARNING statement is not executed. By using RETURN to single-step, the debugger skips from the IF statement directly to the DISPLAY statement.

Additionally, using RETURN at a COPY or GOSUB statement takes you to the first executable statement immediately following the COPY or GOSUB statement, allowing you to skip over the statements in the subroutine.

Step/Into

To view the contents of the COPY or GOSUB routine, you must select STEP/INTO while positioned at the COPY or GOSUB statement. This steps you "out" of the current statements and "into" the referenced subroutine. Selecting STEP/INTO instead of RETURN keeps you "in line," and allows you to execute the subroutine statements. Select RETURN to move through the subroutine one statement at a time, or select STEP/END or STEP/RET to move to the next RETURN statement. Then, use RETURN to leave the subroutine.

Step/Ret

Selecting this option takes you to the next RETURN statement, providing you with a quick exit from a subroutine. Note that the remaining statements in the subroutine are executed although you do not "see" this processing occur.

Step/End

This option allows you to step directly to the end of the current event point. All statements from the current statement line up to (but not including) the last statement in an event point execute. At this point, using RETURN executes the final statement and moves you directly to the first statement in the next logical event point of the current process. This sequence (STEP/END followed by RETURN) effectively continues a trap operation, even if there are no more break points in subsequent event points of a process. It allows you to continue your testing procedure without having to leave the debugger to set additional break points.

You can proceed through all the event points of a process in the same manner. While in the second and subsequent routines, you can perform all normal debugging operations, such as setting additional break points and modifying field values. To leave the debugging facility, simply select GO to move to the normal end of a process.

[F3] Select (Clear or Set Trap)

Select this option to reverse the active/inactive "trap" status of the current statement line. You need to move the cursor to the desired line and then press SELECT (F3) to initiate a TRAP (if it is not already brightened), or deactivate a TRAP (one that is currently brightened). SELECT lets you "toggle" back and forth between active and inactive "trap" status. Remember, you can use SELECT on a TRAP statement as well as on any other statement type.

Add Mode (Debugger Screen)

Select this icon to change the point of execution. While you are stepping through ILF code, the debugger highlights the next statement that will be executed. If you position the cursor to another executable line of ILF code and select the ADD MODE option, then that line will become highlighted and will be the next line executed. You can use this feature to skip sections of code, change a value and repeat a test, or step into a normally unreachable section of code. Care must be taken not to introduce logic problems such as executing an END READ without having executed the corresponding BEG READ first.

Restrictions to the ADD MODE option are as follows:

- You cannot change your point of execution if you are currently stopped at a TRAP statement.
- You cannot change your point of execution to continue with a LABEL or comment (*) statement.

Watch Points

A WATCH POINT can be defined for a variable (field, work field, or PDF) that is either in the PCF or mentioned in ILF statements in your current process. When you define a WATCH POINT, APPX remembers the current value of the field and will watch that field for the remainder of the current process, including child processes. If the value of that field is changed, APPX will display a *Watch Point Trigger* pop-up window as shown in *Figure 4-5-6* and will step into the ILF debugger as soon as possible, allowing you to view the code which modified the field.

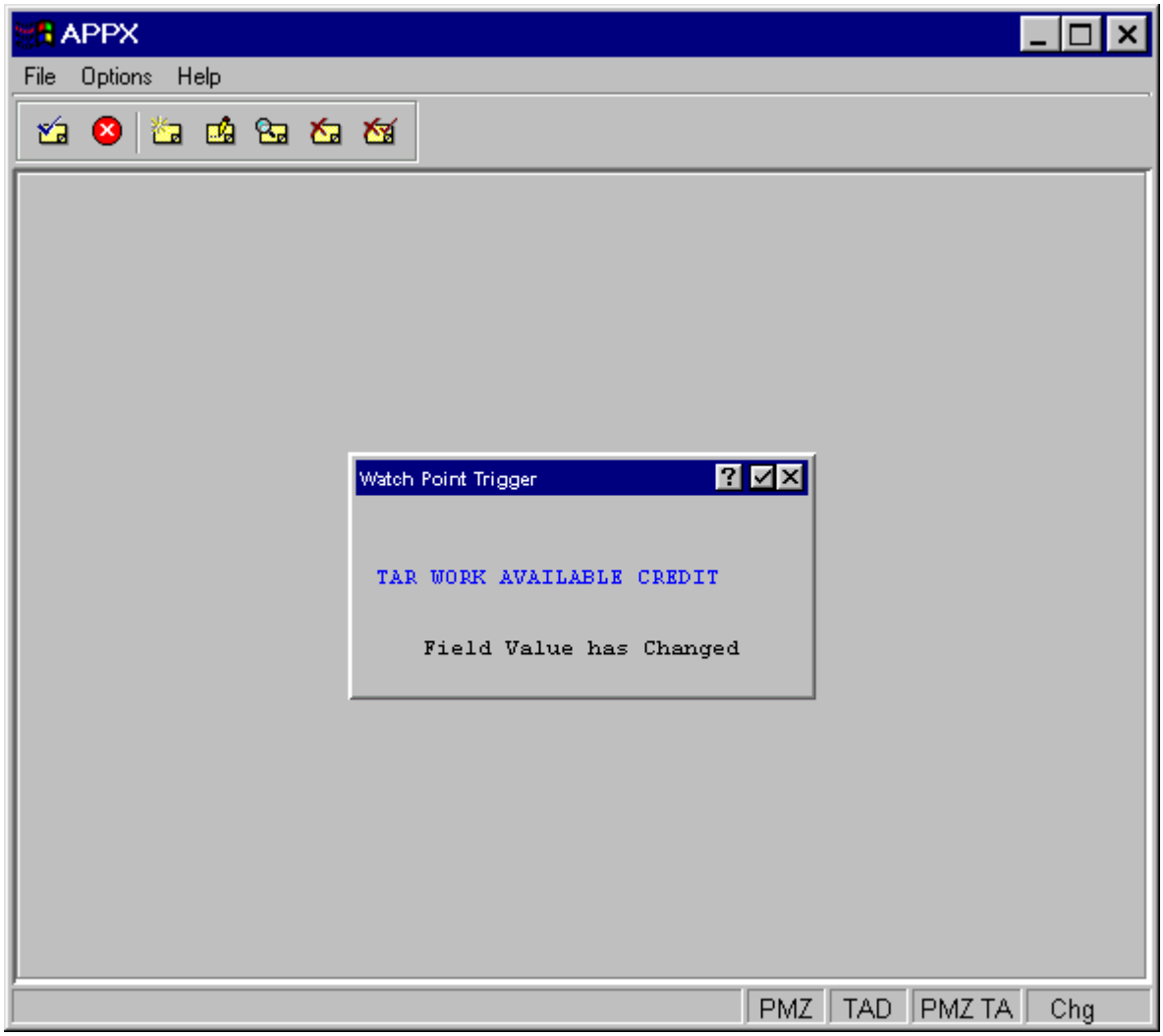


Figure 4-5-6. Watch Point Trigger Pop-Up

You define a WATCH POINT by selecting the DELETE MODE icon from the APPX toolbar. As shown in *Figure 4-5-7,* you will be prompted for an Application ID, Field Name, and Occurrence Number for each field or fields you want to track. You will receive a warning if APPX cannot set a TRAP on the field that you specify. This happens if the selected field is not in the PCF and also not accessed by ILF statements

in the current process. You can add a reference to the field if you need to. You can remove previously defined watch points by deleting them. Select the DELETE MODE and then CONFIRM DELETE icons to accomplish this. Select END to return to the debugging process.

When a watch point field's value changes, APPX will replace its saved field value with the current field value in order to prevent excessive watch points.

The cause of a watch point TRAP may not be immediately obvious. It may not be initiated by an ILF statement. For example, if you are watching a field defined in your PCF, you may notice that APPX TRAPs the first statement in the *Post PCF Read* event point even though that statement does not refer to the field being watched. This is because APPX has read in an entire record and the value of the field in the new record is not the same as the value of the field in the previous record.

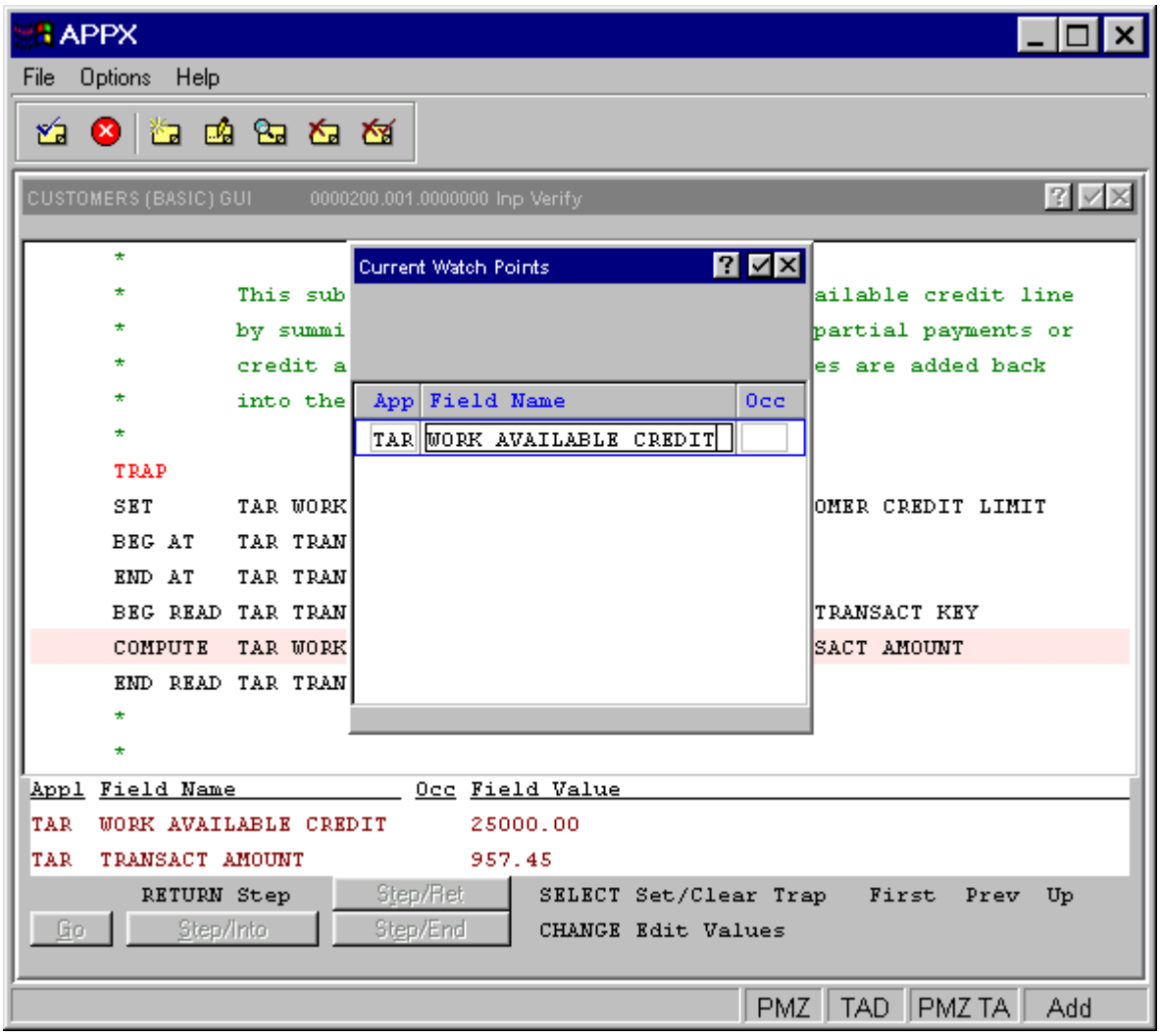


Figure 4-5-7. Current Watch Points Overlay

Inquire Mode (Debugger Screen)

Select this icon to examine and modify the contents of any field known to the current process. A field is known to the current process if any of the following are true:

- the field is part of the PCF
- the field is mentioned in ILF code
- the file containing the field is mentioned in ILF code
- the field is a PDF
- the field is in a file related through data lookup

Restrictions to the INQUIRE MODE option include:

- the *Scan* option cannot be used to select fields
- only the first occurrence of a field may be viewed
- any errors are reported by displayed question marks

Selecting this icon will open the data area of the screen, allowing you to enter the Application ID, Field Name, and Occurrence Number (if applicable) of the field you want to examine. Press ENTER to view the current contents of the field. Press ENTER again to return to normal debugging mode.

Change Mode (Debugger Screen)

This icon allows you to enter or change the values currently displayed in the data values area of the screen. You first position the cursor to any desired statement line in an event point and then select CHANGE MODE. The cursor moves to the data value area, makes the field values modifiable, and allows you to enter or change the values currently on the screen. After entering the desired values, press ENTER to return to normal debugging mode. From this point forward, the ILF routine operates based on the new values you entered.

Other Options

The next six options (*First, Last, Prev, Next, Up*, and *Down*) are the same options described previously for the ILF editor. These options allow you to move through statements in an event point so that you may examine the statement and set or clear break points as desired.

Suggested Uses for the Debugger

At this point, you have learned about the mechanics of the ILF debugger: how to invoke it, what information is provided on the screen, how to clear and set TRAPs dynamically, and how to execute portions of a routine. How and under what circumstances you make use of the debugging facility is up to you, but the following are some suggested ways to make the most effective use of this tool.

- Perhaps the most fundamental use of the debugger occurs when execution of a process produces results known or suspected to be erroneous. The debugger allows you to evaluate statements individually or in groups, to examine exactly how the values were generated. Most of the other uses of the debugger supplement this basic principle.
- You can set the value of a field(s) in the current statement to watch how this value affects subsequent processing. You usually set a field to a known or desired value and evaluate its effect on the remainder of the process. You may need to subsequently enter the ILF editor and make one or more modifications to statements in other ILF routines, based on the results of your testing.
- You can directly enter another ILF routine to examine the logic in a previous or subsequent event point that may affect the routine you are testing with the debugger.
- If you suspect a problem is within a particular event point, you can set a TRAP in that routine and enter the debugger. By using `RETURN`, you can effectively test the entire routine (and only this routine), and evaluate the results to see if they meet your expectation.
- As you test and evaluate a particular ILF routine, you may not uncover a logic problem. You may want to proceed with the testing, even if you failed to set break points in subsequent routines. By using STEP/END with `RETURN`, you can continue testing from one event point to the next without leaving the debugger.

Leaving the Debugger

There are two ways to leave the debugger:

Select

GO until no more break points are encountered. You will end up at the normal termination point of the process you executed.

Select

STEP/END until the final statement of a routine executes, allowing the process to continue executing in the normal manner.

Clearing Out Break Points

To clear out hard-coded break points (TRAP statements), you can invoke the ILF editor and delete them. Alternatively, there are processes on the Utilities, Toolbox menu in Application Design that "Turn All TRAPs Off" and "Turn All TRAPs On" in a given application. The first process converts TRAP statements into "tftft TRAP" comments (where "tftft" is replaced by the actual T/F indicators if any are present on the TRAP statement) and the second process converts the comments back to TRAP statements. Remember that TRAP statements are not intercepted in the end user environment, so ordinary users will never be aware of their existence.

If TRAP statements are not deleted or turned off, they remain a permanent part of a routine. The break points that you set dynamically within the debugger by using SELECT (F3) are automatically cleared when you leave the debugger. You must reactivate any dynamic break points when you return to the debugger.

CHAPTER 4-6

ILF Keyword Reference

* (Comment)

The * (Comment) statement allows you to enter permanent comments, notation, or documentation in an ILF routine. Comment lines have no effect on the processing logic and are intended to provide the designer with a means of explaining what the statement coding does or how it operates.

```
          *        ..............................................................
                   (1)
```

```
(1) Comment (text) (64 characters maximum)
```

Using the Statement

You can use a * (comment) statement at any point in an ILF routine, including the beginning or end.

Restrictions

The "*" (asterisk) must be in the first position of the keyword area.

Statement Ignored

Comments are ignored during execution.

Related Statements

** (Comment)

Example

```
          *        The following set of statements computes the value of
          *        interest due, where:
          *        A = Amount of loan
          *        R = Annual rate of interest
          *        M = Number of months to repay the loan
```

** (Comment)

The ** (Comment) statement is identical to the * (Comment) statement except that, when the ILF routine is listed (using the PRINT TECHNICAL DOCUMENTATION option in Application Design), a top-of-page is generated before the comment is listed.

```
          **       ..............................................................
                   (1)
```

```
(1) Comment (text) (64 characters maximum)
```

Using the Statement

You can include ** (comment) statements anywhere in an ILF routine, including the beginning or end.

Restrictions

The “**” (asterisks) must be in the first two positions of the keyword area.

Statement Ignored

Comments are ignored during execution.

Related Statements

* (Comment)

Example

Notice that the second comment line has only one “*”; if it had two, each of the comments would print at the top of a new page.

```
          **       This statement series is a separate computation of
          *        extended price for assembled components.
```

AND

The **AND** statement adds a second or subsequent conditional check to a preceding statement, such as the series “IF A is true **AND** B is true, then ...”. The **AND** statement itself compares the contents of two fields, or the contents of a field with a constant, to determine if a condition is true.

```
    ..... AND      ... ...................... ... .. ... ...................... ...
    (1)            (2) (3)                    (4) (5)(6) (7)                    (8)
```

(1) T/F execution conditions	(5) Relation (LT,GT,LE,GE,EQ,NE,IN,EX)
(2) 1st application ID	(6) 2nd application ID
(3) 1st field name or PDF	(7) 2nd field name, PDF, or constant
(4) 1st occurrence (constant/index)	(8) 2nd occurrence (constant/index)

Sets True/False Status Indicator

The **AND** statement may modify the current value of the next internal true/false status indicator. If the preceding statement is true and the **AND** statement is false, the current status of the next indicator is changed to *F.* See Using AND and OR Statements for more information.

Using the Statement

The **AND** statement specifies a relationship between two values. The valid relational operators are:

- less than (LT), greater than (GT), less than or equal to (LE), greater than or equal to (GE)
- equals (EQ), not equal to (NE), includes (IN), and excludes (EX).

The relation *IN* means "includes". The relation is true if the value in the second field is included in the first field, and false otherwise. If the relation is true, the predefined field TEXT AT POSITION is set to the starting position of the second field within the first field. The first field must be an alpha field; the second can be either an alpha field or a constant. Trailing blanks are ignored if the second field is a constant; however, you may specify fixed spaces (by entering underscores) to include one or more trailing spaces as part of a character string that is being searched.

Normally, an **AND** statement is used immediately after an IF statement. However, the **AND** statement will operate successfully when used after any statement that sets the true/false status indicator. See AND and OR with Statements Other Than IF.

An **AND** statement may also be used in conjunction with one or more OR statements. For example:

```
IF      A
AND     B
OR      C
AND     D
```

Another example is:

```
IF      A
OR      B
AND     C
OR      D
```

Each of these examples could be expanded to add more **AND** and OR statements in various sequences but, no matter how many there are, the rules are always the same:

- Each **AND** or OR statement performs only two checks at most: it checks the current value of the next true/false status indicator, and it evaluates its own condition. It does not matter how many statements preceded the **AND**/OR statement, nor what other statements follow the **AND**/OR statement.
- If the current status of the next indicator is already false, the **AND** statement is bypassed since it is no longer relevant. If the current status is true and the **AND** statement is true, the current status is

left unchanged as *T*. However, if the **AND** statement is false, the current status of the next indicator is changed to *F*.

- Conversely, for an OR statement, if the current status of the next indicator is already true, the OR statement is bypassed since it is no longer relevant. If the current status is false and the OR statement is false, the current status is left unchanged as *F*. However, if the OR statement is true, the current status of the next indicator is changed to *T*.

Restrictions

The **AND** statement supports comparisons between like data types only (alpha-to-alpha, numeric-to-numeric, date-to-date, logic-to-logic). When comparing two date fields, results can be unpredictable if you are using a date field without a century component. Confirm that your System Administrator has set up the *Pivot Year* environment variable to avoid unexpected results.

Related PDFs

TEXT AT POSITION

Related Statements

IF, OR

Example

This example shows how to perform statement printing if the customer is not a miscellaneous customer and does not have a zero balance. If either condition is false, the level 1 true/false status indicator will be *F* and the subroutine to print statements will not be executed.

```
         IF       TAR CUSTOMER MISC CUSTOMER     NE     1
         AND      TAR CUSTOMER BALANCE           NE     0
    T    GOSUB    :PRINT STATEMENTS
```

APPEND

The purpose of the **APPEND** statement is to join together the contents of two fields or character strings. Specifically, **APPEND** attaches the contents of a second, or source, field to the end of the current contents of the first, or destination, field, optionally inserting one or more blanks between the fields.

```
••••• APPEND   ••• ••••••••••••••••••••• ••• •  ••• ••••••••••••••••••••• •••
(1)            (2) (3)                    (4)(5) (6) (7)                    (8)
```

(1) T/F execution conditions	(5) Number of spaces between fields
(2) Destination application ID	(6) Source application ID
(3) Destination field name or PDF	(7) Source field name, PDF, or constant

(4) Destination occ (constant/index)	(8) Source occurrence (constant/index)

Using the Statement

The destination field must be an alpha field. The source field can be either alpha or constant.

The length of the destination field is determined by finding the last nonblank character in the field. Then, the number of spaces specified in the statement is added to the end of the destination field.

Spaces are added until the specified number of spaces has been added or the end of the destination field has been reached, whichever comes first. Then, the characters of the source field are copied and appended after the inserted spaces. Characters are copied from the source field to the destination field until the last nonblank character of the source field has been copied (or the end of the destination field has been reached).

Restrictions

The *justification-type* and the *pad character* are ignored. The characters copied are stored left justified in the destination field after the specified amount of blanks is added. No padding is performed, even if the source field is shorter than the space remaining in the destination field. No other templating takes place.

Related Statements

SET, SET TEMP

Examples

The following example shows how to set a field to a constant that is longer than 22 characters:

```
          SET      TGL WORK COLUMN HEADING        =      Total Customer-Related
          APPEND   TGL WORK COLUMN HEADING        1      Sales
```

This pair of statements generates a column heading, “Total Customer-Related Sales.”

The next example shows how you can use **APPEND** to generate a message from a number of “pieces”, depending on conditional checks:

```
          SET      TGL WORK MESSAGE FIELD         =      PLEASE
          IF       --- ERRORS                     GE     1
    T     APPEND   TGL WORK MESSAGE FIELD         1      RE-ENTER CUSTOMER ID
    F     APPEND   TGL WORK MESSAGE FIELD         1      CONTINUE DATA ENTRY
```

The following is another example that properly formats line three of a customer’s address:

```
          SET      TAR WORK ADDRESS           003 =  TAR CUSTOMER CITY
          APPEND   TAR WORK ADDRESS           003 0      ,
```

```
          APPEND   TAR WORK ADDRESS          003 1  TAR CUSTOMER STATE
          APPEND   TAR WORK ADDRESS          003 2  TAR CUSTOMER ZIP CODE
```

AT FIELD

AT FIELD is used to see if the cursor is positioned at a specified field within the current image of a menu or input process, usually as one part of a conditional check.

```
    ••••• AT FIELD ••• •••••••••••••••••••• ••• (AT APPEARANCE # •••)
    (1)            (2) (3)                  (4)                  (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or predefined field
(4) Occurrence (constant/index)

(5) Appearance (constant/index)
```

Sets True/False Status Indicator

If the cursor is positioned on the field specified (including the occurrence and appearance numbers if so specified), the next level of the true/false status indicator is set to *T*; otherwise, it is set to *F*. If you leave the occurrence number blank, cursor positioning on any occurrence of the specified field results in a true condition. If you leave the appearance number blank, cursor positioning on any appearance of the specified field results in a true condition.

If you leave field name blank, the **AT FIELD** statement returns a true value if the cursor is positioned in an item corresponding to any field within the specified application. If you leave all the parameters blank, the **AT FIELD** statement returns a true value if the cursor is positioned in any item within the current image.

Restrictions

The **AT FIELD** statement is valid only within the *Option Intercept* or Global Option Intercept event point. If you attempt to add this statement in any other event point, you will receive a warning that states "Warning - This statement has no effect in this event point". However, if the **AT FIELD** statement is encountered at compile time or runtime in any other event point, it is simply ignored. No other errors or warnings are generated.

Related PDFs

The system automatically sets the predefined fields, CURSOR AT OCCURRENCE and CURSOR AT APPEARANCE, to the corresponding values of the occurrence and appearance numbers for the field on which the cursor is positioned.

Example

The following example shows how to use **AT FIELD** as the first conditional check to determine the course of further processing. In this example, **AT FIELD** is used with an IF statement to check which option a user has selected while positioned on the designated field. The intention of the statement series is, "If a user is positioned on field A, and then selects `OPTION` `3`, take the following actions...".

```
        AT FIELD TAR CUSTOMER ID                (AT APPEARANCE #    )
T       IF       --- OPTION                     EQ     USER 3
TT      SET      TAR WORK MESSAGE               =      ENTER CUSTOMER ID
TF      SET      TAR WORK MESSAGE               =      ENTER INVOICE NUMBER
```

BEG AT

BEG AT specifies the starting key value for a range of records to be read from a file. You may use a **BEG AT** preceding a BEG READ/END READ loop, or before a READNEXT statement. If you want to specify the ending value for the range, you also include an END AT statement following a **BEG AT**.

```
••••• BEG AT   ••• •••••••• •• ••• •••••••••••••••••••• •••
(1)            (2) (3)      (4)(5) (6)                   (7)
```

(1) T/F execution conditions	(5) Field application ID
(2) File application ID	(6) Field name
(3) File name	(7) Occurrence (constant/index)
(4) Relation (IN,EX)	

Using the Statement

When using a **BEG AT** statement, you specify a relation to indicate whether to start at a record with a key that includes (IN) or excludes (EX) the specified value.

> Note that you can specify any field, including a non-key field, in the **BEG AT** statement. If you specify a constant, it is assumed to be an alpha-type constant unless the file is sequential, in which case the value is assumed to be a numeric-type constant. The length of the alpha-type constant is determined by discarding all trailing blanks; hence, an all-blank constant will result in a length of zero, and will cause a subsequent BEG READ or READNEXT to position to the first record in the file, regardless of the relation (IN or EX).

If a BEG READ is not preceded by a BEG AT statement, records will be read from the beginning of the file. If a READNEXT is not preceded by a BEG AT statement, records are read from the current position in the file.

Restrictions

If the file to be read is sequential, the field used in the **BEG AT** statement must refer to a numeric domain, and the key field for the BEG READ or READNEXT statement must be blank.

Since **BEG AT** saves the contents of the specified field regardless of the field-type or format, the field must match the field-type and format of the key field used to access the file in a subsequent BEG READ or READNEXT statement. For example, if the key field in the subsequent BEG READ statement is a binary numeric field with nine digits, you cannot use a seven-digit numeric field or a packed-decimal numeric field in the **BEG AT** statement to set the starting value. This applies only to indexed files; for sequential files, any numeric field can be specified.

To ensure that the format of the field specified in the **BEG AT** is appropriate, use either the specified file's key, a field that is part of the file's key, or a field that is the same domain-type as the file's key. In any case, set the field's content to the desired starting value before executing the **BEG AT** statement.

You cannot use the **BEG AT** statement with one-record files.

Related Statements

BEG READ, END AT, END READ, READNEXT

Examples

The following example uses **BEG AT** (with END AT) to specify a range of records to be read in the file specified in the subsequent BEG READ statement. The example includes a transaction file of records to be posted to the master (CUSTOMER) file. The first part of the key to the transaction file is the customer ID. You want to restrict the reading of the transaction file to only the range of record keys that includes a specific customer ID.

```
BEG AT   TAR TRANFILE IN TAR CUSTOMER ID
END AT   TAR TRANFILE IN TAR CUSTOMER ID
BEG READ TAR TRANFILE              HOLD 0 KEY IS  TRANFILE KEY
*
*        (Process each TRANFILE record for specified customer)
*
END READ TAR TRANFILE
```

The preceding example processes records on the TRANFILE file with a key value within the range specified by the **BEG AT**/END AT pair of statements. If the END AT is not included, processing continues until the end of the file is reached. When used with **BEG READ**, and there are no records on the file that meet the specified starting range criteria, no statements within the BEG READ, END READ are executed.

The next example shows **BEG AT** used preceding a READNEXT statement:

```
        BEG AT   TAR ORDER    IN TAR TRANFILE CUSTOMER ID
        END AT   TAR ORDER    IN TAR TRANFILE CUSTOMER ID
        LABEL    :NEXT ORDER RECORD
        READNEXT TAR ORDER                  HOLD 0 FT 0 BY ORDER CUST ID
    T   ...      (Process each order record for specified customer)
    T   GOTO     :NEXT ORDER RECORD
        *
        *        (Terminate order processing for specified customer)
```

In this example, if the ORDER file contains one or more records matching the specified customer ID, the true/false status indicator is set to *T* by the READNEXT statement.

BEG LOOP

The **BEG LOOP** statement initiates the execution of a loop to repeat the same processing logic for a specified number of times. A matching END LOOP statement is always used in conjunction with a **BEG LOOP** to end the loop.

```
    ••••• BEG LOOP •• = ••• TO •••  STEP •••
    (1)             (2)  (3)    (4)       (5)
```

```
(1) T/F execution conditions

(2) Loop index (A to Z, AI to ZI)

(3) "From" value (constant/index)

(4) "To" value (constant/index)

(5) "Step" value (constant/index)
```

Using the Statement

BEG LOOP incorporates an index counter that is initialized to the starting or “from” value. Looping continues until the index counter is greater than the ending or “to” value (unless the increment has a minus sign, in which case the “from” value is higher than the “to” value and looping stops when the counter is less than the “to” value). The increment or “step” value is added to the index counter following each execution of the loop.

The index counter for the loop can be any of the predefined fields A to Z (variable decimal) or AI to ZI (integer). The “from,” “to,” and “step” values can be any of the predefined fields A to Z or AI to ZI, or they can be specified as a constant integer value, either positive or negative.

Upon completion of the loop, processing resumes with the statement following the matching END LOOP statement.

If the step value is zero, the loop will continue to execute until a GOTO or END statement is encountered. You can also "break out" of the loop by making the END LOOP statement conditional by using T/F indicators. If a condition is met that prohibits the END LOOP statement from being executed, the END LOOP statement is bypassed and the loop immediately ends.

If the A to Z predefined fields are used for the "from," "to," or "step" values, you can increment the step in fractional amounts.

Restrictions

Every loop must eventually be ended by a matching END LOOP using the same index counter.

> Note that, although you are allowed to mix data types within a **BEG LOOP** statement, performance may degrade slightly.

Compilation Errors

The system generates a compile error if a matching END LOOP is not found after a **BEG LOOP** statement, or if an unmatched END LOOP is detected.

Statement Ignored

If the "step" value is positive and the "from" value is greater than the "to" value or, if the "step" value is negative and the "from" value is less than the "to" value, the entire loop is not executed.

Related Statements

END, END LOOP, GOTO

Example

The following example is a simple loop that increments the current monthly budgets by 5 percent for all 12 months:

```
          BEG LOOP AI = 001 TO 012  STEP 001
          COMPUTE  TGL BUDGET AMOUNT          AI  *      1.05
          END LOOP AI
```

After the twelfth repetition (where AI=12), control is passed to the next statement following the END LOOP statement.

The next example presents a nested loop that repeats its processing logic within each occurrence of the main loop. This loop also incorporates a conditional check to see if the value of field C equals that of field D, at which point the internal loop terminates and returns control to the main loop.

```
          BEG LOOP A  = 001 TO 010  STEP 001
          *
          *        (Perform processing if needed)
```

```
        *
        BEG LOOP B  = 001 TO 005  STEP 001
        *
        *        (Perform processing)
        *
        IF       --- C                          EQ --- D
    T   SET      --- B                          =      5
        END LOOP B
        *
        END LOOP A
```

There can be more **BEG LOOP** statements in an ILF routine than END LOOP statements. For example:

```
    T   BEG LOOP A  = 001 TO 010  STEP 001
    F   BEG LOOP A  = 001 TO 005  STEP 001
        *
        *        (Perform processing)
        *
        END LOOP A
```

In the preceding example, only one of the **BEG LOOP** statements can execute.

> Note also that mixing integer and variable decimal data types is allowed. This means that the following types of statements can be defined:

```
        BEG LOOP AI = B   TO CI   STEP D
    or
        BEG LOOP A  = 001 TO XI   STEP ZI
```

In both of these examples, the number of repetitions and the step-values are variables, with values set within the context of the ILF routines.

BEG READ

The **BEG READ** statement initiates a “Read Next Record” loop, in which the next record that meets any previously specified range criteria is read. You end the loop with an END READ statement for the same file. Usually, you precede a **BEG READ** statement with a BEG AT and/or an END AT statement to specify the range of records to be read.

```
    ••••• BEG READ ••• ••••••••••••••••••••• HOLD • KEY IS  ••••••••••••••••••••
    (1)            (2) (3)                        (4)        (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) File name (or field name for RDBMS)
```

```
(4) Hold type (0=do not hold, 1=hold,
    2=hold with potential recovery)

(5) Access key field name
```

Using the Statement

When using this statement in conjunction with an RDBMS table, you can specify a field name instead of a file name to limit the volume of data being passed between APPX and the RDBMS. See the Partial-Record I/O section in Chapter 4-3: Specifying Statements for more information.

When using a **BEG READ** statement, you specify whether the record is to be held for updating. If no update is to occur, enter *0*; if updating is possible or mandatory, enter *1*; if you want to hold the record for potential recovery, enter a *2*. While a record is held, no other user may update or delete it. The *HOLD with potential recovery* specification holds the record and saves a copy of it so that a subsequent rewrite or delete can be performed, even if the hold on the record is subsequently lost (by reading another record with hold before completing the rewrite or delete). "Recover" can rehold the record as long as the saved copy is identical to the original record that was read. Note that performance is improved with *hold type 1*.

You can specify a keypath for indexed files that will serve as the access path to the file. If you leave this parameter blank, the primary key is used.

Records are read until the specified key has a value larger than the value of the field specified in the last END AT statement for the file. For sequential files, records are read until the relative record number for the file is larger than the record number specified in the last END AT statement for the file. If no END AT was executed, the "Read Next Record" loop is repeated until an end-of-file condition occurs.

Execution continues with the statement following the corresponding END READ statement if:

- The end-of-file is reached,
- A record with a key value greater than that specified in the last END AT statement is read, or
- There were no records in the specified range.

If a "Read Next Record" loop terminates for any reason, the record area contains the record that is causing the loop to end.

The file position that you can define by BEG AT/END AT, and maintained and used by the **BEG READ** "Read Next Record" loop, is unique to that specific "Read Next Record" loop. If the file being read by the "Read Next Record" loop is accessed within the loop by other file-access statements, the file position will not be affected when the END READ is executed. For example:

```
          BEG AT   TAR CUSTOMER IN     2
          END AT   TAR CUSTOMER IN     7
          BEG READ TAR CUSTOMER               HOLD 0 KEY IS  CUSTOMER NO
          SET      TAR CUSTOMER NO                =      10
```

```
READ     TAR CUSTOMER               HOLD 0 FT 0 BY CUSTOMER NO
END READ TAR CUSTOMER
```

Customer 10 is outside of the range set up by BEG AT/END AT but the READ statement will be successful (assuming the customer 10 record is on file) because the range specified applies only to the **BEG READ** "Read Next Record" loop. When the END READ executes, the file is repositioned to read the next record in the logical range for the "Read Next Record" loop.

Assuming all customer records are on file, the looping logic for this example is:

- · **BEG READ** reads the customer 2 record,
- READ reads the customer 10 record,
- END READ reads the customer 3 record and returns control to the statement immediately following **BEG READ**,
- READ reads the customer 10 record,
- END READ reads the customer 4 record,

and so on, until the customer 8 record is read, which ends the loop. Processing then continues with the statement following the END READ.

If a READNEXT is executed within the BEG READ "Read Next Record" loop, its position in the file is maintained completely independently of the "Read Next Record" loop. For example:

```
BEG AT   TAR CUSTOMER IN     2
END AT   TAR CUSTOMER IN     4
BEG READ TAR CUSTOMER               HOLD 0 KEY IS  CUSTOMER NO
READNEXT TAR CUSTOMER               HOLD 0 FT 0 BY CUSTOMER NO
END READ TAR CUSTOMER
```

Again assuming all customer records are on file, **BEG READ** reads the customer 2 record. READNEXT reads the customer 1 record (the first record in the file) which is outside the BEG AT/END AT range but, because the range applies only to the **BEG READ** "Read Next Record" loop, the READNEXT succeeds. END READ reads the customer 3 record because this is the next logical record for the **BEG READ** "Read Next Record" loop. READNEXT now reads the customer 2 record, the next logical record in its thread. Continuing in this way, END READ reads the customer 4 record, and READNEXT reads the customer 3 record. Finally, END READ reads the customer 5 record, which ends the **BEG READ** "Read Next Record" loop.

You can also nest **BEG READ** "Read Next Record" loops within a "Read Next Record" loop. The following example loops through the CUSTOMER file and, for each customer record read, loops through the TRANSACT file records for that customer.

```
BEG READ TAR CUSTOMER               HOLD 0 KEY IS  CUSTOMER NO
BEG AT   TAR TRANSACT IN TAR CUSTOMER NO
END AT   TAR TRANSACT IN TAR CUSTOMER NO
BEG READ TAR TRANSACT               HOLD 0 KEY IS  TRANSACT KEY
*
```

```
*        (process transactions for customer)
*
END READ TAR TRANSACT
*
*        (process customer information)
*
END READ TAR CUSTOMER
```

Restrictions

For sequential files, you must leave the key field blank. Note that the **BEG READ** statement does not generate an "In Use" message if the record to be held is currently being held by another user. The **BEG READ** statement either waits until the record is released or, if a TIMEOUT statement with a nonblank label was previously encountered, transfers processing control to the specified label. You cannot use the **BEG READ** statement with one-record files.

Compilation Errors

If there is no END READ statement that corresponds to the **BEG READ** statement, an error occurs at compile time.

Execution Errors

Once the **BEG READ** "Read Next Record" loop terminates, re-executing an END READ statement without first re-executing a **BEG READ** generates a fatal file-processing error.

Related Statements

BEG AT, END AT, END READ, READ, READNEXT, TIMEOUT

Example

The following example demonstrates a common use of the **BEG READ** statement where a range is specified:

```
BEG AT   TAR TRANFILE IN TAR CUSTOMER NO
END AT   TAR TRANFILE IN TAR CUSTOMER NO
BEG READ TAR TRANFILE              HOLD 0 KEY IS  TRANFILE KEY
*
*        (process each TRANFILE record for specified customer)
*
END READ TAR TRANFILE
```

In this example, the system reads the TRANFILE file beginning with the first record where the TRANFILE CUSTOMER NO is equal to or greater than the CUSTOMER NO supplied. (TRANFILE CUSTOMER NO must be the first part of TRANFILE KEY.) The **BEG READ** "Read Next Record" loop

terminates when the TRANFILE CUSTOMER NO exceeds the value specified in the END AT statement. If no BEG AT is present, the read begins with the first record in the file. If no END AT is present, the read continues to the end of the file.

BLANK

You use a **BLANK** statement in input and output processes to display an item as entirely blank. A typical use occurs when entering key values to retrieve associated descriptions from a related file (for example, when entering a transaction type and displaying the description). If a matching key is not found on the “description” file, you may want to blank any previously displayed description.

```
••••• BLANK    ••• •••••••••••••••••••• ••• (AT APPEARANCE # •••)
(1)            (2) (3)                  (4)                  (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or predefined field

(4) Occurrence (constant/index)

(5) Appearance (constant/index)
```

Using the Statement

The appearance number can be used if the same field appears more than once on the image. If the application ID, field name, occurrence number, and appearance number are not specified (left blank), the **BLANK** statement affects all of the items painted on the associated image.

BLANK is one of the *characteristic-type* statements. Refer to the common attributes of all the statements in this category in the Characteristic-Type Statements section of this manual.

The operation of a **BLANK** statement affects only the next generation of an image. In other words, the item characteristics are reset each time an image is produced.

The system ignores any **BLANK** statements specified in non-image related event points.

Related Statements

BLINK, BOLD, BRIGHT, COLOR, DIM, DISPLAY, FONT, INVERSE, NO BOLD, NO ULINE, NORMAL, PICTURE, ULINE

Example

In the following example, if the customer number entered on an order is not a valid customer, the customer name displays as blank.

```
        SET      TAR CUSTOMER NO               =  TAR ORDER CUSTOMER NO
        READ     TAR CUSTOMER              HOLD 0 FT 0 BY CUSTOMER NO
    F   BLANK    TAR CUSTOMER NAME             (AT APPEARANCE #    )
```

BLINK

BLINK is used in input or menu processes to cause an item to blink on an image.

```
    ••••• BLINK    ••• •••••••••••••••••••• ••• (AT APPEARANCE # •••)
    (1)            (2) (3)                  (4)                  (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or predefined field

(4) Occurrence (constant/index)

(5) Appearance (constant/index)
```

Using the Statement

BLINK is often used in conjunction with an ERROR or WARNING statement to highlight the fields in error.

The appearance number can be used if the same field appears more than once on the image. If the application ID, field name, occurrence number, and appearance number are not specified (left blank), the **BLINK** statement affects all of the items painted on the associated image.

BLINK is one of the *characteristic-type* statements. Refer to the common attributes of all the statements in this category in the Characteristic-Type Statements section of this manual.

The operation of a **BLINK** statement affects only the next generation of an image. In other words, the item characteristics are reset each time an image is produced.

Restrictions

> Note that, because characteristic statements operate on items other than fields, these statements are meaningful only in event points that are associated with an image.

BLINK is a terminal display attribute only and, therefore, is not allowed on images in output processes.

Statement Ignored

The system ignores any BLINK statements specified in non-image-related event points.

Related Statements

BLANK, BOLD, BRIGHT, COLOR, DIM, DISPLAY, ERROR, FONT, INVERSE, NO BOLD, NO ULINE, NORMAL, PICTURE, ULINE, WARNING

Example

In the following example, the customer number is verified while entering an order. If the customer is not on file, the corresponding customer number on the order blinks.

```
          SET      TAR CUSTOMER NO                =  TAR ORDER CUSTOMER NO
          READ     TAR CUSTOMER               HOLD 0 FT 0 BY CUSTOMER NO
    F     BLINK    TAR ORDER CUSTOMER NO          (AT APPEARANCE #    )
    F     DISPLAY  TAR ORDER CUSTOMER NO          (AT APPEARANCE #    )
```

BOLD

BOLD is used to print an item with a higher density print attribute. Many printers achieve the bolded effect by the use of a multi-strike mechanism.

```
    ••••• BOLD     ••• ••••••••••••••••••••• ••• (AT APPEARANCE # •••)
    (1)            (2) (3)                  (4)                   (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or predefined field

(4) Occurrence (constant/index)

(5) Appearance (constant/index)
```

Using the Statement

The appearance number can be used if the same field appears more than once on the image. If the application ID, field name, occurrence number, and appearance number are not specified (left blank), the **BOLD** statement affects all of the items painted on the associated image.

BOLD is one of the *characteristic-type* statements. Refer to the common attributes of all the statements in this category in the Characteristic-Type Statements section of this manual.

The operation of a **BOLD** statement affects only the next generation of an image. In other words, the item characteristics are reset each time an image is produced.

Restrictions

BOLD is a printed output display attribute only and therefore is not allowed in images in non-output processes.

Statement Ignored

The system ignores any **BOLD** statements specified in non-image-related event points.

Related Statements

BLANK, BLINK, BRIGHT, COLOR, DIM, DISPLAY, FONT, INVERSE, NO BOLD, NO ULINE, NORMAL, PICTURE, ULINE

Example

In the following example, if the customer number entered on an estimate is a valid customer, the customer name is printed in bold characters on the report.

```
        SET      TAR CUSTOMER NO                =  TAR ESTIMATE CUSTOMER NO
        READ     TAR CUSTOMER               HOLD 0 FT 0 BY CUSTOMER NO
    T   BOLD     TAR CUSTOMER NAME              (AT APPEARANCE #    )
    T   DISPLAY  TAR CUSTOMER NAME              (AT APPEARANCE #    )
```

BRIGHT

BRIGHT is used to display an item with high intensity.

```
    ••••• BRIGHT   ••• ••••••••••••••••••••• ••• (AT APPEARANCE # •••)
    (1)            (2) (3)                   (4)                  (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or predefined field

(4) Occurrence (constant/index)

(5) Appearance (constant/index)
```

Using the Statement

The appearance number can be used if the same field appears more than once on the image. If the application ID, field name, occurrence number, and appearance number are not specified (left blank), the **BRIGHT** statement affects all of the items painted on the associated image.

BRIGHT is one of the *characteristic-type* statements. Refer to the common attributes of all the statements in this category in the Characteristic-Type Statements section of this manual.

The operation of a **BRIGHT** statement affects only the next generation of an image. In other words, the item characteristics are reset each time an image is produced.

Restrictions

> Note that, because characteristic statements operate on items rather than fields, these statements are meaningful only in event points that are associated with an image.

BRIGHT is a terminal display attribute only and, therefore, is not allowed in images in output processes.

Statement Ignored

The system ignores any **BRIGHT** statements specified in non-image-related event points.

Related Statements

BLANK, BLINK, BOLD, COLOR, DIM, DISPLAY, FONT, INVERSE, NO BOLD, NO ULINE, NORMAL, PICTURE, ULINE

Example

In the following example, if the customer number entered on an order is a valid customer, the customer name displays with high intensity.

```
          SET      TAR CUSTOMER NO                =  TAR ORDER CUSTOMER NO
          READ     TAR CUSTOMER               HOLD 0 FT 0 BY CUSTOMER NO
    T     BRIGHT   TAR CUSTOMER NAME              (AT APPEARANCE #    )
    T     DISPLAY  TAR CUSTOMER NAME              (AT APPEARANCE #    )
```

CALC

The **CALC** statement allows you to perform complex calculations in a single statement (refer to the COMPUTE statement for simpler calculations involving only two fields or a field and a constant). The result of the calculation is placed in any of the index predefined fields, A to Z or AI to ZI.

```
••••• CALC     •• = ••••••••••••••••••••••••••••••••••••••••••••••••••••••••
(1)            (2)  (3)
```

```
(1) T/F execution conditions

(2) Receiver index PDF (A to Z, AI to ZI)

(3) Algebraic computation (59 characters max)
```

Using the Statement

The algebraic computation can contain index predefined fields A to Z and AI to ZI, numeric constants, operators (+, -, *, and /), and any number of levels of parentheses. The **CALC** statement performs operations according to the standard algebraic rules of precedence:

- Unary negation (-) (i.e., reversing the signs on numbers),
- Then, for the innermost pair of parentheses:
- Multiplication (*) and division (/), which have precedence over
- Addition (+) and subtraction (-)
- Then, proceed to each succeeding layer of parentheses, if applicable.

Precision for the floating-point variables A to Z is a maximum of 15 total decimal positions. Precision for the integer variables AI to ZI is a maximum of 10 total decimal positions.

Standard Math Functions

APPX supports the use of several standard math functions in the **CALC** statement. To use one of these functions, specify the function name, followed by its arguments in parentheses. For instance:

```
          CALC     A  = @ABS(B)
```

In this case, the absolute value of *B* (the argument) is assigned to *A*.

The function arguments can be expressions that may contain other functions. For instance, the following **CALC** statements are legal:

```
          CALC     A  = @POW(B,1/3)
```

... takes the cube root of *B*

```
          CALC     A  = @SQRT(@POW(B-C,2)+@POW(D-E,2))
```

... computes the absolute distance between two Cartesian points

APPX does *not* check for illegal argument values (such as taking the square root of a negative number). In general, you should check arguments for validity before using them in certain functions. If an error occurs, standard OVERFLOW processing takes place (you can tell *where* an error occurred, but not *what* caused it). The math functions fall into three groups described in corresponding sections: basic functions, exponential functions, and trigonometric functions.

Restrictions

The fields referenced in a **CALC** statement are restricted to the index predefined fields (A to Z and AI to ZI). Therefore, you frequently precede a **CALC** statement with one or more SET statements to move values into such fields, as in the following example:

```
*
*        A = Unit price
*        B = Quantity ordered
*        C = Extension
*
SET      --- A                          =  TOE ORDER UNIT PRICE
SET      --- B                          =  TOE ORDER QUANTITY
CALC     C = A*B
SET      TOE ORDER AMOUNT               =  --- C
```

> Note the use of comments to clarify the variable names used in the calculation. You may find it helpful to include comments whenever predefined variable names are used.

Compilation Errors

If the algebraic expression is invalid, a compilation error occurs. Invalid algebraic expressions include those with too few operands per operator and those with unbalanced parentheses.

Execution Errors

A runtime error can occur if an intermediate or final calculation results in overflow or division by zero.

Related Statements

COMPUTE, SET, OVERFLOW

Example

The following example demonstrates a calculation involving three index variables (B, C, and DI). The results are then placed in the receiver index variable (AI).

```
**       The following statements calculate the total estimated monthly
*        compensation (AI) for a salesperson. This formula includes a
*        fixed monthly base salary ($1,800), an annual bonus (B), and a
*        commission based on the number of units sold (DI) over 100
*        times the average sale (C) less $50 per unit. The bonus and
```

```
*        comission are annualized so they need to be divided by 12
*        to derive a monthly amount.
*
SET      --- B                          =  TSA SALES ANNUAL BONUS
SET      --- C                          =  TSA SALES AVG UNIT SALE
SET      --- DI                         =  TSA SALES NO UNITS SOLD
CALC     AI = 1800 + (B + (C - 50) * (DI - 100) ) / 12
SET      TSA SALES MONTHLY SALARY       =  --- AI
```

In this example, the constant *50* is subtracted from the predefined field *C*, and the constant *100* is subtracted from the predefined field *DI*, before the two intermediate results are multiplied together. Next, *B* is added to the result of the multiplication, and that result is divided by 12. Finally, that result is added to 1800, rounded to an integer, and stored in *AI*.

In this example, note that only the predefined field *AI* is affected. The fields *B, C,* and *DI* remain unaltered.

CALC - Basic Functions

Function

Description

Example

@ABS(X)	Returns the positive value of X, regardless of the sign of X.	**@ABS(3) = 3** **@ABS(-3) = 3**
@CEIL(X)	Returns the smallest integer not less than X.	**@CEIL(-2.1) =-2** **@CEIL(-2.0) =-2** **@CEIL(0) = 0** **@CEIL(2.0) = 2** **@CEIL(2.1) = 3**
@FLOOR(X)	Returns the largest integer not greater than X.	**@FLOOR(-2.1) =-3** **@FLOOR(-2.0) =-2** **@FLOOR(0) = 0** **@FLOOR(2.0) = 2** **@FLOOR(2.1) = 2**
@MOD(X,Y)	Returns the remainder of X divided by Y. Y cannot be zero. The result has the same sign as X, regardless of the sign of Y.	**@MOD(5,2) = 1** **@MOD(-5,3) =-2** **@MOD(10,5) = 0** **@MOD(5,0) = ERROR**

@POW(X,Y)	Returns the value of X raised to the power of Y. An error occurs if zero is raised to the zero or a negative power, or if a negative number is raised to a non-integer power. Overflow and underflow errors are also possible, especially with large values of Y.	**@POW(5,2) = 25** **@POW(1.5,2.5) = 2.755676...** **@POW(25,-.5) = .2** **@POW(0,2) = 0** **@POW(0,0) = ERROR** **@POW(0,-2) = ERROR** **@POW(-.6,2.5) = ERROR** **@POW(12345,67890)=OVERFLOW** **@POW(.12345,67890)=UNDERFLOW**
@SQRT(X)	Returns the square root of X. An error occurs if X is negative. This function is equivalent to @POW(X,.5).	**@SQRT(9) = 3** **@SQRT(2) = 1.414...** **@SQRT(0) = 0** **@SQRT(-2) = ERROR**

CALC - Exponential Functions

Function

Description

Example

`@LOG(X)`	Returns the natural log (base e) of X. An error occurs if X is zero or negative.	`@LOG(100) = 4.60517...`
		`@LOG(0 = ERROR`
		`@LOG(-2) = ERROR`

`@EXP(X)`	Returns the value of e raised to the power of X.	`@EXP(2) = 7.38905...`
		`@EXP(1) = 2.71828...`
		`@EXP(0) = 1`
		`@EXP(-2.5) = .08208...`

`@LOG10(X)`	Returns the base 10 log of X. An error occurs if X is 0 or negative. To compute the antilog, you can use @POW, such as @POW(10,X).	`@LOG10(100) = 2`
		`@LOG10(50) = 1.69897...`
		`@LOG10(.5) =-.30102...`
		`@LOG10(0) = ERROR`
		`@LOG10(-5) = ERROR`

CALC - Trigonometric Functions

The standard trigonometric functions (sine, cosine, tangent, and the hyperbolic equivalents) take angle values expressed in radians. Large angles in excess of +/-2π may cause errors or return meaningless values.

The inverse functions (arc sine, arc cosine, and arc tangent) return angles in radians. The arguments must be in the range generated by the standard function to prevent errors. Return values are in the range (-π/2, π/2), except for `@ATAN2`, which returns a normalized angle in the proper quadrant (between -π and π), based on the signs of its two arguments.

The following functions are available:

`@SIN(X)`	`sine of X`
`@SINH(X)`	`hyperbolic sine of X`
`@ASIN(X)`	`arc sine of X`
`@COS(X)`	`cosine of X`
`@COSH(X)`	`hyperbolic cosine of X`
`@ACOS(X)`	`arc cosine of X`
`@TAN(X)`	`tangent of X`
`@TANH(X)`	`hyperbolic tangent of X`
`@ATAN(X)`	`arc tangent of X`
`@ATAN2(X,Y)`	`normalized arc tangent of X/Y`

CALL

You use the **CALL** statement to invoke a program function or routine written in any non-APPX programming language (such as COBOL, “C,” or assembler). Some systems require non-APPX code to be linked to APPX. You can pass values or parameters to the called routine, and share (receive back) them in the calling process. You do this by including one or more PASS statements before specifying the **CALL**. The sequence of PASS statements is very important, and controls which values are used and passed between the calling process and the called routine. The returned value from CALL functions is put into a four-byte binary predefined field called RETURN CODE.

```
••••• CALL     ••• •••••••••••••••••••• ••• RESIDENT? •  END? •  FAIL •
(1)            (2) (3)                    (4)          (5)     (6)     (7)
```

(1) T/F execution conditions	(5) Resident? (Y/N)
(2) Application ID	(6) End? (Y/N)
(3) Program name (field/constant)	(7) Fail action (0=nothing, 1=warning,
(4) Occurrence (constant/index)	2=error, 3=cancel)

Sets True/False Status Indicator

If the called routine cannot be invoked, the next true/false status indicator is set to false. If the invocation completes successfully, the next true/false indicator is set to true. If the routine cannot be invoked successfully, the *FAIL* parameter determines what additional action to take.

Resident?

On machines that run dynamic loading, a *Y* in this field determines whether or not the non-APPX program will stay loaded for future use.

End?

The *END?* parameter is currently ignored.

Fail

See the FAIL (or FT) Parameter section for a complete discussion of this parameter.

Using the Statement

You specify a program to be called by supplying the program as a constant or by supplying an application ID and a field name containing the program name. The format for the program/routine name is system-specific. See the APPX installation guide for your type of system for information on how to structure this field.

Passing Values

The CALL statement passes the current PASS list, if any. Refer to the PASS Lists section for details.

Related Statements

PASS, RUN

Example

The following group of statements calls a routine (as named in TAR field INT CALCULATOR) which is used to compute the total interest on a loan. You need to pass the principal, annual rate of interest, and term of the loan in months.

```
PASS     TAR RATE PRINCIPAL            FIELD            SHARE? N
PASS     TAR RATE ANNL INT             FIELD            SHARE? N
PASS     TAR RATE TERM MONTHS          FIELD            SHARE? N
PASS     TAR RATE TOTAL INTEREST       FIELD            SHARE? Y
CALL     TAR INT CALCULATOR            RESIDENT? Y  END? N  FAIL 3
```

Notice that the *SHARE?* specification is *Y* (yes) for the total interest field, so that the calling process can receive the computed value back.

Using the CALL Statement to Access User-Provided Functions

External routines linked in via call.c should be referenced by prefixing their names with a "." in the CALL statement. For example, one might use

```
          CALL        .myfunc                    RESIDENT? Y  END? N  FAIL 3
```

to call a user-provided function named “myfunc”.

The calling conventions used depend on the type of data passed to APPX, and whether or not you are passing the data as shared or not.

Alpha fields passed to a user-provided function, whether as shared or unshared data items, should be declared, and used, in the C function as “char *”.

Binary fields are handled differently, based on a number of criteria. APPX will pass a BINARY field to a called C function as either a 1-byte, 2-byte or 4-byte integer, depending on how the field is defined. This is important to take into account, when declaring the parameters to your C function. If the parameter definitions don’t match the sizes of fields sent to the function by APPX, you may notice that the values of your parameters are off by 256, 64 or some other multiple of 2, when your function runs. Guidelines used by APPX to determine how a binary field is passed to a called function include:

- If the field, as defined in APPX, includes a range check to verify that it is within the size which can be stored in one byte, and the field includes 3 digits or less, then APPX will pass the field as a 1-byte integer. For signed APPX fields, the range check should be -127 <= x <= 127, and for unsigned APPX fields, the range check should be 0 <= x <= 256. The C declaration used to access this parameter would be “char” (signed) or “unsigned char” (unsigned), if it has been passed with SHARE?=N; “char *” or “unsigned char *” respectively, if passed with SHARE?=Y.
- If the field, as defined in APPX, includes a range check to verify that it is within the size which can be stored in two bytes, and the field includes 5 digits or less, then APPX will pass the field as a 2-byte integer. For signed APPX fields, the range check should be -32767 <= x <= 32767, and for unsigned APPX fields, the range check should be 0 <= x <= 65535. The C declaration used to access this parameter would be “short int” (signed) or “unsigned short int” (unsigned), if it has been passed with SHARE?=N; “short int *” or “unsigned short int *” respectively, if passed with SHARE?=Y.
- Otherwise, the field is passed as a 4-byte integer. The C declaration used to access this parameter would be “int” (signed) or “unsigned int” (unsigned), if it has been passed with SHARE?=N; “int *” or “unsigned int *” respectively, if passed with SHARE?=Y.

CANCEL

You use a **CANCEL** statement to cancel a process after detecting a serious processing problem. You can use **CANCEL** in any event point in any process to clear the screen, display the cancel message, and cancel the process. **CANCEL** provides an orderly termination of a process.

```
..... CANCEL   ..............................................................
(1)            (2)
```

```
(1) T/F execution conditions

(2) Cancel message (constant or TEMPn predefined field)
```

Using the Statement

Instead of displaying a fixed **CANCEL** message (as shown in the example), you can display a variable message. You do this by setting one of the TEMPn predefined fields to the desired message, and entering *TEMPn* in place of specific message text in the **CANCEL** statement.

Related PDFs

TEMPn

Related Statements

SET, SET TEMP

Example

The following example demonstrates a conditional execution of the **CANCEL** statement if a specified file is not available.

```
          IF EXIST TAR CUSTOMER            FAIL 0
    F     CANCEL   Customer File Does Not Exist -- Contact designer for assistance!
```

CLOSE

The CLOSE statement closes the currently opened instance of a specified file (either disk or cache) when required processing is completed. Note that, while OPEN statements may be needed to switch between disk and cache versions of a file or used for performance reasons, CLOSE statements are normally not necessary since APPX takes care of closing files automatically.

```
    ••••• CLOSE    ••• ••••••••  PHYSICAL?      •
    (1)            (2) (3)                    (4)
```

```
(1) T/F execution conditions

(2) Application ID

(3) File name

(4) Physical close? (Y/N)
```

Using the Statement

The CLOSE statement applies only to the last OPEN operation for the file specified. If no preceding OPEN statement for the specified file is encountered, the file for the current database is closed.

The CLOSE statement closes the currently opened instance of a file, either disk or cache. Like memory files, there can be only a single instance of a cached file. There is not a separate instance of a cached file for each database.

Although a file cannot be physically opened again once it has already been opened, it may be logically opened many times, once for each process in the process family. In like manner, you can use the **CLOSE** statement to logically close the file in the current process or to physically close the file for all processes by specifying *N* or *Y*, respectively, in the *Physical?* parameter.

Regardless of whether a file is logically or physically closed, you must open it again before using it in the current process. However, the file may still be logically open in other processes in the process family, and those processes can continue to use the file without having to reopen it.

Restrictions

> Note that when a file is physically closed, all holds previously specified are lost, and all file-read positions are set to the beginning of the file or the beginning of the range in the file, if so specified.

Statement Ignored

If a file is not currently open, the **CLOSE** operation has no effect.

Related Statements

OPEN

Example

The following example performs a logical close on the ORDER file:

```
          CLOSE    TAR ORDER     PHYSICAL?      N
```

CNV BIN

The CNV BIN instruction converts an alpha field containing a binary number to or from a numeric field. This may be useful when operating on single characters of alpha-type fields, such as when you want to convert an alpha field from lower case to upper case.

```
••••• CNV BIN  ••• ••••••••••••••••••••• ••• =  ••• ••••••••••••••••••••• •••
(1)            (2) (3)                    (4)    (5) (6)                    (7)
```

(1) T/F execution conditions	(5) Source application ID
(2) Destination application ID	(6) Source field name, PDF, or constant
(3) Destination field name or PDF	(7) Source occurrence (constant/index)
(4) Destination occ (constant/index)	

Using the Statement

If the source field is an alpha field, it is evaluated as a one-, two-, or four-byte binary value, depending on its length, and converted to the numeric destination field.

If the source field is a numeric field, its numeric value is converted into a one-, two-, or four-byte binary number, depending on the length of the destination field. The source field will be rounded to a whole number.

Restrictions

For an alpha source field, a numeric overflow condition occurs if the destination field is not large enough to store the converted value. The destination field must be numeric.

For a numeric source field, the destination must be alpha and between one, two, or four bytes in size. Based on the size of the destination field and the sign, the numeric value of the second field used in the conversion must be within the range specified below, or a numeric overflow condition occurs:

one byte signed:	-128 through +127
two bytes signed:	-32,768 through +32,767
four bytes signed:	-2,147,483,648 through +2,147,483,647
one byte unsigned:	0 through 255
two bytes unsigned:	0 through 65,535
four bytes unsigned:	0 through 4,294,967,295

Overflow processing is dependent on the use of OVERFLOW statements within the current event point.

Compilation Errors

The alpha field must be one, two, or four characters in length or a compile error occurs.

Related Statements

OVERFLOW

Example

The following example demonstrates how to convert an ASCII letter from lower case to upper case:

```
          CNV BIN  TAR WORK NUMBER               =  TAR ALPHA FIELD
          COMPUTE  TAR WORK NUMBER               -      32
          CNV BIN  TAR ALPHA FIELD               =  TAR WORK NUMBER
```

In this example, the alpha character is converted into a binary number (WORK NUMBER); then, the decimal value of *32* is subtracted from the field, producing the binary value of an uppercase letter. Finally, this value is converted back to its alpha representation.

CNV PORT

CNV PORT works like CNV TEXT except that the portable format is designed for import/export purposes and is immune to differences in numeric and date masks.

CNV PORT ensures that you can import anything that you export.

CNV TEXT

The CNV TEXT statement is bidirectional (data may be converted and then converted back to the original format) and reformats, or converts, data from one data type to another. The CNV TEXT statement sets the destination field equal to the sending field, but in the format of the destination field, including all display mask specifications. CNV TEXT also converts an entire APPX record to an alpha field consisting of a string of the original APPX fields converted to simple alpha fields, or vice-versa.

```
••••• CNV TEXT ••• ••••••••••••••••••••• ••• =  ••• ••••••••••••••••••••• •••
(1)            (2) (3)                    (4)    (5) (6)                    (7)
```

(1) T/F execution conditions	(5) Source application ID
(2) Destination application ID	(6) Source field name, PDF, or constant
(3) Destination field name or PDF	(7) Source occurrence (constant/index)
(4) Destination occ (constant/index)	

Sets True/False Status Indicator

If the conversion is successful, the next true/false status indicator is set to *T*; otherwise, it is set to *F*.

Using the Statement

When converting fields, both fields can be specified as either a field name or a predefined field. One of the fields (either the destination or source) must be of a simplified alpha type. A simplified alpha type is one that is not compressed, tokenized, word-wrapped, or masked.

When converting files, a file name must be entered as either the destination or source, and an alpha field name must be entered as the other. If needed, use Option 88 to save the statement.

Source Field Considerations

Either the source field or destination field can be any of the basic data types: alpha, date/time, logic, or numeric. The other field must be simplified alpha.

Converting to/from Alpha Fields

Alpha fields require no data conversion. The data is copied from the source field to the destination field or vice-versa. If the size of the data to be copied is larger than the destination area can contain, the excess is truncated. If the amount of data to be copied is smaller than the destination can contain, blank padding is added. The formatting characteristics of the destination field, such as justification, are imposed upon the converted destination data.

Converting to/from Date/Time Fields

Date/time fields are converted from their specific date/time types into an alpha date form during the conversion operation or vice-versa. This alpha form is either Gregorian or Julian (based on the source field's data type), and it uses the source field's display mask.

Converting to/from Logic Fields

If the sending field is a "logical Yes" (a value of *1*), the destination field is set to a value of the uppercase *Y* (in the English language version), followed by pad spaces. If the value of the sending field is "logical No," the destination field is set to *N,* followed by pad spaces. If the value of the sending field is blank, the destination field is set to a value of blank. The conversion fails if the value of the sending field is something other than *1*, *0,* or " ". Note these rules also work in reverse.

Converting to/from Numeric Fields

Numeric sending fields are converted from their packed or binary numeric types into an alpha form during the conversion operation. The alpha form is based on the source field's display mask. If the sending field is null-valued, the alpha form is set to blank. This conversion can also be performed in reverse.

When converting numeric fields to alpha, remember that a display mask on the numeric field can be used to control the format of the result. For example, a display mask can prevent producing negative trailing signs or integer results which may be undesirable in Excel.

Restrictions

When all occurrences of the sending field are selected as the **CNV TEXT** data (by entering occurrence number 000), all occurrences are treated as a single, rectangular alpha field. When the data is copied from the sending field, a check is made to ensure that soft spaces are compressed, ensuring that the next "word" is separated from the previous one by just one space.

Compilation Errors

If either the source or destination field is not a simplified alpha type, a compilation error occurs. Attempting to convert invalid data is an unrecoverable error.

Invalid dates cannot be converted and cause **CNV TEXT** to fail.

Execution Errors

Numeric values that cannot be represented by their display masks cannot be converted, and cause the digit placeholders of the mask to be set to *?* to indicate a failed conversion.

Example

The following example moves the current date into an alpha field then converts an entire record into an alpha field:

```
          SET DATE TAR ORDER DATE FIELD
          CNV TEXT TAR WORK ALPHA FIELD           =  TAR ORDER DATE FIELD
          CNV TEXT TAR WORK RECORD                =  TAR TERMS
```

The alpha fields will include all display formatting characters.

COLOR

The **COLOR** statement changes the color of a displayed field's foreground and/or background based on the contents of the predefined fields COLOR FOREGROUND and COLOR BACKGROUND. All fields displayed on the screen can be affected by using the global **COLOR** statement, as shown below, or a specific field's colors can be changed by identifying the field.

```
          SET      --- COLOR FOREGROUND           =      RED
    ••••• COLOR    ••• ••••••••••••••••••••• ••• (AT APPEARANCE # •••)
    (1)            (2) (3)                    (4)                   (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or predefined field

(4) Occurrence (constant/index)

(5) Appearance (constant/index)
```

Using the Statement

The appearance number can be used if the same field appears more than once on an image. If the application ID, field name, occurrence number, and appearance number are not specified (left blank), the **COLOR** statement affects all of the items painted on the associated image using either default colors or the colors specified in the predefined fields COLOR FOREGROUND and COLOR BACKGROUND.

COLOR is one of the *characteristic-type* statements. Refer to the common attributes of all the statements in this category in the Characteristic-Type Statements section of this manual.

The operation of a **COLOR** statement affects only the next generation of an image. In other words, the item characteristics are reset each time an image is produced,

Restrictions

> Note that, because the **COLOR** statement operates on items rather than fields, this statement is meaningful only in event points that are associated with an image.

Statement Ignored

The system ignores any **COLOR** statements specified in non-image-related event points.

Related Statements

BLANK, BLINK, BOLD, BRIGHT, DIM, DISPLAY, FONT, INVERSE, NO BOLD, NO ULINE, NORMAL, PICTURE, ULINE

Example

The **COLOR** statement is used to cause the current foreground and/or background colors of a field to change on the image. In this example, the customer's credit status displays in red if the status is BAD. If the status is not BAD, it is displayed in blue.

```
        IF       TAR CUSTOMER CREDIT STATUS     EQ     BAD
   T    SET      --- COLOR FOREGROUND           =      RED
   F    SET      --- COLOR FOREGROUND           =      BLUE
        COLOR    TAR CUSTOMER CREDIT STATUS     (AT APPEARANCE #    )
        DISPLAY  TAR CUSTOMER CREDIT STATUS     (AT APPEARANCE #    )
```

COMMIT

The **COMMIT** statement causes an immediate commit of all changes made to the tables stored within an RDBMS. All record locks are forfeited.

```
     ••••• COMMIT
     (1)
```

```
(1) T/F execution conditions
```

Using the Statement

The COMMIT MODE predefined field determines if and when APPX commits your changes automatically. It is initially set based on the Commit Mode specification from your process (in Additional Attributes) but can be modified. If you change COMMIT MODE, the change will take affect at the next COMMIT or ROLLBACK.

Based on the structure of your application, there may be other places where you want to COMMIT a transaction. That's what the COMMIT statement is for. One thing to keep in mind is that the frequency of COMMITs will affect performance. The more COMMITs you do, the slower your application will be. For interactive processes, you can probably ignore the performance issue, but for batch-mode operations, COMMITing too often can really slow things down.

Another issue to be aware of is that APPXIO files do not support transaction processing. If you ROLLBACK a change to an RDBMS-hosted file, any changes made to APPXIO will not be rolled back. Related to this is the issue of visibility. Changes made to an RDBMS-hosted file are invisible to other users until they are committed. Changes made to an APPXIO-hosted file are visible immediately.

Related PDFs

COMMIT MODE

Related Statements

ROLLBACK, SAVEPNT

Example

In the following example, if the flag indicates it is OK to commit, the COMMIT takes place and processing continues. Otherwise, changes made since the savepoint designated as TRX POINT 1 are discarded and the program branches to an applicable routine.

```
          SAVEPNT      TRX POINT 1
          *
          <perform RDBMS processing>
          *
          IF       TAR WORK OK TO COMMIT          NE     Y
     T    ROLLBACK     TRX POINT 1
     T    GOTO         :TRX POINT 1 STATUS
          COMMIT
```

COMPUTE

You can use the COMPUTE statement for simple arithmetic computations involving two fields, or one field and a constant. The COMPUTE statement adds to (+), hash-adds to (P), subtracts from (-), hash-subtracts from (M), multiplies (*), or divides (/) the first (or destination) operand by the value specified in the second (or source) operand.

```
••••• COMPUTE  ••• ••••••••••••••••••••• ••• •  ••• ••••••••••••••••••••• •••
(1)            (2) (3)                    (4)(5) (6) (7)                    (8)
```

(1) T/F execution conditions	(5) Operator (+, -, *, /, P, M)
(2) Destination application ID	(6) Source application ID
(3) Destination field name or PDF	(7) Source field name, PDF, or constant
(4) Destination occ (constant/index)	(8) Source occurrence (constant/index)

Using the Statement

The destination field can be packed decimal or binary, and the source field can be packed decimal, binary, or alpha. The packed decimal fields can have either a fixed or variable decimal position, and binary fields always have a fixed decimal position.

If the second field is an alpha field type (valid only for *P* and *M* operators), characters other than *0* to *9*, except trailing spaces, are converted to *0*. Then, up to the first 31 characters are converted to a packed decimal integer before the computation is performed. Once converted, the remainder of the computation is identical to a computation that contains a packed decimal integer for the second field. The hash total operators ignore overflow.

If the result of a *P* or *M* calculation results in a value greater than the destination field can hold, the result "wraps around" from zero. For example, if a destination field has a maximum of five digits, *99999 P 1* results in a value of *00000*. The most significant number is discarded. As another example, assuming that the packed decimal number could contain a value up to *999*, *996 P 8* results in a value of *4*.

Similarly, if the result of the *P* or *M* calculation results in a negative value, the result "wraps around" from the largest value the destination field could hold plus *1*. For example, assuming that the packed decimal number could contain a value up to *9999*, then *3 M 6* results in a value of *9997*.

Restrictions

If there is a division by zero, or the result is too large to be stored in the destination field, an overflow situation occurs. Overflow processing is dependent on the use of the OVERFLOW statement.

For the *P* and *M* operators, the destination field must be a packed decimal numeric type.

Execution Errors

If OVERFLOW has not been specified, an overflow condition causes an unrecoverable processing error.

Statement Ignored

If either field contains a null value, no computation takes place, and the statement following the **COMPUTE** is executed.

Related Statements

OVERFLOW

Example

The following is an example of a **COMPUTE** statement:

```
          COMPUTE  TAR ORDER2 SUBTOTAL           +  TAR ORDER3 LINE AMOUNT
```

In this example, the value in the field ORDER3 LINE AMOUNT is added to the value in ORDER2 SUBTOTAL. Note that the value in ORDER3 LINE AMOUNT is not changed by the computation.

COPY

With the COPY statement, you can include an exact copy of the ILF statements in a subroutine process within the one containing the COPY statement. The copied statements are added to the current event point during compilation. There are two primary advantages to using COPY rather than simply re-entering the desired statements in the current ILF routine:

- You can standardize certain routines, such as a calculation that is performed at several points in your application, and
- If a change is required to a copied routine, you make the changes in only one place and they are automatically reflected where the routine has been copied.

```
    ••••• COPY     ••• ••••••••••••••••••••••••••••
    (1)            (2) (3)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Subroutine name
```

Using the Statement

You can use a related statement, SUBR, to execute a subroutine without actually copying it into the current process. This statement is particularly useful if a subroutine must be used in more than one event point in a process, or if the subroutine is very large and therefore extends the compilation time of a process.

Compilation Errors

An error is not generated if a statement references a non-existing application. A “Missing Subroutine” error is generated if a non-existing subroutine process name is encountered within a valid application.

Recursive use of a COPY statement, by which a COPY statement attempts to copy itself, is not allowed. A “Copy Used Recursively” error is generated in this case.

Related Statements

GOTO, GOSUB, LABEL, SUBR

Example

The following example copies a subroutine process that performs a common routine to verify general ledger account numbers:

```
          COPY     TGL VERIFY ACCOUNT
```

CREATE

The CREATE statement makes certain that a file exists and is opened for use. If the file does not already exist, this statement creates the file. CREATE is often used following a SCRATCH statement; the combination has the effect of clearing all records in a file.

```
    ••••• CREATE   ••• ••••••••  SHARE? •  FAIL • CACHE •••••••
    (1)            (2) (3)             (4)     (5)      (6)
```

```
(1) T/F execution conditions

(2) Application ID

(3) File name

(4) Share the file? (Y/N)

(5) Fail action (0=nothing, 1=warning, 2=error, 3=cancel)

(6) Cache type (NONE, LOAD, AUTO, NOLOAD)
```

Sets True/False Status Indicator

The next level of the internal true/false status indicator is set to *T* if the file is successfully created. The indicator is set to *F* if creation fails for any reason other than the file already exists. In this case, the *FAIL* parameter determines what additional action to take. In addition, on a false condition, the predefined field STATUS CODE returns a token indicating the reason for an I/O command failure. This token can be used as a key to read the system message file to obtain a descriptive I/O error message (see STATUS CODE for details).

Using the Statement

You specify the application ID and file name for the file to be created in the statement. The *SHARE?* parameter determines whether or not other users are permitted to update the file while it is open. Additional information that can affect the location of the file may be contained in the predefined fields DATABASE or NEXT DATABASE.

You can use the NEXT DATABASE predefined field to specify in which database you want the file created. If NEXT DATABASE is blank, the current database (located in DATABASE) is used to create the file. Note that NEXT DATABASE is modifiable, whereas DATABASE is not, and that NEXT DATABASE is set to blank by the CREATE statement.

The CREATE statement can also create and open a cached instance of a file. Specifying a *CACHE* option other than *NONE* causes a cached instance of a file to be created in memory and opened instead of the normal disk instance of the file. This can result in significant reductions in File I/O during posting and reporting processes. Once you have created a cached instance of a file, all I/O that follows will reference the cached instance of the file, not the disk instance, until you OPEN the disk version. If you rewrite, delete, or write records to the cached version of the file, the disk version of the file will not be updated. If the disk version of the file changes after you create and load the cached version of the file, the cached version of the file will not reflect those changes.

If NEXT DATABASE is specified for the CREATE statement which creates a cached file, then the value of NEXT DATABASE determines the source database for automatically loading the cache. Otherwise, the current database determines the source of the file used to load the cache.

After creating a cache, you can switch between the cache and disk versions by using the OPEN statement.

Valid values for the *CACHE* option are:

- NONE creates and opens the file as a normal, non-cached file. This is the default value.
- NOLOAD creates and opens the cached version of the file without loading any records into it. Records are added to cache via the WRITE statement.
- LOAD creates and opens the cached version of the file and loads it from the disk version of the file.
- AUTO creates and opens the cached version of the file and dynamically populates the cache from the disk version of the file as records are individually read.

Related PDFs

DATABASE, NEXT DATABASE

Related Statements

CLOSE, IF EXIST, OPEN, SCRATCH

Example

The following example scratches and then recreates and opens a file called MONTHTRX, allows it to be updated by other users, and generates an error message if the operation fails for any reason.

```
          SCRATCH  TGL MONTHTRX            FAIL 0 CACHE? N
          CREATE   TGL MONTHTRX  SHARE? Y  FAIL 0 CACHE NONE
    F     ERROR    Unable to Create TGL MONTHTRX File!
```

DATE ADD

The DATE ADD statement adds (or subtracts by adding a negative value) a number of time units (years, months, days) to a specified date/time field, to determine a future (or past) date/time. For example, you may want to add 10 days to the DUE DATE field to determine when a 10-day discount period for prompt payment expires.

```
••••• DATE ADD ••• •••••••••••••••••••• ••• •  ••• •••••••••••••••••••• •••
(1)            (2) (3)                   (4)(5) (6) (7)                   (8)
```

(1) T/F execution conditions	(6) Source application ID
(2) Destination application ID	(7) Source field name, PDF, or constant
(3) Destination field name or PDF	(8) Source occurrence (constant/index)
(4) Destination occ (constant/index)	

```
(5) Unit (8:centuries, 7:years, 6:months, 5:days, 4:hours, 3:minutes,
          2:seconds, 1:hundredths)
```

Using the Statement

The *unit* (parameter 5) indicates the units of the source field. The result is returned in the destination field.

When you add “months” to a date, the destination “day” does not change unless the resulting date is beyond the end of the month. In other words, if you add one month to February 28, you get March 28; however, if you add one month to January 31, you get February 28 (except in leap years, when you get February 29). When you add “years” to a date, the destination month and day do not change, even in leap years.

Restrictions

The destination field must be a date/time field type and the source field must be a numeric field type. Also, results can be unpredictable if you are using a date field without a century component. Confirm that your System Administrator has set up the *Pivot Year* environment variable to avoid unexpected results.

Execution Error

If the destination field is null, a runtime error occurs.

Related Statements

DATE BTW

Example

The following example calculates a discount date 10 days in the future and displays it on an image:

```
SET      TAR WORK DISCOUNT DATE         =  TAR INVOICE DUE DATE
DATE ADD TAR WORK DISCOUNT DATE         5      10
DISPLAY  TAR WORK DISCOUNT DATE         (AT APPEARANCE #    )
```

> Note that the INVOICE DUE DATE field is moved into a work field called DISCOUNT DATE before the computation is performed so that the INVOICE DUE DATE field is not modified.

DATE BTW

The DATE BTW statement calculates the number of date/time units (years, months, days), between two date/time fields. It can be used to generate aging categories for reports, by comparing a value for date due with the current date to determine which customers have balances of less than 30 days, 30 to 60 days, over 60 days, and so forth.

```
••••• DATE BTW ••• ••••••••••••••••••••• ••• •  ••• ••••••••••••••••••••• •••
(1)            (2) (3)                    (4)(5) (6) (7)                    (8)
```

(1) T/F execution conditions

(2) From application ID

(6) To application ID

(7) To field name, PDF, or constant

(3) From field name or PDF	(8) To occurrence (constant/index)
(4) From occurrence (constant/index)	

```
(5) Unit (8:centuries, 7:years, 6:months, 5:days, 4:hours, 3:minutes,
          2:seconds, 1:hundredths)
```

Using the Statement

The *unit* (parameter 5) indicates the units of the result of the calculation. The result is returned in the predefined field BETWEEN.

If the “from” date is greater than the “to” date, BETWEEN is negative. If the “from” date is less than the “to” date, BETWEEN is positive.

When you use “months” in **DATE BTW**, the result is zero if the “from” month and year are the same as the “to” month and year. In other words, if “from” is February 1, 2000, and “to” is February 28, 2000, BETWEEN is zero. If the “from” month is different from the “to” month, BETWEEN is zero unless there is at least a 28-day difference in the two dates. In other words, January 31 to February 5 is zero months, but January 31 to February 28 -- or January 1 to February 1 -- is one month.

Restrictions

Both fields in the statement must be date/time field types. Also, results can be unpredictable if you are using a date field without a century component. Confirm that your System Administrator has set up the *Pivot Year* environment variable to avoid unexpected results.

Execution Error

If either date is null, a runtime error occurs.

Related PDFs

BETWEEN

Related Statements

DATE ADD

Example

The following example uses the DATE BTW statement to compare the date due with the current date to determine categories of aging for an accounts receivable report:

```
          SET DATE TAR WORK CURRENT DATE
          DATE BTW TAR INVOICE DUE DATE           5  TAR WORK CURRENT DATE
          IF       --- BETWEEN                    LT     30
    T     SET      TAR WORK CURRENT AMOUNT        =  TAR INVOICE AMOUNT
```

```
    F     IF       --- BETWEEN                    LT     60
    FT    SET      TAR WORK OVER 30 DAYS          =  TAR INVOICE AMOUNT
    FF    SET      TAR WORK OVER 60 DAYS          =  TAR INVOICE AMOUNT
```

DELETE

The DELETE statement deletes the current record from a specified file.

```
    ••••• DELETE   ••• ••••••••
    (1)            (2) (3)
```

```
(1) T/F execution conditions

(2) Application ID

(3) File name
```

Using the Statement

For process types other than input, the current process control file record is implicitly read and held if a REWRITE or DELETE statement exists for the file in any event point in the process; thus, an explicit read with hold is generally not necessary. If, however, another file is read and held within the event point that contains the REWRITE or DELETE, and this READ occurs before the REWRITE or DELETE, the operating system loses the "hold" on the process control file record. In this case, the record must then be explicitly reread and held or the rewrite/delete fails.

> Note that care should be taken when deleting the current process control file record in an input process. The normal file processing for the process control file assumes that the record still exists when it needs to rewrite or delete the record.

Restrictions

The record you want to delete from a specified file must have previously been read with the *Hold?* specification set to *1* (hold) or *2* (recover), or the delete must be done on the record currently being held in the process control file.

For input processes, a record from the process control file must be explicitly held by a statement to delete the record.

The DELETE statement cannot be used with sequential or one-record files.

Execution Errors

In an input process, if a record is rewritten or deleted with ILF statements, a non-recoverable runtime error occurs if the input processor also attempts to rewrite or delete the record.

The DELETE statement does not modify the current true/false status indicators. If the deletion fails, a non-recoverable runtime error occurs and the process will be canceled.

Related Statements

BEG READ, END READ, READ, READNEXT, REWRITE

Example

The following example reads the CUSTOMER file and, if the customer record exists (if the read succeeds), the routine deletes that customer record.

```
          READ     TAR CUSTOMER              HOLD 1 FT 0 BY CUSTOMER NO
    T     DELETE   TAR CUSTOMER
```

DIM

DIM displays an item with low intensity.

```
    ••••• DIM      ••• ••••••••••••••••••••• ••• (AT APPEARANCE # •••)
    (1)            (2) (3)                  (4)                  (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or predefined field

(4) Occurrence (constant/index)

(5) Appearance (constant/index)
```

Using the Statement

The appearance number can be used if the same field appears more than once on the image.

If the application ID, field name, occurrence number, and appearance number are not specified (left blank), the **DIM** statement affects all of the items painted on the associated image.

DIM is one of the *characteristic-type* statements. Refer to the common attributes of all the statements in this category in the Characteristic-Type Statements section of this manual.

The operation of a **DIM** statement affects only the next generation of an image. In other words, the item characteristics are reset each time an image is produced.

Restrictions

> Note that because characteristic statements operate on input items, they are only meaningful in event points that are associated with an input image.

DIM is a terminal display attribute only and, therefore, is not allowed on images in output processes.

Statement Ignored

The system ignores any **DIM** statements specified in non-image-related event points.

Related Statements

BLANK, BLINK, BOLD, BRIGHT, COLOR, DISPLAY, FONT, INVERSE, NO BOLD, NO ULINE, NORMAL, PICTURE, ULINE

Example

In this example, the field CUSTOMER CREDIT STATUS is made dim on the image.

```
        IF       TAR CUSTOMER CREDIT STATUS     NE     BAD
   T    DIM      TAR CUSTOMER CREDIT STATUS     (AT APPEARANCE #    )
        DISPLAY  TAR CUSTOMER CREDIT STATUS     (AT APPEARANCE #    )
```

DISABLE

DISABLE is used to conditionally make a button unavailable to users in input processes or on menus. This can be used to deactivate buttons when certain conditions or data entered dictate.

```
    ••••• DISABLE  ••• ••••••••••••••••••••• ••• (AT APPEARANCE # •••)
    (1)            (2) (3)                   (4)                   (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or button reference name

(4) Occurrence (constant/index)

(5) Appearance (constant/index)
```

Using the Statement

DISABLE is used in input or menu processes to make a button unavailable for user selection. This can be used for buttons that are conditionally unavailable depending on prior entries, user qualities, databases, interactive phase, and so forth.

The application ID must be left blank if a button reference name (assigned when the button is painted on the image) is used. Otherwise, use an application ID and field name that contains the name of an alpha field containing the button reference name.

Restrictions

DISABLE statements are only valid in image-related event points.

Related Statements

ENABLE

Example

The following example disables the SUBMIT ORDER button if a valid approval code has not been entered.

```
          IF       TOE APPROVAL CODE VALID        NE     1
    T     DISABLE      SUBMIT ORDER               (AT APPEARANCE #    )
```

DISPLAY

The DISPLAY statement converts the contents of the field associated with the specified item(s) from internal to external format and displays them to the user.

```
    ••••• DISPLAY  ••• •••••••••••••••••••• ••• (AT APPEARANCE # •••)
    (1)            (2) (3)                  (4)                  (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or predefined field

(4) Occurrence (constant/index)

(5) Appearance (constant/index)
```

Using the Statement

The appearance number can be used if the same field appears more than once on an image. If the application ID, field name, occurrence number, and appearance number are not specified (left blank), the DISPLAY statement affects all of the items painted on the associated image.

DISPLAY is one of the *characteristic-type* statements. Refer to the common attributes of all the statements in this category in the Characteristic-Type Statements section of this manual.

The operation of a DISPLAY statement affects only the next generation of an image. In other words, the item characteristics are reset each time an image is produced,

Restrictions

> Note that, because the **DISPLAY** statement operates on items rather than fields, this statement is meaningful only in event points that are associated with an image.

Statement Ignored

The system ignores any **DISPLAY** statements specified in non-image-related event points.

Related Statements

BLANK, BLINK, BOLD, BRIGHT, COLOR, DIM, FONT, INVERSE, NO BOLD, NO ULINE, NORMAL, PICTURE, ULINE

Example

The DISPLAY statement is used to cause the current contents of a field to appear on the image. In this example, the customer's credit status displays as *BAD* if the customer's balance exceeds the credit limit. If it does not, *OK* displays.

```
       IF       TAR CUSTOMER BALANCE          GT TAR CUSTOMER CREDIT LIMIT
T      SET      TAR CUSTOMER CREDIT STATUS    =      BAD
F      SET      TAR CUSTOMER CREDIT STATUS    =      OK
       DISPLAY  TAR CUSTOMER CREDIT STATUS    (AT APPEARANCE #    )
```

ENABLE

ENABLE is used to conditionally make a button available to users in input processes or on menus. This can be used to activate buttons when certain conditions or data entered dictate.

```
••••• ENABLE   ••• ••••••••••••••••••••• ••• (AT APPEARANCE # •••)
(1)            (2) (3)                   (4)                  (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or button reference name

(4) Occurrence (constant/index)

(5) Appearance (constant/index)
```

Using the Statement

ENABLE is used in input or menu processes to make a button available for user selection. This can be used for buttons that are conditionally available depending on prior entries, user qualities, databases, interactive phase, and so forth.

The application ID must be left blank if a button reference name (assigned when the button is painted on the image) is used. Otherwise, use an application ID and field name that contains the name of an alpha field containing the button reference name.

Restrictions

ENABLE statements are only valid in image-related event points.

Related Statements

DISABLE

Example

The following example enables the SUBMIT ORDER button only if a valid approval code has already been entered.

```
          IF       TOE APPROVAL CODE VALID        EQ     1
    T     ENABLE       SUBMIT ORDER               (AT APPEARANCE #    )
```

END

The END statement terminates the ILF routine that it is contained in, and returns control to APPX.

```
    ••••• END
    (1)
```

```
(1) T/F execution conditions
```

Using the Statement

Although an END statement can be used at any point in the ILF routine, it is not necessary at the end of an event point.

Related Statements

CANCEL, END PROC, EXIT, LOGOUT

Example

In this example, if the customer has a zero balance, the ILF routine terminates and control is returned to APPX.

```
        IF      TAR CUSTOMER BALANCE          EQ     0
T       END
```

END AT

The END AT statement is used in conjunction with a BEG READ/END READ loop, or a READNEXT statement, and may optionally be used in conjunction with a BEG AT statement. END AT specifies the ending key value for a range of records to be read.

```
••••• END AT   ••• •••••••• •• ••• ••••••••••••••••••••• •••
(1)            (2) (3)      (4)(5) (6)                    (7)
```

(1) T/F execution conditions	(5) Field application ID
(2) File application ID	(6) Field name
(3) File name	(7) Occurrence (constant/index)
(4) Relation (IN,EX)	

Using the Statement

The key used to access records in the file is specified in the subsequent BEG READ or READNEXT statement if the file is an indexed file. If the file is sequential, the key field for BEG READ or READNEXT must be blank, and the field used in the END AT statement must be numeric.

The END AT statement specifies an application ID and file name to identify the file upon which the range is being specified, and which is read in the subsequent BEG READ or READNEXT. The field specified by END AT must contain the ending value of the key of the last record that is to be read or, if the file is a sequential file, the record number of the last record to be read.

A relation is specified to indicate whether to end at a record with a key that includes (IN) or excludes (EX) the specified value.

END AT temporarily saves the contents of the specified field and, if the file is indexed, its length. If this length is less than the key length used in a subsequent BEG READ or READNEXT statement, only the leading portion of the key is constrained. In other words, the field value is padded with HEX FFs if the relation is IN, or with HEX 00s if the relation is EX. If this length is longer than the key specified in the subsequent BEG READ or READNEXT, the extra characters are ignored.

Since END AT saves the contents of the specified field regardless of field type or format, the field must exactly match the field type and format of the field used to access the file in a subsequent BEG READ or READNEXT statement. For example, if the key in the subsequent BEG READ statement is a binary numeric field with nine digits, you should not use a seven-digit numeric field or a packed decimal numeric field in the END AT statement to set the starting value. Again, this applies only to indexed files; sequential files can specify any valid record number.

> Note that any field, including non-key fields, can be used with the END AT statement. This statement uses the contents of the field as an end marker. Unless the file is sequential, constant fields are assumed to be alphanumeric; for sequential files, constants are assumed to be numeric. The length of the alphanumeric constant is determined by discarding all trailing blanks; hence, an all-blank constant results in a length of zero and causes a subsequent BEG READ or READNEXT to include all of the records through the end of the file, regardless of the relation (IN or EX).

Restrictions

If an END AT is not executed before a BEG READ statement, records are read through the end of the file. If an END AT is not executed before a READNEXT statement, the READNEXT is successful unless an end-of-file condition is encountered.

If an END AT is followed by a BEG AT for the same file, the ending key value is lost.

The END AT statement cannot be used with one-record files.

Related Statements

BEG AT, BEG READ, END READ, READNEXT

Example

The following example uses END AT (with BEG AT) to specify a range of records to be read in the file specified in the subsequent BEG READ statement. The example includes a transaction file of records to be posted to the master (CUSTOMER) file. The first part of the key to the transaction file is the customer ID. You want to restrict the reading of the transaction file to only the range of record keys that includes a specific customer ID.

```
BEG AT   TAR TRANFILE IN TAR CUSTOMER ID
END AT   TAR TRANFILE IN TAR CUSTOMER ID
BEG READ TAR TRANFILE              HOLD 0 KEY IS  TRANFILE KEY
*
```

```
          *        (Process each TRANFILE record for specified customer)
          *
          END READ TAR TRANFILE
```

The preceding example processes records on the TRANFILE file with a key value within the range specified by the BEG AT/END AT pair of statements. If the END AT is not included, the processing continues until the end of the file is reached. When used with BEG READ, and there are no records on the file that meet the specified ending range, no statements within the BEG READ, END READ are executed.

The next example shows END AT used preceding a READNEXT statement:

```
          BEG AT   TAR ORDER    IN TAR TRANFILE CUSTOMER ID
          END AT   TAR ORDER    IN TAR TRANFILE CUSTOMER ID
          LABEL    :NEXT ORDER RECORD
          READNEXT TAR ORDER                  HOLD 0 FT 0 BY ORDER CUST ID
    T     ...      (Process each order record for specified customer)
    T     GOTO     :NEXT ORDER RECORD
          *
          *        (Terminate order processing for specified customer)
```

In this example, if the ORDER file contains one or more records matching the specified customer ID, the true/false status indicator is set to *T* by the READNEXT statement.

END LOOP

The **END LOOP** statement marks the end of loop execution.

```
    ••••• END LOOP ••
    (1)            (2)
```

```
(1) T/F execution conditions

(2) Loop index (A to Z, AI to ZI)
```

Using the Statement

The parameter entered is a predefined field A to Z or AI to ZI that corresponds to the index counter specified in the matching BEG LOOP statement.

If the index counter plus the step value is outside the “to” range set in the BEG LOOP statement, the index counter is left unchanged, and execution continues with the statement following the **END LOOP**.

Otherwise, the index counter is incremented by the step value, and execution continues with the statement immediately following the corresponding BEG LOOP statement.

Restrictions

A BEG LOOP statement must come before the **END LOOP** statement. There must be an **END LOOP** statement for each BEG LOOP statement. See the BEG LOOP section for additional information.

Execution Errors

An error occurs if a BEG LOOP statement does not get executed.

Related Statements

BEG LOOP

Example

The following example is a simple loop that increments the current monthly budgets by 5 percent for all 12 months:

```
          BEG LOOP AI = 001 TO 012  STEP 001
          COMPUTE  TGL BUDGET AMOUNT          AI  *      1.05
          END LOOP AI
```

After the twelfth repetition (where AI=12), control is passed to the next statement following the END LOOP statement.

The next example presents a nested loop that repeats its processing logic within each occurrence of the main loop. This loop also incorporates a conditional check to see if the value of field C equals that of field D, at which point the internal loop terminates and returns control to the main loop.

```
          BEG LOOP A  = 001 TO 010  STEP 001
          *
          *        (Perform processing if needed)
          *
          BEG LOOP B  = 001 TO 005  STEP 001
          *
          *        (Perform processing)
          *
          IF       --- C                          EQ --- D
    T     SET      --- B                          =      5
          END LOOP B
          *
          END LOOP A
```

There can be more BEG LOOP statements in an ILF routine than END LOOP statements. For example:

```
    T     BEG LOOP A  = 001 TO 010  STEP 001
    F     BEG LOOP A  = 001 TO 005  STEP 001
          *
          *        (Perform processing)
```

```
          *
          END LOOP A
```

In the preceding example, only one of the BEG LOOP statements can execute.

END PROC

The END PROC statement allows you to end the current process and return control to APPX. For example, after a child process has an END PROC statement executed, control may be returned to the parent process. The END PROC statement has the same effect as a user selecting the `END` option while in a process.

```
    ••••• END PROC
    (1)
```

```
(1) T/F execution conditions
```

Using the Statement

There are no restrictions as to where you can insert the **END PROC** statement within a process. You can control if it is to be executed by setting appropriate true/false execution conditions.

Related Statements

CANCEL, END, EXIT, LOGOUT

Example

In this example, if the customer has a zero balance, the current process terminates and control is returned to APPX.

```
          IF       TAR CUSTOMER BALANCE           EQ     0
    T     END PROC
```

END READ

The **END READ** statement indicates the end of a BEG READ "Read Next Record" loop.

```
    ••••• END READ ••• ••••••••••••••••••••
    (1)            (2) (3)
```

```
(1) T/F execution conditions

(2) Application ID
```

```
(3) File name (or field name for RDBMS)
```

Using the Statement

When using this statement in conjunction with an RDBMS table, you can specify a field name instead of a file name to limit the volume of data being passed between APPX and the RDBMS. See the Partial-Record I/O section in Chapter 4-3: Specifying Statements for more information.

The **END READ** statement acts as a GOTO and a placeholder; the **END READ** "goes to" the statement after the corresponding BEG READ. The **END READ** statement is also used by the BEG READ statement as a placeholder, to know where to continue execution when the loop is finished.

Restrictions

The application ID and file name must match those specified in a previous BEG READ statement.

The **END READ** statement cannot be used with one-record files.

Execution Errors

A non-recoverable file processing error occurs if an **END READ** statement executes without, or before, the corresponding BEG READ statement. A non-recoverable error also occurs if, once loop termination has taken place, the **END READ** is re-executed without or before another corresponding BEG READ statement. All loops are considered terminated when an event point ends.

Related Statements

BEG AT, BEG READ, END AT, READNEXT

Example

The following example demonstrates a common use of the END READ statement where a range is specified:

```
          BEG AT   TAR TRANFILE IN TAR CUSTOMER ID
          END AT   TAR TRANFILE IN TAR CUSTOMER ID
          BEG READ TAR TRANFILE               HOLD 0 KEY IS  TRANFILE KEY
          *
          *        (process each TRANFILE record for specified customer)
          *
          END READ TAR TRANFILE
```

In this example, the system reads the TRANFILE file, beginning with the first record where the TRANFILE CUSTOMER ID is equal to or greater than the first one found in the TRANFILE (TRANFILE CUSTOMER ID must be the first part of TRANFILE KEY). The BEG READ "Read Next Record" loop terminates when the key in TRANFILE exceeds the one specified in the END AT statement. If no BEG AT is present, the read begins with the first record in the file.

ERROR

ERROR can be used in any event point in an input process to display an error message, or to print an error message in an output or action process.

```
••••• ERROR    ••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••
(1)            (2)
```

```
(1) T/F execution conditions

(2) Error message (constant or TEMPn predefined field)
```

Using the Statement

Variable error messages can be displayed or printed by setting the predefined field TEMPn to the required message and entering TEMPn in place of a message in the ERROR statement.

When an error occurs, the image is redisplayed to the user with the error message blinking on line two of the status surface, and the terminal beeps. If more than one ERROR, WARNING, or MESSAGE statement is executed, the user can display the additional messages. The user must correct the problem before processing can continue with the next image.

In output processes, errors are printed on the report; in update, status, and subroutine processes errors are sent to the associated log. Each time an ERROR statement executes, the value of the predefined field ERRORS is incremented by one.

Related PDFs

ERRORS

Related Statements

WARNING, MESSAGE

Example

In this example, an ERROR statement followed by a BLINK statement are used to alert a user that the customer has exceeded their credit limit.

```
          IF       TAR CUSTOMER BALANCE           GT TAR CUSTOMER CREDIT LIMIT
    T     ERROR    Customer's balance exceeds credit limit!
    T     BLINK    TAR CUSTOMER BALANCE
```

EXIT

The EXIT statement ends the current subprocess family and returns control to the previous related or detached process.

```
••••• EXIT
(1)
```

```
(1) T/F execution conditions
```

Using the Statement

EXIT is used to end a subprocess family. There are no restrictions as to where you can insert the EXIT statement within a process. You can control if it is to be executed by setting appropriate true/false execution conditions.

Related Statements

CANCEL, END, END PROC, LOGOUT

Example

In this example, if the customer has a zero balance, the current subprocess family terminates and control is returned to the previous related or detached process.

```
      IF       TAR CUSTOMER BALANCE           EQ     0
T     EXIT
```

FONT

The FONT statement changes a field's font based on the contents of the predefined field FONT. All fields displayed on the screen can be affected by using the global FONT statement as shown below, or a specific field's font can be changed by identifying the field.

```
      SET      --- FONT                       =      COURIER
••••• FONT     ••• ••••••••••••••••••••• ••• (AT APPEARANCE # •••)
(1)            (2) (3)                    (4)                  (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or predefined field

(4) Occurrence (constant/index)
```

```
(5) Appearance (constant/index)
```

Using the Statement

The appearance number can be used if the same field appears more than once on an image. If the application ID, field name, occurrence number, and appearance number are not specified (left blank), the FONT statement affects all of the items painted on the associated image using either the default font or the font specified in the predefined field FONT.

The operation of a FONT statement affects only the next generation of an image. In other words, the item characteristics are reset each time an image is produced,

Restrictions

> Note that, because the **FONT** statement operates on items rather than fields, this statement is meaningful only in event points that are associated with an image.

Statement Ignored

The system ignores any **FONT** statements specified in non-image-related event points.

Related Statements

BLANK, BLINK, BOLD, BRIGHT, COLOR, DIM, DISPLAY, INVERSE, NO BOLD, NO ULINE, NORMAL, PICTURE, SET, ULINE

Example

The FONT statement is used to cause the current font on a field to change on the image. In this example, the customer's credit status displays in Courier font while all other fields are displayed in the default font designated by the System Administrator.

```
          SET      --- FONT                        =      COURIER
          FONT     TAR CUSTOMER CREDIT STATUS     (AT APPEARANCE #    )
          DISPLAY  TAR CUSTOMER CREDIT STATUS     (AT APPEARANCE #    )
```

GOSUB

The GOSUB statement transfers processing control to the statement immediately following the referenced label or subroutine. Execution of a subsequent RETURN statement transfers control to the statement following the GOSUB.

```
    ••••• GOSUB    ••• ••••••••••••••••••••••••••••••
    (1)            (2) (3)
```

```
(1) T/F execution conditions

(2) Application ID (optional)

(3) Label or subroutine name
```

Sets True/False Status Indicator

The **GOSUB** statement stores the current values of the internal true/false status indicators, clears the values before executing the subroutine, and restores them upon returning from the subroutine. In addition, **when implemented**, GOSUB will set the next status indicator to the value indicated in the RETURN statement, if any. (Note: when designing subroutines, be sure that no true/false levels are expected upon entering the subroutine.)

Using the Statement

The **GOSUB** statement specifies a label name or subroutine process. The **GOSUB** statement first tries to locate a corresponding LABEL within the event point being executed. If no such LABEL exists, APPX attempts to locate a subroutine process with that name in the specified application. If the application ID in the **GOSUB** statement is blank, it is assumed to be the current application.

If the GOSUB reference is to a label within the event point, the application ID should be blank or the current application.

If the reference is to a subroutine process, the following statements are generated automatically at process compile time, at the end of the event point:

```
          END
          LABEL    (label from GOSUB statement)
          COPY     (application ID and label from GOSUB statement)
          RETURN
```

This allows subroutine processes to be invoked by GOSUB as well as by COPY or SUBR.

Passing Values

The GOSUB statement passes the current PASS list, if any. Refer to the PASS Lists section for details.

Compilation Errors

If there is no corresponding LABEL and no subroutine process with the name identified in the **GOSUB** for the referenced application, and if the referenced application is a valid application in System Administration, a non-recoverable error occurs at runtime.

Statement Ignored

If there is no corresponding LABEL and the application ID refers to a nonexistent application, the statement is ignored.

Related Statements

COPY, END, GOTO, LABEL, PASS, RECEIVE, RETURN, SUBR

Example

```
          *
          *        On Account or Regular posting?
          *
          IF       TAR RECEIPT3 APPLY TYPE        NE     1
    T     GOSUB        :REGULAR POST
    F     GOSUB        :ON ACCOUNT POST
          *
          END
          *
          LABEL    :REGULAR POST
          *        (perform regular posting)
          RETURN
          *
          LABEL    :ON ACCOUNT POST
          *        (perform on account posting)
          RETURN
```

Notice that, in this example, the subroutine does not use the values of the true/false status indicators established before the GOSUB. Instead, APPX stores the current values of those indicators, executes the subroutine, and then restores the original values upon completion of the subroutine. This allows you to have the use of the full range of true/false values within a subroutine.

GOTO

The **GOTO** statement transfers processing control to the statement immediately following the referenced label.

```
    ••••• GOTO     ••••••••••••••••••••••••••••
    (1)            (2)
```

```
(1) T/F execution conditions

(2) Label
```

Using the Statement

The LABEL to which the **GOTO** statement transfers processing control can be anywhere in the same event point as the **GOTO**.

Compilation Errors

A process compile error occurs if the referenced LABEL is not defined within the same event point as the **GOTO** statement, and also not defined in any ILF subroutines copied into the event point with the COPY statement.

Related Statements

COPY, GOSUB, LABEL, OVERFLOW, RETURN, TIMEOUT

Example

In the following example, “on account” payments are processed by invoice until no more payment amount is left to apply.

```
        LABEL    :GET NEXT APPLY TO
        *        any more to apply?
        IF       TAR WORK ON ACCOUNT AMOUNT     EQ     0
    T   GOTO     :END PROCESS ON ACCOUNT
        *        (process 'on account' payment)
        GOTO     :GET NEXT APPLY TO
        *
        LABEL    :END PROCESS ON ACCOUNT
        *        (perform other processing)
```

IF

The **IF** statement compares the contents of two fields, or compares the contents of a field with a constant, and sets the next level of the internal true/false flag.

```
••••• IF      ••• •••••••••••••••••••• ••• •• ••• •••••••••••••••••••• •••
(1)           (2) (3)                   (4) (5)(6) (7)                   (8)
```

(1) T/F execution conditions	(5) Relation (LT,GT,LE,GE,EQ,NE,IN,EX)
(2) 1st application ID	(6) 2nd application ID
(3) 1st field name or PDF	(7) 2nd field name, PDF, or constant
(4) 1st occurrence (constant/index)	(8) 2nd occurrence (constant/index)

Sets True/False Status Indicator

In comparing the contents of two fields, or a field with a constant, the next level of the true/false status indicators is set to *T* if the condition is true, or *F* if the condition is false. See Chapter 4-2: True/False Status Indicators for details on using these indicators.

Using the Statement

The **IF** statement specifies a relationship between two values. The valid relational operators are:

- less than (LT), greater than (GT), less than or equal to (LE), greater than or equal to (GE)
- equals (EQ), not equal to (NE), includes (IN), and excludes (EX).

The relation *IN* means "includes". The relation is true if the value in the second field is included in the first field, and false otherwise. If the relation is true, the predefined field TEXT AT POSITION is set to the starting position of the second field within the first field. The first field must be an alpha field; the second can be either an alpha field or a constant. Trailing blanks are ignored if the second field is a constant; however, you may specify fixed spaces (by entering underscores) to include one or more trailing spaces as part of a character string that is being searched.

Restrictions

The **IF** statement supports comparisons between like data types only (alpha-to-alpha, numeric-to-numeric, date-to-date, logic-to-logic). When comparing two date fields, results can be unpredictable if you are using a date field without a century component. Confirm that your System Administrator has set up the *Pivot Year* environment variable to avoid unexpected results.

Related PDFs

TEXT AT POSITION

Related Statements

AND, OR

Example

In this example, if the customer's balance is greater than his credit limit, an error message is given and the balance field blinks; otherwise, no action is taken.

```
          IF       TAR CUSTOMER BALANCE           GT TAR CUSTOMER CREDIT LIMIT
    T     ERROR    Customer's balance exceeds credit limit!
    T     BLINK    TAR CUSTOMER BALANCE
```

IF DIFF

The IF DIFF statement compares the current contents of a record (file name), field, group, or predefined field with its stored, default, or original value.

```
••••• IF DIFF  ••• •••••••••••••••••••• ••• ••••••••••••••
(1)            (2) (3)                  (4) (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) File name, field name, or predefined field

(4) Occurrence (constant/index)

(5) Comparison type (STORE, DEFAULT, or ORIGINAL)
```

Sets True/False Status Indicator

If the two values compared are different, the next level of the true/false status indicators is set to *T; otherwise, it is set to F.* See *Chapter 4-2: True/False Status Indicators* for details on using these indicators.

Using the Statement

The IF DIFF statement allows you to determine if the contents of the operand have changed from the comparison type value. The operand can identify a record (file name), an individual field or a group (field name), or a PDF. Valid comparison types are STORE, DEFAULT, and ORIGINAL and can be selected via Scan.. IF DIFF will compare the current value of the operand against the STORE/RESTORE area (STORE), the default record (DEFAULT), or the original (on-disk) buffer (ORIGINAL).

ORIGINAL is only valid for the PCF since APPX does not store the original on-disk values for other files (see notes below). To make it easier to use IF DIFF in a subroutine, you can omit the operand name if the comparison type is ORIGINAL (since that implies the PCF record).

IF DIFF will also populate the --- COMPARE memory file with a list of differences. If no differences are found, this file will be empty. If differences are found, the COMPARE file will hold one record for each changed field, composed of the following fields:

COMPARE FIELD NAME	X(22), primary key
COMPARE DESCRIPTION	X(30)
COMPARE CURRENT VALUE	X(80)
COMPARE OTHER VALUE	X(80)
COMPARE OFFSET	9(5)
COMPARE LENGTH	9(5)

The COMPARE FIELD NAME field will contain the name of the changed field. COMPARE DESCRIPTION will contain the descriptive name of the changed field. COMPARE CURRENT VALUE will contain the value of the field in the current record buffer, as converted by CNV TEXT. COMPARE OTHER VALUE will contain the value of the field in the other record buffer (store, default, or original) as converted by CNV TEXT. The COMPARE OFFSET and COMPARE LENGTH fields will contain the offset of the field within the record buffer and the length of the field, respectively

The following notes may provide a better understanding of how to use the ORIGINAL comparison type in this statement:

1. Any process that has a PCF can compare to the original-record buffer.
2. Since file-level event points have a PCF, they can compare to the original-record buffer.
3. The original-record buffer is now shared between the file-level event points and the foreground process. Prior to the implementation of the IF DIFF statement in Release 4.2, you could modify a record in the file-level Post Read event point and the foreground input process would not notice that the record had changed.
4. For file-level event points, each stream maintains its own original-record buffer.
5. For release 4.2, APPX only keeps an original-record buffer for PCF files. However, this restriction may be lifted in a future release.
6. The original-record buffer is managed in the manner described below:

1. In Add mode, the default-record buffer is copied to the original-record buffer just before the Set Run-Time Defaults event point.
2. In Inquire, Delete, or Change mode, the original-record buffer is copied from disk just before the Post PCF Read event point.
3. For file-level event points, the original-record buffer is copied from disk just before the Post Read event point.
4. In Change mode, the current-record is copied to the original-record buffer just before the Post PCF Update event point.*
5. In Add mode, the current-record is copied to the original-record buffer just before the Post PCF Update event point.*

*Note: For process types other than Input, APPX never updates the PCF automatically, so rules 6.d and 6.e only apply to Input processes.

IF EXIST

The **IF EXIST** statement determines the existence of a specified file.

```
••••• IF EXIST ••• ••••••••            FAIL • CACHE? •
(1)            (2) (3)                    (4)      (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) File name

(4) Fail action (0=nothing, 1=warning, 2=error, 3=cancel)

(5) Cache? (Y=cache version, N=disk version)
```

Sets True/False Status Indicator

The next true/false status indicator is set to *T* if the file exists, and to *F* if it does not. If the file does not currently exist, the *FAIL* parameter determines what additional action to take. In addition, on a false condition, the predefined field STATUS CODE returns a token indicating the reason for an I/O command failure. This token can be used as a key to read the system message file to obtain a descriptive I/O error message (see STATUS CODE for details).

Using the Statement

The application ID and file name to locate are specified in the statement. Additional information that can affect the location of the file may be contained in the predefined fields DATABASE or NEXT DATABASE.

The file name references a file that is defined in the data dictionary of the specified application ID. You can set the NEXT DATABASE predefined field to the database ID of the database in which the file should be located.

If NEXT DATABASE is blank, the DATABASE predefined field is used. The NEXT DATABASE field is set to blank by the IF EXIST statement, regardless of whether or not the file was successfully located.

Execution Errors

Serious problems encountered while trying to open the file, such as a file structure/data dictionary mismatch, cause a non-recoverable runtime error to occur, regardless of the *FAIL* parameter.

Related PDFs

DATABASE, NEXT DATABASE

Related Statements

CLOSE, CREATE, OPEN, SCRATCH

Example

In the example below, the General Ledger Parameters file is accessed to determine if it exists. If it does not exist, an error is displayed or printed; if it does exist, the record is read; and if the record does not exist, an error is displayed or printed.

```
        IF       TAR PARAM KEEP HIST CALEND     EQ     0
    T   IF EXIST TGL PARAM                FAIL 0 CACHE? N
    TF  ERROR    General Ledger Parameters File Does Not Exist
    TT  READ     TGL PARAM                   HOLD 0 FT 0 BY
    TTF ERROR    General Ledger Parameters Record Not on File
```

INPUT

The INPUT statement allows you to alter the flow of processing by invoking an APPX input process from within an ILF routine.

```
    ••••• INPUT    ••• ••••••••••••••••••••••••••••••   ••••••••••   END? •  FAIL •
    (1)            (2) (3)                              (4)              (5)     (6)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Input process name

(4) Invocation type (SUBPROCESS, RELATED, DETACHED)

(5) End current process first? (Y/N)

(6) Fail action (0=nothing, 1=warning, 2=error, 3=cancel)
```

Sets True/False Status Indicator

The next status indicator is set to *T* if the process is invoked successfully, and to *F*, otherwise. If the process cannot be invoked successfully, the *FAIL* parameter determines what additional action to take.

Using the Statement

In addition to the application ID and input process name to be invoked, you specify the invocation type (subprocess, related, or detached). As with all other child invocations, this parameter controls the level of data sharing between the parent and the input process that is invoked.

If *END?* is set to *Y*, the current process will end before invoking the input process. The input process will return to the parent of the current process when it is finished. If *END?* is set to *N*, the input process returns to the current process (beginning with the first statement after the **INPUT** statement).

If the application ID is blank, the value in the predefined field NEXT APPLICATION is used. If the process name is blank, the value in the predefined field NEXT PROCESS NAME is used.

Passing Values

The INPUT statement passes the current PASS list, if any. Refer to the PASS Lists section for details.

Related PDFs

NEXT APPLICATION, NEXT PROCESS NAME

Related Statements

INQUIRY, JOB, MENU, OUTPUT, PASS, QUERY, STATUS, SUBR, UPDATE

Example

The following example checks to see if it is the last processing day in the month. If so, the **INPUT** statement is executed, invoking an end-of-month data entry operation.

```
          IF       TAR WORK END OF MONTH          EQ     Y
    T     INPUT    TAR END OF MONTH ENTRY               SUBPROCESS   END? Y  FAIL 3
```

INQUIRY

The INQUIRY statement allows you to alter the flow of processing by invoking an APPX inquiry process from within an ILF routine.

```
    ••••• INQUIRY  ••• ••••••••••••••••••••••••••••   ••••••••••   END? •  FAIL •
    (1)            (2) (3)                            (4)              (5)     (6)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Inquiry process name

(4) Invocation type (SUBPROCESS, RELATED, DETACHED)

(5) End current process first? (Y/N)

(6) Fail action (0=nothing, 1=warning, 2=error, 3=cancel)
```

Sets True/False Status Indicator

The next status indicator is set to *T* if the process is invoked successfully, and to *F*, otherwise. If the process cannot be invoked successfully, the *FAIL* parameter determines what additional action to take.

Using the Statement

In addition to the application ID and inquiry process name to be invoked, you specify the invocation type (subprocess, related, or detached). As with all other child invocations, this parameter controls the level of data sharing between the parent and the inquiry process that is invoked.

If *END?* is set to *Y*, the current process will end before invoking the inquiry process. The inquiry process will return to the parent of the current process when it is finished. If *END?* is set to *N*, the inquiry process returns to the current process (beginning with the first statement after the **INQUIRY** statement).

If the application ID is blank, the value in the predefined field NEXT APPLICATION is used. If the process name is blank, the value in the predefined field NEXT PROCESS NAME is used.

Passing Values

The INQUIRY statement passes the current PASS list, if any. Refer to the PASS Lists section for details.

Related PDFs

NEXT APPLICATION, NEXT PROCESS NAME

Related Statements

INPUT, JOB, MENU, OUTPUT, PASS, QUERY, STATUS, SUBR, UPDATE

Example

The following example checks to see if it is the last processing day in the month. If so, the **INQUIRY** statement is executed, invoking an end-of-month file inquiry operation.

```
          IF       TAR WORK END OF MONTH          EQ     Y
    T     INQUIRY  TAR END OF MONTH INQUIRY             SUBPROCESS   END? Y  FAIL 3
```

INVERSE

Items are usually displayed in normal video mode on an image. INVERSE displays an item in an inverse video mode.

```
    ••••• INVERSE  ••• ••••••••••••••••••••• ••• (AT APPEARANCE # •••)
    (1)            (2) (3)                   (4)                   (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or predefined field
```

```
(4) Occurrence (constant/index)

(5) Appearance (constant/index)
```

Using the Statement

The appearance number can be used if the same field appears more than once on the image.

If the application ID, field name, occurrence number, and appearance number are not specified (left blank), the INVERSE statement affects all of the items painted on the associated image.

INVERSE is one of the *characteristic-type* statements. Refer to the common attributes of all the statements in this category in the Characteristic-Type Statements section of this manual.

The operation of an INVERSE statement affects only the next generation of an image. In other words, the item characteristics are reset each time an image is produced.

Restrictions

> Note that because characteristic statements operate on items rather than fields, they are only meaningful in event points that are associated with an image.

INVERSE is a terminal display attribute only and therefore is not allowed on images in output processes.

Statement Ignored

The system ignores any INVERSE statements specified in non-image event points.

Related Statements

BLANK, BLINK, BOLD, BRIGHT, COLOR, DIM, DISPLAY, FONT, NO BOLD, NO ULINE, NORMAL, PICTURE, ULINE

Example

In the following example, if the customer's credit status is *BAD*, it displays in inverse video.

```
        IF       TAR CUSTOMER CREDIT STATUS     EQ     BAD
   T    INVERSE  TAR CUSTOMER CREDIT STATUS     (AT APPEARANCE #    )
        DISPLAY  TAR CUSTOMER CREDIT STATUS     (AT APPEARANCE #    )
```

JOB

The JOB statement allows you to alter the flow of processing by invoking an APPX job process from within an ILF routine.

```
    ••••• JOB      ••• ••••••••••••••••••••••••••••••   ••••••••••   END? •  FAIL •
    (1)            (2) (3)                              (4)              (5)     (6)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Job process name

(4) Invocation type (SUBPROCESS, RELATED, DETACHED)

(5) End current process first? (Y/N)

(6) Fail action (0=nothing, 1=warning, 2=error, 3=cancel)
```

Sets True/False Status Indicator

The next status indicator is set to *T* if the process is invoked successfully, and to *F*, otherwise. If the process cannot be invoked successfully, the *FAIL* parameter determines what additional action to take.

Using the Statement

In addition to the application ID and job process name to be invoked, you specify the invocation type (subprocess, related or detached). As with all other child invocations, this parameter controls the level of data sharing between the parent and the job process that is invoked.

If *END?* is set to *Y*, the current process will end before invoking the job process. The job process will return to the parent of the current process when it is finished. If *END?* is set to *N*, the job process returns to the current process (beginning with the first statement after the **JOB** statement).

If the application ID is blank, the value in the predefined field NEXT APPLICATION is used. If the process name is blank, the value in the predefined field NEXT PROCESS NAME is used.

Passing Values

The JOB statement passes the current PASS list, if any. Refer to the PASS Lists section for details.

Related PDFs

NEXT APPLICATION, NEXT PROCESS NAME

Related Statements

INPUT, INQUIRY, MENU, OUTPUT, PASS, QUERY, STATUS, SUBR, UPDATE

Example

The following example checks to see if it is the last processing day in the month. If so, the **JOB** statement is executed, invoking an end-of-month file update operation.

```
          IF       TAR WORK END OF MONTH          EQ     Y
    T     JOB      TAR END OF MONTH UPDATE              SUBPROCESS   END? Y  FAIL 3
```

LABEL

The **LABEL** statement defines a position in an ILF routine that may be referenced by GOTO, GOSUB, OVERFLOW, or TIMEOUT statements.

```
          LABEL    ..............................
                   (1)
```

```
(1) Label
```

Using the Statement

The LABEL statement defines a position in an ILF routine that may be referenced by a GOTO, GOSUB, OVERFLOW, or TIMEOUT statement. Each label must be unique within the event point that contains it.

Compilation Errors

A process compile error occurs if multiple labels of the same name are defined within the same event point or in any ILF subroutines copied into that event point with the COPY statement.

Related Statements

COPY, GOSUB, GOTO, OVERFLOW, TIMEOUT

Example

In the following example, “on account” payments are processed by invoice until no more payment amount is left to apply.

```
          LABEL    :GET NEXT APPLY TO
          *        any more to apply?
          IF       TAR WORK ON ACCOUNT AMOUNT     EQ     0
    T     GOTO     :END PROCESS ON ACCOUNT
          *        (process 'on account' payment)
          GOTO     :GET NEXT APPLY TO
          *
          LABEL    :END PROCESS ON ACCOUNT
          *        (perform other processing)
```

LOCAL

A new LOCAL statement allows fields to be declared to be LOCAL to the current context.

LOCAL saves the named field's value. When the current context completes, it restores the saved value before returning to the invoking context.

LOCAL works in subroutines called via a GOSUB, and in Start of Process event points.

Description:

A LOCAL statement is (nearly) equivalent to a RECEIVE statement that executes without a matching PASS statement (such a RECEIVE statement will set the next T/F to F, but a LOCAL statement doesn't change the T/F flags).

When you execute a LOCAL (or RECEIVE) statement, the engine decides the point in time that the original value should be restored according to the following rules:

1. If you execute a LOCAL (or RECEIVE) while there is a GOSUB active (in the current event point), the original value is restored when you execute a RETURN statement.
2. If there is no GOSUB active, the original value is restored when the event point ends (with one exception described in the next rule). That means that you can add LOCAL statements to the beginning of an event point to protect variables that you want to use in that event point.
3. If you execute a LOCAL statement (and there is no active GOSUB) in the Start of Process event point, you are protecting a variable for the life of the process (the original value is restored when the process ends).

LOGOUT

The **LOGOUT** statement causes APPX to end and returns control to the host operating system.

```
    ••••• LOGOUT   SYSTEM? •
    (1)                   (2)
```

```
(1) T/F execution conditions

(2) Log off operating system?
```

Using the Statement

If SYSTEM is specified as *Y*, the user is logged off of the system if the operating system provides this capability.

Related Statements

CANCEL, END, END PROC, EXIT

MARK TRX

The MARK TRX keyword is reserved for future use.

MENU

The MENU statement allows you to alter the flow of processing by invoking an APPX menu process from within an ILF routine.

```
••••• MENU     ••• ••••••••••••••••••••••••••••••   ••••••••••   END? •  FAIL •
(1)            (2) (3)                              (4)              (5)     (6)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Menu process name

(4) Invocation type (SUBPROCESS, RELATED, DETACHED)

(5) End current process first? (Y/N)

(6) Fail action (0=nothing, 1=warning, 2=error, 3=cancel)
```

Sets True/False Status Indicator

The next status indicator is set to *T* if the process is invoked successfully, and to *F*, otherwise. If the process cannot be invoked successfully, the *FAIL* parameter determines what additional action to take.

Using the Statement

In addition to the application ID and menu process name to be invoked, you specify the invocation type (subprocess, related, or detached). As with all other child invocations, this parameter controls the level of data sharing between the parent and the menu process that is invoked.

If *END?* is set to *Y*, the current process will end before invoking the menu process. The menu process will return to the parent of the current process when it is finished. If *END?* is set to *N,* the menu process returns to the current process (beginning with the first statement after the **MENU** statement).

If the application ID is blank, the value in the predefined field NEXT APPLICATION is used. If the process name is blank, the value in the predefined field NEXT PROCESS NAME is used.

If the application ID is supplied but the menu name is left blank, the menu is obtained from the Application Parameters file for the specified application.

If both the application ID and the menu name are blank, the application used is the Startup Application for the current database (specified in the Database file in System Administration), and the menu name is then obtained from the Application Parameters file for that application.

Passing Values

The MENU statement passes the current PASS list, if any. Refer to the PASS Lists section for details.

Related PDF's

NEXT APPLICATION, NEXT PROCESS NAME

Related Statements

INPUT, INQUIRY, JOB, OUTPUT, PASS, QUERY, STATUS, SUBR, UPDATE

Example

The following example checks to see if it is the last processing day in the month. If so, the **MENU** statement is executed, invoking an end-of-month file menu.

```
          IF       TAR WORK END OF MONTH          EQ     Y
    T     MENU     TAR END OF MONTH MENU                SUBPROCESS   END? Y  FAIL 3
```

MESSAGE

MESSAGE can be used to display a message in an input process, or to print a message in an output, update or status process.

```
    ••••• MESSAGE  ••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••
    (1)            (2)
```

```
(1) T/F execution conditions

(2) Message (constant or TEMPn predefined field)
```

Using the Statement

Variable messages can be displayed or printed by setting a predefined field TEMPn to the required message and entering *TEMPn* in place of a message in the **MESSAGE** statement.

When a message occurs, the image is redisplayed to the user with the message on line two of the status surface. If more than one ERROR, WARNING, or **MESSAGE** statement is executed, the user can display the additional messages.

In output processes, messages are printed on the report; in update, status, and subroutine processes they are printed to a log.

Related Statements

ERROR, WARNING

Example

In the example below, the application designer is advising the user to use a different transaction entry process to enter invoices for miscellaneous customers.

```
          MESSAGE  Use Standard Invoice Entry for Miscellaneous Customers!
```

NO BOLD

NO BOLD is used to print an item with normal print density (the reverse of BOLD).

```
    ••••• NO BOLD  ••• ••••••••••••••••••••• ••• (AT APPEARANCE # •••)
    (1)            (2) (3)                   (4)                   (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or predefined field

(4) Occurrence (constant/index)

(5) Appearance (constant/index)
```

Using the Statement

The appearance number can be used if the same field appears more than once on the image.

If the application ID, field name, occurrence number, and appearance number are not specified (left blank), the NO BOLD statement affects all of the items painted on the associated image.

NO BOLD is one of the *characteristic-type* statements. Refer to the common attributes of all the statements in this category in the Characteristic-Type Statements section of this manual.

The operation of a NO BOLD statement affects only the next generation of an image. In other words, the item characteristics are reset each time an image is produced.

Restrictions

> Note that, because characteristic statements operate on items rather than fields, these statements are meaningful only in event points that are associated with an image.

NO BOLD is a printer attribute only and therefore has no effect on images displayed on a terminal screen.

Statement Ignored

The system ignores any NO BOLD statements specified in non-image-related event points.

Related Statements

BLANK, BLINK, BOLD, BRIGHT, COLOR, DIM, DISPLAY, FONT, INVERSE, NO ULINE, NORMAL, PICTURE, ULINE

Example

In the following example, if the customer's credit status is not **BAD**, it is printed with normal intensity on the report.

```
          IF       TAR CUSTOMER CREDIT STATUS     NE     BAD
    T     NO BOLD  TAR CUSTOMER CREDIT STATUS     (AT APPEARANCE #    )
          DISPLAY  TAR CUSTOMER CREDIT STATUS     (AT APPEARANCE #    )
```

NO INPUT

NO INPUT is used in input processes to prevent a user from modifying a particular field.

```
    ••••• NO INPUT ••• ••••••••••••••••••••• ••• (AT APPEARANCE # •••)
    (1)            (2) (3)                   (4)                  (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or predefined field

(4) Occurrence (constant/index)

(5) Appearance (constant/index)
```

Using the Statement

The NO INPUT statement can be used to make fields conditionally unmodifiable, depending on prior entries. The appearance number can be used if the same field appears more than once on the image.

The application ID, field name, occurrence, and appearance may all be left blank, in which case the statement is applied to all items on the image.

When NO INPUT is executed for an item, it is dimmed on the image.

Restrictions

NO INPUT statements are only valid on image-related event points.

Related Statements

OK INPUT

Example

The example below prevents the operator from entering a General Ledger account number into a customer's record if the General Ledger Chart of Accounts file does not exist.

```
          IF EXIST TGL BALANCE1            FAIL 0
    F     NO INPUT TAR CUSTOMER ACCOUNT           (AT APPEARANCE #    )
```

NO ULINE

Items can be displayed with an accompanying underline on the image. NO ULINE is used to reverse an underline.

```
    ••••• NO ULINE ••• •••••••••••••••••••• ••• (AT APPEARANCE # •••)
    (1)            (2) (3)                   (4)                   (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or predefined field

(4) Occurrence (constant/index)

(5) Appearance (constant/index)
```

Using the Statement

The appearance number can be used if the same field appears more than once on the image.

If the application ID, field name, occurrence number, and appearance number are not specified (left blank), the NO ULINE statement affects all of the items painted on the associated image.

NO ULINE is one of the *characteristic-type* statements. Refer to the common attributes of all the statements in this category in the Characteristic-Type Statements section of this manual.

The operation of a NO ULINE statement affects only the next generation of an image. In other words, the item characteristics are reset each time an image is produced.

Restrictions

> Note that because characteristic statements operate on items rather than fields, they are only meaningful in event points that are associated with an image.

Statement Ignored

The system ignores any **NO ULINE** statements specified in non-image event points.

Related Statements

BLANK, BLINK, BOLD, BRIGHT, COLOR, DIM, DISPLAY, FONT, INVERSE, NO BOLD, NORMAL, PICTURE, ULINE

Example

The following example erases an underline from beneath the 2nd appearance of the customer's name.

```
          NO ULINE TAR CUSTOMER NAME              (AT APPEARANCE # 002)
          DISPLAY  TAR CUSTOMER NAME              (AT APPEARANCE # 002)
```

NORMAL

Items can be displayed in an inverse video mode on an image. In contrast, NORMAL is used to display an item in a NORMAL video mode.

```
    ••••• NORMAL   ••• ••••••••••••••••••••• ••• (AT APPEARANCE # •••)
    (1)            (2) (3)                   (4)                  (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or predefined field

(4) Occurrence (constant/index)
```

```
(5) Appearance (constant/index)
```

Using the Statement

The appearance number can be used if the same field appears more than once on the image.

If the application ID, field name, occurrence number, and appearance number are not specified (left blank), the NORMAL statement affects all of the items painted on the associated image.

NORMAL is one of the *characteristic-type* statements. Refer to the common attributes of all the statements in this category in the Characteristic-Type Statements section of this manual.

The operation of a NORMAL statement affects only the next generation of an image. In other words, the item characteristics are reset each time an image is produced.

Restrictions

> Note that because characteristic statements operate on items rather than fields, they are only meaningful in event points that are associated with an image.

NORMAL is a terminal display attribute only and therefore is not allowed on images in output processes.

Statement Ignored

The system ignores any NORMAL statements specified in non-image event points.

Related Statements

BLANK, BLINK, BOLD, BRIGHT, COLOR, DIM, DISPLAY, FONT, INVERSE, NO BOLD, NO ULINE, PICTURE, ULINE

Example

The example below “turns off” inverse video for the customer’s name.

```
          NORMAL   TAR CUSTOMER NAME              (AT APPEARANCE #    )
          DISPLAY  TAR CUSTOMER NAME              (AT APPEARANCE #    )
```

OK INPUT

OK INPUT is used in input processes to make an item modifiable. This can be used for fields that are conditionally modifiable depending on prior entries.

```
    ••••• OK INPUT ••• •••••••••••••••••••• ••• (AT APPEARANCE # •••)
    (1)            (2) (3)                    (4)                  (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or predefined field

(4) Occurrence (constant/index)

(5) Appearance (constant/index)
```

Using the Statement

OK INPUT is used in input processes to make an item modifiable. This can be used for fields that are conditionally modifiable, depending on prior entries. The appearance number can be used if the same field appears more than once on the image.

The application ID, field name, occurrence, and appearance may all be left blank, in which case the statement is applied to all items on the image.

Restrictions

OK INPUT statements are only valid in image-related event points.

Related Statements

NO INPUT

Example

The following example allows a user to enter a General Ledger account number into a customer's record if the General Ledger Chart of Accounts file exists.

```
          IF EXIST TGL BALANCE1            FAIL 0
    T     OK INPUT TAR CUSTOMER ACCOUNT           (AT APPEARANCE #    )
```

OPEN

The OPEN statement logically and physically opens a specified file immediately. Normally, files are not opened until an I/O operation is performed on the file, such as a READ or a WRITE, and are handled by APPX automatically. However, OPEN statements may be necessary to switch between disk and cache versions of a file. Also, OPEN statements may be desirable for performance reasons, for example, by checking certain conditions before requiring a file to be opened.

```
    ••••• OPEN     ••• ••••••••  SHARE? •  FAIL • CACHE? •
    (1)            (2) (3)             (4)     (5)      (6)
```

```
(1) T/F execution conditions

(2) Application ID

(3) File name

(4) Share the file? (Y/N)

(5) Fail action (0=nothing, 1=warning, 2=error, 3=cancel)

(6) Cache? (Y=cache version, N=disk version)
```

Sets True/False Status Indicator

The next true/false status indicator is set to *T* if the file exists and was successfully opened, and to *F* if it cannot be opened. If the process cannot be invoked successfully, the *FAIL* parameter determines what additional action to take. In addition, on a false condition, the predefined field STATUS CODE returns a token indicating the reason for an I/O command failure. This token can be used as a key to read the system message file to obtain a descriptive I/O error message (see STATUS CODE for details).

Using the Statement

A file is *physically* opened only once for each user, but may be *logically* opened in several different processes for each user. For example, if a user is entering sales orders, the customer file may be logically opened both for the process that enters the order itself and the related child process that enters the sales line items. Additionally, the OPEN statement allows you to open either the cached version of a file or the disk version of a file.

The *SHARE?* parameter determines whether other users should be permitted to update the file while it is open. If some other user has opened the file already in a restrictive sharing mode, the current user is presented with an “In Use” display, unless a TIMEOUT statement with a nonblank label was executed before the **OPEN** statement. In this case, processing control transfers to the label specified.

The application ID and file name for the file to open are specified in the statement. Additional information that can affect the location of the file may be contained in the predefined fields DATABASE and NEXT DATABASE.

The file name references a file that is defined in the data dictionary of the application ID. You can set the NEXT DATABASE predefined field to the database ID of the database containing the file that you want opened.

If NEXT DATABASE is blank, the DATABASE predefined field is used. Note that the NEXT DATABASE field is modifiable, whereas the DATABASE field is not.

The NEXT DATABASE field is set to blank by the **OPEN** statement, regardless of whether or not the file is successfully opened.

If NEXT DATABASE is specified in conjunction with a cache file, then NEXT DATABASE is cleared and ignored. Like memory files there is only one instance of a cached file, not one for each database.

Execution Errors

Serious problems encountered while trying to open a file, such as a file structure/data dictionary mismatch, cause a non-recoverable runtime error to occur, regardless of the *FAIL* parameter.

If *CACHE?* has been set to *Y* and a cached version of the file does not exist, then the OPEN statement will fail. Therefore, a CREATE statement must be used to create the cached version of a file before the OPEN statement can be used to open it.

Related PDFs

DATABASE, NEXT DATABASE

Related Statements

BEG READ, CLOSE, CREATE, END READ, READ, READNEXT, SCRATCH, TIMEOUT, WRITE

Example

The following example opens the Customer file in the database labeled ARC. Other users are allowed access to the file. If the OPEN fails, a warning message is generated.

```
SET      --- NEXT DATABASE              =      ARC
OPEN     TAR CUSTOMER  SHARE? Y  FAIL 1 CACHE? N
```

OR

The OR statement adds a second or subsequent conditional check to a preceding statement, such as the series “IF A is true OR B is true, then...”. The OR statement itself compares the contents of two fields, or compares the contents of a field with a constant, to determine if a condition is true.

```
..... OR      ... ..................... ... .. ... ..................... ...
(1)           (2) (3)                   (4) (5)(6) (7)                   (8)
```

(1) T/F execution conditions	(5) Relation (LT,GT,LE,GE,EQ,NE,IN,EX)
(2) 1st application ID	(6) 2nd application ID
(3) 1st field name or PDF	(7) 2nd field name, PDF, or constant
(4) 1st occurrence (constant/index)	(8) 2nd occurrence (constant/index)

Sets True/False Status Indicator

The OR statement may modify the current value of the next internal true/false status indicator. If the preceding statement is false and the OR statement is true, the current status of the next indicator is changed to *T*. See Using AND and OR Statements for more information.

Using the Statement

The OR statement specifies a relationship between two values. The valid relational operators are:

- less than (LT), greater than (GT), less than or equal to (LE), greater than or equal to (GE)
- equals (EQ), not equal to (NE), includes (IN), and excludes (EX).

The relation *IN* means "includes". The relation is true if the value in the second field is included in the first field, and false otherwise. If the relation is true, the predefined field TEXT AT POSITION is set to the starting position of the second field within the first field. The first field must be an alpha field; the second can be either an alpha field or a constant. Trailing blanks are ignored if the second field is a constant; however, you may specify fixed spaces (by entering underscores) to include one or more trailing spaces as part of a character string that is being searched.

Normally, an OR statement is used immediately after an IF statement. However, the OR statement will operate successfully when used after any statement that sets the true/false status indicator. See AND and OR with Statements Other Than IF.

An OR statement may also be used in conjunction with one or more AND statements. For example:

```
IF      A
AND     B
OR      C
AND     D
```

Another example is:

```
IF      A
OR      B
AND     C
OR      D
```

Each of these examples could be expanded to add more AND and OR statements in various sequences but, no matter how many there are, the rules are always the same:

- Each AND or OR statement performs only two checks at most: it checks the current value of the next true/false status indicator, and it evaluates its own condition. It does not matter how many statements preceded the AND/OR statement, nor what other statements follow the AND/OR statement.
- If the current status of the next indicator is already false, the AND statement is bypassed since it is no longer relevant. If the current status is true and the AND statement is true, the current status is

left unchanged as *T*. However, if the AND statement is false, the current status of the next indicator is changed to *F*.

- Conversely, for an OR statement, if the current status of the next indicator is already true, the OR statement is bypassed since it is no longer relevant. If the current status is false and the OR statement is false, the current status is left unchanged as *F*. However, if the OR statement is true, the current status of the next indicator is changed to *T*.

Restrictions

The OR statement supports comparisons between like data types only (alpha-to-alpha, numeric-to-numeric, date-to-date, logic-to-logic). When comparing two date fields, results can be unpredictable if you are using a date field without a century component. Confirm that your System Administrator has set up the *Pivot Year* environment variable to avoid unexpected results.

Related PDFs

TEXT AT POSITION

Related Statements

AND, IF

Example

This example shows how to bypass statement processing if the customer is a miscellaneous customer or has a zero balance.

```
          IF       TAR CUSTOMER MISC CUSTOMER     EQ     1
          OR       TAR CUSTOMER BALANCE           EQ     0
    T     GOTO     :NO STATEMENT
          *
          *        (processing to print customer statements)
          *
          LABEL    :NO STATEMENT
```

OUTPUT

The OUTPUT statement allows you to alter the flow of processing by invoking an APPX output process from within an ILF routine.

```
    ••••• OUTPUT   ••• ••••••••••••••••••••••••••••••   ••••••••••   END? •  FAIL •
    (1)            (2) (3)                              (4)              (5)     (6)
```

```
(1) T/F execution conditions

(2) Application ID
```

```
(3) Output process name

(4) Invocation type (SUBPROCESS, RELATED, DETACHED)

(5) End current process first? (Y/N)

(6) Fail action (0=nothing, 1=warning, 2=error, 3=cancel)
```

Sets True/False Status Indicator

The next status indicator is set to *T* if the process is invoked successfully, and to *F*, otherwise. If the process cannot be invoked successfully, the *FAIL* parameter determines what additional action to take.

Using the Statement

In addition to the application ID and output process name to be invoked, you specify the invocation type (subprocess, related, or detached). As with all other child invocations, this parameter controls the level of data sharing between the parent and the output process that is invoked.

If *END?* is set to *Y*, the current process will end before invoking the output process. The output process will return to the parent of the current process when it is finished. If *END?* is set to *N*, the output process returns to the current process (beginning with the first statement after the **OUTPUT** statement).

If the application ID is blank, the value in the predefined field NEXT APPLICATION is used. If the process name is blank, the value in the predefined field NEXT PROCESS NAME is used.

Passing Values

The OUTPUT statement passes the current PASS list, if any. Refer to the PASS Lists section for details.

Related PDF's

NEXT APPLICATION, NEXT PROCESS NAME

Related Statements

INPUT, INQUIRY, JOB, MENU, PASS, QUERY, STATUS, SUBR, UPDATE

Example

The following example checks to see if it is the last processing day in the month. If so, the **OUTPUT** statement is executed, invoking an end-of-month report.

```
          IF       TAR WORK END OF MONTH          EQ     Y
    T     OUTPUT   TAR END OF MONTH REPORT              SUBPROCESS   END? Y  FAIL 3
```

OVERFLOW

The OVERFLOW statement provides a means for intercepting numeric overflow conditions which can be caused by division by zero or by a result too large to be stored in a numeric destination field. The OVERFLOW statement specifies a label in much the same manner as a GOTO statement.

```
••••• OVERFLOW ••••••••••••••••••••••••••••
(1)            (2)
```

```
(1) T/F execution conditions

(2) Label
```

Using the Statement

If a statement subsequent to the OVERFLOW statement references a numeric field and causes an overflow condition, processing control is transferred to the statement immediately following the specified label. The LABEL statement referenced by the OVERFLOW statement can be anywhere in the same event point as the OVERFLOW.

The OVERFLOW statement may be respecified at any time with another label, or the intercept may be disabled by an OVERFLOW statement with no label specified. Intercepts are automatically disabled when the event point that they are contained in is complete.

Compilation Errors

A process compile error occurs if the referenced LABEL is not defined within the same event point as the **OVERFLOW** statement, and also not defined in any ILF subroutines copied into the event point with the COPY statement.

Related Statements

CALC, COMPUTE, COPY, LABEL, SET, SET TOT

Example

In the following example, the OVERFLOW statement prevents the event point from failing with an overflow condition caused either by WORK AVG SALES not being large enough to store the integer component of WORK TOTAL SALES or by a division by zero error should the number of salesreps be zero.

```
          OVERFLOW :WORK AVG SALES OVERFLOW
          SET      TAR WORK AVG SALES             =  TAR WORK TOTAL SALES
          COMPUTE  TAR WORK AVG SALES             /  TAR WORK NO SALESREPS
          GOTO     :DISPLAY RESULTS
          *
```

```
          LABEL    :WORK AVG SALES OVERFLOW
          SET      TAR WORK AVG SALES             =      0
          WARNING  Numeric Overflow - WORK AVG SALES set to zero!
          *
          LABEL    :DISPLAY RESULTS
          DISPLAY  TAR WORK AVG SALES
```

PASS

The PASS statement causes the indicated area to be passed to the next subroutine, process, or external program that is executed via a GOSUB, CALL, RUN, or process-type statement (i.e., INPUT, INQUIRY, JOB, MENU, OUTPUT, QUERY, STATUS, SUBR, UPDATE), or to the next automatic or optional child process that is run. See the next section, PASS Lists, for a discussion of how PASS lists are maintained and cleared.

```
    ••••• PASS     ••• •••••••••••••••••••• ••• ••••••••••••••  SHARE? •
    (1)            (2) (3)                   (4) (5)                    (6)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or predefined field

(4) Occurrence (constant/index)

(5) Area type (FIELD)

(6) Share? (Y/N)
```

Using the Statement

The PASS statement has two uses:

- To explicitly pass an area from one process to another that would not otherwise be passed using implicit interprocess communication specifications.
- To pass areas to an external program written in another programming language.

In the first case, the receiving program must explicitly receive the area with a RECEIVE statement, unless it receives the passed area based on implicit IPC specifications.

In the second case, the order areas are passed in is significant. Areas are passed in the order that the PASS statements are executed in.

Area type should be FIELD until such time other values have been implemented.

If *SHARE?* is *Y*, the **PASS** statement sends a pointer to the area value. Any changes made to this area in the receiving process or external program are available to the calling process.

If *SHARE?* is *N*, the **PASS** statement sends a copy of the area value. Changes made to the area in the receiving process (or program) do not modify the value in the calling process.

Related Statements

CALL, GOSUB, INPUT, INQUIRY, JOB, MENU, OUTPUT, QUERY, RECEIVE, RUN, STATUS, SUBR, UPDATE

Example

```
        PASS     TAR CUSTOMER NAME             FIELD            SHARE? N
        PASS     TAR WORK STRING LENGTH        FIELD            SHARE? Y
        GOSUB    :COMPUTE NAME LENGTH
        END
        *
        LABEL    :COMPUTE NAME LENGTH
        RECEIVE  --- TEMP 2K                   FIELD              FAIL N
        RECEIVE  --- A1                        FIELD              FAIL N
        CNV BIN  --- TEMP 1                    =      0
        APPEND   --- TEMP 2K                   0  --- TEMP 1
        IF       --- TEMP 2K                   IN --- TEMP 1
    T   SET      --- A1                        =  --- TEXT AT POSITION
    T   COMPUTE  --- A1                        -      1
        RETURN
```

In this example, the initial value of CUSTOMER NAME is returned following the subroutine. Even if the subroutine altered the value of this field, it would still be returned in its initial state because the PASS statement does not share the field. The WORK STRING LENGTH field (paired with the value of the AI field), however, is modified to contain the length of the value in the CUSTOMER NAME field. This new value is returned following the subroutine because the WORK STRING LENGTH field was passed as a shared field.

PASS Lists

As discussed in the previous section, the PASS statement allows an area to be passed to the next subroutine, process, or external program that is executed via a GOSUB, CALL, RUN, or process-type statement (i.e., INPUT, INQUIRY, JOB, MENU, OUTPUT, QUERY, STATUS, SUBR, UPDATE), or to the next automatic or optional child process that is run. Prior to APPX 4.1, there was only one PASS list which was cleared upon completion of a CALL or RUN statement. Now it is possible to have a series of PASS lists. Each PASS statement adds its area to a single list until the execution of a GOSUB, CALL, RUN, or process-type statement, or the invocation of a automatic or optional child process. Upon such execution, only the values in the current PASS list are passed. However, if executing an APPX

subroutine or process, it can in turn execute PASS statements. These statements then build another PASS list until terminated in the same manner. Each PASS list is cleared when execution returns to the routine or level which established it.

The following code provides an example of the use of three separate PASS lists. The first PASS list holds values for CUSTOMER NAME and CUSTOMER ADDRESS. The second PASS list contains the value for AREA MAIL CODE, and this is the only value passed to the :PROCESS MAIL CODE subroutine. The second PASS list is then cleared upon return from this subroutine while the first PASS list remains intact until the return from the :PROCESS NAME & ADDRESS subroutine. As illustrated in our example, values from the first PASS list do not have to be RECEIVE'd before initiating another PASS list. Finally, a third PASS list is established for the CUSTOMER BALANCE value, which is finally cleared upon return from the :PROCESS BALANCE subroutine.

```
         PASS     TAR CUSTOMER NAME              FIELD            SHARE? N
         PASS     TAR CUSTOMER ADDRESS           FIELD            SHARE? N
         GOSUB    :PROCESS NAME & ADDRESS
         PASS     TAR CUSTOMER BALANCE           FIELD            SHARE? N
         GOSUB    :PROCESS BALANCE
         END
         *
         LABEL    :PROCESS NAME & ADDRESS
         RECEIVE  TAR WORK NAME
         *        (code to process name)
         PASS     TAR AREA MAIL CODE
         GOSUB    :PROCESS MAIL CODE
         RECEIVE  TAR WORK ADDRESS
         *        (code to process address)
         RETURN
         *
         LABEL    :PROCESS MAIL CODE
         RECEIVE  TAR WORK MAIL CODE
         *        (code to process mail code)
         RETURN
         *
         LABEL    :PROCESS BALANCE
         RECEIVE  TAR WORK BALANCE
         *        (code to process balance)
         RETURN
```

PICTURE

The PICTURE statement displays a graphics file or graphical instruction set on the image. This statement is used to place pictures, buttons, or other graphics items on a displayed image.

```
    ••••• PICTURE  ••• ••••••••••••••••••••• ••• (AT APPEARANCE # •••)
    (1)            (2) (3)                   (4)                  (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or predefined field

(4) Occurrence (constant/index)

(5) Appearance (constant/index)
```

Using the Statement

The appearance number can be used if the same field appears more than once on an image.

PICTURE is one of the *characteristic-type* statements. Refer to the common attributes of all the statements in this category in the Characteristic-Type Statements section of this manual.

The operation of a PICTURE statement affects only the next generation of an image. In other words, the item characteristics are reset each time an image is produced,

Restrictions

> Note that, because the PICTURE statement operates on items rather than fields, this statement is meaningful only in event points that are associated with an image.

Statement Ignored

The system ignores any PICTURE statements specified in non-image-related event points.

Related Statements

BLANK, BLINK, BOLD, BRIGHT, COLOR, DIM, DISPLAY, FONT, INVERSE, NO BOLD, NO ULINE, NORMAL, ULINE

Example

The PICTURE statement is used to cause the current contents of a graphics item to appear on the image. In this example, if no picture is available for the current product, a default “picture not available” graphics is displayed on the image in the field defined as WORK PICTURE FILE. Otherwise, the picture file associated with this product is displayed.

```
        IF       TIC PRODUCT PICTURE FILE       EQ
   T    SET      TIC WORK PICTURE FILE          =  TIC WORK PICTURE NOT AVAIL
   F    SET      TIC WORK PICTURE FILE          =  TIC PRODUCT PICTURE FILE
        PICTURE  TIC WORK PICTURE FILE          (AT APPEARANCE #    )
        DISPLAY  TIC WORK PICTURE FILE          (AT APPEARANCE #    )
```

POSITION

POSITION can be used in the image event points (*Pre-Display*, *Option Intercept*, *Pre-Display/Verify*) to position the cursor to either a specific item or a specific row and column position.

```
    ••••• POSITION ••• •••••••••••••••••••• ••• (AT APPEARANCE # •••)
    (1)            (2) (3)                  (4)                  (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or predefined field

(4) Occurrence (constant/index)

(5) Appearance (constant/index)
```

Using the Statement

To position the cursor to a specific row and column, the designer must leave the application ID, field name, occurrence number, and appearance number blank. If all of the fields are left blank, the POSITION statement positions the cursor to the row and column specified by the CURSOR ROW and CURSOR COLUMN predefined fields, respectively. Note that the CURSOR ROW and CURSOR COLUMN predefined fields refer to the physical row and column of the terminal, rather than to a relative position within the image, window, region, or surface. If the CURSOR ROW or CURSOR COLUMN fields contain a value larger than the size of the terminal, the POSITION statement substitutes the terminal limit for that coordinate. If the CURSOR ROW or CURSOR COLUMN fields contain a negative value, the POSITION statement substitutes *1* for the negative coordinate.

Restrictions

The POSITION statement only affects the cursor position if the image is redisplayed within the same image execution cycle (i.e., if the same image is redisplayed for a higher entry level, or because of a change to the image).

Statement Ignored

If the POSITION statement appears in an invalid event point, it is ignored at runtime. This means that no errors or warnings are generated at compile or runtime because of an invalid context.

Related PDFs

CURSOR COLUMN, CURSOR ROW

Example

In the following example, if the customer's credit limit is zero, the cursor is positioned on this field.

```
        IF       TAR CUSTOMER CREDIT LIMIT      EQ     0
    T   POSITION TAR CUSTOMER CREDIT LIMIT      (AT APPEARANCE #    )
```

QUERY

The QUERY statement allows you to alter the flow of processing by invoking an APPX query process from within an ILF routine.

```
    ••••• QUERY    ••• ••••••••••••••••••••••••••••••   ••••••••••   END? •  FAIL •
    (1)            (2) (3)                              (4)              (5)     (6)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Query process name

(4) Invocation type (SUBPROCESS, RELATED, DETACHED)

(5) End current process first? (Y/N)

(6) Fail action (0=nothing, 1=warning, 2=error, 3=cancel)
```

Sets True/False Status Indicator

The next status indicator is set to *T* if the process is invoked successfully, and to *F*, otherwise. If the process cannot be invoked successfully, the *FAIL* parameter determines what additional action to take.

Using the Statement

In addition to the application ID and query process name to be invoked, you specify the invocation type (subprocess, related, or detached). As with all other child invocations, this parameter controls the level of data sharing between the parent and the query process that is invoked.

If *END?* is set to *Y*, the current process will end before invoking the query process. The query process will return to the parent of the current process when it is finished. If *END?* is set to *N*, the query process returns to the current process (beginning with the first statement after the **QUERY** statement).

If the application ID is blank, the value in the predefined field NEXT APPLICATION is used. If the process name is blank, the value in the predefined field NEXT PROCESS NAME is used.

Passing Values

The QUERY statement passes the current PASS list, if any. Refer to the PASS Lists section for details.

Related PDFs

NEXT APPLICATION, NEXT PROCESS NAME

Related Statements

INPUT, INQUIRY, JOB, MENU, OUTPUT, PASS, STATUS, SUBR, UPDATE

Example

The following example checks to see if it is the last processing day in the month. If so, the **QUERY** statement is executed, invoking an end-of-month query process.

```
          IF       TAR WORK END OF MONTH          EQ     Y
    T     QUERY    TAR END OF MONTH QUERY               SUBPROCESS   END? Y  FAIL 3
```

READ

The READ statement reads a record from a specified file, using the specified key.

```
    ••••• READ     ••• •••••••••••••••••••• HOLD • FT • BY ••••••••••••••••••••
    (1)            (2) (3)                        (4)  (5)   (6)
```

```
(1) T/F execution conditions

(2) Application ID

(3) File name or field name

(4) Hold type (0=do not hold, 1=hold, 2=hold with potential recovery)

(5) Fail Type (0=nothing, 1=warning, 2=error, 3=cancel)

(6) Access key field name (blank if one-record file)
```

Sets True/False Status Indicator

The next status indicator is set to *T* if the record is read successfully, and to *F* if the record is not found. If the record is not on file, the *FAIL* (or *FT*) parameter determines what additional action to take. In addition, on a false condition, the predefined field STATUS CODE returns a token indicating the reason for an I/O command failure. This token can be used as a key to read the system message file to obtain a descriptive I/O error message (see STATUS CODE for details).

Using the Statement

When using this statement in conjunction with an RDBMS table, you can specify a field name instead of a file name to limit the volume of data being passed between APPX and the RDBMS. See the Partial-Record I/O section in Chapter 4-3: Specifying Statements for more information.

The READ statement allows a record to be read as "held" for update. While a record is held, no other user may update or delete that record. If the record is to be rewritten or deleted, the *HOLD* parameter must be set to *1* (hold) or *2* (recover). Otherwise, it should be set to *0* (no). The recover specification holds the record and internally saves a copy of the record so that a subsequent rewrite or delete can be performed, even if the hold is lost (by reading another record with hold before completing the rewrite or delete). "Recover" can rehold the record for update, as long as the saved copy is identical to the original record that was read. Note that performance is improved with hold type *1*.

> Note also that the READ statement does not generate an "In Use" message if the record to be read as "held" is currently being held by another user. The READ statement simply waits until the record to read has been freed up by the other user, unless a TIMEOUT statement with a nonblank label was previously encountered. If a TIMEOUT statement is used, processing control transfers to the specified label.

The keypath and position established by the process interpreter for the process control file is not affected by a "Read Next Record" loop.

Restrictions

The contents of a specified key field must be set before executing a **READ**. For one-record files, the key field specification must be left blank. For consecutive files, the key field specification must contain a numeric field that must be set to the relative record number of the record to be read, or a numeric constant indicating that record number. For indexed files, the key field specification must identify a key field in the file to be read.

Compilation Errors

If any of the above restrictions fail, an error is generated at compile time.

Related Statements

BEG READ, CLOSE, DELETE, END READ, OPEN, READNEXT, REWRITE, TIMEOUT

Example

In the following example, the original transaction date is set from the Unpaid Invoices file if a record exists for that invoice, or from the Cash Receipts file if no such record is found.

```
*       Set original transaction date so post can compute days to pay
SET     TAR UNPAID CUSTOMER NO         =  TAR RECEIPT2 CUSTOMER NO
SET     TAR UNPAID APPLY TO            =  TAR RECEIPT3 PAID INVOICE
```

```
        READ     TAR UNPAID                  HOLD 0 FT 0 BY UNPAID KEY
T       SET      TAR POST ORIGINAL TRX DATE     =  TAR UNPAID INVOICE DATE
F       SET      TAR POST ORIGINAL TRX DATE     =  TAR RECEIPT2 DATE RECEIVED
```

In the second example below, the Accounts Receivable Parameters record is read for update and the process is canceled if the record is not found (no key is specified because Parameters is a one-record file).

```
        READ     TAR PARAM                   HOLD 1 FT 0 BY
F       CANCEL   Accounts Receivable Parameters Record Not on File
```

READNEXT

The READNEXT statement reads the next record in a specified file.

```
••••• READNEXT ••• ••••••••••••••••••••• HOLD • FT • BY ••••••••••••••••••••
(1)            (2) (3)                      (4)  (5)   (6)
```

```
(1) T/F execution conditions

(2) Application ID

(3) File name or field name

(4) Hold type (0=do not hold, 1=hold, 2=hold with potential recovery)

(5) Fail Type (0=nothing, 1=warning, 2=error, 3=cancel)

(6) Access key field name (blank if consecutive file)
```

Sets True/False Status Indicator

Normally, the READNEXT statement sets the next status indicator to *F* or *T* depending on whether the end of the file is or is not reached, respectively.

If an END AT statement is executed before a READNEXT, the status indicator is also set to *F* if the record read has a key value beyond the one specified in the END AT statement. Note that the current record area contains the record that is beyond the END AT range and that, unlike an end-of-file condition, the position in the key path does not become undefined.

If the READNEXT sets the next status indicator to *F*, the *FAIL* or *(FT)* parameter determines what additional action to take. In addition, on a false condition, the predefined field STATUS CODE returns a token indicating the reason for an I/O command failure. This token can be used as a key to read the system message file to obtain a descriptive I/O error message (see STATUS CODE for details).

Using the Statement

When using this statement in conjunction with an RDBMS table, you can specify a field name instead of a file name to limit the volume of data being passed between APPX and the RDBMS. See the Partial-Record I/O section in Chapter 4-3: Specifying Statements for more information.

If a key path is specified, the next record is read using the path specified. If no keypath is specified, the keypath used by the most recently executed READ or READNEXT statement for the same file is used. If no READ or READNEXT statement for the same file was previously executed, or the file was logically closed and reopened, the primary key path is used.

If a BEG AT statement for this file is executed before the READNEXT, the position in the file is modified before executing the READNEXT, based on the value and relation specified in the BEG AT statement and the keypath identified by the READNEXT statement.

If the READNEXT identifies a key path that is different than the key path used in the most recently encountered READ or READNEXT statement for the same file, and no subsequent BEG AT is encountered, the file position is reset to the first record in the key path specified in the READNEXT statement.

The key path and position established by the process interpreter for the process control file has no effect on, and is not affected by, the READNEXT position, nor does the key path and position established for a "Read Next Record" loop have any effect.

The READNEXT statement allows a record to be read and held for update. While a record is held, no other user may update or delete that record. If the record is to be rewritten or deleted, the *HOLD* specification must be set to *1* (hold) or *2* (recover). Otherwise, it should be set to *0* (no). The recover specification holds the record and internally saves a copy of the record so that a subsequent rewrite or delete can be performed, even if the hold is lost (by reading another record with hold before completing the rewrite or delete). "Recover" can rehold the record for update, as long as the saved copy is identical to the original record that was read. Note that performance is improved with hold type *1*.

> Note also that the READNEXT statement does not generate an "In Use" message if the record to be held is currently being held by another user. The READNEXT statement waits until the record to be read is freed up by the other user, unless a TIMEOUT statement with a nonblank label was previously encountered. In that case, processing control transfers to the specified label.

Restrictions

For consecutive files, the key field must be left blank because there is only one path (the relative record number) for accessing consecutive files.

The **READNEXT** statement cannot be used with one-record files.

Execution Errors

Once the end of the file is reached, the current position within the file becomes undefined, and a subsequent **READNEXT** will result in a non-recoverable runtime error unless a position is re-established (see the *Sets True/False Status Indicator* section above).

Related Statements

BEG AT, END AT, READ, TIMEOUT

Example

In the following example, a range of records is read from the Transaction file and summed into the field, WORK CUSTOMER BALANCE. When the range of records is exhausted, the predefined field *A* is then set to the balance.

```
        BEG AT   TAR TRANSACT IN TAR WORK APPLY TO KEY
        END AT   TAR TRANSACT IN TAR WORK APPLY TO KEY
        LABEL    :NEXT TRANSACTION RECORD
        READNEXT TAR TRANSACT               HOLD 0 FT 0 BY TRANSACT KEY
  T     COMPUTE  TAR WORK CUSTOMER BALANCE      +  TAR TRANSACT AMOUNT
  T     GOTO     :NEXT TRANSACTION RECORD
        SET      --- A                          EQ TAR WORK CUSTOMER BALANCE
```

RECEIVE

The RECEIVE keyword can be used for different purposes. The RECEIVE statement works with the PASS statement to implement true subroutines. Or, it works without the PASS statement to essentially declare a local variable (a field that is automatically restored to its initial value when the subroutine, event point, or process ends).

```
    ••••• RECEIVE  ••• ••••••••••••••••••••• ••• ••••••••••••••     FAIL •
    (1)            (2) (3)                   (4) (5)                   (6)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or predefined field

(4) Occurrence (constant/index)

(5) Area type (FIELD)

(6) Fail action (0=nothing, 1=warning, 2=error, 3=cancel)
```

Sets True/False Status Indicator

The RECEIVE statement sets the next status indicator to T if a matching PASS statement was found and to F if not. If you RECEIVE without a PASS statement, you are simply declaring a local variable that is restored to its initial value when the subroutine, event point, or process ends.

Using the Statement

The RECEIVE statement can be used in three locations:

- Inside a subroutine marked by the LABEL statement. Field values are restored when the RETURN is executed.
- Inside an event point. Field values are restored when the event point ends.
- Inside the *Start of Process* event point. Field values are restored when the process ends.

Data types of related PASS and RECEIVE statements do not have to match. The same rules that are used for SET statements convert the PASS field into the RECEIVE field (and back again). Fields are received in the same order they are passed so passing and receiving the same field names is not required.

Related Statements

PASS

Example

```
        PASS     TAR CUSTOMER NAME             FIELD            SHARE? N
        PASS     TAR WORK STRING LENGTH        FIELD            SHARE? Y
        GOSUB    :COMPUTE NAME LENGTH
        END
        *
        LABEL    :COMPUTE NAME LENGTH
        RECEIVE  --- TEMP 2K                   FIELD              FAIL N
        RECEIVE  --- A1                        FIELD              FAIL N
        CNV BIN  --- TEMP 1                    =      0
        APPEND   --- TEMP 2K                   0  --- TEMP 1
        IF       --- TEMP 2K                   IN --- TEMP 1
    T   SET      --- A1                        =  --- TEXT AT POSITION
    T   COMPUTE  --- A1                        -      1
        RETURN
```

In this example, the initial value of CUSTOMER NAME is returned following the subroutine. Even if the subroutine altered the value of this field, it would still be returned in its initial state because the PASS statement does not share the field. The WORK STRING LENGTH field (paired with the value of the AI field), however, is modified to contain the length of the value in the CUSTOMER NAME field. This new value is returned following the subroutine because the WORK STRING LENGTH field was passed as a shared field.

REFRESH

REFRESH can be used to immediately redisplay the image in an input process without waiting for it to be redisplayed normally in accordance with the process cycle.

```
    ••••• REFRESH
    (1)
```

```
(1) T/F execution conditions
```

Using the Statement

The REFRESH statement instantly redisplays the current image without waiting for the processing cycle to complete.

This statement is useful when changes to the display must be shown immediately. Normally, an image is redisplayed only when the user initiates the processing cycle by pressing ENTER. By using the REFRESH statement, however, the image can be redisplayed prior to the execution of the processing cycle. This is especially useful, as shown in the following example, when a user's idle scrolling display must be updated to reflect any new records added by other users.

Related Statements

DISPLAY

Example

```
          IF       TAR WORK 30 SECONDS ELAPSED    EQ     Y
    T     REFRESH
```

RELEASE

The RELEASE keyword is reserved for future use.

RESTORE

The RESTORE statement restores the contents of a field or a record from the area assigned as the store or default area for the field or record.

```
    ••••• RESTORE  ••• ••••••••••••••••••••• ••• •••••••••••••
    (1)            (2) (3)                   (4) (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) File name, field name, or predefined field

(4) Occurrence (constant/index)

(5) Area type (RECORD, FIELD, DEFAULT RECORD, DEFAULT FIELD,
               ORIGINAL RECORD, ORIGINAL FIELD)
```

Using the Statement

The RESTORE statement can recover data values from the store, default, or original record area. The contents of the store record area are established by the STORE command. If no STORE command was executed, the store record area contains the default values, which are defined in the data dictionary. The RESTORE statement does not alter the contents of the store, default, or original area.

Fields that are part of a record do not have an independent store or default area. All fields that are part of a record are restored from their proper location in the area assigned for the record. Therefore, it is possible to store an entire record and then restore specific fields, or to store only some fields and then restore the entire record. A field or record can be stored any number of times, and can also be restored any number of times, independently. These statements only cause the contents of the specified field or record to be stored in, or restored from, the store or default area assigned to the field or record. The store and default areas are allocated if a STORE or RESTORE statement is present. They are initialized to the default value and, therefore, a RESTORE statement can be executed even if no STORE statement is present for the field or record.

The contents of predefined fields can also be restored using the RESTORE statement.

No editing or justification is performed. All data is copied unchanged.

Restrictions

If the FIELD option is specified as the data item type, the data item to be restored must be a field name in the data dictionary. If the RECORD option is specified, the data item to be restored must be a file name in the data dictionary.

Related Statements

STORE

Example

In the following example, the record MTDPOST is filled with its default values from the data dictionary, so that you only have to supply those values that are different for this particular routine.

```
          RESTORE  TAR MTDPOST                    DEFAULT RECORD
          SET      TAR MTDPOST ACCOUNT           =  TAR ADJUST2 ACCOUNT
          SET      TAR MTDPOST POST YEAR         =  TAR ADJUST2 POST YEAR
          SET      TAR MTDPOST POST MONTH        =  TAR ADJUST2 POST MONTH
```

RETURN

The RETURN statement transfers execution to the statement following the most recent GOSUB statement.

```
    ••••• RETURN   ••• ••••••••••••••••••••• •••
    (1)            (2) (3)                   (4)
```

```
(1) T/F execution conditions

(2) Optional application ID

(3) Optional logic field name, logic PDF, or constant

(4) Optional occurrence (constant/index)
```

Restores True/False Status Indicators

When you invoke a subroutine using the keyword GOSUB, APPX temporarily stores the true/false status indicators and then clears them so that all five indicators are available for use within the subroutine. When RETURN transfers control back to the statement following the GOSUB, the system restores the indicators to the values that existed at the time the GOSUB was invoked. In addition, **when implemented**, GOSUB will set the next status indicator to the value indicated by the logic field or constant in the RETURN statement, if any.

Execution Errors

If a **RETURN** is encountered with no previous GOSUB, a non-recoverable error occurs at runtime.

Related Statements

GOSUB

Example

In this example, the same routine is used to post to both the adjustment account and the receivable account. The actual code is copied into the event point from a subroutine process.

```
          *        post adjustment
          GOSUB    :POST MTDPOST RECORD
          *        post receivable
```

```
          COMPUTE  TAR MTDPOST AMOUNT             *      -1
          SET      TAR MTDPOST ACOUNT             =  TAR ADJUST2 RECEIV ACCOUNT
          GOSUB    :POST MTDPOST RECORD
          *
          END
          *
          LABEL    :POST MTDPOST RECORD
          COPY     TAR POST MTDPOST RECORD
          RETURN
```

REWRITE

The REWRITE statement rewrites the current record to the specified file.

```
    ••••• REWRITE  ••• •••••••••••••••••••• FAIL •
    (1)            (2) (3)                       (4)
```

```
(1) T/F execution conditions

(2) Application ID

(3) File name or field name

(4) Fail action (0=nothing, 1=warning, 2=error, 3=cancel)
```

Sets True/False Status Indicator

If the record is rewritten successfully, the next level of the true/false status indicator is set to *T*. The status indicator is set to *F* if the rewrite fails (for example, a unique alternate key is already on file). If the record is not rewritten successfully, the *FAIL* parameter determines what additional action to take. In addition, on a false condition, the predefined field STATUS CODE returns a token indicating the reason for an I/O command failure. This token can be used as a key to read the system message file to obtain a descriptive I/O error message (see STATUS CODE for details).

Using the Statement

When using this statement in conjunction with an RDBMS table, you can specify a field name instead of a file name to limit the volume of data being passed between APPX and the RDBMS. See the Partial-Record I/O section in Chapter 4-3: Specifying Statements for more information.

Once a record is rewritten, the hold on that record is released. If another rewrite (or a delete) is to be attempted, the record must be read again with *HOLD?* equal to *1* or *2*.

Restrictions

The record must have been previously read with the *HOLD?* specification set to *1* or *2*, or the rewrite must be done on the record currently being held by the process control file.

Because of the nature of various file systems, only one record may be held for update at a time. If a designer does not take this into consideration, the hold on a record may be lost without notice because of multiple reads with hold. If the record is read with *HOLD?* equal to *1*, and the hold on the record is subsequently lost, the rewrite will fail. If the record is read with *HOLD?* equal to 2 (hold for recovery), and the hold is subsequently lost, APPX attempts to reread the record and hold it for update. If the record that is reread is changed in any way from the original record read, the rewrite will fail.

> Note also that a designer should not rewrite a record held by the process control file in an input process. APPX assumes that the process control file record is still held when it needs to rewrite or delete the record. If you need to modify the process control file record in an input process, you can do so in the *Pre-PCF Update* event point for that process. If the record must be rewritten before the *Pre-PCF Update* event point, you must reread it with hold or a non-recoverable file processing error will occur.

Execution Errors

A non-recoverable file processing error occurs if a rewrite of a record is attempted without having read the record with hold.

Related Statements

BEG READ, DELETE, END READ, READ, READNEXT, WRITE

Example

In this example, a record is read and updated.

```
        READ     TAR ADJUST1                HOLD 1 FT 0 BY ADJUST1 KEY
    F   ERROR    Unexpected Error READ TAR ADJUST1
    T   SET      TAR ADJUST1 JOURNAL OK         =      0
    T   REWRITE  TAR ADJUST1                FAIL 0
    TF  ERROR    Unexpected Error REWRITE TAR ADJUST1
```

ROLLBACK

Executing a **ROLLBACK** causes the RDBMS to discard all changes made during the current transaction or, if the ROLLBACK statement includes a savepoint name, to discard only changes made since that savepoint. As with the COMMIT statement, all record locks are forfeited. However, APPX never automatically issues a ROLLBACK.

```
••••• ROLLBACK ••• •••••••••••••••••••• •••
(1)            (2) (3)                  (4)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or constant

(4) Occurrence (constant/index)
```

Using the Statement

The ROLLBACK statement can refer to a savepoint established by a previous SAVEPNT statement. If you include a savepoint name (either as a constant or as a field value) in a ROLLBACK statement, changes made since that savepoint are discarded but changes made prior to the savepoint are not. Without a savepoint, the entire transaction is discarded.

> Note, however, that APPXIO files do not support transaction processing. If you ROLLBACK a change to an RDBMS-hosted file, any changes made to APPXIO will not be rolled back. Related to this is the issue of visibility. Changes made to an RDBMS-hosted file are invisible to other users until they are committed. Changes made to an APPXIO-hosted file are visible immediately.

Related PDFs

COMMIT MODE

Related Statements

COMMIT, SAVEPNT

Example

In the following example, if the flag indicates it is OK to commit, the COMMIT takes place and processing continues. Otherwise, changes made since the savepoint designated as TRX POINT 1 are discarded and the program branches to an applicable routine.

```
          SAVEPNT      TRX POINT 1
          *
          <perform RDBMS processing>
          *
          IF       TAR WORK OK TO COMMIT          NE     Y
    T     ROLLBACK     TRX POINT 1
    T     GOTO         :TRX POINT 1 STATUS
          COMMIT
```

RUN

The RUN statement transfers processing control from the current process to an external program.

```
    ••••• RUN      ••• ••••••••••••••••••••• •••              END? •  FAIL •
    (1)            (2) (3)                    (4)                 (5)     (6)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Program name or field containing it

(4) Occurrence (constant/index)

(5) End current process first? (Y/N)

(6) Fail action (0=nothing, 1=warning, 2=error, 3=cancel)
```

Sets True/False Status Indicator

If the designer specifies that a RUN is not required and the invocation fails, the next true/false flag is set to *F*. If the invoked process completes successfully, the next true/false flag is set to *T*. If the process cannot be invoked successfully, the *FAIL* parameter determines what additional action to take.

Using the Statement

RUN is used to invoke an external program in the same way that it would be invoked from the operating system command processor. As parameters are passed using the PASS statement, they are assigned a number. Each parameter can be referred to in the command line by using dollar sign ($) plus the parameter number. Using dollar sign ($) plus * refers to any unused remaining parameters. If no dollar signs ($) are used, APPX ignores all parameters. If you need to use a dollar sign ($) in the command line, precede it with a backslash (\). On the DOS platform, use ~ (tilde).

The *END?* parameter is presently ignored.

If you PASS a file to the RUN statement, APPX supplies the full operating system path name.

The value returned from a RUN function is put in the four-byte binary predefined field RETURN CODE.

Passing Values

The RUN statement passes the current PASS list, if any. Refer to the PASS Lists section for details.

Related PDFs

RETURN CODE

Related Statements

CALL, PASS

Example

In the following example, RUN is used to invoke the operating system's "display" utility. The name of this program is stored in the field OS DISPLAY UTILITY in application ABC.

```
          RUN      ABC OS DISPLAY UTILITY                      END? Y  FAIL 3
```

SAVEPNT

The SAVEPNT statement allows you to establish a savepoint within a transaction.

```
    ••••• SAVEPNT  ••• ••••••••••••••••••••• •••
    (1)            (2) (3)                   (4)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or constant

(4) Occurrence (constant/index)
```

Using the Statement

The SAVEPNT statement provides for rolling back a partial transaction; it marks a point within a transaction and establishes a savepoint name, either as a constant or as a value within the field specified. A subsequent ROLLBACK statement can then refer to the savepoint, discarding changes made after the savepoint but keeping changes made prior to the savepoint. Without a savepoint, the entire transaction is discarded.

> Note, however, that APPXIO files do not support transaction processing. If you ROLLBACK a change to an RDBMS-hosted file, any changes made to APPXIO will not be rolled back. Related to this is the issue of visibility. Changes made to an RDBMS-hosted file are invisible to other users until they are committed. Changes made to an APPXIO-hosted file are visible immediately.

Related PDFs

COMMIT MODE

Related Statements

COMMIT, ROLLBACK

Example

In the following example, if the flag indicates it is OK to commit, the COMMIT takes place and processing continues. Otherwise, changes made after the savepoint designated as TRX POINT 1 are discarded and the program branches to an applicable routine.

```
        SAVEPNT      TRX POINT 1
        *
        <perform RDBMS processing>
        *
        IF       TAR WORK OK TO COMMIT          NE     Y
   T    ROLLBACK     TRX POINT 1
   T    GOTO         :TRX POINT 1 STATUS
        COMMIT
```

SCAN

The SCAN statement provides a means for extending the functionality of the SCAN option during input processes. (See the User Manual for a full discussion of the SCAN option.)

```
••••• SCAN     ••• ••••••••••••••••••••• ••• =  ••• ••••••••••••••••••••• •••
(1)            (2) (3)                    (4)    (5) (6)                    (7)
```

(1) T/F execution conditions	(5) Source application ID
(2) Destination application ID	(6) Source field name, PDF, or constant
(3) Destination field name or PDF	(7) Source occurrence (constant/index)
(4) Destination occ (constant/index)	

Using the Statement

The SCAN statement becomes effective only when the user has selected the SCAN option, or when the predefined field OPTION is set to SCAN with statements.

The SCAN option permits the user to select a record in a referenced file, and then automatically moves the key for that file into the corresponding fields of the input record. The SCAN statement provides a more general capability for moving fields from a user-selected record into corresponding fields of another record.

The source field of the **SCAN** statement must be a field in an indexed file. Multiple **SCAN** statements may be executed. When the user selects the SCAN option, the names of the files containing the source fields of the **SCAN** statement(s), as well as the name of the file containing the data lookup field on which the cursor is positioned (if data lookup is defined), are displayed to the user for selection. If there is only one such file, that file is selected automatically.

If the user scans and selects a record from the selected file, the values in the source field of each **SCAN** statement (for source fields in the selected file only) are moved into the destination field of the **SCAN** statement. If any of these destination fields are defined as input items, the new values for these fields are displayed on the image.

The **SCAN** statement can be used to handle situations in which a field is both a key to the referenced file and part of a key to another referenced file. It can also be used to automatically set fields other than the key field when the user selects a record in the referenced file. Lastly, the **SCAN** statement can be used to access the process control file.

Restrictions

The **SCAN** statement can be used only in image-related event points in input processes.

Related PDFs

OPTION

Related Statements

AT FIELD

Example

In the following example, if the SCAN option is selected when the cursor is positioned at the substitute product ID field, the SCAN statement is executed to enable the user to scan the PRODUCT file.

```
          IF       --- OPTION                     EQ     SCAN
    T     AT FIELD TIC SUBSTITUTE PRODUCT ID      (AT APPEARANCE #    )
    TT    SCAN     TIC SUBSTITUTE PRODUCT ID      =  TIC PRODUCT ID
```

SCRATCH

The SCRATCH statement closes a file if it is open, and then deletes the file.

```
    ••••• SCRATCH  ••• ••••••••            FAIL • CACHE? •
    (1)            (2) (3)                     (4)      (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) File name

(4) Fail action (0=nothing, 1=warning, 2=error, 3=cancel)
```

```
(5) Cache? (Y=cache version, N=disk version)
```

Sets True/False Status Indicator

The SCRATCH statement modifies the current status indicators by setting the next indicator to *T* if the file does not exist following the SCRATCH, and to *F* if the file still exists. If the next indicator is set to *F*, the *FAIL* parameter determines what additional action to take. In addition, on a false condition, the predefined field STATUS CODE returns a token indicating the reason for an I/O command failure. This token can be used as a key to read the system message file to obtain a descriptive I/O error message (see STATUS CODE for details).

> Note that the next indicator is set to *T* if the file does not exist when the SCRATCH statement executes. Scratching a nonexistent file is considered successful because the file does not exist following the SCRATCH statement.

Using the Statement

The SCRATCH statement allows you to indicate whether to scratch the cached version of a file or the disk version.

When scratching a file, you can specify a database other than the current one by setting the predefined field NEXT DATABASE. If NEXT DATABASE is blank, the current database (contained in DATABASE) is used. If NEXT DATABASE is specified in conjunction with *CACHE?=Y*, then NEXT DATABASE is cleared and ignored as cache files, like memory files, have only one version.

> Note that NEXT DATABASE is modifiable, whereas DATABASE is not, and that NEXT DATABASE is set to blank by the SCRATCH statement.

It is possible to have several versions of a file open at the same time by using the OPEN statement on different databases. The version of a file that is scratched is the last database accessed with the OPEN statement for that file. Refer to the OPEN statement for additional explanation.

Restrictions

If a file is scratched, it must be created again before it can be used.

Statement Ignored

A SCRATCH statement can fail when the file is physically open for use by another user, or when the current user has insufficient privileges to scratch the file.

Related PDFs

DATABASE, NEXT DATABASE

Related Statements

CLOSE, CREATE, IF EXIST, OPEN

Example

In the example below, a work file called TEMP is initialized by using the SCRATCH and CREATE statements.

```
          SCRATCH  TAR TEMP                FAIL 0 CACHE? N
          CREATE   TAR TEMP      SHARE? Y  FAIL 0 CACHE NONE
```

SET

SET copies the contents of a source field to a destination field.

```
••••• SET      ••• ••••••••••••••••••••• ••• =  ••• ••••••••••••••••••••• •••
(1)            (2) (3)                    (4)    (5) (6)                    (7)
```

(1) T/F execution conditions	(5) Source application ID
(2) Destination application ID	(6) Source field name, PDF, or constant
(3) Destination field name or PDF	(7) Source occurrence (constant/index)
(4) Destination occ (constant/index)	

Using the Statement

The contents of the source field are not modified. If the source and destination fields are of different types, conversion is performed automatically. The destination field must be either a field name or a modifiable predefined field. The source field can be a field name, a predefined field, or a constant. Constants are considered to be the same field type as the destination.

Specific information regarding the SET statement is provided in the following sections:

- When the SET Destination is Alpha
- When the SET Destination is a Number
- When the SET Destination is a Date
- When the SET Destination is a Logic Field

Restrictions

The destination field must be either a field name or a modifiable predefined field.

For an alpha destination, if the source field is null, the intermediate alpha form is set to blank. If the converted value is too large to fit in the destination, it is truncated.

For a numeric destination, if the destination field is not large enough to hold the integer part of the source field value, a numeric overflow condition results. Overflow processing is dependent on the use of OVERFLOW statements within the event point containing the **SET** statement. If the source field is null or blank, the destination field is set to its null value. In addition, a valid source field must contain only digits, an optional decimal point, an optional leading or trailing sign, and optional leading and/or trailing spaces.

For a date destination, unless the source field is null, it must contain exactly 16 decimal digits. A date constant must be exactly 16 characters in length, without leading or embedded blanks, and with trailing blanks. Only those characters of the date constant that are within the range of the date parts of the destination date field must be valid date values.

Execution Errors

For numeric and date destinations, a non-recoverable error occurs at runtime if a restriction (as described in the appropriate section) is not complied with.

Related Statements

CNV TEXT, OVERFLOW

Example

The following subroutine is an example of the use of the SET statement. Note that some fields are alpha, some are numeric, some are dates, and some are constants. The example includes both alpha and numeric constants (since constants are deemed to be the same field type as the destination field).

```
          LABEL    DEFAULT MTDPOST
          SET      TAR MTDPOST FULL              =  TAR ADJUST2 FULL
          SET      TAR MTDPOST AMOUNT            =  TAR ADJUST2 AMOUNT
          COMPUTE  TAR MTDPOST AMOUNT            *      -1
          IF       TAR PARAM POST TO GL          EQ     1
     T    IF       TAR MTDPOST AMOUNT            GE     0
     TT   SET      TGL POST ADJUSTMENT           =      0
     TF   SET      TGL POST ADJUSTMENT           =      1
          SET      TAR MTDPOST TRX DATE          =  TAR ADJUST2 TRX DATE
          SET      TAR MTDPOST APPLY TO          =  TAR ADJUST2 APPLY TO
          RETURN
```

When SET Destination is Alpha

Formatting characteristics

The data dictionary record for an alpha destination field contains formatting characteristics that affect the results of the **SET** statement. These characteristics include justification, pad character, and word wrap.

Source alpha

Alpha source fields require no conversion before the beginning of the copying operation.

Source date

The destination field is set to blank if the date is null or blank; otherwise, it is set to 16 digits representing century, year, month, etc. If necessary, the destination is padded with blanks. A non-recoverable error occurs if the destination is less than 16 characters long.

Source dates are converted to an intermediate Gregorian alpha date form before the copy operation, with a mask value of *0807060504030201*.

Source number

The source is converted from its packed or binary form into an intermediate alpha form before the copy operation. The intermediate alpha form is in the format *xxxxxxxx.yyy-*, with the number of *x* and *y* digits, and the existence of the sign, determined by the source field. If the source field is null, the intermediate alpha form is set to blank. If the converted value is too large to fit in the destination, it is truncated. You can use the CNV TEXT statement if formatting is desired.

Source logic

If the value of the source field is *1* (yes), the destination field is set to a value of *Y*. If the value of the source field is *0* (no), the destination field is set to a value of *N*. If the value of the source is blank, the destination field is set to blank.

When SET Destination is a Number

The numerical value of the source field, determined as described below, is copied to the destination field. No formatting is performed on the destination field, which is stored in the binary or packed form appropriate to the destination field's definition in the data dictionary. The source field value is rounded to the decimal positions of the destination field. If the destination field is not large enough to hold the integer part of the source field value, a numeric overflow condition results. Overflow processing is dependent on the use of OVERFLOW statements within the event point that contains the **SET** statement.

Source number

Number source fields are copied using conversion routines appropriate to the type and decimal positions of the source and destination fields. If the source field is null, the destination field is set to the destination field's null value.

Source alpha

The source field is tested for a valid numeric value. To be valid, the source field must contain only digits, an optional decimal point, an optional leading or trailing sign, and optional leading and/or trailing spaces. If the source field is blank, the destination field is set to the destination field's null value. If the source field does not contain a number, a non-recoverable error occurs.

Source date

The source field is converted to a packed decimal Gregorian date intermediate form, if necessary, before the copy operation. The integer packed decimal number representing the date is then copied to the destination field, ignoring the decimal positions of the destination field. For example, if the source field contains the value for January 1, 1988, and the destination field is defined with four digits to the left of the decimal point and two digits to the right of the decimal point, the destination field receives the value *880101.00*.

Source logic

If the value of the source field is *1* (yes), the destination field is set to a value of *1*. If the value of the source field is *0* (no), the destination field is set to a value of *0*. If the value of the source field is blank, the destination field is set to null, according to the destination field's definition in the data dictionary.

When SET Destination is a Date

If the source field is null, the destination date field is set to the null value for the date storage type of the destination field.

Source date

Date source fields are copied using conversion routines appropriate to the date type of the source and destination fields. The value of the source field is then copied to the corresponding common date parts of the destination field. For example, if the destination field contains date parts of century, year, month, day, and hour, and the source field contains date parts of day, hour, and minute, the copy operation copies only the day and hour parts of the source field to the day and hour parts of the destination field. The remaining parts are filled with null values.

Source alpha

Unless the source field is null, it must contain exactly 16 decimal digits, which are then converted into a packed decimal number intermediate form. The packed decimal number is then treated as a packed decimal Gregorian date with all date parts present, and the intermediate form is copied using a conversion routine appropriate to the date type of the destination field. A source field that contains something other than 16 digits causes a non-recoverable error to occur.

Source number

If the source field is negative, the destination field is set to the null value for the date type of the destination field.

Otherwise, the number is converted to an intermediate form, packed decimal number, padded with leading zeros or truncation on the left, so that the number of digits in the intermediate form matches the number of digits in the date parts of the destination field. The intermediate form is then treated as a Gregorian packed decimal date and is copied using a conversion routine appropriate to the date type of the destination field. Any assumed decimal positions in the source field are ignored, and the packed decimal number is processed as if it were an integer value.

Source logic

A Gregorian packed decimal date intermediate form is used. If the value of the source field is *1* (yes), the intermediate form is set to a value of all *1*s. If the value of the source field is *0* (no), the intermediate form is set to a value of *0.* If the value of the source field is blank, the intermediate form is set to a value of *-1* (the null value for packed decimal dates). The intermediate form is then copied using a conversion routine appropriate to the date type of the destination field.

Source constant

A date constant must be exactly 16 characters in length, without leading or embedded blanks, and with trailing blanks. The 16 characters are treated as if they were a Gregorian alpha date of the form *ccyymmddhhmmssth*. However, only those characters of the date constant that are within the range of the date parts of the destination date field must be valid date values. For readability, the "unused" components of the date constant may be set to "-." For example, if the destination field is a *yymmdd* field, a valid date constant representing March 15, 1988, would be *--880315*--------.

Date constants are copied using conversion routines appropriate to a Gregorian alpha date and the date type of the destination field.

When SET Destination is a Logic Field

The destination field is set to blank, 1 (yes), or 0 (no), based on the value of the source field.

Source logic

The source field is copied directly to the destination field.

Source alpha

If the value of the first position of the source field is *0, N*, or *n*, the destination field is set to a value of *0* (no). If the value of the first position of the source field is blank, the destination field is set to a value of blank. Otherwise, the destination field is set to a value of *1* (yes).

Source number

If the source field is null, the destination field is set to blank. Otherwise the destination field is set to *0* (no) if the source field is equal to *0*, or *1* (yes) if the source field is not equal to *0*.

Source date

If the source field is null, the destination field is set to blank. If the source field is a Gregorian binary date and has a value of *0*, the destination field is set to *0* (no). Otherwise, the destination field is set to a value of *1* (yes).

SET DATE

The SET DATE statement places the current date/time in the destination field.

```
••••• SET DATE ••• ••••••••••••••••••••• •••
(1)            (2) (3)                   (4)
```

```
(1) T/F execution conditions

(2) Destination application ID

(3) Destination field name or PDF

(4) Destination occurrence (constant/index)
```

Using the Statement

> Note that to set a date/time field to a constant value, you use the SET command.

Restrictions

The destination field must be a date/time field type.

Related Statements

SET, SET DAY, SET MNTH

Example

In the example below, the current date is supplied for the user if a date is blank.

```
         *
         *       If blank, set adjustment date to system date
         *
```

```
         IF       TAR ADJUST2 ADJ DATE           EQ
    T    SET DATE TAR ADJUST2 ADJ DATE
```

SET DAY

The SET DAY statement extracts the day of the week from a date/time field.

```
••••• SET DAY  ••• ••••••••••••••••••••• ••• •  ••• ••••••••••••••••••••• •••
(1)            (2) (3)                   (4)(5) (6) (7)                   (8)
```

(1) T/F execution conditions	(5) Type (S=short name, L=long name)
(2) Destination application ID	(6) Source application ID
(3) Destination field name or PDF	(7) Source field name, PDF, or constant
(4) Destination occ (constant/index)	(8) Source occurrence (constant/index)

Using the Statement

If the type (item 5) is *S* (short), the result is returned in the destination field as three characters (for example, “Wed”). If the type is *L* (long), the result is returned in the destination field as up to nine characters (for example, “Wednesday”). If the source field is blank, the destination field is set to blank.

Restrictions

The destination must be an alpha field type and the source must be a date/time field type containing at least year, month, and day.

Execution Errors

If the source field is one of the Gregorian date/time field types, a non-recoverable error occurs if it does not contain the year, month, and day parts. If the source field is one of the Julian date/time field types, a non-recoverable error occurs if it does not contain the year and day parts.

Related Statements

SET, SET DATE, SET MNTH

Example

The following example begins with an adjustment date, and retrieves and displays all of the days of the week, beginning with that day.

```
         *
         *        Set and display days of the week
         *
         SET      TAR WORK DATE                 =  TAR ADJUST2 ADJ DATE
```

```
          BEG LOOP X  = 001 TO 007  STEP 001
          SET DAY  TAR WORK DAY DESCRIPT      X   S  TAR WORK DATE
          DATE ADD TAR WORK DATE                  5      1
          DISPLAY  TAR WORK DAY DESCRIPT      X   (AT APPEARANCE #    )
          END LOOP X
```

SET MNTH

The SET MNTH statement extracts the month of the year from a date/time field.

```
••••• SET MNTH ••• ••••••••••••••••••••• ••• •  ••• ••••••••••••••••••••• •••
(1)            (2) (3)                    (4)(5) (6) (7)                    (8)
```

(1) T/F execution conditions	(5) Type (S=short name, L=long name)
(2) Destination application ID	(6) Source application ID
(3) Destination field name or PDF	(7) Source field name, PDF, or constant
(4) Destination occ (constant/index)	(8) Source occurrence (constant/index)

Using the Statement

If the type (item 5) is *S* (short), the result is returned in the destination field as three characters (for example, “Sep”). If the type is *L* (long), the result is returned in the destination field as up to nine characters (for example, “September”). If the source field is blank, the destination field is set to blank.

Restrictions

The destination must be an alpha field type and the source must be a Gregorian date/time field type with at least the month part.

Execution Errors

A non-recoverable error occurs at runtime if the source field does not include the month part.

Related Statements

SET, SET DATE, SET DAY

Example

The following example with TAR WORK FISCAL MONTH set to the first month of the accounting year. The routine then retrieves and displays all the months in a year beginning with that start month.

```
          *
          *        Set and display calendar months
          *
          SET      TAR WORK MM                    =  TAR WORK FISCAL MONTH
```

```
BEG LOOP X  = 001 TO 012  STEP 001
SET MNTH TAR WORK MONTH DESCRIPT    X   S  TAR WORK MM
DATE ADD TAR WORK MM                    6      1
DISPLAY  TAR WORK MONTH DESCRIPT    X   (AT APPEARANCE #    )
END LOOP X
```

SET TEMP

The SET TEMP statement copies the contents of a field to or from the selected predefined TEMP field. The statement identifies substrings of the selected predefined TEMP field and the field specified in the statement.

```
••••• SET TEMP ••••• AT ••• FOR ••• •••• ••• OF  ••• ••••••••••••••••••••• •••
(1)            (2)      (3)     (4) (5)  (6)     (7) (8)                    (9)
```

(1) T/F execution conditions	(6) Field start (constant/index)
(2) TEMP length (constant/index)	(7) Field application ID
(3) TEMP start (constant/index)	(8) Field name, PDF, or constant
(4) Length to copy (constant/index)	(9) Field occurrence (constant/index)
(5) Direction (FROM or INTO)	

Using the Statement

The TEMP field to be used is specified by the length supplied; all predefined TEMP fields have the same characteristics except for their length. The user specifies which TEMP field is to be used by entering the appropriate length.

The TEMP length, starting character, and length to copy can be either predefined fields (A through Z or AI through ZI), or they can be specified as constant positive integer values. Note that the best performance can be obtained by specifying the TEMP starting length and length to copy as constants, or by using the integer AI through ZI fields as parameters.

The direction “from/into” indicates which field is the source and which field is the destination. If it specifies the FROM option, data is copied from the field specified in the statement to the selected predefined TEMP field. If the direction specifies the INTO option, data is copied from the selected predefined TEMP field into the field specified in the statement. This field must be alphanumeric.

The TEMP starting parameter identifies the starting position for the selected predefined TEMP field. Characters are copied to or from the selected predefined TEMP field beginning with this position. This can either be a constant or an index (predefined fields A through Z or AI through ZI). If this parameter is a fraction, its value is rounded to the nearest integer. (The contents of the actual predefined field, A through Z or AI through ZI, remain unchanged.)

The length parameter identifies the number of characters to be copied. This can either be a constant or an index. If this parameter is a fraction, its value is rounded to the nearest integer. (The contents of the actual predefined field, A through Z or AI through ZI, remain unchanged.) If, after rounding, the length parameter is greater than the maximum length available in the source field, the latter is used as the length parameter. This maximum length is determined by calculating the number of bytes that remain in the source field from its specified start position to the end of the field.

The field starting parameter determines the location in the field where copying begins. This can either be a constant or an index. If this parameter is a fraction, its value is rounded to the nearest integer. (The contents of the actual predefined field, A through Z or AI through ZI, remain unchanged.) The number of characters to copy is determined by the length parameter described above, and can be constrained by the length of the destination field. In other words, characters are copied up to the number specified by the length parameter, or until the destination field is filled, whichever comes first.

The justification type and pad character are ignored. The characters copied are always stored left justified in the destination field. If padding is required, the space character is used. Ending spaces are copied unchanged. The word-wrap option is ignored. No formatting takes place.

Restrictions

The INTO option is invalid for constants and for predefined fields that are defined to be non-modifiable. It is valid to use the selected predefined TEMP field also as the field specified in the statement. The copy performs the same as it would if the source field were a different field.

Execution Errors

If, after rounding, the value of the TEMP starting parameter is less than *1*, or greater than the length of the selected predefined TEMP field, a non-recoverable runtime error occurs.

If, after rounding, the value of the length parameter is less than *1*, a non-recoverable runtime error occurs.

If, after rounding, the value of the field starting parameter is less than *1*, or greater than the size of the field specified in the statement, a non-recoverable error occurs.

Related PDFs

A thru Z, A1 thru Z1, TEMPn

Related Statements

SET

Example

In the following example, SET TEMP is used to obtain characters 2, 3, and 4 of WORK CHECK.

```
          SET TEMP 79    AT 001 FOR 003 FROM 002 OF  TAP WORK CHECK
```

In this next example, the fifth and sixth characters of TEMP 80 are moved into the seventh and eighth character positions of WORK CHECK.

```
          SET TEMP 80    AT 005 FOR 002 INTO 007 OF  TAP WORK CHECK
```

SET TOT

The SET TOT statement is used to obtain the value of an end-of-range total that was accumulated automatically or using the TOTAL statement. SET TOT retrieves the total value and places it in the destination field. The value is dependant on the current sort level. Refer to the TOTAL statement for additional information.

```
••••• SET TOT  ••• ••••••••••••••••••••• ••• =  ••• ••••••••••••••••••••• •••
(1)            (2) (3)                    (4)    (5) (6)                    (7)
```

(1) T/F execution conditions	(5) Source application ID
(2) Destination application ID	(6) Source field name, PDF, or constant
(3) Destination field name or PDF	(7) Source occurrence (constant/index)
(4) Destination occ (constant/index)	

Using the Statement

SET TOT is used in *End-of-Rang*e frames in output and update processes.

The source and destination fields can be the same. In this case, the total for that field is moved from the total area into the current area for that field.

Restrictions

Both the source and destination fields must be numeric.

Execution Errors

If the destination field is not large enough to hold the integer portion of the totaled field, a non-recoverable error occurs, unless overflow processing is present in the event point.

Related Statements

OVERFLOW, TOTAL

Example

In the following example, the value in the WORK CASH RECEIVED work field is placed in the WORK AMOUNT work field.

```
          SET TOT  TAR WORK AMOUNT               =  TAR WORK CASH RECEIVED
```

STATIC

The STATIC keyword is reserved for future use.

STATUS

The STATUS statement allows you to alter the flow of processing by invoking an APPX status process from within an ILF routine.

```
    ••••• STATUS   ••• ••••••••••••••••••••••••••••••   ••••••••••   END? •  FAIL •
    (1)            (2) (3)                              (4)              (5)     (6)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Status process name

(4) Invocation type (SUBPROCESS, RELATED, DETACHED)

(5) End current process first? (Y/N)

(6) Fail action (0=nothing, 1=warning, 2=error, 3=cancel)
```

Sets True/False Status Indicator

The next status indicator is set to *T* if the process is invoked successfully, and to *F*, otherwise. If the process cannot be invoked successfully, the *FAIL* parameter determines what additional action to take.

Using the Statement

In addition to the application ID and status process name to be invoked, you specify the invocation type (subprocess, related, or detached). As with all other child invocations, this parameter controls the level of data sharing between the parent and the status process that is invoked.

If *END?* is set to *Y*, the current process will end before invoking the status process. The status process will return to the parent of the current process when it is finished. If *END?* is set to *N*, the status process returns to the current process (beginning with the first statement after the **STATUS** statement).

If the application ID is blank, the value in the predefined field NEXT APPLICATION is used. If the process name is blank, the value in the predefined field NEXT PROCESS NAME is used.

Passing Values

The STATUS statement passes the current PASS list, if any. Refer to the PASS Lists section for details.

Related PDFs

NEXT APPLICATION, NEXT PROCESS NAME

Related Statements

INPUT, INQUIRY, JOB, MENU, OUTPUT, PASS, QUERY, SUBR, UPDATE

Example

The following example checks to see if it is the last processing day in the month. If so, the **STATUS** statement is executed, invoking an end-of-month file status operation.

```
          IF       TAR WORK END OF MONTH          EQ     Y
    T     STATUS   TAR END OF MONTH STATUS              SUBPROCESS   END? Y  FAIL 3
```

STORE

The STORE statement saves the contents of a field or record in the area assigned as the store area for the field or record. These contents can later be restored into the field or record with the RESTORE statement.

```
    ••••• STORE    ••• ••••••••••••••••••••• ••• •••••••••••••
    (1)            (2) (3)                    (4) (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) File name, field name, or predefined field

(4) Occurrence (constant/index)

(5) Data item type (RECORD, FIELD, DEFAULT RECORD, DEFAULT FIELD)
```

Using the Statement

The STORE statement allows the designer to store the values in a field or record so that temporary use of the field or record for accessing other data records or some other processing does not cause the contents of the field or record to be lost permanently.

The data item type specification can specify *FIELD, RECORD, DEFAULT FIELD*, or *DEFAULT RECORD*. If the *FIELD* or *DEFAULT FIELD* option is specified, the data item to be saved must be a field in the data dictionary. If the *RECORD* or *DEFAULT RECORD* option is specified, the data item to be saved must be specified as a file name. In this case, the current contents of the file's record area are saved. The occurrence number has no meaning for the STORE statement with the *RECORD* option.

Fields that are part of a record do not have an independent store or default area. All fields that are part of a record are saved in their proper location in the area assigned for storing the record. Therefore, it is possible to store an entire record and then restore specific fields, or to store only some fields and then restore the entire record. A field or record can be stored any number of times, and it can also be restored any number of times, independently. These statements only cause the contents of the specified field or record to be stored in, or restored from, the store or default area assigned to the field or record. The store area and default area are allocated if a STORE or RESTORE statement is present. They are initialized to the default value; therefore, a RESTORE statement can be executed even if no STORE statement exists for the field or record.

No editing or justification is performed. All data is copied unchanged, and a subsequent RESTORE statement restores the contents of a field or record to the state it was in when the most recent STORE statement was executed.

You can alter the default values of a field or record by using the STORE statement to move current field or record values into the *DEFAULT FIELD* or *DEFAULT RECORD* area.

Restrictions

If the *FIELD* or *DEFAULT FIELD* option is specified, the data item to be saved must be a field in the data dictionary. If the *RECORD* or *DEFAULT RECORD* option is specified, the data item to be saved must be specified as a file name. The occurrence number has no meaning for the STORE statement with the *RECORD* option.

Because fields that are part of a record do not have a separate store or default area assigned to them, storing a record causes any previously saved contents of fields that are part of that record to be lost. Executing a STORE statement for a field that is part of a record after the record is saved with a STORE statement causes the stored record to be modified.

The contents of predefined fields can also be saved using the STORE statement. Therefore, it is invalid to use the STORE statement for predefined fields that are non-modifiable, since the RESTORE command is invalid for those fields.

Related Statements

RESTORE

Example

In the following example, the amount field RECEIPT2 CHECK AMOUNT is saved, presumably because its contents may be affected during the processing of unpaid invoices. After invoice processing is completed, the field is restored.

```
          STORE    TAR RECEIPT2 CHECK AMOUNT      FIELD
          SET      TAR WORK COUNT APPLY TO        =      0
          BEG AT   TAR UNPAID   IN TAR RECEIPT2 CUSTOMER NO
          END AT   TAR UNPAID   IN TAR RECEIPT2 CUSTOMER NO
          BEG READ TAR UNPAID                HOLD 0 KEY IS  UNPAID ALTERNATE KEY
          *        (process unpaid invoices
          END READ TAR UNPAID
          RESTORE  TAR RECEIPT2 CHECK AMOUNT      FIELD
```

SUBR

The SUBR statement allows you to alter the flow of processing by invoking an APPX subroutine process from within an ILF routine.

```
    ••••• SUBR     ••• ••••••••••••••••••••••••••••••   ••••••••••   END? •  FAIL •
    (1)            (2) (3)                              (4)              (5)     (6)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Subroutine process name

(4) Invocation type (SUBPROCESS, RELATED, DETACHED)

(5) End current process first? (Y/N)

(6) Fail action (0=nothing, 1=warning, 2=error, 3=cancel)
```

Sets True/False Status Indicator

The next status indicator is set to *T* if the process is invoked successfully, and to *F*, otherwise. If the process cannot be invoked successfully, the *FAIL* parameter determines what additional action to take.

Using the Statement

In addition to the application ID and subroutine process name to be invoked, you specify the invocation type (subprocess, related, or detached). As with all other child invocations, this parameter controls the level of data sharing between the parent and the subroutine process invoked.

If *END?* is set to *Y*, the current process will end before invoking the subroutine process. The subroutine process will return to the parent of the current process when it is finished. If *END?* is set to *N*, the subroutine process returns to the current process (beginning with the first statement after the **SUBR** statement).

If the application ID is blank, the value in the predefined field NEXT APPLICATION is used. If the process name is blank, the value in the predefined field NEXT PROCESS NAME is used.

Passing Values

The SUBR statement passes the current PASS list, if any. Refer to the PASS Lists section for details.

Related PDF's

NEXT APPLICATION, NEXT PROCESS NAME

Related Statements

COPY, GOSUB, INPUT, INQUIRY, JOB, MENU, OUTPUT, PASS, QUERY, STATUS, UPDATE

Example

The following example checks to see if it is the last processing day in the month. If so, the **SUBR** statement is executed, invoking an end-of-month calculation routine.

```
          IF       TAR WORK END OF MONTH          EQ     Y
    T     SUBR     TAR END OF MONTH CALCULATION         SUBPROCESS   END? Y  FAIL 3
```

TIMEOUT

The TIMEOUT statement provides a means for intercepting file-handling statements. The TIMEOUT statement specifies an amount of time and a label.

```
    ••••• TIMEOUT  ••••••••••••••••••••••••••••••    ••• •••••••••••••••••••• •••
    (1)            (2)                               (3) (4)                   (5)
```

```
(1) T/F execution conditions

(2) Label

(3) Application ID
```

```
(4) Numeric field name, PDF, or constant

(5) Occurrence (constant/index)
```

Using the Statement

If any of the read statements take longer than the amount of time specified, processing control is transferred to the statement immediately following the specified label.

Statements that can cause a TIMEOUT condition are BEG READ/END READ, READNEXT, and READ.

The numeric field or constant represents an amount of time in hundredths of a second. If the amount of time is zero and the requested record is being held by another user, the TIMEOUT condition occurs immediately. If the amount of time is not zero, a TIMEOUT condition occurs if the requested record remains held for the specified length of time. Note, however, that although APPXIO honors TIMEOUT values, other data access methods such as Oracle may not. In this case, the amount of time specified will be treated as if it were zero.

The TIMEOUT statement may be respecified at any time with another label, or the intercept may be disabled by a TIMEOUT statement with no label specified. If the intercept is disabled, the statements BEG READ/END READ, READ, and READNEXT will wait indefinitely for the requested record to become available.

Compilation Errors

A process compile error occurs if the referenced label is not defined within the same event point as the TIMEOUT or in a subroutine copied into the event point with the COPY statement.

Related Statements

BEG READ, END READ, READ, READNEXT

Example

In the following example, if the READ on the customer file takes longer than the amount of time defined in WORK WAIT COUNTER, processing control transfers to the statement following the label :CUST RECORD NOT AVAILABLE.

```
          TIMEOUT  :CUST RECORD NOT AVAILABLE        TAR WORK WAIT COUNTER
          READ     TAR CUSTOMER              HOLD 0 FT 0 BY CUSTOMER NO
          *        (process customer record)
          *        (branch or END to avoid executing customer not available code)
          *
          LABEL    :CUST RECORD NOT AVAILABLE
          *        (handle customer record not available condition)
```

TOTAL

The TOTAL statement accumulates subtotals and grand totals in output and update processes.

```
••••• TOTAL    ••• ••••••••••••••••••••• ••• •  ••• ••••••••••••••••••••• •••
(1)            (2) (3)                  (4)(5) (6) (7)                   (8)
```

(1) T/F execution conditions	(5) Operator (+, -, P, M)
(2) 1st application ID	(6) 2nd application ID
(3) 1st field name or PDF	(7) 2nd field name, PDF, or constant
(4) 1st occurrence (constant/index)	(8) 2nd occurrence (constant/index)

Using the Statement

TOTAL adds to (+), hash-adds to (P), subtracts from (-), or hash-subtracts from (M) the first operand using the amount indicated in the second operand. However, rather than storing the result in the the first operand, the total is accumulated internally within APPX.

TOTAL is a special use of a numeric field that occurs only in output and update processes. TOTAL maintains running totals that can be retrieved and used in *End of Range* frames. Totaled fields can also be automatically displayed on end of range images.

APPX dynamically maintains a numeric field for each sort level of a totaled field. It does not alter the contents of the field being totaled. As each level is subtotaled, the corresponding value of the totaled field is added to the next level and then cleared. In a subtotal image, a user can define the field being totaled on the image and request a display of the “range” value. This causes APPX to display the value that was totaled for that sort level.

Using the sample code in the *Example* section below, assume that a cash receipts report is being printed with subtotals by account number and date, as well as grand totals. In this case, APPX allocates three work fields for total receipts. When the TOTAL statement is executed, APPX adds to the account number subtotal. When the account number subtotal prints, APPX adds the account number subtotal to the date subtotal and then zeros the account number subtotal. When the date subtotal prints, APPX adds the date subtotal to the grand total and then zeros the date subtotal. At the end of the report, APPX prints the grand total.

A user can retrieve a totaled value by using the SET TOT statement at the subtotal or grand total level. Thus, in our example, you can retrieve the value of WORK CASH RECEIVED in any subtotal or grand total image with the following statement:

```
          SET TOT  TAR WORK AMOUNT               =  TAR WORK CASH RECEIVED
```

Related Statements

SET TOT

Example

In the following example, the TOTAL statement is used to accumulate daily cash receipts. Subtotals or grand totals can be printed on reports, or extracted using the SET TOT statement.

```
          TOTAL    TAR WORK CASH RECEIVED         +  TAR RECEIPT2 CHECK AMOUNT
```

TRAP

The TRAP statement temporarily halts execution at runtime to permit debugging. Debugging permits "browsing" through the statements in an ILF routine, setting and clearing TRAP statements in those event points, stepping through the statements or event points, and inspecting and/or modifying variables.

```
    ••••• TRAP
    (1)
```

```
(1) T/F execution conditions
```

Using the Statement

A TRAP statement can be used in any ILF routine in any event point in any process. The true/false indicators can be used to make a TRAP conditional. See Chapter 4-5: The Debugger for details on using the debugger.

Restrictions

TRAP is executed only when the application is invoked from "Test Application" within the application design utilities, or when invoked from the Toolbox there.

Statement Ignored

TRAP is ignored if the application is invoked from "Select Database."

Example

In the following example, the TRAP statement allows the designer to explore and observe the execution of this routine, which assigns a batch control number to a group of adjustments.

```
          LABEL    :CHECK TRXCNTRL
          TRAP
          IF       TAR PARAM TRXCNTRL             EQ     1
    T     SET      TAR TRXCNTRL CONTROL NO         =  TAR PARAM ADJUST CONTROL
    T     COMPUTE  TAR PARAM ADJUST CONTROL       +      1
    T     SET      TAR TRXCNTRL APPLICATION       =      TAR
    T     SET      TAR TRXCNTRL FILE POSTED       =      ADJUSTMENTS
```

```
    T     SET      TAR TRXCNTRL OPERATOR ID       =  --- USER ID
    T     READ     TAR TRXCNTRL              HOLD 1 FT 1 BY TRXCNTRL KEY
    TT    GOTO     :CHECK TRXCNTRL
```

ULINE

Items are normally displayed without being underlined on the image. ULINE displays an item with an underline.

```
    ••••• ULINE    ••• •••••••••••••••••••• ••• (AT APPEARANCE # •••)
    (1)            (2) (3)                  (4)                  (5)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Field name or predefined field

(4) Occurrence (constant/index)

(5) Appearance (constant/index)
```

Using the Statement

The appearance number can be used if the same field appears more than once on the image.

If the application ID, field name, occurrence number, and appearance number are not specified (left blank), the ULINE statement affects all of the items painted on the associated image.

ULINE is one of the *characteristic-type* statements. Refer to the common attributes of all the statements in this category in the Characteristic-Type Statements section of this manual.

The operation of a ULINE statement affects only the next generation of an image. In other words, the item characteristics are reset each time an image is produced.

Restrictions

> Note that, because characteristic statements operate on items rather than fields, the characteristic statements are only meaningful in event points that are associated with an image.

Statement Ignored

The system ignores any **ULINE** statements specified in non-image event points.

Related Statements

BLANK, BLINK, BOLD, BRIGHT, COLOR, DIM, DISPLAY, FONT, INVERSE, NO BOLD, NO ULINE, NORMAL, PICTURE

Example

In the example below, the customer's name is underlined on the image.

```
          ULINE    TAR CUSTOMER NAME              (AT APPEARANCE #    )
          DISPLAY  TAR CUSTOMER NAME              (AT APPEARANCE #    )
```

UPDATE

The UPDATE statement allows you to alter the flow of processing by invoking an APPX update process from within an ILF routine.

```
    ••••• UPDATE   ••• ••••••••••••••••••••••••••••••   ••••••••••   END? •  FAIL •
    (1)            (2) (3)                              (4)              (5)     (6)
```

```
(1) T/F execution conditions

(2) Application ID

(3) Update process name

(4) Invocation type (SUBPROCESS, RELATED, DETACHED)

(5) End current process first? (Y/N)

(6) Fail action (0=nothing, 1=warning, 2=error, 3=cancel)
```

Sets True/False Status Indicator

The next status indicator is set to *T* if the process is invoked successfully, and to *F*, otherwise. If the process cannot be invoked successfully, the *FAIL* parameter determines what additional action to take.

Using the Statement

In addition to the application ID and update process name to be invoked, you specify the invocation type (subprocess, related, or detached). As with all other child invocations, this parameter controls the level of data sharing between the parent and the update process that is invoked.

If *END?* is set to *Y*, the current process will end before invoking the update process. The update process will return to the parent of the current process when it is finished. If *END?* is set to *N*, the update process returns to the current process (beginning with the first statement after the **UPDATE** statement).

If the application ID is blank, the value in the predefined field NEXT APPLICATION is used. If the process name is blank, the value in the predefined field NEXT PROCESS NAME is used.

Passing Values

The UPDATE statement passes the current PASS list, if any. Refer to the PASS Lists section for details.

Related PDF's

NEXT APPLICATION, NEXT PROCESS NAME

Related Statements

INPUT, INQUIRY, JOB, MENU, OUTPUT, PASS, QUERY, STATUS, SUBR

Example

The following example checks to see if it is the last processing day in the month. If so, the **UPDATE** statement is executed, invoking an end-of-month file update operation.

```
          IF       TAR WORK END OF MONTH          EQ     Y
    T     UPDATE   TAR END OF MONTH UPDATE              SUBPROCESS   END? Y  FAIL 3
```

USE CC

The USE CC statement takes the child constraints placed on the current process by the parent process, and applies them to the specified file.

```
    ••••• USE CC   ••• ••••••••
    (1)            (2) (3)
```

```
(1) T/F execution conditions

(2) Application ID

(3) File name
```

Using the Statement

A USE CC statement applies the child constraints in force on the current process to the file specified.

Restrictions

USE CC is only allowed...

Statement Ignored

USE CC is ignored if there are no child constraints in force for the current process.

Example

In the following example, ...

```
          IF       TAR WORK END OF MONTH          =      Y
    T     USE CC   TAR CUSTOMER
```

VERIFY

The VERIFY statement causes various functions from the Database Management's Verify File Menus to be executed from within a process.

```
    ••••• VERIFY   ••• •••••••••••••••••••• ••• TYPE •••••••••••••• FAIL •
    (1)            (2) (3)                  (4)      (5)               (6)
```

```
(1) T/F execution conditions

(2) Application ID

(3) File name or field name

(4) Occurrence (constant/index)

(5) Verify type (FIELD, RECORD)

(6) Fail action (0=nothing, 1=warning, 2=error, 3=cancel)
```

Sets True/False Status Indicator

The next level of the internal true/false status indicator is set to *T* if the VERIFY operation is successful; otherwise, the indicator is set to *F*. If the VERIFY fails, the *FAIL* parameter determines what additional action to take.

Using the Statement

A file can be verified to insure that it is structurally sound and that each field (or a specific field) contains data that corresponds to the field type (number, date, etc.).

The *TYPE* parameter indicates what component of the file to verify. The only valid choices, among those presented in a drop-down list, are:

- *FIELD* to verify the contents of a particular field within every record of the file (field name required).

- *RECORD* to verify the contents of all fields within the current record (file name required).

The application ID and file name for the file (or field) to verify are specified in the statement. Additional information that can affect the location of the file may be contained in the predefined field DATABASE.

The file name references a file that is defined in the data dictionary for the selected application ID. You can set the NEXT DATABASE predefined field to the database ID of the database containing the file that you want verified if it is different than the current database.

Execution Errors

Serious problems encountered while trying to verify a file or its contents, such as a file structure/data dictionary mismatch, cause a non-recoverable runtime error to occur, regardless of the *FAIL* parameter.

Related PDFs

DATABASE, NEXT DATABASE

Related Statements

OPEN, CLOSE

Example

The following example verifies the contents of the *CUSTOMER BALANCE* field and returns an error if an inconsistency is found between the Data Dictionary field type and the actual contents of the field in all CUSTOMER file records.

```
          VERIFY   TAR CUSTOMER BALANCE           TYPE FIELD          FAIL 0
    F     CANCEL   Verification of TAR CUSTOMER BALANCE Field Failed!
```

WARNING

WARNING can be used to display a warning message in an input process, or to print a warning message in an output or action process.

```
    ••••• WARNING  ••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••
    (1)            (2)
```

```
(1) T/F execution conditions

(2) Warning message (constant or TEMPn predefined field)
```

Using the Statement

Variable warning messages can be displayed or printed by setting the predefined field TEMPn to the required message, and entering *TEMPn* in place of a message in the **WARNING** statement.

When a warning occurs, the image is redisplayed to the user with the warning message blinking on line two of the status surface. If more than one ERROR, WARNING, or MESSAGE statement is encountered, the user can display the additional messages.

In output processes, errors and warnings are printed on the report and in update, status, and subroutine processes, they are printed to a log.

Related PDFs

TEMPn

Related Statements

ERROR, MESSAGE

Example

In the following example, a warning is issued if the user does not enter a tax code on a customer's record.

```
          IF       TAR CUSTOMER TAX CODE          EQ
    T     WARNING  No sales tax will be collected for this customer.
```

WRITE

The WRITE statement writes a new record to the specified file.

```
    ••••• WRITE    ••• •••••••••••••••••••• FAIL •
    (1)            (2) (3)                        (4)
```

```
(1) T/F execution conditions

(2) Application ID

(3) File name or field name

(4) Fail action (0=nothing, 1=warning, 2=error, 3=cancel)
```

Sets True/False Status Indicator

If a record is written successfully, the current true/false status indicators are modified by setting the next indicator to *T*. If a write fails because of an "Already on File" condition for a unique key in the file, the next indicator is set to *F*. In this case, the *FAIL* parameter determines what additional action to take. In addition, on a false condition, the predefined field STATUS CODE returns a token indicating the reason for an I/O command failure. This token can be used as a key to read the system message file to obtain a descriptive I/O error message (see STATUS CODE for details).

Using the Statement

When using this statement in conjunction with an RDBMS table, you can specify a field name instead of a file name to limit the volume of data being passed between APPX and the RDBMS. See the Partial-Record I/O section in Chapter 4-3: Specifying Statements for more information.

Execution of the **WRITE** statement does not modify the position in the file for subsequent BEG READ/ END READ or READNEXT operations.

Related Statements

DELETE, REWRITE

Example

In the following example, invoice batch control information is transferred to a file called EDIT.

```
          CREATE   TAR EDIT      SHARE? Y  FAIL 0 CACHE NONE
          BEG READ TAR INVOICE1               HOLD 0 KEY IS  INVOICE1 KEY
          SET      TAR EDIT KEY                   =  TAR INVOICE1 KEY
          SET      TAR EDIT FILE NAME             =      INVC
          WRITE    TAR EDIT                   FAIL 0
    F     CANCEL   Unexpected Error WRITE TAR EDIT
          END READ TAR INVOICE1
```

CHAPTER 4-7

Predefined Fields and Processes

Runtime Application

APPX includes a component called the *runtime application*. This application provides a number of *predefined fields* (PDFs) and predefined processes that are described in detail in this chapter. You can make use of the features provided in this application as you develop your own applications.

To refer to the runtime application, you designate the application ID with three hyphens (---) which APPX interprets as the runtime application. (The actual application ID is *0LA*.)

Predefined Fields (PDFs)

In the runtime application, there are a number of fields that contain various "points of reference," such as who the current user is and the company address. Although the names of these fields are pre-specified, you can use them as you would any other fields in your own applications. You can reference predefined fields in statements, for example, and set the values of some of them (such as ALTERNATE IMAGE NUMBER) to control processing flow. APPX keeps track of the values of predefined fields at all times, and maintains their values automatically.

When you use a predefined field, you designate the application ID with three hyphens (---). APPX interprets the three hyphens as the runtime application. (The actual application ID is *0LA*.)

Using Predefined Fields (PDFs)

You can make use of predefined fields in three ways:

- You can display and optionally allow a user to modify the value of a predefined field by painting it on an image. For example, you may want to display the value of COMPANY NAME on an input screen for the user's information.
- Using statements, you can check the value of any predefined field and, based on the result, control the flow of processing. For example, you may want to determine if there were any errors encountered during an output process and, if there were errors, perform a special error-handling routine. You can examine the value of the ERRORS predefined field and, if it is greater than zero, pass control to the error-handling routine.
- You can use statements to set the value of certain predefined fields and control how a process operates. For example, you may want to control which alternate image is to appear on the screen by examining a user's response to a preliminary question. Depending on the response, you can set the value of ALTERNATE IMAGE NUMBER to the appropriate value, and cause the proper image to be displayed.

Categories of PDFs

There are five categories of predefined fields:

- ***Current status PDFs***, which provide information about the current runtime environment, such as the current user ID.
- ***Process control PDFs***, which limit or affect an operation within the application.
- ***Terminal management PDFs***, which can be used to control and provide information about the current cursor position.
- ***Report management PDFs***, which can be used to control and provide information about the disposition and production of reports, such as the print destination.
- ***General purpose PDFs*** are used as index work variables or work fields in ILF statements.

Current Status PDFs

Current status PDFs provide information about the current runtime environment, such as the current user ID. The current status PDFs are:

- APPLICATION
- BILLING CODE
- COLOR BACKGROUND
- COLOR FOREGROUND
- COMPANY ADDRESS 1
- COMPANY ADDRESS 2
- COMPANY ADDRESS 3
- COMPANY NAME
- DATABASE
- DISPOSITION OPTIONS*
- ENTRY LEVEL
- ERRORS
- FONT
- INTERACTIVE PHASE
- KEYMAP NAME
- LANGUAGE ID
- MODE
- NEXT LANGUAGE ID
- OPERATIONAL PHASE

- PROCESS NAME
- PROCESS TYPE
- RECORD NUMBER
- SELECTED APPLICATION
- SELECTED DATABASE
- SELECTED VERSION
- SOURCE APPLICATION
- SOURCE VERSION
- STATUS CODE
- TEXT AT POSITION
- USER ID
- VERSION
- WARNINGS

*Used only during conversions from SPEED II to APPX. See the *SPEED II to APPX Conversion Manual* for more information.

Process Control PDFs

Process control PDFs limit or affect an operation within the application. The process control PDFs are:

- ACCESS PATH
- ALTERNATE IMAGE NUMBER
- AUTO READNEXT
- CANCEL OK
- CHILD ID
- COMMIT MODE
- DEFAULT MODE
- EXACT KEY REQUIRED
- INCLUDE CHILDREN
- INCLUDE RECORD
- NEXT APPLICATION
- NEXT CHILD ID
- NEXT DATABASE
- NEXT PROCESS NAME
- NEXT VERSION
- OPTION

- PROCESS CODE
- RETURN CODE
- SCROLL TO END
- SUBMIT DATE
- SUBMIT MODE
- SUBMIT NOTIFY USER
- SUBMIT PRIORITY
- SUBMIT PROCESS
- SUBMIT QUEUE

Terminal Management PDFs

Terminal management PDFs can be used to control and provide information about the current cursor position. The terminal management PDFs are:

- CURSOR AT APPEARANCE
- CURSOR AT OCCURRENCE
- CURSOR COLUMN
- CURSOR ROW
- PAN COLUMN
- PAN ROW

Report Management PDFs

Report management PDFs can be used to control and provide information about the disposition and production of reports, such as the print destination. The report management PDFs are:

- FORM LENGTH
- FORM WIDTH
- LINES LEFT ON PAGE
- LOG BANNER
- LOG COPIES
- LOG FILE DISPOSITION
- LOG FILE PATHNAME
- LOG FORM ID
- LOG FORMAT
- LOG HARD COPY

- LOG MODE
- LOG NOTIFY USER
- LOG ON SCREEN
- LOG PRINTER ID
- LOG PRIORITY
- LOG REPORT OPTIONS
- LOG SPOOL DATE
- NEW PAGE AFTER
- NEW PAGE BEFORE
- PAGE NUMBER
- PRINT BANNER
- PRINT COPIES
- PRINT FILE BROWSABLE
- PRINT FILE DISPOSITION
- PRINT FILE PATHNAME
- PRINT FORM ID
- PRINT FORMAT
- PRINT HARD COPY
- PRINT MODE
- PRINT NOTIFY USER
- PRINT ON SCREEN
- PRINT PAGE ESTIMATE
- PRINT PRINTER ID
- PRINT PRIORITY
- PRINT RECORD LIMIT
- PRINT REPORT OPTIONS
- PRINT SPOOL DATE
- PRINT SUMMARY ONLY
- SKIP AFTER
- SKIP BEFORE
- SUBHEADING
- SUBMIT OPTIONS

General Purpose PDFs

General purpose PDFs are used as index work variables or work fields in ILF statements. The general purpose PDFs are:

- A, B, C...Z
- AI, BI, CI...ZI
- BETWEEN
- TEMP 1
- TEMP 2
- TEMP 4
- TEMP 30
- TEMP 30 CTR
- TEMP 30 LJ
- TEMP 30 RJ
- TEMP 32
- TEMP 79
- TEMP 80
- TEMP 132
- TEMP 256
- TEMP 512
- TEMP 2K
- TEMP 8K
- TEMP 32K

Information Headings for PDF's

Part II provides a detailed description of each PDF. The following is a brief description of the possible headings for each PDF.

- *Description* explains in as much detail as necessary how the PDF is used.
- *Example* gives a brief example of the usage of the PDF.
- *Domain* describes the domain characteristics of the PDF.
- *Process Sharing* describes how the PDF is shared between processes, if at all.
- *Modifiable?* Yes/No — either the PDF can be modified, or it cannot.
- *Initial Value* is the value to which the PDF is initialized by APPX.

- *Default Value* is the value that is assigned to the field by the system at runtime, if the value is different from the *Initial Value.*
- *Valid Values* describes the valid options for each PDF, if a table of valid values is used.
- *Related PDFs* lists any other PDFs that may be related to, or similar to, this one.

Key to Table 4-7-1: Predefined Fields Attributes

The following table specifies the attributes of each APPX predefined field (PDF). The following items are included in the table:

- The PDF name, which is the exact name of the PDF as you would specify it in a statement.
- The size of the PDF.
- The sign, indicating if the PDF is a signed value.
- The storage type, or how the PDF value is stored.
- The IPC characteristics, provided in the following three columns:
- Pass—the extent to which the PDF value is *passed* to other processes.
- Rec—the extent to which the PDF value is *received* by other processes.
- Share—the extent to which the PDF value is *shared* with parent processes.

The following abbreviations are used to describe the IPC Characteristics:

- sub —communicated with subprocess invocations.
- manual—must be manually communicated with statements.
- prohib—communication is prohibited.
- rcv—communicated with any fully or partially receptive process.
- rel/sub—communicated with all subprocess and related invocations.
- all—communicated with all invocations regardless of the receptivity level.
- Modifiable?—tells whether the PDF value can be changed with statements.
- Initial value—(how the PDF is initialized). When this column contains an asterisk (*), the PDF initial value is controlled by values entered in system administration or at runtime by a user.

Table 4-7-1: Predefined Fields Attributes

Table 4-7-1. Predefined Field Attributes

	DOMAIN INFO			IPC CHARACTERISTICS				
Predefined Field	Size	Sign	Store	Pass	Rec	Share	Modify	Init Val
A, B, C, ... Z	15	N	var dec	sub	rcv	sub	yes	0
AI, BI, CI, ... ZI	10.0	N	binary	sub	rcv	sub	yes	0

	DOMAIN INFO			IPC CHARACTERISTICS				
Predefined Field	Size	Sign	Store	Pass	Rec	Share	Modify	Init Val
ACCESS PATH	30		alpha	rel/sub	all	rel/sub	no	*
ALTERNATE IMAGE NUMBER	3.0		binary	sub	rcv	sub	yes	0
APPLICATION	3		alpha	man	prohib	prohib	no	*
AUTO READNEXT	1		logic	all	man	man	yes	*
BETWEEN	15.0	N	pack dec	sub	rec	sub	yes	0
BILLING CODE	30		alpha	all	all	rel/sub	yes	null
CANCEL OK	1		logic	rel/sub	all	rel/sub	yes	*
CHILD ID	4		alpha	man	prohib	prohib	no	*
COLOR BACKGROUND	12		alpha	rel/sub	all	rel/sub	yes	*
COLOR FOREGROUND	12		alpha	rel/sub	all	rel/sub	yes	*
COMMIT MODE	20		alpha	rel/sub	all	rel/sub	yes	*
COMPANY ADDRESS 1	30		alpha	all	all	sub	no	*
COMPANY ADDRESS 2	30		alpha	all	all	sub	no	*
COMPANY ADDRESS 3	30		alpha	all	all	sub	no	*
COMPANY NAME	30		alpha	all	all	sub	no	*
CURSOR AT APPEARANCE	5.0		binary	sub	rcv	sub	no	0
CURSOR AT OCCURRENCE	5.0		binary	sub	rcv	sub	no	0
CURSOR COLUMN	7.0	N	binary	sub	rcv	sub	yes	1
CURSOR ROW	7.0	N	binary	sub	rcv	sub	yes	1
DATABASE	3		alpha	all	all	sub	no	*
DEFAULT MODE	22		token	all	all	prohib	yes	*
ENTRY LEVEL	1.0		binary	man	prohib	prohib	no	0
ERRORS	10.0		binary	sub	rcv	sub	yes	0
EXACT KEY REQUIRED	1		logic	rel/sub	all	rel/sub	yes	*
FONT	12		alpha	rel/sub	all	rel/sub	yes	*
FORM LENGTH	5.0		binary	sub	rcv	sub	no	*
FORM WIDTH	5.0		binary	sub	rcv	sub	no	*
INCLUDE CHILDREN	1		logic	sub	rcv	sub	yes	yes
INCLUDE RECORD	1		logic	sub	rcv	sub	yes	null
INTERACTIVE PHASE	22		token	man	prohib	prohib	no	null
KEYMAP NAME	80		alpha	rel/sub	all	rel/sub	yes	*
LANGUAGE ID	3		alpha	all	all	sub	no	*

	DOMAIN INFO			IPC CHARACTERISTICS				
Predefined Field	Size	Sign	Store	Pass	Rec	Share	Modify	Init Val
LINES LEFT ON PAGE	10.0	N	binary	binary	rcv	sub	no	0
LOG BANNER	1		logic	rel/sub	all	rel/sub	yes	0
LOG COPIES	5.0		binary	rel/sub	all	rel/sub	yes	*
LOG FILE DISPOSITION	10		token	rel/sub	all	rel/sub	yes	*
LOG FILE PATHNAME	256		alpha	rel/sub	all	rel/sub	yes	
LOG FORM ID	10		alpha	rel/sub	all	rel/sub	yes	*
LOG FORMAT	10		alpha	rel/sub	all	rel/sub	yes	
LOG HARD COPY	1		logic	rel/sub	all	rel/sub	yes	*
LOG MODE	10		token	rel/sub	all	rel/sub	yes	*
LOG NOTIFY USER	1		logic	rel/sub	all	rel/sub	yes	*
LOG ON SCREEN	1		logic	rel/sub	all	rel/sub	yes	*
LOG PRINTER ID	10		alpha	rel/sub	all	rel/sub	yes	*
LOG PRIORITY	10		alpha	rel/sub	all	rel/sub	yes	*
LOG REPORT OPTIONS	256		alpha	rel/sub	all	rel/sub	yes	null
LOG SPOOL DATE	C-th		greg-bin	rel/sub	all	rel/sub	yes	*
MODE	7		token	man	prohib	prohib	no	null
NEW PAGE AFTER	1		logic	sub	rcv	sub	yes	no
NEW PAGE BEFORE	1		logic	sub	rcv	sub	yes	no
NEXT APPLICATION	3		alpha	sub	rcv	sub	yes	null
NEXT CHILD ID	4		alpha	sub	rcv	sub	yes	null
NEXT DATABASE	3		alpha	sub	rcv	sub	yes	null
NEXT LANGUAGE ID	3		alpha	sub	rcv	sub	yes	null
NEXT PROCESS NAME	30		alpha	sub	rcv	sub	yes	null
NEXT VERSION	2		alpha	sub	rcv	sub	yes	null
OPERATIONAL PHASE	22		token	all	all	sub	no	*
OPTION	22		token	sub	all	sub	yes	null
PAGE NUMBER	10.0		binary	sub	all	sub	yes	0
PAN COLUMN	7.0	N	binary	sub	all	sub	yes	1
PAN ROW	7.0	N	binary	sub	all	sub	yes	1
PRINT BANNER	1		logic	rel/sub	all	rel/sub	yes	0
PRINT COPIES	5.0		binary	rel/sub	all	rel/sub	yes	*
PRINT FILE BROWSABLE	1		logic	rel/sub	all	rel/sub	yes	*

	DOMAIN INFO			IPC CHARACTERISTICS				
Predefined Field	Size	Sign	Store	Pass	Rec	Share	Modify	Init Val
PRINT FILE DISPOSITION	10		token	rel/sub	all	rel/sub	yes	*
PRINT FILE PATHNAME	256		alpha	rel/sub	all	rel/sub	yes	
PRINT FORM ID	10		alpha	rel/sub	all	rel/sub	yes	*
PRINT FORMAT	10		alpha	rel/sub	all	rel/sub	yes	
PRINT HARD COPY	1		logic	rel/sub	all	rel/sub	yes	*
PRINT MODE	10		token	rel/sub	all	rel/sub	yes	*
PRINT NOTIFY USER	1		logic	rel/sub	all	rel/sub	yes	*
PRINT ON SCREEN	1		logic	rel/sub	all	rel/sub	yes	*
PRINT PAGE ESTIMATE	10.0		binary	rel/sub	all	rel/sub	yes	*
PRINT PRINTER ID	10		alpha	rel/sub	all	rel/sub	yes	*
PRINT PRIORITY	10		alpha	rel/sub	all	rel/sub	yes	*
PRINT RECORD LIMIT	10.0		binary	rel/sub	all	rel/sub	yes	0
PRINT REPORT OPTIONS	256		Alpha	rel/sub	all	rel/sub	yes	null
PRINT SPOOL DATE	C-th		greg-bin	rel/sub	all	rel/sub	yes	null
PRINT SUMMARY ONLY	1		logic	rel/sub	all	rel/sub	yes	no
PROCESS CODE	4		alpha	rel/sub	all	rel/sub	yes	null
PROCESS NAME	30		alpha	man	prohib	prohib	no	*
PROCESS TYPE	10		token	man	prohib	prohib	no	*
RECORD NUMBER	10.0		binary	man	man	man	no	0
RETURN CODE	10.0		binary	rel/sub	all	rel/sub	yes	0
SCROLL TO END	1		logic	sub	all	sub	yes	0
SELECTED APPLICATION	3		alpha	all	all	rel/sub	no	null
SELECTED DATABASE	3		alpha	all	all	rel/sub	no	
SELECTED VERSION	2		alpha	all	all	rel/sub	no	null
SKIP AFTER	2.0		binary	sub	rcv	sub	yes	0
SKIP BEFORE	2.0		binary	sub	rcv	sub	yes	0
SOURCE APPLICATION	3		alpha	all	all	sub	no	*
SOURCE VERSION	2		alpha	all	all	sub	no	*
SUBHEADING	79		alpha	rel/sub	all	rel/sub	yes	null
SUBMIT DATE	C-th		greg-bin	rel/sub	all	rel/sub	yes	null
SUBMIT MODE	10		alpha	rel/sub	all	rel/sub	yes	null
SUBMIT NOTIFY USER	1		logic	rel/sub	all	rel/sub	yes	0

	DOMAIN INFO			IPC CHARACTERISTICS				
Predefined Field	Size	Sign	Store	Pass	Rec	Share	Modify	Init Val
SUBMIT OPTIONS	256		alpha	rel/sub	all	rel/sub	yes	null
SUBMIT PRIORITY	10		Alpha	rel/sub	all	rel/sub	yes	null
SUBMIT PROCESS	1		Logic	rel/sub	all	rel/sub	yes	0
SUBMIT QUEUE	10		alpha	rel/sub	all	rel/sub	yes	null
TEMP 1	1		alpha	sub	all	sub	yes	null
TEMP 2	2		alpha	sub	all	sub	yes	null
TEMP 4	4		alpha	sub	all	sub	yes	null
TEMP 30	30		alpha	sub	all	sub	yes	null
TEMP 30 CTR	30		alpha	sub	all	sub	yes	null
TEMP 30 LJ	30		alpha	sub	all	sub	yes	null
TEMP 30 RJ	30		alpha	sub	all	sub	yes	null
TEMP 32	32		alpha	sub	all	sub	yes	null
TEMP 79	79		alpha	sub	all	sub	yes	null
TEMP 80	80		alpha	sub	all	sub	yes	null
TEMP 132	132		alpha	sub	all	sub	yes	null
TEMP 256	256		alpha	sub	all	sub	yes	null
TEMP 512	512		alpha	sub	all	sub	yes	null
TEMP 2K	2k		alpha	sub	all	sub	yes	null
TEMP 8K	8k		alpha	sub	all	sub	yes	null
TEMP 32K	32k		alpha	sub	all	sub	yes	null
TEXT AT POSITION	5.0		binary	sub	rcv	sub	no	0
USER ID	3		alpha	all	all	all	no	*
VERSION	2		alpha	man	prohib	prohib	no	*
WARNINGS	10.0		binary	sub	all	sub	yes	0

* Controlled in System Administration or at runtime.

A, B, C...Z

A through Z are index work variables for use in ILF statements. They are floating-point variables that can have up to 15 places to the left of the decimal and up to 15 to the right, subject to a maximum of 15 total digits. These fields and their related fields, integer variables AI, BI, CI...ZI, are the only fields that can be used in a CALC statement.

```
SET      --- A                          =  TAP INVOICE QUANTITY
SET      --- B                          =  TAP INVOICE PRICE
CALC     C = A*B
```

A though Z variables can also be used as index work variables in BEG LOOP/END LOOP and SET TEMP statements and in place of occurrence numbers and/or appearance numbers wherever such numbers occur.. For example,

```
BEG LOOP A  = 001 TO 010  STEP 001
SET      TAP WORK COUNTER           A   =      0
END LOOP A
```

AI, BI, CI...ZI

AI through ZI are index work variables for use in ILF statements. They are integer variables that can be used in computations which require only integers. Before setting these variables, any numbers with decimal places are automatically rounded up to the next higher integer. The range of values allowed is from –2,147,483,648 to +2,147,483,647, inclusive. These fields and their related fields, floating point variables A, B, C...Z, are the only fields that can be used in a CALC statement. Note that in the example below quantity and price are both rounded to integers as part of setting the variables AI and BI.

```
SET      --- AI                         =  TAP INVOICE QUANTITY
SET      --- BI                         =  TAP INVOICE PRICE
CALC     CI = AI*BI
```

AI through ZI variables can also be used as index work variables in BEG LOOP/END LOOP and SET TEMP statements and in place of occurrence numbers and/or appearance numbers wherever such numbers occur. For example,

```
BEG LOOP AI = 001 TO 010  STEP 001
SET      TAP WORK COUNTER           AI  =      0
END LOOP AI
```

ACCESS PATH

ACCESS PATH contains the current key access path of a file. **At the present time, it cannot be modified.**

However, **when implemented**, it may be set in the *Option Intercept* event point in input processes. This will allow the designer to set the key access path of a file during a user scan operation. Normally, a file is scanned using the primary key of the file but setting this predefined field would override that path.

```
IF       --- ACCESS PATH               EQ TAR CUSTOMER NAME
```

ALTERNATE IMAGE NUMBER

ALTERNATE IMAGE NUMBER can be set in the *Select Image* event point in both input processes and output processes.

If set to a number other than zero, APPX will display the image that carries the value of this predefined field. If no such image exists, APPX will display the first image.

If set to zero, APPX will display no image and continue to the next frame.

Any normal frame that includes a SET --- ALTERNATE IMAGE NUMBER statement in its *Select Image* event point, is considered an optional frame.

```
          SET      --- ALTERNATE IMAGE NUMBER     =      2
```

APPLICATION

This field identifies the application that contains the process being executed. This value does not change if a COPY or GOSUB statement is executed that references a different application.

```
          IF       --- APPLICATION                EQ     TAP
          AND      --- VERSION                    EQ     00
    T     SET      --- PRINT ON SCREEN            =      Y
```

Related PDFs: PROCESS NAME, PROCESS TYPE, VERSION

AUTO READNEXT

This field is used in input processes to determine whether or not APPX automatically retrieves the next record after it finishes storing the current one. It can be changed during data entry by selecting the SET ATTRIBUTES option. In general, it is established by the designer when the input process is defined. This field can also be used in inquiry processes to control information flow.

```
          IF       --- MODE                       EQ     CHANGE
    T     SET      --- AUTO READNEXT              =      1
```

Default Value:

The default value is blank but can be changed to *Y* or *N* by the designer when the input process is defined, or by the user while the input process is being executed. Blank means *Y* in all modes except delete mode, when it means *N*. If AUTO READNEXT is blank when the process begins, the first record is not read from the file being maintained.

Related PDFs: EXACT KEY REQUIRED

BETWEEN

A value is placed in this field when a DATE BTW statement is executed in an event point. It contains the number of date/time units (centuries, years, months, days, hours, minutes, seconds, hundredths) between the two dates being compared. The unit contained in this field is defined by the DATE BTW statement.

```
          DATE BTW TAP WORK DATE                  5  TAP TODAYS DATE
          IF       --- BETWEEN                    GT     10
```

In this example, the unit in BETWEEN is "days," indicated by the value *5* date unit specified in the DATE BTW statement.

BILLING CODE

When the usage log is implemented, this field will initially contain the current billing code. The billing code will be saved as part of each record in the usage log. This field will affect future usage log records written for jobs executed by the current user.

```
          SET DATE TAP TODAYS DATE
          APPEND   --- BILLING CODE               1  TAP TODAYS DATE
```

CANCEL OK

This field controls whether or not a process or family of processes can be cancelled.

When each process is defined, the field *User Cancel OK* can be entered as *Y* or *N*. When the process is invoked, the predefined field CANCEL OK is set to the value that the designer specified for that process.

If there is a child process, and the child's CANCEL DISPOSITION is set to *prohibit*, that value takes precedence for the child process.

The value of CANCEL OK can be toggled back and forth in any fashion by the designer. The example below would prevent a family of processes from being cancelled.

```
          SET      --- CANCEL OK                  =      0
```

CHILD ID

This predefined field will contain the value of the CHILD ID that the designer entered when the application was designed. A CHILD ID can be assigned to either optional or automatic child processes. That value is assigned to this field when the child process is invoked.

```
        IF      --- CHILD ID                EQ     50
T       SET     --- CANCEL OK               =      1
```

Related PDFs: INCLUDE CHILDREN, NEXT CHILD ID

COLOR BACKGROUND

Combined with the ILF statement COLOR, this field is used to conditionally set the background color of a specific field (as shown in the example below) or all fields on an image. The field can be set to either a constant (as shown) or a field containing the color name.

```
        IF      TAR CUSTOMER BALANCE        LT     0
T       SET     --- COLOR BACKGROUND        =      RED
T       COLOR   TAR CUSTOMER BALANCE
```

Default Value: Taken from system administration files.

Related PDFs: COLOR FOREGROUND, FONT

COLOR FOREGROUND

Combined with the ILF statement COLOR, this field is used to conditionally set the foreground color of a specific field or all fields (as shown in the example below) on an image. The field can be set to either a constant or a field containing the color name (as shown).

```
        IF      TAR CUSTOMER PAST DUE       GT     10000.00
T       SET     --- COLOR FOREGROUND        =  SYS COLOR RED
T       COLOR
```

Default Value: Taken from system administration files.

Related PDFs: COLOR BACKGROUND, FONT

COMMIT MODE

This field is used to define or change the COMMIT MODE of any process. COMMIT MODE determines those points in time when APPX will commit changes made to the tables stored within an RDBMS. It is only applicable in conjunction with an RDBMS; it cannot be used with APPXIO. Also, not all File Managers support this feature.

This field can be set in the Additional Attributes of all processes but can be added or overridden by use of this field. Note that when COMMIT MODE is changed via a SET statement, the change takes effect at the next COMMIT or ROLLBACK statement.

```
SET      --- COMMIT MODE                =      COMPATIBLE
```

Valid Values:

COMPATIBLE	Changes are committed as soon as all logical locks are freed. This provides maximum compatibility between APPXIO files and RDBMS tables. Note that both APPXIO and RDBMS locks must be released.
AUTOCOMMIT*	Commit each modification as it is made. ***This value will be implemented in a future release.**
PCF CYCLE	Commit modifications as each PCF record is processed, normally after the *Post PCF Update* event point.
END	Only commit modifications at the end of the process family.
MANUAL	Modifications must be committed manually via use of the COMMIT statement. APPX will not automatically COMMIT any changes.

COMPATIBLE is the default value while use of AUTOCOMMIT, when implemented, is the least effective in terms of performance.

Based on the structure of your application, there may be other places where you want to COMMIT a transaction. That's what the COMMIT statement is for. One thing to keep in mind is that the frequency of COMMITs will affect performance. The more COMMITs you do, the slower your application will be. For interactive processes, you can probably ignore the performance issue, but for batch-mode operations, COMMITing too often can really slow things down.

Another issue to be aware of is that APPXIO files do not support transaction processing. If you ROLLBACK a change to an RDBMS-hosted file, any changes made to APPXIO will not be rolled back. Related to this is the issue of visibility. Changes made to an RDBMS-hosted file are invisible to other users until they are committed. Changes made to an APPXIO-hosted file are visible immediately.

APPX also supports savepoints. A savepoint is a partial transaction. You mark the beginning of a savepoint with the SAVEPNT statement - the only argument is a name of your choosing. Executing a SAVEPNT tells the RDBMS that it should write a marker into the current transaction. If you find that you want to roll back part of a transaction, you can refer to a savepoint name in the ROLLBACK statement. The transaction is still open and you can commit other changes, but changes made since the savepoint was created are discarded.

If using COMPATIBLE mode, it is helpful to understand when locks are applied and released. When you read an RDBMS record with hold, you obtain a physical lock and a logical lock. As soon as you delete that record, rewrite it, or read another record, the logical lock is forfeited, but the physical lock is held until the transaction is committed (or rolled back). The physical lock may, depending on the database, prevent other users from reading the record, or it may prevent other users from locking the record. APPXIO, on the other hand, releases the physical lock at the same time as the logical lock. In COMPATIBLE mode, APPX commits changes as soon your task releases all the logical locks, both APPXIO and RDBMS.

Related Statements

COMMIT, ROLLBACK, SAVEPNT

COMPANY ADDRESS 1

This field contains the first line of the company address that was entered for the current database in system administration. (The current database is defined when the *Select Database* option is chosen, and that value is then placed into the DATABASE predefined field.)

```
          DISPLAY  --- COMPANY ADDRESS 1          (AT APPEARANCE #    )
```

Related PDFs: COMPANY ADDRESS 2, COMPANY ADDRESS 3, COMPANY NAME, DATABASE

COMPANY ADDRESS 2

This field contains the second line of the company address that was entered for the current database in system administration. (The current database is defined when the *Select Database* option is chosen, and that value is then placed into the DATABASE predefined field.)

```
          DISPLAY  --- COMPANY ADDRESS 2          (AT APPEARANCE #    )
```

Related PDFs: COMPANY ADDRESS 1, COMPANY ADDRESS 3, COMPANY NAME, DATABASE

COMPANY ADDRESS 3

This field contains the third line of the company address that was entered for the current database in system administration. (The current database is defined when the *Select Database* option is chosen, and that value is then placed into the DATABASE predefined field.)

```
          DISPLAY  --- COMPANY ADDRESS 3          (AT APPEARANCE #    )
```

Related PDFs: COMPANY ADDRESS 1, COMPANY ADDRESS 2, COMPANY NAME, DATABASE

COMPANY NAME

This field contains the name of the company that was entered for the current database in system administration. (The current database is defined when the *Select Database* option is chosen, and that value is then placed into the DATABASE predefined field.)

```
          DISPLAY  --- COMPANY NAME              (AT APPEARANCE #    )
```

Related PDFs: COMPANY ADDRESS 1, COMPANY ADDRESS 2, COMPANY ADDRESS 3, DATABASE

CURSOR AT APPEARANCE

This field has significance only for input processes, and only if the cursor is positioned on an input item. (The item need not be modifiable.) Under these circumstances, this field will contain the appearance number of the item. In other words, if the cursor is positioned on the first appearance of the item, CURSOR AT APPEARANCE will have the value *1*; if the cursor is positioned on the second appearance, *2*; and so forth.

Remember that APPEARANCE refers to multiple appearances of a unique field on an image, and that OCCURRENCE refers to multiple occurrences of a field, as defined in the data dictionary.

```
        IF      --- CURSOR AT APPEARANCE       EQ     1
   T    SET     --- CURSOR COLUMN              =      30
```

Default Value: Wherever the cursor is, if it is sitting on a field.

Related PDFs: CURSOR AT OCCURRENCE

CURSOR AT OCCURRENCE

This field has significance only for input processes, and only if the cursor is positioned on an input item. (The item need not be modifiable.) Under these circumstances, this field will contain the occurrence number of the item. In other words, if the cursor is positioned on the first occurrence of the item, CURSOR AT OCCURRENCE will have the value *1*; if the cursor is positioned on the second occurrence, *2*; and so forth.

Remember that APPEARANCE refers to multiple appearances of a unique field on an image, and that OCCURRENCE refers to multiple occurrences of a field, as defined in the data dictionary.

```
        IF      --- CURSOR AT OCCURRENCE       EQ     1
   T    SET     --- CURSOR COLUMN              =      30
```

Default Value: Wherever the cursor is, if it is sitting on a field.

Related PDFs: CURSOR AT APPEARANCE

CURSOR COLUMN

This field contains the number of the column (horizontal position) where the cursor currently sits. It can be used to change the location of the cursor if used along with CURSOR ROW and the POSITION statement as shown in the following example.

```
SET      --- CURSOR COLUMN              =      25
SET      --- CURSOR ROW                 =      5
POSITION                                (AT APPEARANCE #    )
```

This example positions the cursor at row 5, column 25. The positioning is absolute; that is, if the current image begins at row 15 and column 1, the cursor will be outside the current image after this example is executed.

The POSITION statement can also be used to set the value of this field as shown below:

```
POSITION TAP VENDOR NAME                (AT APPEARANCE #    )
```

This example positions the cursor at the input item called VENDOR NAME and sets the value of CURSOR COLUMN (and CURSOR ROW) accordingly.

Default Value: Wherever the cursor is.

Related PDFs: CURSOR ROW

CURSOR ROW

This field contains the number of the row (vertical position) where the cursor currently sits. It can be used to change the location of the cursor, if used along with CURSOR COLUMN and the POSITION statement as shown in the following example.

```
SET      --- CURSOR COLUMN              =      25
SET      --- CURSOR ROW                 =      5
POSITION                                (AT APPEARANCE #    )
```

This example positions the cursor at row 5, column 25. The positioning is absolute; that is, if the current image begins at row 15 and column 1, the cursor will be outside the current image after this example is executed.

The POSITION statement can also be used to set the value of this field as shown below:

```
POSITION TAP VENDOR NAME                (AT APPEARANCE #    )
```

This example positions the cursor at the input item called VENDOR NAME and sets the value of CURSOR ROW (and CURSOR COLUMN) accordingly.

Default Value: Wherever the cursor is.

Related PDFs: CURSOR COLUMN

DATABASE

This is the default database used when opening database-related files and evaluating security classes. This field is initialized to the value of SELECTED DATABASE when a detached child is invoked. Once set, this value never changes during the life of a process.

```
          IF       --- DATABASE                  EQ     TST
    T     SET      --- PRINT ON SCREEN           =      Y
```

Related PDFs: NEXT DATABASE, SELECTED DATABASE

DEFAULT MODE

This field is used only for input and query processes. For input processes, it initially contains the value of the *Default Mode* parameter entered by the designer when the input process was defined. For query processes, the initial value is null. The only appropriate places to change this parameter are in the *Pre-Invocation* or *Start of Process* event points.

In query processes, a value of *ADD* or blank will reset any previous setup for the query; a value of *CHANGE* or *INQUIRE* will allow the user to modify any previous setup for the query. A value of *DELETE* will delete the sort work files.

```
          SET      --- DEFAULT MODE              =      INQUIRE
```

Initial Value:

The Initial Value is initialized to blank when a child is invoked detached. For an input process, it is set to the value of MODE before invoking a child whose *Default Mode* is blank.

Valid Values: ADD, DELETE, INQUIRE, CHANGE, or blank.

Related PDFs: MODE

ENTRY LEVEL

This field is used only for input processes and has meaning only during the *DATA ADDITION* interactive phase. It contains a number that essentially indicates the number of times the operator has selected ENTER (assuming no errors were encountered). ENTRY LEVEL is not incremented until after the *Verify* and *Pre-Display/Verify* event points have completed successfully.

An entry level of *1* indicates the first time fields are modifiable on the image. An entry level of *2* indicates that the previously entered data has been verified plus additional fields may have been retrieved (or computed) and/or displayed, and such fields may now be available for entry or modification.

The entry level for any given item on an image — that is, the time at which the item becomes modifiable — is defined when the designer paints the image.

The entry level for an optional child can be set as an additional attribute. If set, that optional child cannot be invoked until the proper entry level is reached.

```
        IF      --- ENTRY LEVEL               EQ     1
        AND     --- INTERACTIVE PHASE         EQ     DATA ADDITION
T       SET     TAP WORK VENDOR NO            =  TAP VENDOR NO
```

Related PDFs: INTERACTIVE PHASE, MODE

ERRORS

This field contains a count of the number of errors generated so far within the current subprocess family, either by the use of the ERROR statement or by data lookup error conditions. It can also be set to zero at any time, and then interrogated at a later time, to determine if any errors were encountered between those two points.

```
        IF      --- ERRORS                    EQ     0
T       MESSAGE Transmission of Payments Has Completed Successfully.
```

Related PDFs: WARNINGS

EXACT KEY REQUIRED

This field is used in input processes to determine whether or not APPX will require the end user to enter a valid key to access a record in the file being maintained. It can be changed during data entry by selecting the SET ATTRIBUTES option. In general, it is established by the designer when the input process is defined.

This field can also be used in inquiry processes to control the process flow.

```
        IF      --- MODE                      EQ     DELETE
T       SET     --- EXACT KEY REQUIRED        =      Y
```

Default Value: *N*, but can be changed by the designer when the input process is defined.

Related PDFs: AUTO READNEXT

FONT

Combined with the ILF statement FONT, this field is used to conditionally set the font type of a specific field or all fields (as shown in the example below) on an image. The field can be set to either a constant or a field containing the font name (as shown).

```
        IF       TAR CUSTOMER TYPE              EQ     A
    T   SET      --- FONT                       =  SYS FONT ARIAL BOLD
    T   FONT
```

Default Value: Taken from system administration files.

Related PDFs: COLOR BACKGROUND, COLOR FOREGROUND

FORM LENGTH

This field pertains only to output processes and contains the value of the form length that was entered in the Forms file in system administration for the form selected in the printer disposition.

```
        IF       --- FORM LENGTH                EQ     66
    T   DISPLAY  TAP WORK TEXT                  (AT APPEARANCE #    )
```

Related PDFs: FORM WIDTH, LINES LEFT ON PAGE

FORM WIDTH

This field pertains only to output processes and contains the value of the form width that was entered in the Forms file in system administration for the form selected in the printer disposition. This parameter can also be entered when the output process is defined in application design. If the parameter in the output process is blank, this field will determine the form width.

```
        IF       --- FORM WIDTH                 EQ     132
    T   DISPLAY  TAP WORK TEXT                  (AT APPEARANCE #    )
```

Related PDFs: FORM LENGTH, LINES LEFT ON PAGE

INCLUDE CHILDREN

This field determines whether or not child processes will be invoked from a parent process. It affects both automatic and optional child processes. Which child processes will be invoked and which will be skipped depends on where this field is modified. This field is immediately reset to *Y* after one or more child processes have been skipped.

When a process (particularly a JOB or INPUT process) invokes a child process, the value of INCLUDE CHILDREN is reset at the end of that child process to its value prior to the child process invocation.

If INCLUDE CHILDREN is set to *N* in the *End of Frame* event point of a frame that calls automatic child processes, all child processes for that frame will be skipped.

If INCLUDE CHILDREN is set to *N* in a *Pre-Invocation* event point, only that child process is skipped.

If INCLUDE CHILDREN is set to *N* in an *Post Invocation* event point, any remaining automatic child processes will be skipped.

```
          SET      --- INCLUDE CHILDREN           =      N
```

Related PDFs: CHILD ID, NEXT CHILD ID

INCLUDE RECORD

This field is used only in query processes and determines whether or not any given record, on a file being sorted, will be included in the sort. Before each record is presented to query selection, this field is reset to null (blank).

If it is left blank, then the normal query selection — whether end user-entered or designer-entered — will determine whether or not the record is included in the sort.

If this field is set to *N*, the record will not be included in the sort even if it would have passed the normal query selection.

If this field is set to *Y*, the record *will be included* in the sort, *even if it would not have passed the normal query selection*.

```
          IF       TAP VENDOR NO                  GT     55555
    T     SET      --- INCLUDE RECORD             =      N
```

INTERACTIVE PHASE

This field is set during input processes and indicates which data entry phase is currently in force. It works with the predefined field MODE and, although they work together, each has a specific use.

MODE is determined when the user selects an input process from a menu and selects a mode option (*ADD, CHG, DEL,* or *INQ*). INTERACTIVE PHASE determines what the user may do while in that mode. For example, if you are in add mode entering data into a scrolling frame, you can move the cursor to a previously added record and select the CHANGE ITEM option. At this time you will still be in add mode, but the interactive phase will change from data addition to data modification.

```
        IF      --- INTERACTIVE PHASE          EQ     DATA ADDITION
        AND     --- ENTRY LEVEL                EQ     1
    T   SET     TAP WORK VENDOR NO             =  TAP VENDOR NO
```

Valid Values:

NONE

DATA ADDITION (add mode before a record is saved)

DATA DELETION (user is prompted to “Confirm Delete”)

DATA INQUIRE (the record is displayed, cannot be modified)

DATA MODIFICATION (the record is displayed, can be modified)

DATA SCROLL (multiple records are being prepared for display)

KEY ENTRY (user is prompted to enter a key)

MENU ENTRY (currently not used)

Related PDFs: ENTRY LEVEL, MODE

KEYMAP NAME

This field represents the current keymap file name being used to map APPX function and option keys to the user’s keyboard. It can be set or reset by setting this field to either a constant or an alpha field with a keymap name for a value. Note that the keymap name must be the name of a valid keymap file residing in the default APPX data directory.

```
        IF      --- DATABASE                   EQ     ARC
    T   SET     --- KEYMAP NAME                =      VT320
```

LANGUAGE ID

This is the current language that all images are being presented to the user in. This field is only used in installations using the Language Translation feature. This field is initialized to the value of NEXT LANGUAGE ID when a detached child is invoked. Once set, this value never changes during the life of a process. Refer to the *Language Translation Manual* for further information.

```
        IF      --- LANGUAGE ID                EQ     ESP
    T   DISABLE     SPANISH BUTTON
```

Related PDFs: NEXT LANGUAGE ID

LINES LEFT ON PAGE

This field is used in output processes and indicates how many lines are left to print on the current page, based on FORM LENGTH and the number of lines that have already been printed.

```
        IF      --- LINES LEFT ON PAGE         LE     4
   T    GOSUB       :SPECIAL FOOTER
```

Related PDFs: FORM LENGTH, FORM WIDTH

LOG BANNER

This field contains the value of the *Banner?* option, which was entered in the *Report/Log Default Values* fields in system administration, for error and modification logs. If supported by the operating system, LOG BANNER will print a banner page before the log. The system-wide defaults for these values are entered in the System Parameters file. These values can be overridden for each new application, database, and user.

```
        SET     --- LOG BANNER                 =      Y
```

Default Value: Taken from system administration files.

LOG COPIES

This field contains the value of the *Copies* option, which was entered in the *Report/Log Default Values* fields in system administration, for error and modification logs. LOG COPIES contains the number of copies of the log that will be produced. The system-wide defaults for these values are entered in the System Parameters file. These values can be overridden for each new application, database, and user.

```
        SET     --- LOG COPIES                 =      2
```

Default Value: Taken from system administration files.

LOG FILE DISPOSITION

This field contains the value of the *Disposition* option, which was entered in the *Report/Log Default Values* fields in system administration, for error and modification logs. LOG FILE DISPOSITION indicates whether the log file will be scratched, saved in the user's print library, or requeued after printing. The system-wide defaults for these values are entered in the System Parameters file. These values can be overridden for each new application, database, and user.

```
SET      --- LOG FILE DISPOSITION       =      SAVE
```

Default Value: Taken from system administration files.

Valid Values:

SCRATCH (scratch after printing)

SAVE (place in user's print library after printing)

REQUEUE (place back in print queue after printing)

Related PDFs: LOG MODE

LOG FILE PATHNAME

This field contains the full directory path tree to the print file and the name of the print file.

```
SET      --- TEMP 80                    =  --- LOG FILE PATHNAME
```

LOG FORM ID

This field contains the value of the *Form* option, which was entered in the *Report/Log Default Values* fields in system administration, for error and modification logs. LOG FORM ID contains a key to the Forms file in system administration, which retrieves the record that identifies the form on which the log will be printed. The system-wide defaults for these values are entered in the System Parameters file. These values can be overridden for each new application, database, and user.

```
SET      --- LOG FORM ID                =      STANDARD
```

Default Value: Taken from system administration files.

LOG FORMAT

This field contains formatting information send to the printer when printing a log. The use of the field is not currently implemented and is reserved for future use.

LOG HARD COPY

This field contains the value of the *Hard Copy?* option, which was entered in the *Report/Log Default Values* fields in system administration, for logs. The system-wide defaults for these values are entered in the System Parameters file. These values can be overridden for each new application, database, and user.

```
SET      --- LOG HARD COPY              =      Y
```

Default Value: Taken from system administration files.

LOG MODE

This field contains the value of the *Mode* option, which was entered in the *Report/Log Default Values* fields in system administration, for error and modification logs. LOG MODE indicates whether the file will be held in the print queue, spooled directly to the printer, or kept in the user's print library instead of being directed to the print queue. The system-wide defaults for these values are entered in the System Parameters file. These values can be overridden for each new application, database, and user.

```
SET      --- LOG MODE                   =      HOLD
```

Default Value: Taken from system administration files.

Valid Values:

HOLD (hold in the print queue until an operator releases it)

SPOOL (spool directly to the printer)

KEEP (place in user's print library, not print queue)

Related PDFs: LOG FILE DISPOSITION

LOG NOTIFY USER

This field contains the value of the *Notify?* option, which was entered in the *Report/Log Default Values* fields in system administration, for error and modification logs. If supported by the operating system, LOG NOTIFY USER can alert the user when the log has finished printing. The system-wide defaults for these values are entered in the System Parameters file. These values can be overridden for each new application, database, and user.

```
SET      --- LOG NOTIFY USER            =      Y
```

Default Value: Taken from system administration files.

LOG ON SCREEN

This field contains the value of the *On Screen?* option, which was entered in the *Report/Log Default Values* fields in system administration, for logs. The system-wide defaults for these values are entered in the System Parameters file. The values can be overridden for each new application, database, and user.

```
SET      --- LOG ON SCREEN              =      Y
```

Default Value: Taken from system administration files.

LOG PRINTER ID

This field contains the value of the *Printer* option, which was entered in the *Report/Log Default Values* fields in system administration, for error and modification logs. LOG PRINTER ID contains a key to the Printers file in system administration, which will retrieve the record that identifies the printer on which the log will be printed. The system-wide defaults for these values are entered in the System Parameters file. These values can be overridden for each new application, database, and user.

```
SET      --- LOG PRINTER ID             =      BAND PRINTER
```

Default Value: Taken from system administration files.

LOG PRIORITY

This field contains the value of the *Priority* option, which was entered in the *Report/Log Default Values* fields in system administration, for error and modification logs. If supported by the operating system, LOG PRIORITY can assign a specific priority to the log file in the print queue. The system-wide defaults for these values are entered in the System Parameters file. These values can be overridden for each new application, database, and user.

```
SET      --- LOG PRIORITY               =      10
```

Default Value: Taken from system administration files.

Valid Values: Operating system specific.

LOG REPORT OPTIONS

This field contains a string of printer commands that are passed to the printer for printer control purposes when printing logs. The information is written to a line in the <log report file name>.cfg file. Because these commands are operating system and printer specific, you should review those respective requirements before attempting to use this field. The field can be SET to a constant or alpha field (as shown in the example below) containing the print commands.

```
          SET      --- LOG REPORT OPTIONS         =  SYS COURIER PORTRAIT DUPLX
```

Valid Values: Operating system specific.

LOG SPOOL DATE

This field contains the value of the *Spool Date* option, which can optionally be entered when the input process DISPOSITION is run. If supported by the operating system, LOG SPOOL DATE can cause a spooled log to delay being printed until a specified date and time.

```
          SET      --- LOG SPOOL DATE             =  TAP WORK DATE
```

MODE

This field is used only for input processes, and contains the value that the operator selects by selecting the ADD ITEM, DELETE ITEM, INQUIRE ITEM, or CHANGE ITEM icon when the process first loads from the menu.

```
          IF       --- MODE                       EQ     ADD
    T     SET      TAP WORK VENDOR                =  TAP VENDOR NO
```

Valid Values: ADD, DELETE, INQUIRE, CHANGE

Related PDF's: ENTRY LEVEL, INTERACTIVE PHASE, DEFAULT MODE

NEW PAGE AFTER

This field is used in output processes, and indicates whether or not the report will skip to a new page after a subsequent PRINT statement is executed. This field is also used when you are running something outside of APPX that will affect the screen. If this field is set to Y or left blank, APPX will repaint the screen after the RUN statement is executed.

```
SET      --- NEW PAGE AFTER              =      Y
PRINT    TAP WORK TEXT                   AT 050 ROWS 010 COLUMNS 030
```

Related PDF's: NEW PAGE BEFORE, SKIP AFTER, SKIP BEFORE

NEW PAGE BEFORE

This field is used in output processes, and indicates whether or not the report will skip to a new page before a subsequent PRINT statement is executed.

```
SET      --- NEW PAGE BEFORE             =      Y
PRINT    TAP WORK TEXT                   AT 005 ROWS 002 COLUMNS 020
```

Related PDF's: NEW PAGE AFTER, SKIP AFTER, SKIP BEFORE

NEXT APPLICATION

This field can be used to identify an application ID for use in a subsequent MENU, INPUT, OUTPUT, or similar statement, if the statement itself does not specify an application.

This field is also used to control which source file is accessed when opening, creating, or scratching a source file. It is automatically cleared following an OPEN, CREATE, or SCRATCH statement.

```
SET      --- NEXT APPLICATION            =      TAP
SET      --- NEXT PROCESS NAME           =      VENDORS
INPUT                                          DETACHED     END? N  FAIL 0
```

Related PDF's: NEXT PROCESS NAME, NEXT VERSION

NEXT CHILD ID

This field can be set to choose which child process will be invoked next.

If it is set in the *End of Frame* event point on a frame having automatic child processes, the first child process defined with a CHILD ID that matches NEXT CHILD ID will be invoked. Once that child has been completed, processing will continue normally.

If this field is set in the *Pre- Invocation* event point, processing will immediately skip to the first child encountered that has a matching CHILD ID.

If this field is set in the *Post Invocation* event point, the next child process to be invoked will be the first child process encountered where CHILD ID matches NEXT CHILD ID.

There are two special CHILD IDs: "+" means move to the next (or first) child; "-" means move to the previous (or last) child. These IDs are automatically set by ***APPX*** but can be overridden using statements. They can also be interrogated to determine the direction of process flow.

```
          SET      --- NEXT CHILD ID              =      EOM
```

Related PDF's: CHILD ID, INCLUDE CHILDREN

NEXT DATABASE

This field can be set before the execution of a CREATE, OPEN, or SCRATCH statement to operate on a file from a database other than the one currently identified by the DATABASE field. It is then cleared automatically following the execution of the CREATE, OPEN, or SCRATCH statement.

This field can also be set before invoking the SELECT DATABASE subroutine process to automatically select a specific database. Note that the database is not actually changed to the selected value until the next invocation of a detached process.

```
          SET      --- NEXT DATABASE              =      TST
          SUBR     --- SELECT DATABASE                   SUBPROCESS   END? N  FAIL 0
          INPUT    TAP VENDORS                           DETACHED     END? N  FAIL 0
```

Related PDFs: DATABASE, SELECTED DATABASE

NEXT LANGUAGE ID

This field is used to instantly change the presentation language of the current and subsequent images displayed to a user. It is only used in installations using the Language Translation feature. Refer to the *Language Translation Manual* for further information.

```
          IF       --- OPTION                     EQ     USER 1
    T     SET      --- NEXT LANGUAGE ID           =      DUT
```

Related PDFs: LANGUAGE ID

NEXT PROCESS NAME

This field can be used to identify a specific process for use in a subsequent MENU, INPUT, OUTPUT, or similar statement, if the statement itself does not explicitly specify a process.

```
          SET      --- NEXT APPLICATION           =      TAP
          SET      --- NEXT VERSION               =      00
```

```
SET      --- NEXT PROCESS NAME          =      VENDORS
INPUT                                          DETACHED     END? N  FAIL 0
```

Related PDFs: NEXT APPLICATION, NEXT VERSION

NEXT VERSION

This field can be used to identify a specific version of an application for use in a subsequent MENU, INPUT, OUTPUT, or similar statement.

This field is also used to control which source file is accessed when opening, creating, or scratching a source file. It is automatically cleared following an OPEN, CREATE, or SCRATCH statement.

```
SET      --- NEXT APPLICATION           =      TAP
SET      --- NEXT VERSION               =      00
SET      --- NEXT PROCESS NAME          =      VENDORS
INPUT                                          DETACHED     END? N  FAIL 0
```

This example will fail if the value in NEXT VERSION is not valid for the current database.

Related PDFs: NEXT APPLICATION, NEXT PROCESS NAME

OPERATIONAL PHASE

This field is stored in system administration for each database and application combination. Its value can be changed from initial setup or recovery processing by means of the predefined process BEGIN LIVE OPERATIONS.

There are certain differences between the operating phases. In initial setup and recovery processing, for example, records can be deleted from files even if there is delete protection, and keys can be changed even if there is key protection. In live operations or recovery processing, modification logs are produced for any input process with a non-blank *Mod Log File Name*. In initial setup, mod logs are *never* produced. And for any input item defined with *Modifiable?* equal to blank, it means that the item is modifiable in initial setup and recovery processing, but *not* in live operations. (As an example, this allows the designer to use the same input process for initial setup and live operations, without compromising data integrity.) In addition, images and even entire frames can be skipped if the application is in live operations, by setting the parameter *Skip if Live Operations?* to *Y* for that image or frame.

```
IF       --- OPERATIONAL PHASE          EQ     INITIAL SET UP
```

Default Value: Taken from system administration files for a given database/application combination when a child is invoked as detached.

Valid Values: INITIAL SET UP, LIVE OPERATIONS, RECOVERY PROCESSING

OPTION

This predefined field can be used either to set an option in a manner transparent to the end user, or to intercept the end user's option selections. If statements are used to set OPTION to a specific value, APPX will behave as though the user had selected that option. The field can be intercepted only in the *Option Intercept* event points, and can be intercepted or set either at the frame level or the image level. To set this field, use the *Pre-Display* or *Option Intercept* event points.

```
          IF       --- OPTION                    EQ     USER 99
    T     RUN      --- PROCESS NAME                                END? N  FAIL 0
```

Default Value: Whatever option the user has selected.

Valid Values: USER 0 through USER 255, SCAN, SELECT ACCESS PATH, PREVIOUS ACTIVITY, NEXT RECORD, REDISPLAY, INSERT TEXT, END PARAGRAPH, ADD MODE, DELETE MODE, INQUIRE MODE, CHANGE MODE, SET ENTRY ATTRIBUTES, DIRECT PROCESS 1, DIRECT PROCESS 2, END, HELP ITEM, PRINT SCREEN, CANCEL, SHOW MESSAGES, HELP, RETURN, ACKNOWLEDGE DELETE, SCROLL FIRST, SCROLL LAST, SCROLL PREV, SCROLL NEXT, SCROLL UP, SCROLL DOWN

PAGE NUMBER

This field relays the value of the *Page Number* field as maintained during the printing of a report. It can be used to restart page numbering if, for example, the designer wishes to begin numbering the pages of each customer statement at 1.

```
          IF       TAP WORK RECORD TYPE          EQ     NEXT
    T     SET      --- PAGE NUMBER               =      0
```

Default Value: Whichever page the report is on.

PAN COLUMN

In input processes, this field contains the number of the column (horizontal position) where the cursor currently sits when the user pans beyond the physical limits of the terminal screen. If the user pans beyond the screen limits, CURSOR COLUMN contains the column number where the cursor is positioned within the screen, and PAN COLUMN contains the actual physical column number of the cursor.

```
SET      --- PAN COLUMN                =      100
SET      --- PAN ROW                   =      5
```

Related PDFs: CURSOR COLUMN, CURSOR ROW, PAN ROW

PAN ROW

In input processes, this field contains the number of the row (vertical position) where the cursor currently sits when the user pans beyond the physical limits of the terminal screen. If the user pans beyond the screen limits, CURSOR ROW contains the row number where the cursor is positioned within the screen, and PAN ROW contains the actual physical row number of the cursor.

```
SET      --- PAN COLUMN                =      100
SET      --- PAN ROW                   =      5
```

Related PDFs: CURSOR COLUMN, CURSOR ROW, PAN COLUMN

PRINT BANNER

This field contains the value of the *Banner?* option, which was entered in the *Report/Log Default Values* fields in system administration, for reports. If supported by the operating system, PRINT BANNER will print a banner page before the report. The system-wide defaults for these values are entered in the System Parameters file. These values can be overridden for each new application, database, and user, and overridden again each time the input process DISPOSITION is invoked.

```
SET      --- PRINT BANNER              =      Y
```

Default Value: Taken from system administration files.

PRINT COPIES

This field contains the value of the *Copies* option, which was entered in the *Report/Log Default Values* fields in system administration, for reports. PRINT COPIES contains the number of copies of the report that will be produced. The system-wide defaults for these values are entered in the System Parameters file. These values can be overridden for each new application, database, and user, and overridden again every time the input process DISPOSITION is invoked.

```
SET      --- PRINT COPIES              =      2
```

Default Value: Taken from system administration files.

PRINT FILE BROWSABLE

This field indicates whether or not an on-screen report is browsable by users. Browsable reports allow the user to select a detail line of the report, then select `OPTION` `33` to access the PCF file's default input process. The selected report detail record is immediately available for maintenance or inquiry purposes. There must be a *Default Input Process* defined for the PCF file in the Data Dictionary.

```
SET     --- PRINT FILE BROWSABLE     =     N
```

Default Value: (Y)es.

PRINT FILE DISPOSITION

This field contains the value of the *Disposition* option, which was entered in the *Report/Log Default Values* fields in system administration, for reports. PRINT FILE DISPOSITION indicates whether the report file will be scratched, saved in the user's print library, or requeued after printing. The system-wide defaults for these values are entered in the System Parameters file. These values can be overridden for each new application, database, and user, and overridden again every time the input process DISPOSITION is invoked.

```
SET     --- PRINT FILE DISPOSITION     =     SAVE
```

Default Value: Taken from system administration files.

Valid Values:

SCRATCH (scratch after printing)

SAVE (place in user's print library after printing)

REQUEUE (place back in print queue after printing)

Related PDFs: PRINT MODE

PRINT FILE PATHNAME

This field contains the full directory path tree to the print file and the name of the print file.

```
SET     --- TEMP 80                  =  --- PRINT FILE PATHNAME
```

PRINT FORM ID

This field contains the value of the *Form* option that which was entered in the *Report/Log Default Values* fields in system administration, for reports. PRINT FORM ID contains a key to the Forms file in system administration, which retrieves the record that identifies the form on which the report will be printed. The system-wide defaults for these values are entered in the System Parameters file. These values can be overridden for each new application, database, and user, and overridden again every time the input process DISPOSITION is invoked.

```
          SET      --- PRINT FORM ID              =      WIDE
```

Default Value: Taken from system administration files.

PRINT FORMAT

This field contains formatting information send to the printer when printing a report. The use of the field is not currently implemented and is reserved for future use.

PRINT HARD COPY

This field contains the value of the *Hard Copy?* option, which was entered in the *Report/Log Default Values* fields in system administration, for reports. The system-wide defaults for these values are entered in the System Parameters file. These values can be overridden for each new application, database, and user, and overridden again every time the input process DISPOSITION is invoked.

```
          SET      --- PRINT HARD COPY            =      Y
```

Default Value: Taken from system administration files.

PRINT MODE

This field contains the value of the *Mode* option, which was entered in the *Report/Log Default Values* fields in system administration, for reports. PRINT MODE indicates whether the file will be held in the print queue, spooled directly to the printer, or kept in the user's print library instead of being directed to the print queue. The system-wide defaults for these values are entered in the System Parameters file. The values can be overridden for each new application, database, and user, and overridden again every time the input process DISPOSITION is invoked.

```
          SET      --- PRINT MODE                 =      HOLD
```

Valid Values:

HOLD (hold in the print queue until an operator releases it)

SPOOL (spool directly to the printer)

KEEP (place in user's print library, not print queue)

Related PDFs: PRINT FILE DISPOSITION

PRINT NOTIFY USER

This field contains the value of the *Notify?* option, which was entered in the *Report/Log Default Values* fields in system administration, for reports. If supported by the operating system, PRINT NOTIFY USER can alert the user when the report finishes printing. The system-wide defaults for these values are entered in the System Parameters file. The values can be overridden for each new application, database, and user, and overridden again every time the input process DISPOSITION is invoked.

```
          SET      --- PRINT NOTIFY USER          =      Y
```

Default Value: Taken from system administration files.

PRINT ON SCREEN

This field contains the value of the *On Screen?* option, which was entered in the *Report/Log Default Values* fields in system administration, for reports. The system-wide defaults for these values are entered in the System Parameters file. The values can be overridden for each new application, database, and user, and overridden again every time the input process DISPOSITION is invoked.

```
          SET      --- PRINT ON SCREEN            =      Y
```

Default Value: Taken from system administration files.

PRINT PAGE ESTIMATE

This field contains the value of the *Page Estimate* option, which can optionally be entered when the input process DISPOSITION is executed. **When implemented**, it will be used to estimate percent complete on systems that do not provide a means for knowing the number of records in a file.

```
          SET      --- PRINT PAGE ESTIMATE        =      10
```

PRINT PRINTER ID

This field contains the value of the *Printer* option that was entered in the *Report/Log Default Values* fields in system administration, for reports. PRINT PRINTER ID contains a key to the Printers file in system administration, which retrieves the record that identifies the printer on which the report will be printed. The system-wide defaults for these values are entered in the System Parameters file. The values can be overridden for each new application, database, and user, and overridden again every time the input process DISPOSITION is invoked.

```
          SET      --- PRINT PRINTER ID          =      BAND
```

Default Value: Taken from system administration files.

PRINT PRIORITY

This field contains the value of the *Priority* option, which was entered in the *Report/Log Default Values* fields in system administration, for reports. If supported by the operating system, PRINT PRIORITY can assign a specific priority to the report file in the print queue. The system-wide defaults for these values are entered in the System Parameters file. The values can be overridden for each new application, database, and user, and overridden again every time the input process DISPOSITION is invoked.

```
          SET      --- PRINT PRIORITY            =      10
```

Default Value: Taken from system administration files.

PRINT RECORD LIMIT

This field contains the value of the *Record Limit* option, which can optionally be entered when the input process DISPOSITION is executed. **When implemented**, it will determine the number of records from the process control file that will actually be printed.

```
          SET      --- PRINT RECORD LIMIT        =      10
```

PRINT REPORT OPTIONS

This field contains a string of printer commands that are passed to the printer for printer control purposes when printing reports. The information is written to a line in the <report file name>.cfg file. Because these commands are operating system and printer specific, you should review those respective requirements before attempting to use this field. The field can be SET to a constant or alpha field (as shown in the example below) containing the print commands.

```
          SET      --- PRINT REPORT OPTIONS       =  SYS COURIER PORTRAIT DUPLX
```

Valid Values: Operating system specific.

PRINT SPOOL DATE

This field contains the value of the *Spool Date* option, which can optionally be entered when the input process DISPOSITION is run. If supported by the operating system, PRINT SPOOL DATE can cause a spooled report to delay being printed until a specified date and time.

```
          SET      --- PRINT SPOOL DATE           =  TAP WORK DATE
```

PRINT SUMMARY ONLY

This field contains the value of the *Print Summary Only?* option, which can optionally be entered when the input process DISPOSITION is run. If PRINT SUMMARY ONLY is *Y*, all detail frames in the output process or processes that follow will be executed but not printed; only subheadings, subtotals, and grand totals will be printed.

```
          SET      --- PRINT SUMMARY ONLY         =      Y
```

PROCESS CODE

This field is used to pass information from one process to another among related and subprocess families.

```
          SET      --- PROCESS CODE               =      SUMM
```

Query processes have special rules with respect to "sharing" of work fields and temporary work files among APPX processes. (See Chapter 1-5: Interprocess Communications for more information on data sharing.) PROCESS CODE represents an exception to these special rules, as follows:

The "Setup" (Operator Interactive) phase of Query processing inherits inbound work fields and temporary files according to normal Interprocess Communication rules. But it passes outbound work fields/temp files *only* to the "Execution" phase of Query processing. Query (Execution) inherits inbound files/fields *only* from its associated Query (Setup). It can pass *nothing* to any subsequent job step, with the following exception:

The predefined field PROCESS CODE can be used to pass limited information into and out of Query processes. Up through Release 3.0, the application designer could not count on a Query running to completion (and making PROCESS CODE modifications available) before a subsequent Output or Update starts executing.

An exception to this is that the Query (Runtime) does not make PROCESS CODE available until *after* the subsequent Output/Update's *Start of Process* ILF series has executed, but before the first frame starts executing for the Output/Update

To access a PROCESS CODE (modified in a Query) in an Output's or Update's *Start of Process*, you can put a "dummy" update in after the Query, and before the subsequent Output/Update. This dummy update should be based in Query's PCF, and contain a single END PROC statement in its *Start of Frame* ILF series. This will make PROCESS CODE available in a subsequent Output/Update's *Start of Process* ILF series.

PROCESS NAME

This field identifies the name of the current process that is executing. The value does not change if a COPY or GOSUB statement is executed that references a different process.

```
          IF       --- PROCESS NAME              EQ     VENDORS LIST
          AND      --- PROCESS TYPE              EQ     OUTPUT
     T    SET      --- PRINT ON SCREEN           =      N
```

Related PDFs: PROCESS TYPE, APPLICATION, VERSION

PROCESS TYPE

This field identifies the type of process that is currently executing. The value does not change if a COPY or GOSUB statement is executed that references a different process type.

```
          IF       --- PROCESS NAME              EQ     VENDORS LIST
          AND      --- PROCESS TYPE              EQ     OUTPUT
     T    SET      --- PRINT ON SCREEN           =      N
```

Valid Values: MENU, JOB, INPUT, OUTPUT, UPDATE, QUERY, INQUIRY, STATUS, SUBROUTINE, ACTION

Related PDFs: PROCESS NAME, APPLICATION, VERSION

RECORD NUMBER

When implemented, this field will contain a count of the records that have been read from the process control file.

```
          IF       --- RECORD NUMBER              GT     400
    T     END
```

RETURN CODE

This field is returned whenever a CALL or RUN statement is completed. Its value is determined by the process that was called or run.

```
          RUN           /appx/deal                                  END? N FAIL 0
          IF       --- RETURN CODE                EQ     52
    T     GOSUB         :ACE OF SPADES
```

Assuming that the directory /appx contains the following sample C program, this program will be executed and a return code of *1* to *52* will be passed back to the APPX process that executed the RUN statement.

```
/* deal.c */
#include <stdlib.h>
#include <time.h>
int main( int argc, char ** argv )
{
/*
** Initialize the random number generator
*/
       srand(( unsigned ) time(0) );
/*
** Return a random number between 1 and 52
*/
       return( rand() % 52 + 1 );
}
```

SCROLL TO END

If this field is set to *Y* at the frame level of an input process, and if the frame is a scrolling frame, APPX will read through the process control file and present each record as defined on the related image, scrolling completely through the file until end of file is reached. The input process *Create Executable Modules*, found on the Toolbox menu in application design, is an example of the use of SCROLL TO END.

```
          SET      --- SCROLL TO END              =      Y
```

SELECTED APPLICATION

This field is set to the current contents of NEXT APPLICATION by the SELECT APPLICATION predefined input process. The current value of the predefined field SOURCE APPLICATION remains unaffected when that input process is executed. Instead, the current value of SELECTED APPLICATION is modified, which in turn is used to determine the initial value of SOURCE APPLICATION the next time a child process is invoked as detached. This predefined field can only be used by the APPX applications (application design, system administration, database management, and the runtime application).

```
          IF       --- SELECTED APPLICATION       EQ     TAP
```

Related PDFs: NEXT APPLICATION, NEXT VERSION, SELECTED VERSION, SOURCE APPLICATION, SOURCE VERSION

SELECTED DATABASE

This field is set to the current contents of NEXT DATABASE by the SELECT DATABASE predefined subroutine process. The current value of the predefined field DATABASE remains unaffected when that input process is executed. Instead, the current value of SELECTED DATABASE is modified, which in turn is used to determine the initial value of DATABASE the next time a child process is invoked as detached.

```
          IF       --- SELECTED DATABASE          EQ     TST
```

Related PDFs: DATABASE, NEXT DATABASE

SELECTED VERSION

This field is set to the current contents of NEXT VERSION by the SELECT APPLICATION predefined input process. The current value of the predefined field SOURCE VERSION remains unaffected when that input process is executed. Instead, the current value of SELECTED VERSION is modified, which in turn is used to determine the initial value of SOURCE VERSION the next time a child process is invoked as detached. This predefined field can only be used by the APPX applications (application design, system administration, database management, and the runtime application).

```
          IF       --- SELECTED VERSION           EQ     00
```

Related PDFs: NEXT APPLICATION, NEXT VERSION, SELECTED APPLICATION, SOURCE APPLICATION, SOURCE VERSION

SKIP AFTER

This field is used in output processes and determines the number of lines that are skipped after a subsequent PRINT statement is executed.

```
          SET      --- SKIP AFTER                 =      2
          PRINT    TAP WORK TEXT                  AT 010 ROWS 002 COLUMNS 020
```

Related PDFs: NEW PAGE AFTER, NEW PAGE BEFORE, SKIP BEFORE

SKIP BEFORE

This field is used in output processes and determines the number of lines that are skipped before a subsequent PRINT statement is executed.

```
          SET      --- SKIP BEFORE                =      2
          PRINT    TAP WORK TEXT                  AT 010 ROWS 002 COLUMNS 020
```

Related PDFs: NEW PAGE AFTER, NEW PAGE BEFORE, SKIP AFTER

SOURCE APPLICATION

This field identifies the current application being acted upon by application design.

This predefined field can only be used by the APPX applications (application design, system administration, database management, and the runtime application).

```
        IF      --- SOURCE APPLICATION          EQ     0AD
```

Related PDFs: NEXT APPLICATION, NEXT VERSION, SELECTED APPLICATION, SELECTED VERSION, SOURCE VERSION

SOURCE VERSION

This field identifies the current version of the application being acted upon by application design.

This predefined field can only be used by the APPX applications (application design, system administration, database management, and the runtime application).

```
        IF      --- SOURCE VERSION              EQ     00
```

Related PDFs: NEXT APPLICATION, NEXT VERSION, SELECTED APPLICATION, SELECTED VERSION, SOURCE APPLICATION

STATUS CODE

This field is returned by certain file-related statements (for example, CREATE, OPEN, READ, etc.) that result in a false condition, thereby setting the next true/false status indicator to *F*. STATUS CODE is a token field and contains a value indicating the reason for the I/O command failure, such as:

- FI_AOF - Record Already on File (following a WRITE or REWRITE)
- FI_EOF - End of File (following a READNEXT)
- FI_NOF - Record Not in File (following a READ)

STATUS CODE can be used to set the primary key to the system message file (--- MESSAGE) to obtain a descriptive I/O error message, as shown in the example below.

```
        OPEN    TGL ACCTMS    SHARE? Y  FAIL 0 CACHE? N
    F   ERROR   Unexpected Error OPEN TGL ACCTMS (see error message below)
    F   SET     --- MESSAGE ID                =  --- STATUS CODE
    F   READ    --- MESSAGE                HOLD 0 FT 0 BY MESSAGE ID
    FT  SET     --- TEMP 80                   =  --- MESSAGE TXT
    FT  ERROR   TEMP 80
```

The descriptive message is the same message that would be generated by setting the FAIL (or FT) parameter on the I/O statement appropriately (where 0=nothing, 1=warning, 2=error, 3=cancel).

SUBHEADING

This field allows the end user to enter up to 79 characters to be printed as a subheading with the standard page header. It is available each time the predefined input process DISPOSITION is run.

```
SET      --- SUBHEADING                =      As of
APPEND   --- SUBHEADING                1  TAP WORK ALPHA AS OF DATE
```

SUBMIT DATE

This field contains the value of the *Submit Date* option, which can optionally be entered when the input process DISPOSITION is run, when a process is submitted to background. If supported by the operating system, SUBMIT DATE can cause a background process to delay execution until a specified date and time.

```
SET      --- SUBMIT DATE               =  TAP WORK DATE
```

Related PDFs: SUBMIT MODE, SUBMIT NOTIFY USER, SUBMIT OPTIONS, SUBMIT PRIORITY, SUBMIT PROCESS, SUBMIT QUEUE

SUBMIT MODE

This field contains the value of the *Mode* option, which can optionally be entered when the input process DISPOSITION is run, when a process is submitted to background. If supported by the operating system, a background process can either be run immediately or held in the background queue until a system operator releases it.

```
SET      --- SUBMIT MODE               =      HOLD
```

Valid Values:

HOLD (hold in background queue until released by operator)

RUN (begin execution immediately)

Related PDFs: SUBMIT DATE, SUBMIT NOTIFY USER, SUBMIT OPTIONS, SUBMIT PRIORITY, SUBMIT PROCESS, SUBMIT QUEUE

SUBMIT NOTIFY USER

This field contains the value of the *Notify?* option, which can optionally be entered when the input process DISPOSITION is run, when a process is submitted to background. If supported by the operating system, SUBMIT NOTIFY USER can alert the user when the background process has completed.

```
SET      --- SUBMIT NOTIFY USER          =      Y
```

Related PDFs: SUBMIT DATE, SUBMIT MODE, SUBMIT OPTIONS, SUBMIT PRIORITY, SUBMIT PROCESS, SUBMIT QUEUE

SUBMIT OPTIONS

This field contains a string of commands that are passed to the operating system as command line arguments when running background processes. Because these commands are operating system specific, you should review those requirements before attempting to use this field. The field can be SET to a constant or alpha field (as shown in the example below) containing the submission commands.

```
SET      --- SUBMIT OPTIONS             =  SYS ACCOUNTING MONTH END
```

Valid Values: Operating system specific.

Related PDFs: SUBMIT DATE, SUBMIT MODE, SUBMIT NOTIFY USER, SUBMIT PRIORITY, SUBMIT PROCESS, SUBMIT QUEUE

SUBMIT PRIORITY

This field contains the value of the *Priority* option, which can optionally be entered when the input process DISPOSITION is run, when a process is submitted to background. If supported by the operating system, SUBMIT PRIORITY can assign a specific priority to the background process.

```
SET      --- SUBMIT PRIORITY            =      10
```

Valid Values: Operating system specific.

Related PDFs: SUBMIT DATE, SUBMIT MODE, SUBMIT NOTIFY USER, SUBMIT OPTIONS, SUBMIT PROCESS, SUBMIT QUEUE

SUBMIT PROCESS

This field controls whether or not a process is submitted to background. If the end user selects the *Background* option from the DISPOSITION input screen, this field is automatically set to *Y*.

If background processing is selected for a job stream, any subsequent child process that is invoked as a separate task will be submitted to background independently.

Not all hardware platforms support background processing.

```
SET      --- SUBMIT PROCESS              =      Y
```

Valid Values: Operating system specific.

Related PDFs: SUBMIT DATE, SUBMIT MODE, SUBMIT NOTIFY USER, SUBMIT OPTIONS, SUBMIT PRIORITY, SUBMIT QUEUE

SUBMIT QUEUE

This field contains the value of the *Queue* option, which can optionally be entered when the input process DISPOSITION is run, when a process is submitted to background. If supported by the operating system, SUBMIT QUEUE can assign a specific job queue to the background process.

```
SET      --- SUBMIT QUEUE                =      TAPU
```

Valid Values: Operating system specific.

Related PDFs: SUBMIT DATE, SUBMIT MODE, SUBMIT NOTIFY USER, SUBMIT OPTIONS, SUBMIT PRIORITY, SUBMIT PROCESS

TEMP 1

This field is a temporary alphanumeric work area that can be used by any statement series. Substring manipulation of this field is provided through the use of the SET TEMP statement. This field can also be used in CANCEL, ERROR, MESSAGE, and WARNING statements to provide variable messages.

```
SET      --- TEMP 1                      =      0
```

Related PDFs: See all other TEMP PDFs.

TEMP 2

This field is a temporary alphanumeric work area that can be used by any statement series. Substring manipulation of this field is provided through the use of the SET TEMP statement. This field can also be used in CANCEL, ERROR, MESSAGE, and WARNING statements to provide variable messages.

```
SET      --- TEMP 2                      =      NO
```

Related PDFs: See all other TEMP PDFs.

TEMP 4

This field is a temporary alphanumeric work area that can be used by any statement series. Substring manipulation of this field is provided through the use of the SET TEMP statement. This field can also be used in CANCEL, ERROR, MESSAGE, and WARNING statements to provide variable messages.

```
          SET      --- TEMP 4                    =      NEXT
```

Related PDFs: See all other TEMP PDFs.

TEMP 30

This field is a temporary alphanumeric work area that can be used by any statement series. Substring manipulation of this field is provided through the use of the SET TEMP statement. This field can also be used in CANCEL, ERROR, MESSAGE, and WARNING statements to provide variable messages.

```
          SET      --- TEMP 30                   =      PROCESSING COMPLETE
```

Related PDFs: See all other TEMP PDFs.

TEMP 30 CTR

This field is a temporary alphanumeric work area that can be used by any statement series. Substring manipulation of this field is provided through the use of the SET TEMP statement. The contents of this field are automatically centered. This field can also be used in CANCEL, ERROR, MESSAGE, and WARNING statements to provide variable messages.

```
          SET      --- TEMP 30 CTR               =      PROCESSING COMPLETE
```

Related PDFs: See all other TEMP PDFs.

TEMP 30 LJ

This field is a temporary alphanumeric work area that can be used by any statement series. Substring manipulation of this field is provided through the use of the SET TEMP statement. The contents of this field are automatically left justified. This field can also be used in CANCEL, ERROR, MESSAGE, and WARNING statements to provide variable messages.

```
SET      --- TEMP 30 LJ                =      PROCESSING COMPLETE
```

Related PDFs: See all other TEMP PDFs.

TEMP 30 RJ

This field is a temporary alphanumeric work area that can be used by any statement series. Substring manipulation of this field is provided through the use of the SET TEMP statement. The contents of this field are automatically right justified. This field can also be used in CANCEL, ERROR, MESSAGE, and WARNING statements to provide variable messages.

```
SET      --- TEMP 30 RJ                =      PROCESSING COMPLETE
```

Related PDFs: See all other TEMP PDFs.

TEMP 32

This field is a temporary alphanumeric work area that can be used by any statement series. Substring manipulation of this field is provided through the use of the SET TEMP statement. This field can also be used in CANCEL, ERROR, MESSAGE, and WARNING statements to provide variable messages.

```
SET      --- TEMP 32                   =      PROCESSING COMPLETE
```

Related PDFs: See all other TEMP PDFs.

TEMP 79

This field is a temporary alphanumeric work area that can be used by any statement series. Substring manipulation of this field is provided through the use of the SET TEMP statement. This field can also be used in CANCEL, ERROR, MESSAGE, and WARNING statements to provide variable messages.

```
SET TEMP 79    AT 005 FOR 008 FROM 001 OF  TAP WORK DATA
```

Related PDFs: See all other TEMP PDFs.

TEMP 80

This field is a temporary alphanumeric work area that can be used by any statement series. Substring manipulation of this field is provided through the use of the SET TEMP statement. This field can also be used in CANCEL, ERROR, MESSAGE, and WARNING statements to provide variable messages.

```
          SET TEMP 80    AT 005 FOR 052 FROM 001 OF  TAP WORK DATA
```

Related PDFs: See all other TEMP PDFs.

TEMP 132

This field is a temporary alphanumeric work area that can be used by any statement series. Substring manipulation of this field is provided through the use of the SET TEMP statement. This field can also be used in CANCEL, ERROR, MESSAGE, and WARNING statements to provide variable messages.

```
          SET TEMP 132   AT 005 FOR 052 FROM 002 OF  TAP WORK DATE
```

Related PDFs: See all other TEMP PDFs.

TEMP 256

This field is a temporary alphanumeric work area that can be used by any statement series. Substring manipulation of this field is provided through the use of the SET TEMP statement. This field can also be used in CANCEL, ERROR, MESSAGE, and WARNING statements to provide variable messages.

```
          SET TEMP 256   AT 005 FOR 052 FROM 001 OF  TAP WORK DATA
```

Related PDFs: See all other TEMP PDFs.

TEMP 512

This field is a temporary alphanumeric work area that can be used by any statement series. Substring manipulation of this field is provided through the use of the SET TEMP statement. This field can also be used in CANCEL, ERROR, MESSAGE, and WARNING statements to provide variable messages.

```
          SET TEMP 512   AT 005 FOR 052 INTO 001 OF  TAP WORK DATA
```

Related PDFs: See all other TEMP PDFs.

TEMP 2K

This field is a temporary alphanumeric work area that can be used by any statement series. Substring manipulation of this field is provided through the use of the SET TEMP statement. This field can also be used in CANCEL, ERROR, MESSAGE, and WARNING statements to provide variable messages.

```
          SET TEMP 2K    AT 005 FOR 052 FROM 001 OF  TAP WORK DATA
```

Related PDFs: See all other TEMP PDFs.

TEMP 8K

This field is a temporary alphanumeric work area that can be used by any statement series. Substring manipulation of this field is provided through the use of the SET TEMP statement. This field can also be used in CANCEL, ERROR, MESSAGE, and WARNING statements to provide variable messages.

```
          SET TEMP 8K    AT 005 FOR 052 FROM 001 OF  TAP WORK DATA
```

Related PDFs: See all other TEMP PDFs.

TEMP 32K

This field is a temporary alphanumeric work area that can be used by any statement series. Substring manipulation of this field is provided through the use of the SET TEMP statement. This field can also be used in CANCEL, ERROR, MESSAGE, and WARNING statements to provide variable messages.

```
          SET TEMP 32K   AT 005 FOR 052 FROM 001 OF  TAP WORK DATA
```

Related PDFs: See all other TEMP PDFs.

TEXT AT POSITION

This field is set after an IF, AND, or OR statement using the *IN* relation is executed, which returns a true/false value of *T* (indicating that the first field includes the second field). The field indicates the starting position of the character string within the first field.

```
          IF       TAP WORK DATA                  IN     JUNK
    T     SET      --- AI                         =  --- TEXT AT POSITION
    T     SET      --- TEMP 79                    =      MESS
    T     SET TEMP 79    AT 001 FOR 004 INTO AI  OF  TAP WORK DATA
```

In this example, the first occurrence of the text *JUNK* in the field *TAP WORK DATA* is replaced by the text *MESS*.

USER ID

This field contains the three-character logon ID for the current user.

```
          IF       --- USER ID                    NE TAP WORK USER ID
          AND      TAP OP EXCLUSIVE               EQ     1
    T     ERROR    Cannot Edit Another User's Data
```

VERSION

This identifies the version of the application that contains the process currently being executed. This value does not change if a COPY or GOSUB statement is executed that references a different application.

```
          IF       --- APPLICATION                EQ     TAP
          AND      --- VERSION                    EQ     00
    T     SET      --- PRINT ON SCREEN            =      Y
```

Related PDFs: APPLICATION, PROCESS NAME, PROCESS TYPE

WARNINGS

This field contains a count of the number of warnings generated so far within the current subprocess family, either by the use of the WARNING statement or by data lookup warning conditions. It can also be set to zero at any time, and then interrogated at a later time, to determine if any warnings were encountered between those two points.

```
          SET      TAP WORK NO WARNINGS           =  --- WARNINGS
```

Related PDF: ERRORS

Predefined Processes

APPX provides a number of standard processes in the runtime application that you can invoke, or are automatically invoked, at appropriate points in your applications. For example, APPX contains a standard DISPOSITION input process that allows users to specify certain print-related information that controls how, when, and where a report is to print. To invoke this process as a child in an application, you identify the application ID as "---", the process type as *INPUT*, and the process name as *DISPOSITION*.

When you use a predefined process, you designate the application ID with three hyphens (---). APPX interprets the three hyphens as the runtime application. (The actual application ID is *0LA*.)

You cannot modify any of the predefined processes, although you can, for example, develop your own version of a DISPOSITION process by defining an input process in your own application. Then, instead of executing the runtime application's DISPOSITION process, you invoke the appropriate input process from your own application.

Some of the predefined processes are shown as Optional Child Processes on a menu in *Figure 4-7-1*.

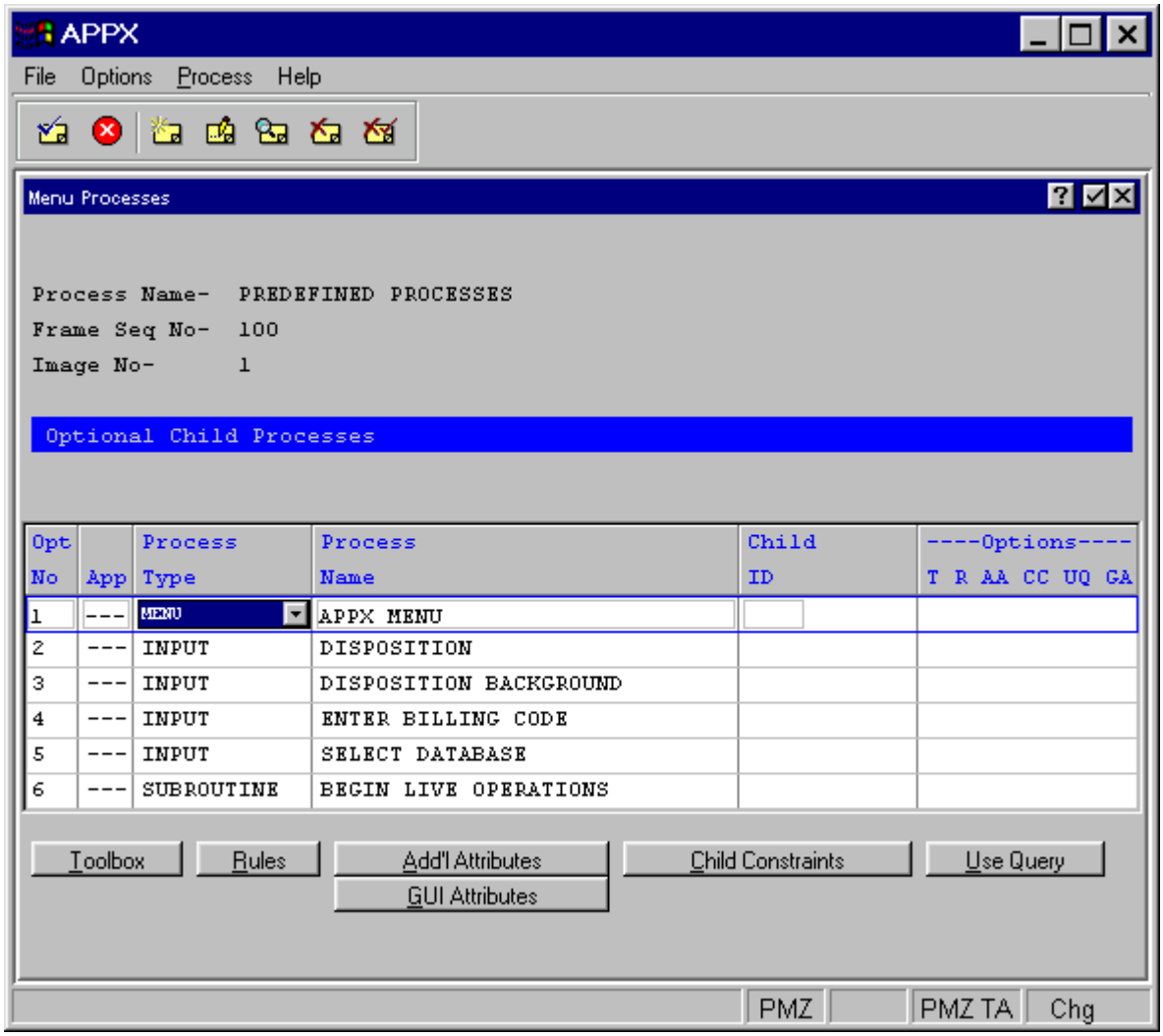


Figure 4-7-1. Some Predefined Processes Shown As Optional Child Processes

Designer-Invoked Processes

The runtime application contains several predefined processes. In general, these processes can be placed into two categories: processes that a designer can invoke, and processes that only APPX can invoke. Most of the predefined processes are only invoked by APPX.

Following is a list of the predefined processes that a designer can invoke (each of which is described in detail later in this chapter):

- **APPX MENU**, which allows a user to select a new database and provides access to system administration, application design, and database management.
- **BEGIN LIVE OPERATIONS**, which transfers the run phase from initial setup or recovery processing to live operations.
- **CHECK GUI CLIENT**, which determines if the current engine is connected to a GUI-enabled client.
- **DISPOSITION**, which allows a user to provide information that controls how and where a report is to print.
- **ENTER BILLING CODE**, which allows a user to enter a value into the predefined field BILLING CODE.
- **GET CMD LINE PARM**, which checks for the existence of a parameter on the command line used to invoke the current engine.
- **GET ENV VAR**, which retrieves the contents of an environment variable.
- **GET FILE FROM CLIENT**, which retrieves a file from the client system and stores it on the server.
- **PROCESS QUERY RECORD**, which potentially adds a PCF record to the result set of a query.
- **SELECT DATABASE**, which allows a user to select the appropriate database.
- **SEND FILE TO CLIENT**, which retrieves a file from the server and stores it on the client system.

APPX MENU

This menu is the default startup process for all users. In addition to allowing a user to select a new database, the APPX MENU process provides access to system administration, application design, and database management. This menu, shown in *Figure 4-7-2,* can be invoked at any time and is typically used as one of the steps in a startup job. For example, consider the following job:

```
100    UPDATE     SYS LOG IN USER
200    MENU       --- APPX MENU
300    UPDATE     SYS LOG OUT USER
```

The first and last steps of the above job may write records to a customized system log file that tracks each user as they enter and exit APPX.

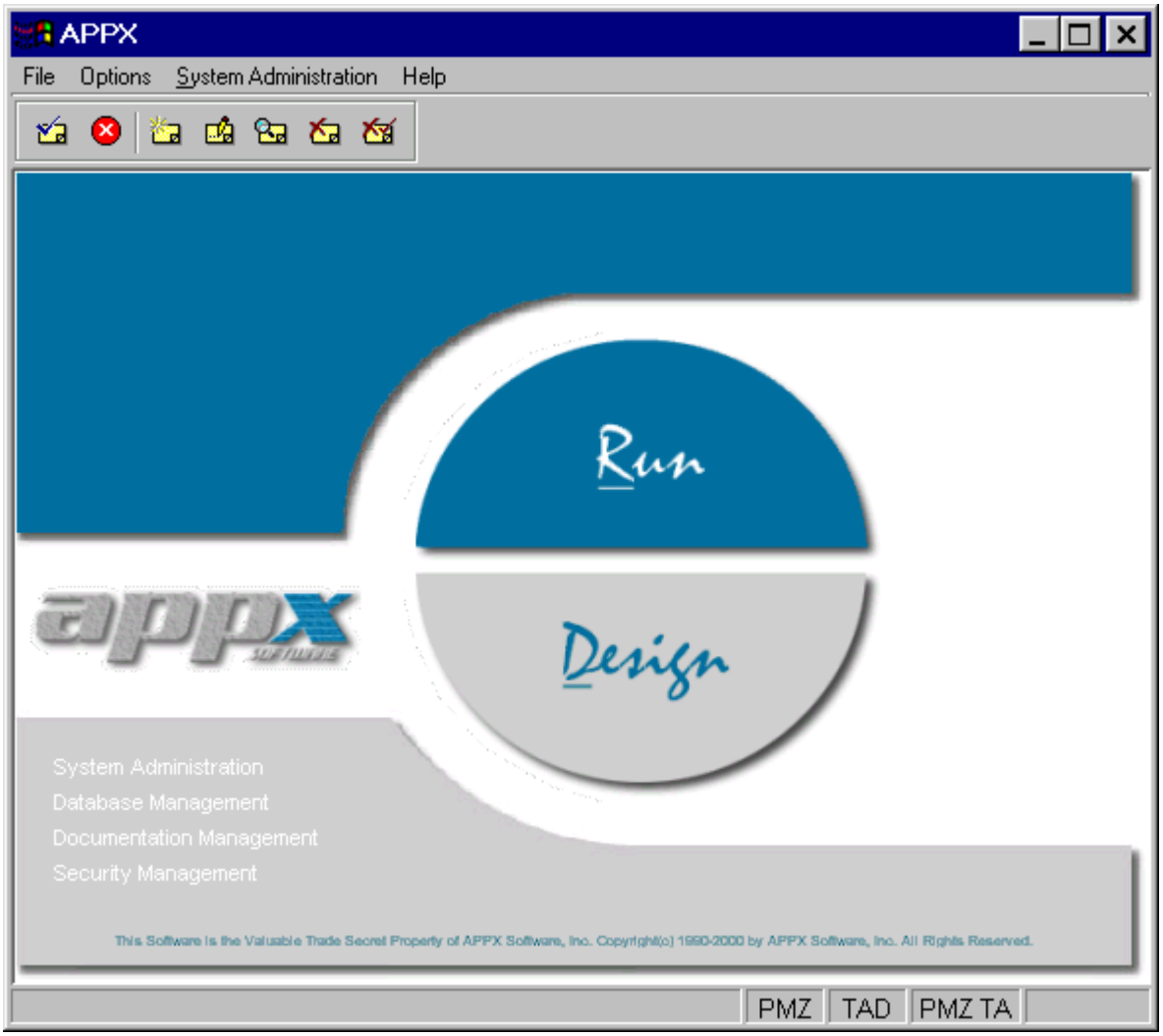


Figure 4-7-2. Menu Invoked by APPX MENU Predefined Process

BEGIN LIVE OPERATIONS

This subroutine process allows an end user to move automatically from either the initial setup operational phase or the recovery processing operational phase, to the live operations phase. The optional child job process that invokes this subroutine should have a Parent Disposition of *End After*; a MENU statement with the proper application ID and a blank process name should be defined in the *Post Invocation* event point of the job process. This will cause the main Live Operations menu for that application to display to the end user after the operational phase is changed.

CHECK GUI CLIENT

This subroutine checks to see if the current engine is connected to a GUI-enabled client. No arguments are passed to the subroutine. It simply sets the predefined field RETURN CODE to 1 if running under a GUI-enabled client.

The subroutine can be invoked via SUBR or GOSUB. However, if invoked via SUBR, it must not be invoked DETACHED.

The following example generates a message or warning depending on whether or not the user is running under a GUI-enabled client.

```
        *
        SUBR     --- CHECK GUI CLIENT                SUBPROCESS   END? N  FAIL 0
        *
        IF       --- RETURN CODE               EQ     1
    T   MESSAGE  User is running under GUI-enabled client.
    F   WARNING  User is not running under GUI-enabled client.
```

DISPOSITION

This is the input process that a designer uses to allow the end user to choose a printer, or choose whether to print a report or log to the screen, or whether to submit a process to background. If DISPOSITION is not invoked, any subsequent output will be directed to the screen or printer according to the defaults set in system administration.

This process, shown in *Figure 4-7-3,* can be invoked at any time, but must not be invoked as DETACHED, and must not be specified as a “Separate Task” (in other words, a designer should never allow this process to be submitted to background). If either of these parameters are wrong, the invocation will have no effect.

Before invoking this process, the designer can override the default values by modifying the corresponding PRINT... or SUBMIT... predefined fields. If the PDFs are modified using SET statements in the *Pre-Invocation* event point, the user will still be able to manually modify the value. However, if the PDFs are modified by using child constraints, the values will display but cannot be changed by the user.

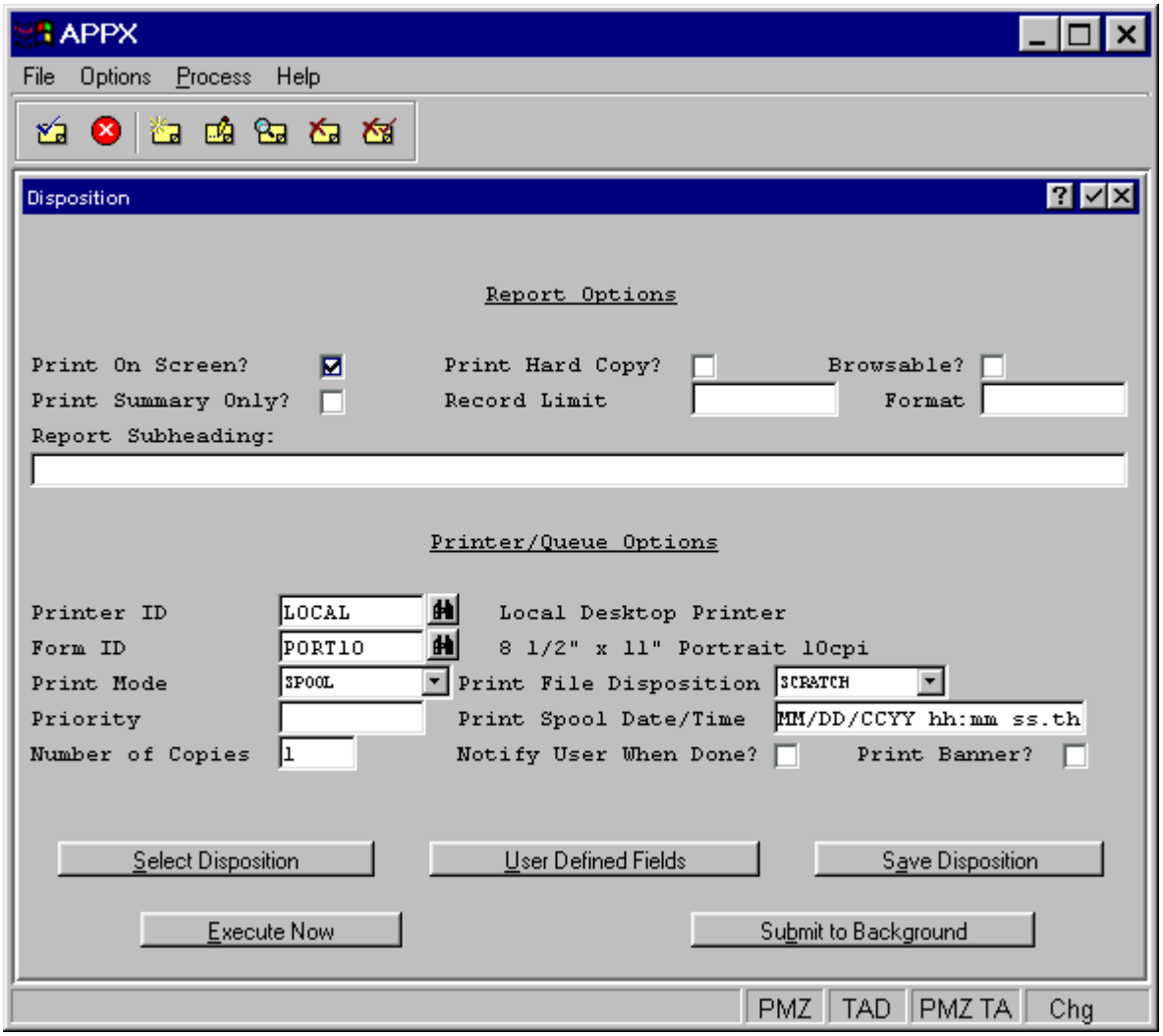


Figure 4-7-3. Disposition Predefined Process

If the end user selects the SUBMIT TO BACKGROUND option, DISPOSITION invokes the DISPOSITION (BACKGROUND) predefined input process as a subprocess.

DISPOSITION (BACKGROUND)

This is the input process that allows an end user to choose a priority, spool date, mode, and other options for background processing. It is not explicitly invoked by the designer, but rather invoked as a subprocess by DISPOSITION, when the end user selects the SUBMIT TO BACKGROUND option. The background disposition screen is shown in *Figure 4-7-4* below:

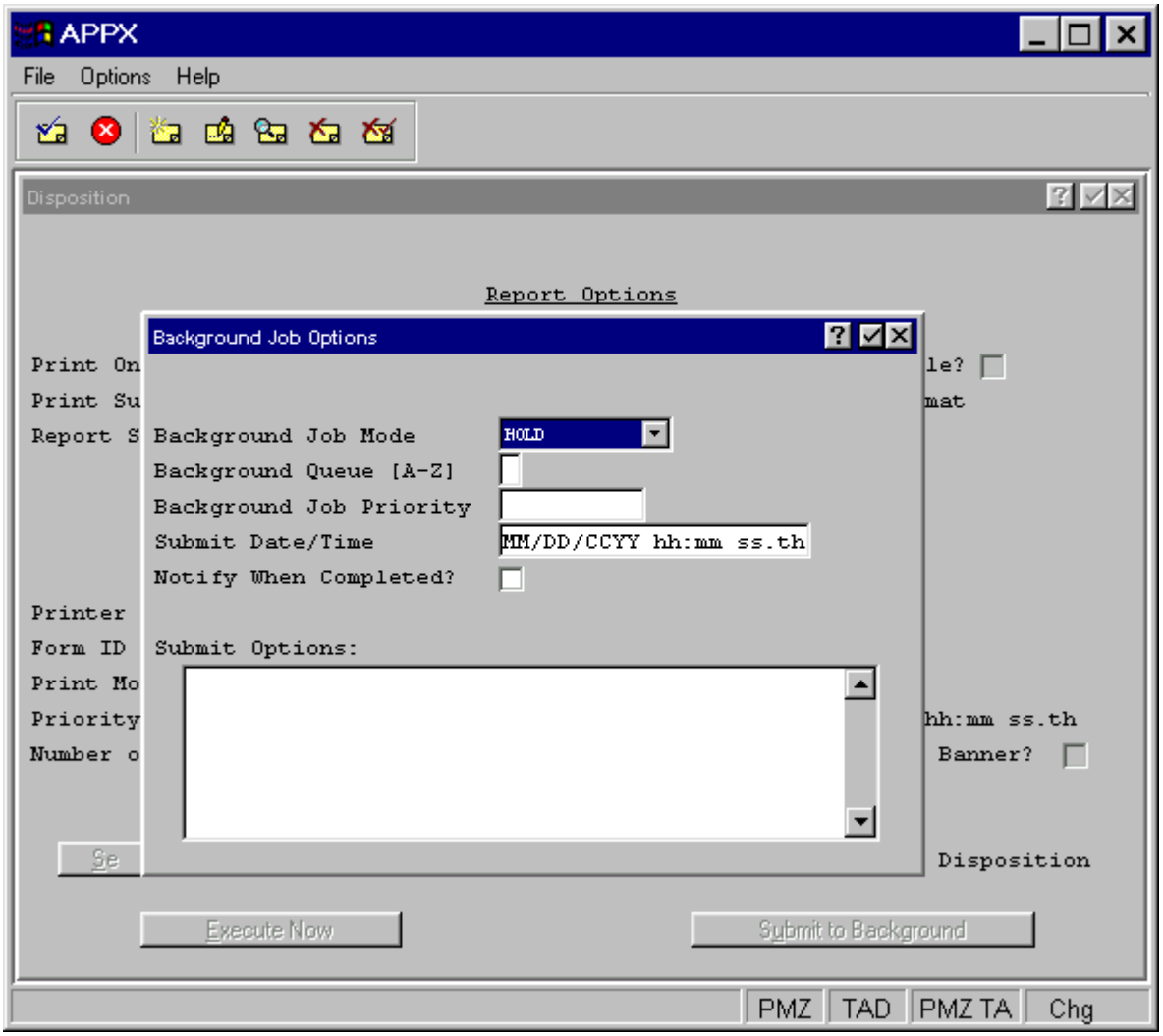


Figure 4-7-4. Background Disposition Predefined Process

ENTER BILLING CODE

This process, shown in *Figure 4-7-5,* allows the user to enter a value into the predefined field BILLING CODE. When the usage log is implemented, this field will be saved as a part of each record in the usage log.

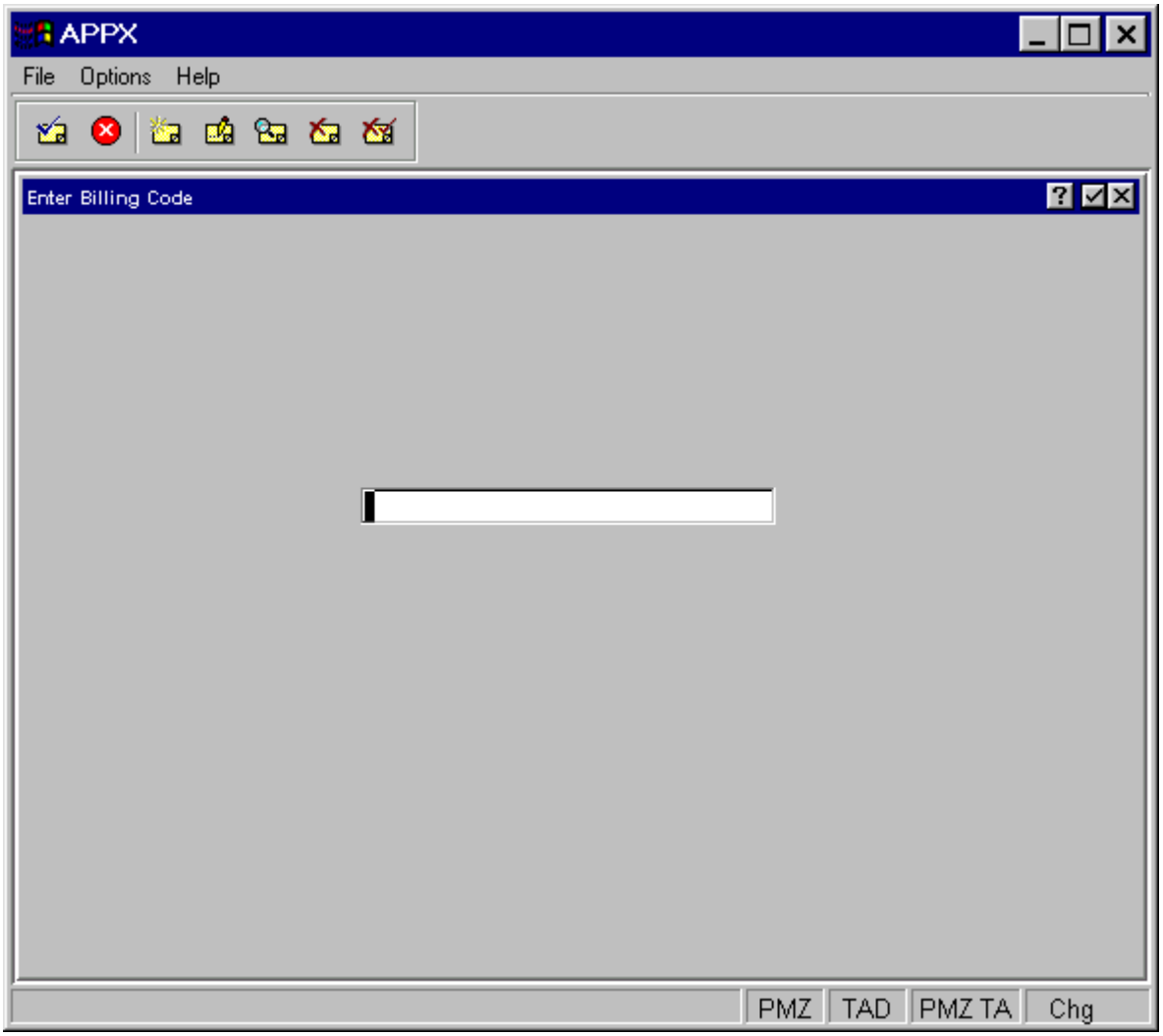


Figure 4-7-5. Enter Billing Code Predefined Process

GET CMD LINE PARM

This subroutine checks for the existence of a parameter on the command line used to invoke the current engine. Two arguments are expected to be passed to the subroutine:

- Parameter to be checked (alpha field, maximum length 132)
- Buffer for returned value (alpha field, maximum length 2048 or 2K)
- Must be PASS'ed shared.
- Returns specified parameter only if it existed on the command line.

The subroutine can be invoked via SUBR or GOSUB.

The following example generates a message or warning depending on whether or not the parameter *-m* existed on the command line when the APPX engine was invoked.

```
          SET      --- TEMP 132                  =      -m
          SET      --- TEMP 256                  =
          *
          PASS     --- TEMP 132                  FIELD            SHARE? N
          PASS     --- TEMP 256                  FIELD            SHARE? Y
          SUBR     --- GET CMD LINE PARM               DETACHED     END? N  FAIL 0
          *
          IF       --- TEMP 256                  EQ --- TEMP 132
     T    APPEND   --- TEMP 132                  1      found
     F    APPEND   --- TEMP 132                  1      not found
     -    APPEND   --- TEMP 132                  1      on command line
     T    MESSAGE  TEMP 132
     F    WARNING  TEMP 132
```

GET ENV VAR

This subroutine retrieves the contents of an environment variable. Two arguments are expected to be passed to the subroutine:

- Desired environment variable (alpha field, maximum length 132)
- Buffer for returned value (alpha field, maximum length 2048 or 2K)
- Must be PASS'ed shared.
- Returns contents of specified environment variable.

The subroutine can be invoked via SUBR or GOSUB.

The following example retrieves the contents of the PATH environment variable and stores it in a work field.

```
          SET      --- TEMP 132                  =      PATH
          SET      --- TEMP 2K                   =
          *
          PASS     --- TEMP 132                  FIELD            SHARE? N
          PASS     --- TEMP 2K                   FIELD            SHARE? Y
          SUBR     --- GET ENV VAR                     DETACHED     END? N  FAIL 0
          *
          SET      CSY WORK PATH ENV VAR         =  --- TEMP 2K
```

GET FILE FROM CLIENT

This subroutine retrieves a file from the client system and stores it on the server. Three arguments are expected to be passed to the subroutine:

- Source pathname (alpha field, maximum length 256)
- If blank, the client will open a dialog browser so the user can select the file.

- Should be PASS'ed shared to allow for return of user-selected file.
- If the source pathname is relative, it will be relative to the location of the APPX client jar file.
- Destination pathname (alpha field, maximum length 256)
- Field for status indicator (numeric field)
- Must be PASS'ed shared.
- Returns status code (0=Failure, 1=Success).

The subroutine can be invoked via SUBR or GOSUB. While the file is in transit, the client will show a progress slider and the APPX session will wait until it completes before returning.

The following example retrieves a user-selected file from the APPX client and stores it on the APPX server. It displays a message (or warning) reporting the name of the file the user selected and whether or not it was retrieved successfully.

```
          SET      --- TEMP 132                  =
          SET      --- TEMP 256                  =      /tmp/testfile.txt
          SET      --- XI                        =      0
          *
          PASS     --- TEMP 132                  FIELD            SHARE? Y
          PASS     --- TEMP 256                  FIELD            SHARE? N
          PASS     --- XI                        FIELD            SHARE? Y
          SUBR     --- GET FILE FROM CLIENT            DETACHED     END? N  FAIL 0
          *
          IF       --- XI                        EQ     1
     T    APPEND   --- TEMP 132                  1      retrieved OK
     F    APPEND   --- TEMP 132                  1      not retrieved
     -    APPEND   --- TEMP 132                  1      from client
     T    MESSAGE  --- TEMP 132
     F    WARNING  --- TEMP 132
```

PROCESS QUERY RECORD

This subroutine potentially adds a PCF record to the result set of a query (provided the record meets the query constraints). No arguments are passed to the subroutine, and no values are returned.

The subroutine can only be invoked via GOSUB. Also, it can only be used within the *Establish PCF Range* event point in a query process (or in a subroutine called from that event point).

The following example processes two ranges of vendor records for query selection. Only vendors 10-19 and 1000-1999 will be selected for printing if on file and then only if they also satisfy any runtime constraints.

```
          *        Process first range of records
          SET      TAP VENDOR VENDOR NO          =      10
          BEG AT   TAP VENDOR   IN TAP VENDOR VENDOR NO
          SET      TAP VENDOR VENDOR NO          =      19
          END AT   TAP VENDOR   IN TAP VENDOR VENDOR NO
```

```
BEG READ TAP VENDOR                HOLD 0 KEY IS  VENDOR VENDOR NO
GOSUB    --- PROCESS QUERY RECORD
END READ TAP VENDOR
*        Process second range of records
SET      TAP VENDOR VENDOR NO           =      1000
BEG AT   TAP VENDOR   IN TAP VENDOR VENDOR NO
SET      TAP VENDOR VENDOR NO           =      1900
END AT   TAP VENDOR   IN TAP VENDOR VENDOR NO
BEG READ TAP VENDOR                HOLD 0 KEY IS  VENDOR VENDOR NO
GOSUB    --- PROCESS QUERY RECORD
END READ TAP VENDOR
```

SELECT DATABASE

Used in conjunction with the NEXT DATABASE predefined field, this subroutine process accesses a different database. This database becomes the current database and is used by any subsequent detached child processes.

To use this subroutine, set the NEXT DATABASE field to the new database ID, then call the SELECT DATABASE predefined subroutine as a subprocess.

```
SET      --- NEXT DATABASE              =      TST
SUBR     --- SELECT DATABASE                  SUBPROCESS   END? N  FAIL 0
```

Another method of changing databases based on user selection is the use of designer-defined subroutines. These subroutines can be executed from menus when a user makes a selection to change from one database to another. For example, if a menu contains an option for a user to change from the current database to database ABC, the corresponding optional child would be a subroutine called, perhaps, SELECT DATABASE ABC. You can create a subroutine for every database defined for your installation and allow users to select them via menu options. These subroutines can also be executed from jobs or other processes when navigation between multiple databases is required.

The aforementioned subroutine would consist of these ILF statements:

```
SET      --- NEXT DATABASE              =      ABC
SUBR     TSY SELECT DATABASE ABC              SUBPROCESS   END? N  FAIL 0
```

SEND FILE TO CLIENT

This subroutine retrieves a file from the server and stores it on the client system. Three arguments are expected to be passed to the subroutine:

- Source pathname (alpha field, maximum length 256)
- Destination pathname (alpha field, maximum length 256)
- If blank, the client will open a dialog browser so the user can select the file.

- Should be PASS'ed shared to allow for return of user-selected file.
- If the destination pathname is relative, it will be relative to the location of the APPX client jar file.
- Field for status indicator (numeric field)
- Must be PASS'ed shared.
- Returns status code (0=Failure, 1=Success).

The subroutine can be invoked via SUBR or GOSUB. While the file is in transit, the client will show a progress slider and the APPX session will wait until it completes before returning.

The following example retrieves a file from the APPX server and stores it in the user-selected file on the APPX client. A message (or warning) is used to report the name of the file the user selected and whether or not it was sent successfully.

```
          SET      --- TEMP 132                  =      /tmp/testfile.txt
          SET      --- TEMP 256                  =
          SET      --- XI                        =      0
          *
          PASS     --- TEMP 132                  FIELD            SHARE? N
          PASS     --- TEMP 256                  FIELD            SHARE? Y
          PASS     --- XI                        FIELD            SHARE? Y
          SUBR     --- SEND FILE TO CLIENT             DETACHED     END? N  FAIL 0
          *
          IF       --- XI                        EQ     1
     T    APPEND   --- TEMP 256                  1      sent OK
     F    APPEND   --- TEMP 256                  1      not sent
     -    APPEND   --- TEMP 256                  1      to client
     T    MESSAGE  --- TEMP 256
     F    WARNING  --- TEMP 256
```