CodeReporter 2.0 ™

The Report Writer for C/C++, Visual Basic & Delphi Developers

dBASE Compatible

FoxPro Compatible

Clipper Compatible

Sequiter Software Inc.

Contents

Introduction

CodeReporter is a comprehensive relational report designer that takes the painstaking work out of creating custom reports.

Use CodeReporter to design intricate reports visually using simple point-and-click commands. The report data can come from any data file in any directory on any drive or network drive. The ties between the data files used in the report are made using simple dBASE expressions. Use only a portion of the data files by creating a query -- again using a simple logical dBASE expression. CodeReporter takes care of the complexities.

The speed of the Sequiter Software's Query Technology is clearly evident in all aspects of CodeReporter. Sort, query, and relate at unmatched speed. CodeReporter can query a 500,000 record data file and begin output in just one second.

Reports designed using CodeReporter under Windows are portable and configurable. Use the same report in any application by loading a report file or by generating source code that can be linked directly into DOS, OS/2, UNIX , and/or Windows applications. Once the report structures are obtained (hard or soft coded), the CodeReporter API functions may modify them to create a truly custom solution.

CodeReporter takes advantage of Windows display and printer drivers when outputting reports. Any part of the report can be made to look unique by changing the size, typeface, and/or color. Using the Windows standard and TrueType fonts, a report will look exactly the same on the printer as it does in the screen preview.

These features and more, make CodeReporter an indispensable tool for any software designer or end user!

Features

CodeReporter integrates the powerful features of many DOS-based report writers with the added benefits of Windows and Sequiter Software's high performance database capabilities.

Relations Relate any number of data files using any one of the following techniques:

- One to One. Have a direct correspondence between the data files.
- One to Many. Have a unique search value from one data file retrieve multiple records in a related file.
- Many to Many. Have any number of duplicate search values from one data file retrieve multiple records in a related file.

The relations may be joined together in any combination, so that relations such as a one to one to many to many to one are completely supported.

Queries Limit the scope of the records used in a report using simple or complex dBASE expressions involving fields from any data file used in the relation. For instance if data file 'A' is related to data file 'B' which in turn is related to data file 'C', and 'C' contains a value that shouldn't be used in the report, an expression such as `C->FIELD_NAME<>'VALUE'` could be used.

Totals Numeric information in a report is often most useful in a summarized format. Long columns of numbers mean very little until the 'bottom line' is seen. CodeReporter can summarize numerical data in a number of ways:

- Sum. Maintain a numerical sum of a group of numbers.
- Highest/Lowest. Store the largest or smallest numeric value encountered.
- Average. Maintain the mean value.

Look Ahead Totals may also be set as "look ahead" totals. This lets a total skip ahead of the actual report output and continue its accumulation -- so that when it is outputted, it displays the sum of information in the report before that information is displayed. This total can then, for example, be used in calculations later in the report to display a percentage of the total.

Fonts CodeReporter makes full use of the fonts available under Windows, including TrueType fonts. Use any number of typefaces, sizes and colors in a report, simply by creating a style. If Windows can display a font, it can be used in a report.

Cut and Paste Move any output object in the report with familiar click-and-drag mouse commands. Familiar cut and paste commands can also be used to make multiple copies of output objects.

Display to Screen Save paper when designing a report by sending test output to the screen. Since CodeReporter uses Windows fonts, the fonts appear the same on paper as on the screen.

Save as Code Once a report is designed using CodeReporter, save it in a soft coded report file for the next CodeReporter session, or save it as C source code. The generated code can then be included in any CodeBase C/C++ application under DOS, Windows or OS/2.

Graphics Customize reports with pictures! Either place static graphic elements such as a company logo, or load graphic elements on-the-fly as the report is running.

Print to a DataFile CodeReporter 2.0 includes the ability to output a report to a data file. This new feature lets a user do a report based on the contents of a previous report, or use CodeReporter as a data transformation tool.

Distribution Application developers may purchase additional copies of CodeReporter at bulk rates and resell them to their end users. Contact Sequiter Software Inc. for further details.

Getting Started

System Requirements

The reports created with CodeReporter and the report functions may be used under any platform. CodeReporter itself, however, must be run under Windows NT or ‘95/’98.

CodeReporter requires a Windows-compatible mouse.

Recommended

The following configuration is a suggested minimum to make full use of CodeReporter:

- IBM 386 or true compatible processor running at 25 MHz or higher
- 4MB RAM or more
- Windows-compatible SVGA color monitor
- A two-button Windows-compatible mouse
- A hard drive with at least 1.6 MB free space for each version of CodeReporter installed. (Depending upon the report, additional space may be required during its execution.)

CodeReporter runs under almost any configuration that runs Windows, however, the more the powerful computer, the faster the reports will be generated.

Registration

Please take a moment to complete and mail in your CodeReporter registration card. Doing so assures you of quick technical support and notification of upgrades.

Installation

When purchased in a bundle with other Sequiter products (such as CodeBase 6), CodeReporter may be installed through the bundled package's installation program . When purchased separately, CodeReporter may be installed by executing the INSTALL.EXE program on the CodeReporter #1 disk.

File Formats

CodeReporter relies on the CodeBase database engine for opening the data, index and memo files used in the creation and display of reports. Therefore, CodeReporter can be used with any of the file formats supported by CodeBase. Presently this includes FoxPro, dBASE IV and Clipper formats. During the installation program, the general file format option that you select is used to install the necessary CodeReporter files needed to support that format.

CodeReporter supports the following index file formats:

- dBASE IV indexes (.MDX). When using this compatibility it is not possible to access the dBASE III PLUS indexes (.NDX) even though dBASE IV may access both.
- FoxPro 2.5 (and higher) compact indexes (.CDX). CodeReporter does not support earlier, non-compact versions of FoxPro indexes.
- Clipper indexes (.NTX). CodeReporter works with both Clipper Summer '87 and Clipper 5.x.

Starting CodeReporter

CodeReporter is a 32-bit Microsoft Windows application --to run it you need Windows NT or Windows ‘95/’98. By default, the CodeReporter executable CREP2.EXE, and its support files, are installed in the C:\CODEBASE\CODEREP sub-directory.

To run the application, locate the CodeReporter program using File Manager (Windows NT) or the Windows Explorer (Windows ‘95/’98). Double-clicking on the CodeReporter program item icon will launch the application.

You can also set up permanent program item icons (Windows NT) or program shortcuts (Windows ‘95/’98) to make it easier to launch CodeReporter. Refer to your Windows user’s guide for more information.

Command-line Arguments

CodeReporter can accept a single command line argument which represents the name of a previously saved CodeReporter report file. CodeReporter will attempt to load the report and any data files associated with it.

Command-line arguments can be associated with permanent program item icons and shortcuts. In addition, under Window ‘95/’98 and NT, command line arguments can be specified by invoking CodeReporter from the command line of a DOS window. For example:

```
c:\codebase\coderep> crep2 myfile.rep
```

Accessing Report Files

Reports created with CodeReporter may be saved in CodeReporter report files, which have a ".REP" extension. These report files contain all the specific information of the report--including the path to the data files in the report's relation and the styles used in the report.

Loading a Report

A previously created report may be retrieved by selecting the **FILE | OPEN** menu option. Use the "File Open" common dialog to locate the desired report file and select the "OK" button.

If an attempt is made to load a report file when one is already loaded, CodeReporter closes the current report.

FILE / OPEN WITH PATH may also be used to open a report file. In addition to prompting for the name of the file, the **FILE / OPEN WITH PATH** menu option prompts for the directory in which the data files of the report may be found.

Use FILE | OPEN WITH PATH when the data files for a report have been moved to another directory.

1.0 Files CodeReporter version 1.0 files may be imported into CodeReporter 2.0 using the **FILE | OPEN OLD FILE** menu option.

Starting a New Report

A new report may be created by selecting the **FILE | NEW** menu option. This closes the current report and invokes the "File Open" common dialog so that the top master data file may be selected.

CodeReporter automatically creates and displays a new group named "BODY" in the new report. The file name for the new report is specified when the file is saved.

Saving a Report

The **FILE | SAVE** menu option causes CodeReporter to save to disk the changes that have been made to a report. If a report has not previously been saved (i.e. a new report), CodeReporter uses the "File Open" common dialog to obtain the report's file name.

With a New Name

A report may be saved under a new name using the **FILE | SAVE AS** menu option. This causes CodeReporter to prompt for a new file name in the same manner as if the report were a new report.

Manual Conventions

Listed below are the typographic conventions used within this manual.

Menu options are listed in a bold, small capital letter, Arial font. If a sub-menu is listed, the main menu listing is followed by the '|' character and the sub-menu listing. In addition, the words "menu option" generally follow the name of the menu.

Example: Choose the **File | New** menu option.

The names of dialog boxes are encased in double quotation marks and use the manual's regular font. In addition, the words "dialog" or "dialog box" generally follow the name of the dialog.

Example: Use the "Easy Expression" dialog box to enter dBASE expressions.

The names of controls within a dialog box or within the CodeReporter design screen are encased in double quotation marks and listed in the manual's regular font. In addition, the type of control is generally listed after the name of the control.

Example: Choose an entry in the "Fields" list box and select the "OK" button.

dBASE expressions are in a typewriter font, and are generally in upper case (the only exception is with literal text that is included in a dBASE expression). Fields referenced within a dBASE expression are qualified with the name of the data file to which they belong and a '->'. See the Expressions chapter for information about dBASE expressions.

Example:

```
'Name: '+ DATA->NAME+'Age: '+STR(DATA->AGE,3,0)
```

All directories listed within this manual are assumed to be subdirectories of the CodeReporter directory (default: C:\CODEREP). When subdirectories are listed, the CodeReporter directory is listed as '.\', which means "this directory." Therefore, the .\EXAMPLES directory is the same as the C:\CODEREP\EXAMPLES directory.

Icons

At various points throughout this manual, the following icons are used to bring attention to important information.

This icon indicates that the particular function only relates to Microsoft Windows applications. This is only available if the report module functions have been compiled with the **S4WINDOWS** switch. This icon incorporates an image copyrighted by Microsoft Corporation.

This icon indicates that the particular function may not be applied to Microsoft Windows applications. This icon incorporates an image copyrighted by Microsoft Corporation.

Report Design Screen

At various points, this manual makes reference to elements within the CodeReporter design screen. These elements are illustrated in Figure 1, on the next page.

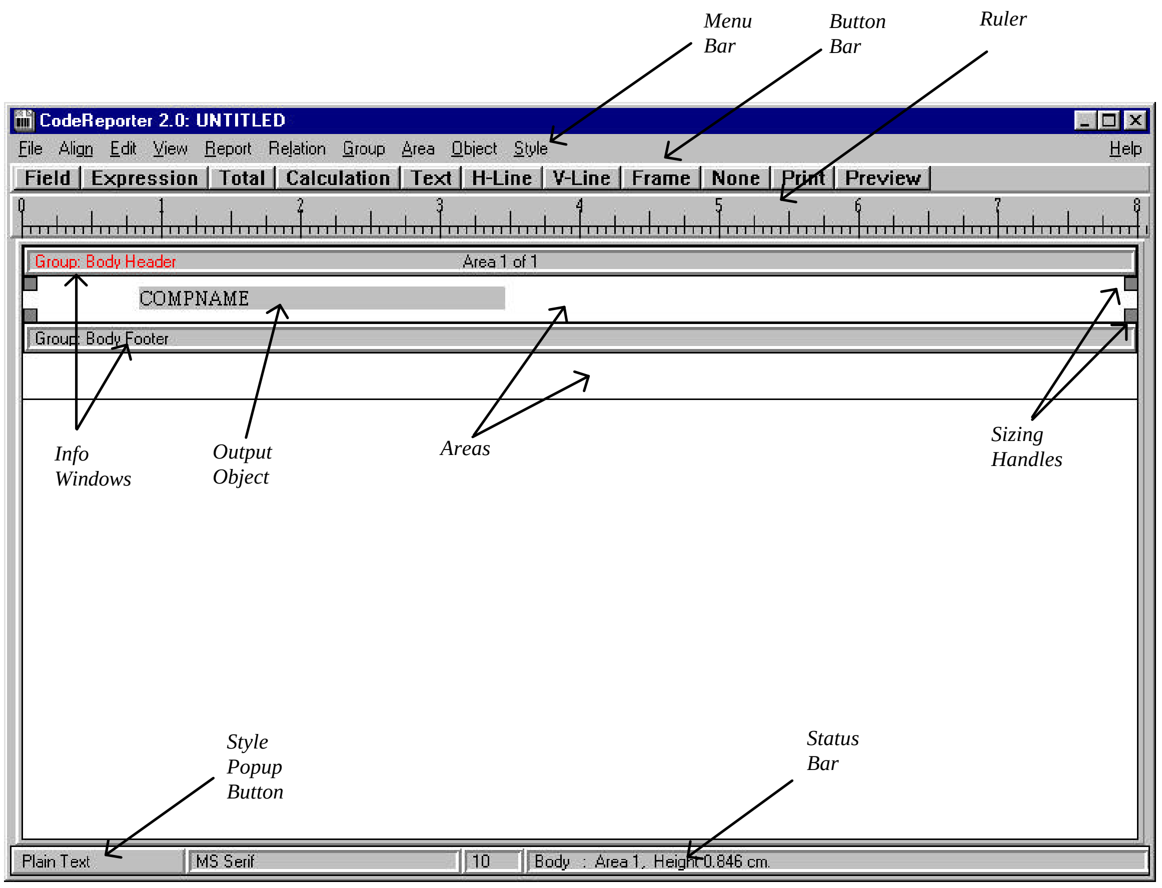

Menu Bar
Button Bar
Ruler
CodeReporter 2.0: UNTITLED
File Align Edit View Report Relation Group Area Object Style Help
Field Expression Total Calculation Text H-Line V-Line Frame None Print Preview
Group: Body Header
Area 1 of 1
COMPNAME
Group: Body Footer
Info Windows
Output Object
Areas
Sizing Handles
Style Popup Button
Status Bar
Plain Text
MS Serif
10
Body : Area 1, Height 0.846 cm.

Figure 1 Report Design Screen

Contacting Sequiter

If you have a comment or question about this product, or any one of Sequiter's products, please feel free to contact us by phone, e-mail, fax, or mail. Your comments and questions are very important to us.

Quality Control

If you are not completely satisfied with product quality or our service, please ask for or address your comments to "Quality Control".

Technical Support

In order to help us serve you better, you must be able to provide the following information to the Technical Support Representative:

1. Your name, phone number, fax number, and the name of your company.
2. The fact that you are using CodeReporter 2.0.
3. Your CodeReporter serial number. This is either on your CodeReporter diskettes or the inside front cover of the CodeBase Reference Guide.
4. Your operating system and its exact version.
5. The file compatibility of the CodeReporter version you are using (FoxPro, dBASE IV, Clipper).
6. The date of the files on your CodeReporter diskette. Periodically Sequiter may issue maintenance releases. In this case, the exact maintenance release you are using can be determined by performing a directory listing on the CodeReporter diskette files and checking the date stamp of the files.
7. The file size of the CodeReporter executable file being used.

   If you wish to submit a report which is not working properly in order that Sequiter can test it, you need to provide the above information and also do the following:

a. E-mail the report to us or send a copy on diskette by mail or courier. It is imperative that the data, index and memo files used in the report be included also.

b. Save the report file without path names (See the Path Names section of the Customizing Reports chapter for information on how to do this).

c. Provide in written or electronic form all the information listed above in 1-7.

d. Specify exactly what the problem is.

We are here to help and will do our best to provide high quality service.

Addresses

Listed below are the Sequiter Software Inc. contact information. Please visit our web site for additional ways of contacting Sequiter Software Inc.

Sequiter Software Inc.
P.O. Box 783
Greenland, NH, USA
03840

Voice: (403) 437-2410
Fax: (403) 436-2999
E-mail: info@sequiter.com
http://www.sequiter.com/

Sequiter Software Inc.
112 Powis Street
London, UK
SE18 6LU

Voice: (44) (181) 316 5001
Fax: (44) (181) 316 6001
Error! Bookmark not defined.

Newsgroups: comp.databases.xbase.codebase

Tutorials

This chapter of the manual is used to quickly familiarize CodeReporter 2.0 users with some of the basic procedures used in creating reports. After going through the tutorials listed in this chapter, a user of CodeReporter 2.0 will be able to:

- Open existing report files with and without paths
- Create new reports from scratch
- Place text, expression, and calculation output objects
- Create and place total output objects
- Create report areas and size them in two ways
- Create a relation manually and load one from disk
- Preview a report
- Use "Swap Header" and "Swap Footer" options

Additional examples of using some of the special features of CodeReporter 2.0 may be found near the end of the following chapters: Relational Reporting, Areas, Output Objects, Columnar Report Wizard, Styles, and Printing. Use the Index to locate the exact pages.

Loading a Report with Paths

This first tutorial describes the steps necessary to load two different reports locate their data files in different manners.

Opening in Current Directory

The first sample report, TUT1.REP simply lists out the contents of the COMPANY.DBF data file.

When this report file was saved, the drive and directory for the data file was not saved within it (See the CodeReporter Options chapter for information on how to do this). The data files for the report are then assumed to be in the same directory as the report file. The advantage in this is that the report and data files may be moved to any drive and any directory as long as they are moved together.

Invoke CodeReporter in one of the ways described in the Getting Started section of this manual. Once it is running, use the **FILE | OPEN** menu option to invoke the "Select Report File" dialog. Use the drives and directories list boxes to navigate to the .\EXAMPLES directory. Once there, select the TUT1.REP report file and select the "OK" button. The report is loaded from the current directory.

Opening With a Path

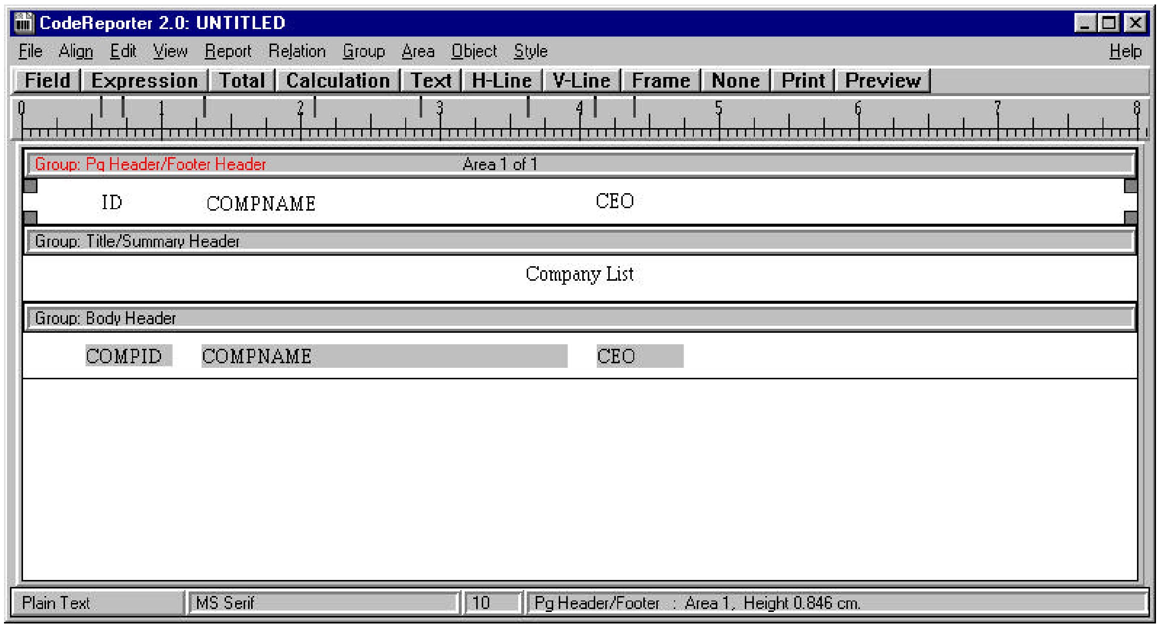
TUT2.REP is just the opposite of TUT1.REP. The different drives and paths for the COMPANY.DBF and STORES.DBF data files are saved within the report.

An attempt to open this report with **FILE | OPEN** will result in CodeReporter warning that the file could not be found and asking if an alternate file should be substituted for each file that could not be found. This is useful when the file names or directories of the data files within the report have changed -- but manually changing each and every file can be quite time consuming, especially if all files are located in the same directory.

The **FILE | OPEN WITH PATH** menu option overrides the drives and paths stored within the report file and replaces them all with the drive and path provided by the report's designer. Use **FILE | OPEN WITH PATH** to open the TUT2.REP file and specify the CodeReporter examples directory: .\EXAMPLES.

Creating a Database List

This next tutorial shows the steps necessary to create a simple report that lists the contents of the COMPANY.DBF data file. This data file, which contains the fields COMPID, COMPNAME, and CEO, is located in the CodeReporter examples directory: .\EXAMPLES.


CodeReporter 2.0: UNTITLED
File Align Edit View Report Relation Group Area Object Style Help
Field Expression Total Calculation Text H-Line V-Line Frame None Print Preview
0 1 2 3 4 5 6 7 8

Group: Pg Header/Footer Header   Area 1 of 1
ID   COMPNAME   CEO
Group: Title/Summary Header
Company List
Group: Body Header
COMPID   COMPNAME   CEO
Plain Text | MS Serif | 10 | Pg Header/Footer : Area 1, Height 0.846 cm.

Figure 2 Completed tutorial three report

Start a new Report

Select the **FILE | NEW** menu option once CodeReporter is running. This removes any currently loaded report and prompts for the first data file of the report (the top master data file). Since the report is to list the contents of the COMPANY.DBF data file, navigate to the CodeReporter examples directory, select COMPANY.DBF and then the "OK" button.

Once this initial setup is completed, CodeReporter displays a blank report design screen. A default group, "Body" is created with one header area in which output objects may be placed.

A sample sketch report would demonstrate that the report only requires one area of the report which repeats for each new composite record. CodeReporter automatically creates this area for each new report, so no new areas need be created. (For information on "areas" and "groups" see the Areas and Groups chapters).

Place repeating elements

The default area should contain fields from the top master data file whose values change with each new composite record. To put CodeReporter into insertion mode so that these fields may be placed within the "Body" area, click on the "Field" button on the button bar. Doing so invokes the "Field Objects" floating list box (Figure 3, below) which contains all of the fields in the composite data file.

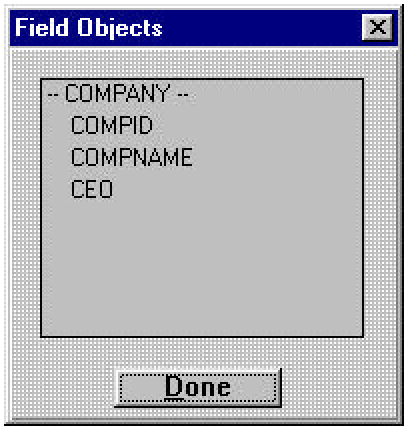


Figure 3 Field objects list box

Use the mouse or the keyboard to select all the fields in the list box, just like normal Windows list boxes (press and hold the left mouse button and drag it over all three fields).

Once the desired fields are selected, position the mouse cursor over the "Body" group's header area. Do not select the "Done" button at this point. Notice that the cursor changes from an arrow to a field insertion cursor (See

Appendix C for all cursors). This indicates that the next click of the left mouse button establishes the point where the upper left corner of the fields are placed.

Position the cross hairs of the field insertion cursor at the left side of the "Body" area and press the left mouse button. This invokes the "Field Layout" dialog. Since the default settings are fine, select the "OK" button to complete the field placement. The report design screen now looks like Figure 4.

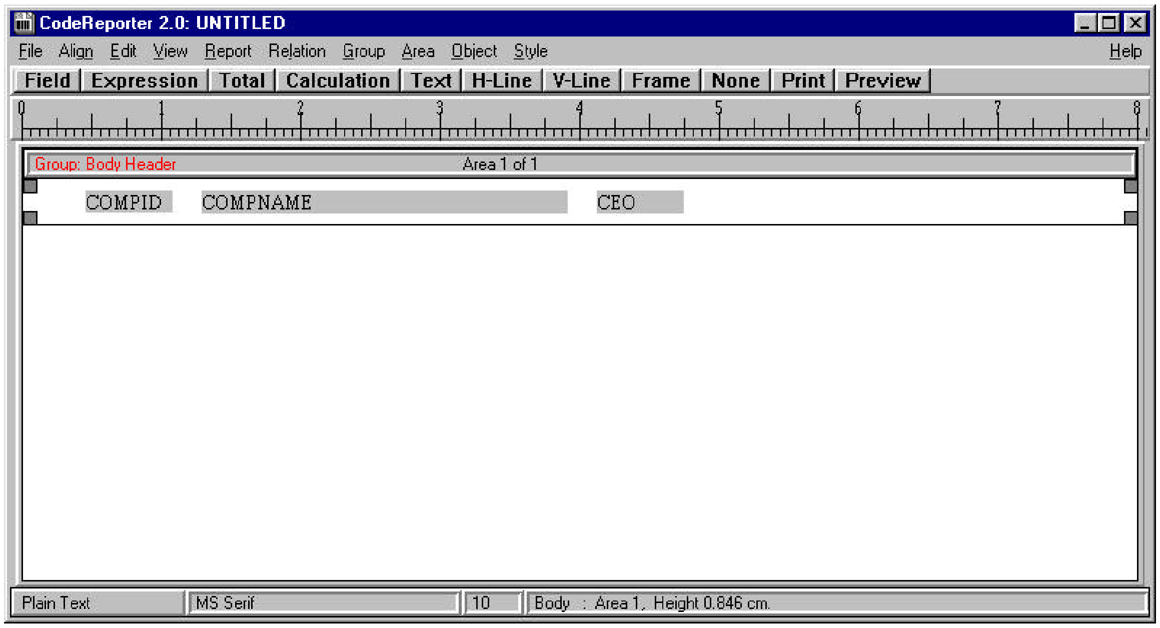
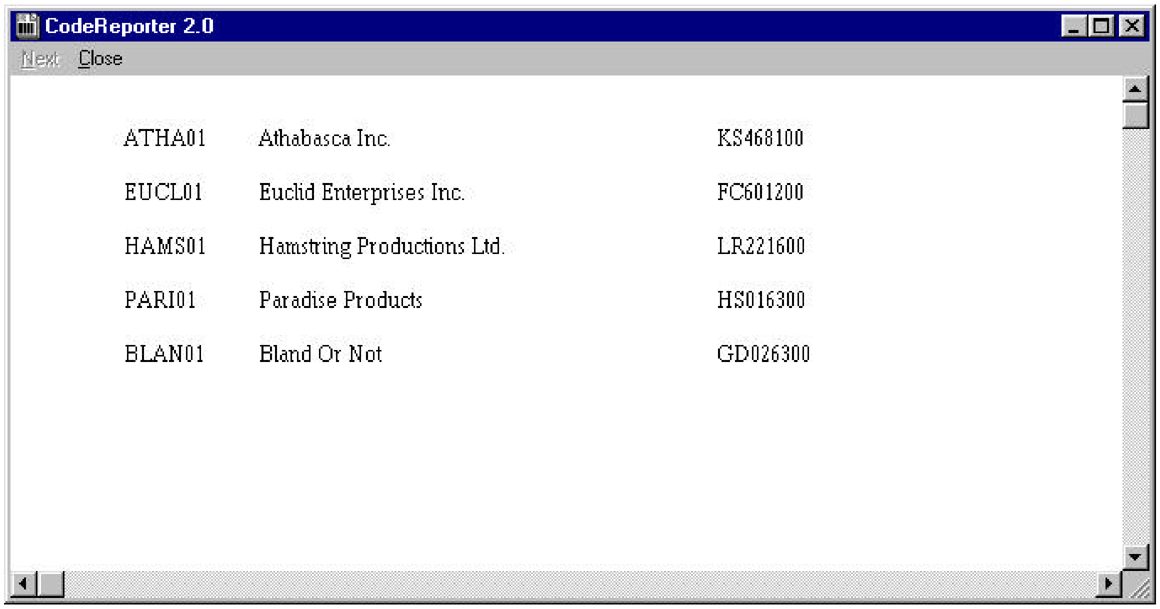

CodeReporter 2.0: UNTITLED
File Align Edit View Report Relation Group Area Object Style Help
Field Expression Total Calculation Text H-Line V-Line Frame None Print Preview
Group: Body Header
Area 1 of 1
COMPID
COMPNAME
CEO
Plain Text
MS Serif
10
Body : Area 1, Height 0.846 cm.

Figure 4 Partially designed tutorial three report

Preview Report

At this point, the report design may be considered finished. It has successfully fulfilled its requirements, namely, to output the contents of the COMPANY.DBF data file. Use the **FILE | PRINT PREVIEW** menu option to view the report. As seen in Figure 5, this report is remarkably bland and relatively obscure. Unless the reader of the report is very familiar with the contents and layout of all the data files accessible, it is impossible to know exactly what this report is displaying.



```
ATHA01   Athabasca Inc.                KS468100
EUCL01   Euclid Enterprises Inc.       FC601200
HAMS01   Hamstring Productions Ltd.    LR221600
PARI01   Paradise Products             HS016300
BLAN01   Bland Or Not                  GD026300
```

Figure 5 Partial tutorial three output

Adding Explanatory Areas

The most obvious addition to help explain the report would be a title like "Contents of COMPANY.DBF". However, placing this title in the "Body" group's header area is not appropriate, since it would be repeated for every record in the composite data file.

Title Area

A title for the report is properly placed within the title report area. Report elements in the title area are outputted at the top of the first page before any other output area's objects, including the page header area. As such, this area may be used as a "cover page" for the report. To create a title area, choose the **AREA | NEW TITLE AREA** from the menu. The new report area is created.

The title "Contents of COMPANY.DBF" or any other appropriate title is considered static text. Once put in the report, its value is set. CodeReporter is put into insertion mode for static text by pressing the "Text" button on the button bar, or by choosing the **OBJECT | TEXT** menu option. Notice the mouse cursor changes to the Text Insertion cursor to indicate the position of the text output object.

Place the mouse cursor within the newly created title area and click the left mouse button. This specifies the upper left corner of the output object's text. When the new text object is placed, the "Enter Text for Text Object" dialog is invoked so that the text for the text object may be entered.

Enter "Company List" or some other descriptive string and choose the "OK" button to complete the object creation. The newly created output object, to be visually appealing, should be horizontally centered within the report design screen. To do this, use the mouse to manually drag the text output object to the approximate center of the area, or choose the **ALIGN | CENTER** menu option.

Page Header

These new additions to the report describe what the report is and when it is completed, however it doesn't identify what each of the report's columns are. Following the steps above, static text objects could be placed beside each of the field output objects to identify the field, but this would also cause needless duplication within the final output.

What this report needs are column titles -- static text objects placed at the top of each column that contain the name of the field. If the report crosses a new page, these column titles should also be reprinted at the top of the new one.

A report area that is outputted at the top of each page is the Page Header area. Each page, with the exception of the first page (if there is a title area), begins with the page header area, no matter what the contents of the composite data file.

A Page Header report area may be created by using the **AREA | NEW PAGE HEADER AREA** menu option. A report area with a default size of .33 inches is created. Within this area, place the following text output objects:

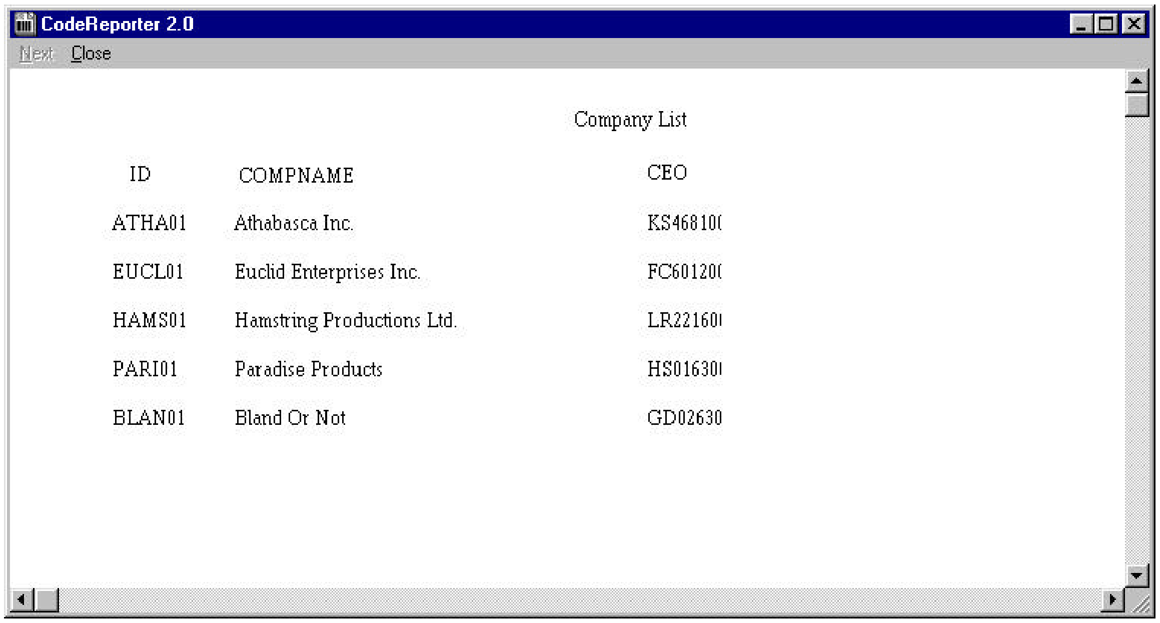
- ID
- COMPNAME
- CEO

and manually move them into position above their respective field objects. When finished, choose the "None" button on the button bar (or the ESC key) to move CodeReporter out of insertion mode.

The report design screen should now look very similar to the one shown in Figure 2 and the previewed output should look like Figure 6, below.

There are two items worth mentioning about this output—the first, is that the Title Area string ‘Company List’ appears on the same page as the records in the report. This is achieved by setting the **REPORT | PREFERENCES - PAGE BREAK AFTER TITLE** checkbox off. Be default the Title Area is displayed on its own separate page.

The second item worth noting, is that the last character of the CEO field output appears to be only partially outputted for some of the records. This is a result of CodeReporter's estimation of the size of the output object based on the font's average character width. The estimation using average character width may be off a little bit when capital letters -- which may be wider than the average width -- are found in the field.



Company List

ID	COMPNAME	CEO
ATHA01	Athabasca Inc.	KS468100
EUCL01	Euclid Enterprises Inc.	FC601200
HAMS01	Hamstring Productions Ltd.	LR221600
PARI01	Paradise Products	HS016300
BLAN01	Bland Or Not	GD02630

Figure 6 Completed tutorial two report

To fully display all of the field, select the CEO field by clicking the object with the left mouse button. Notice that when selected, eight little black squares appear on the object. These are sizing handles, which may be used to change the space used to output the object. Click and hold the left mouse button on one of the right sizing handles and drag it to the right about an 1/8th of an inch. When the sizing handle is released, the CEO output object is resized to the new width and when the object is outputted, the entire contents of the CEO field is displayed.

Statement of Accounts Report

The following tutorial brings together many of the complex features of CodeReporter 2.0 into a single invoice report. The completed report design screen is shown in Figure 7.

Load the Relation

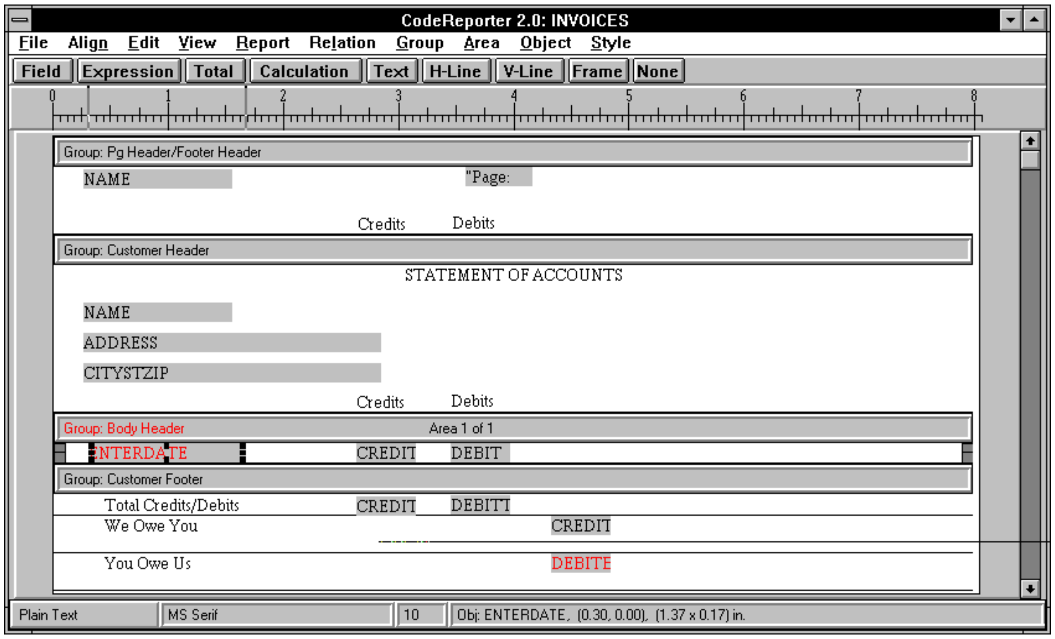
This tutorial uses a relation saved in the TUT3*xxx*.REL relation file. This relation file contains the relation for the two data files used in this report: INVOICES.DBF and CUST.DBF. When a relation file is loaded, all previously used data files are closed, and the new files in the relation are opened and used.

Loading a relation saves time by quickly retrieving an often used relation from disk instead of manually building the same relation for several reports.

While CodeReporter is running, use the **File | Load Relation** menu option to load the relation file. Since relation files are index file specific, load the appropriate version of the relation file as listed in Table 1, below. If the index file version of CodeReporter is unknown, use the **HELP | ABOUT** menu option.

Relation File Name	Index Compatibility
TUT3FOX.REL	FoxPro
TUT3MDX.REL	dBASE IV
TUT3CLI.REL	Clipper

Table 1 Relation file compatibility


CodeReporter 2.0: INVOICES
File Align Edit View Report Relation Group Area Object Style
Field Expression Total Calculation Text H-Line V-Line Frame None

Group: Pg Header/Footer Header
NAME "Page:
Credits Debits
Group: Customer Header
STATEMENT OF ACCOUNTS
NAME
ADDRESS
CITYSTZIP
Credits Debits
Group: Body Header Area 1 of 1
ENTERDATE CREDIT DEBIT
Group: Customer Footer
Total Credits/Debits CREDIT DEBITT
We Owe You CREDIT
You Owe Us DEBITE

Plain Text | MS Serif | 10 | Obj: ENTERDATE, (0.30, 0.00), (1.37 x 0.17) in.

Figure 7 Tutorial 4 completed report screen

Add Body Group Fields

After the relation is successfully loaded, CodeReporter initiates a new report with one group, "Body", containing a single header area. It is within this area that the repeating fields for the composite data file are outputted.

Place CodeReporter into insertion mode for field output objects by selecting the "Field" button on the button bar or by selecting the **OBJECT | FIELD** menu option. Either method will invoke the "Field Objects" floating list box which contains the fields of all the data files in the loaded relation.

Select the following fields from the INVOICE.DBF data file within the list box:

- CREDIT
- DEBIT
- ENTERDATE

Move the mouse cursor over the "Body" group's report area and click the left mouse to place the field objects. The objects are placed in the order in which they were found within the data file. Select the "Field Objects" dialog box's "Done" button to remove it from view.

The position of the field output objects must be changed to suit the report. Use the mouse to drag the ENTERDATE field object to the upper left corner of the report area. The CREDIT and DEBIT fields may also be moved to the top of the report area and spaced as seen in Figure 7.

Modify Object Settings

Output objects have many default settings, some of which are not always applicable. In the case of the ENTERDATE, CREDIT, and DEBIT fields, this is true. ENTERDATE is displayed in MM/DD/YY format, when it should be in MMM DD, CCYY format, and both CREDIT and DEBIT should not be outputted when their values are zero.

Select the ENTERDATE output object and press the Enter key to invoke its Object Menu. From this menu, select the **OBJECT** Settings menu option to invoke the "Object Settings" dialog in which the date format and size of the object may be modified.

Use the "Date Format" drop down combo box to select the MMM DD, CCYY format (or manually type it in), and set the "Width" edit control (in the Size area) to be .75 inches. Selecting the "OK" button makes these two changes and modifies the object appropriately.

Invoke the "Object Settings" dialog for CREDIT output object using the above procedures. The "Display Zero" radio button in the lower right portion of the dialog is enabled. Click on the radio button to disable the display zero option, and select the "OK" button to close the dialog. Repeat these steps for the DEBIT output object.

Change the Size of "Body"

Use the **FILE | PRINT PREVIEW** menu option to view the report. Notice that the lines of data in the report appear to be double spaced. This is a result of the "Body" group being its default size of .33 inches, but the font for the output objects are only approximately .17 inches tall. To eliminate this double spacing, the height of the report area may be changed to that of the output objects.

This may be accomplished in two ways, using the report area's sizing handles, or by directly setting the size using the "Modify Area" dialog. The former method is discussed here, while the later method is used below for sizing the "Customer" group header area.

Use the mouse to drag the lower left or right sizing handle for the "Body" area so that the area is smaller than the three output objects. When the mouse is released, CodeReporter attempts to make the area smaller than the report objects, but displays the warning shown in Figure 8.

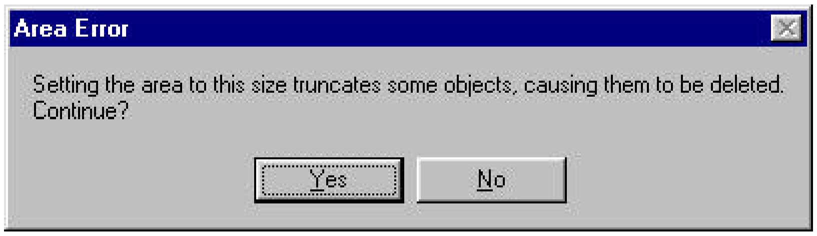

Area Error
Setting the area to this size truncates some objects, causing them to be deleted. Continue?
Yes
No

Figure 8 Area error message

Select the "No" button. This causes CodeReporter to size the area to the smallest size possible without truncating (and deleting) any output objects. (See the Areas chapter for more information on sizing the area)Preview the report again (**FILE | PRINT PREVIEW**), and notice the lines of the report are now single spaced.

Add the Customer Group

The report as it stands is not of much use to the casual reader. All that is listed are some dates and amounts. Without knowing to whom these figures belong and what they represent, the report does not impart any real knowledge.

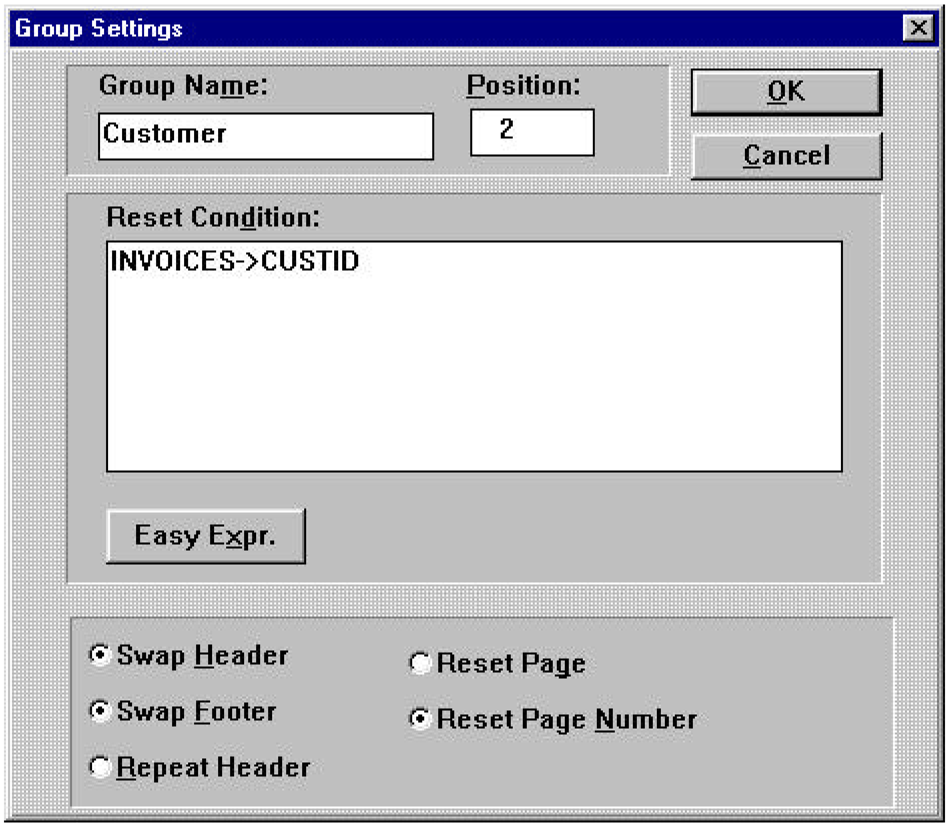
Perhaps the most important information that is needed for this report is an identification of to whom these credits and debits belong. This is done by creating a second group for the report using the **GROUP | NEW** menu option and placing within this group's areas the information identifying the customer.

When the new group is created, the "Group Settings" dialog box (Figure 9) is invoked to allow the report designer the option of changing some of the default actions of the group.

CodeReporter automatically gives the groups within a report a unique name. This unique name, however, is not very descriptive of the purpose of the group. The name of the group may be changed in the "Group Settings" dialog box by using its "Group Name" edit control. Change the default group name, *Group 2,* to be *Customer.*

Since it is unnecessary for the personal information to be displayed for each and every credit and debit, a group reset expression is created to output the areas of the Customer group only when the customer changes. A group reset expression that uniquely identifies the subset to which each customer belongs is:

```
INVOICES->CUSTID
```



Group Name: Customer
Position: 2
Reset Condition: INVOICES->CUSTID
[✓ Swap Header]
[✓ Swap Footer]
Repeat Header
Reset Page
[✓ Reset Page Number]

Figure 9 Group settings dialog for customer group

Enter this expression within the "Reset Condition" entry window.

Whenever the customer identification number changes, the new credits/debits should be associated with a different customer. This condition, known as a group reset condition, causes the Customer group to be outputted.

Notice that no group reset condition was created for the default "Body" group. This is because without a group reset expression, the "Body" group resets for each composite record in the data file -- causing the header area for the group to be outputted for every record.

Also set the "Swap Header", "Swap Footer", and "Reset Page Number" radio buttons within this dialog. These provided some special handling for the invoice report. For more information on these settings, see the Groups chapter.

Select the "OK" button to close the "Group Settings" dialog.

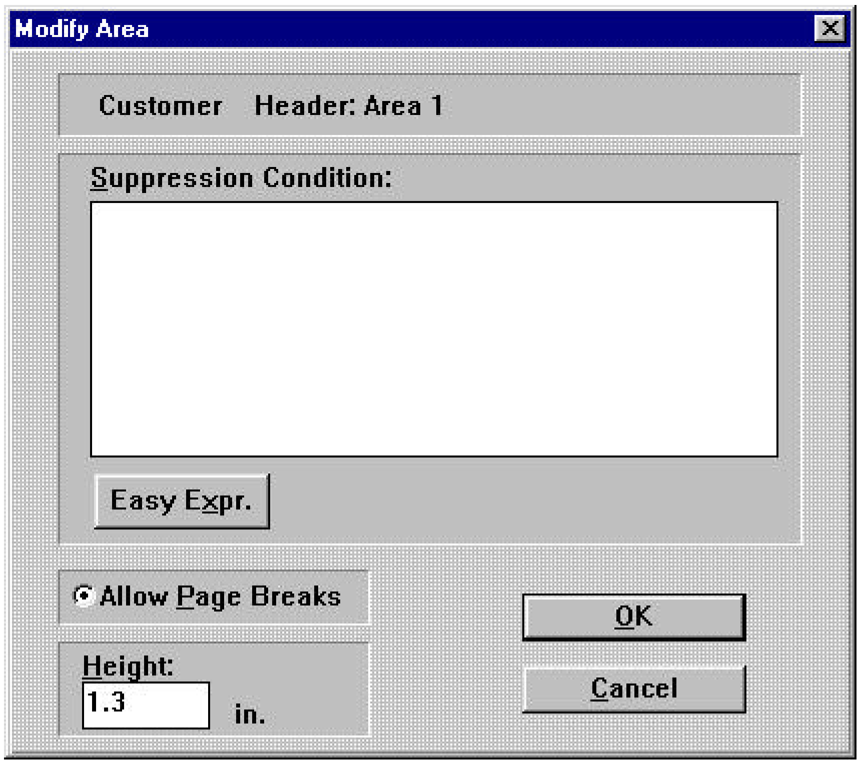
Populate the Customer Header

The customer header area, by virtue of its swap header setting, is outputted at the top of the page for each new customer. It is therefore necessary to add the descriptive information for the report as well as for the customer to the Customer header area.

Create a text output object entitled "STATEMENT OF ACCOUNTS", place it within the Customer header area and use the **ALIGN | CENTER** menu option to center it horizontally.

Before the rest of the descriptive fields are added to the Customer header area, the size of the area must be modified. This may be done with the mouse and the area's sizing handles as described above, or by using the "Modify Area" dialog box.

This dialog is invoked by clicking the right mouse button within an empty portion of the Customer area (not the group's information window).



Height: 1.3 in.

Figure 10 Modify area dialog

Use the "Height" edit control (shown in Figure 10), to set the height of the Customer group to 1.3 inches and select the "OK" button. The header area is automatically resized.

The fields describing the customer are added to this new area through the "Field Objects" floating list box. This is invoked using the "Field" button on the button bar. Use the scroll bar to move the following fields into view:

- NAME - The customer's name
- ADDRESS - The customer's address

- CITYSTZIP - The customer's city, state, and zip code.

These fields may be individually selected and dropped into their positions in the report as shown in Figure 7, or they may be multiply selected, and dropped at the same time.

Multiply select the above fields, position the mouse cursor in the Customer header area and press the left mouse button. This invokes the "Field Layout" dialog. Instead of automatically selecting the "OK" button as done for the "Body" group, select the "Vertical" check box. Selecting the "OK" button this time causes the automatic layout of the three selected fields to be placed vertically, aligned on the left.

Also place two text objects containing "Credits" and "Debits" on the bottom of the Customer header area above the CREDIT and DEBIT field objects.

Page Header Area

Use the **AREA | NEW PAGE HEADER AREA** menu option to create a page header which is outputted at the top of every page within the report (except where swapped with the Customer group header). As such this area should be used to briefly describe the report and tie all the pages together.

Size the area to .5 inches. This area contains four output objects, one field, two text, and one expression. Use the "Field Objects" floating list box (it should still be visible) and select and drop the NAME field within the page header area. Close the "Field Objects" floating list box by selecting its "Done" button.

Copy and Paste

The two text objects created for the Customer header area ("Credit" and "Debit") may be created manually as done in the customer header and placed within the page header area.

As an alternate, multiply select the objects in the Customer header and then use the **EDIT | COPY** menu option to place a duplicate copy of the objects within the Windows clipboard. Select **EDIT | PASTE**, position the mouse cursor within the page header area and click the left mouse button to place the copies of the text objects.

The fourth output object, the expression, is used to output the page number of the report. Select the "Expression" button on the button bar (CodeReporter is put into insertion mode for expression output objects), position the mouse cursor over the upper right portion of the page header area and press the left mouse button. This invokes the "Easy Expression" dialog box in which the expression for the expression output object may be entered.

Type in the following expression (including the quotation marks):

```
"Page: "+STR(PAGENO(), 3,0)
```

This expression, which combines both a string ("Page:") and the numeric page number ("PAGENO()"), may actually have been represented as two objects; a text object for "Page:" and a separate expression output object for "PAGENO()". However, since they are logically related and always are moved and positioned together, it is convenient to combine them into one expression. Select the "OK" button to complete the placement of the output object.

End of Invoice Summary

This statement of account lists a total of the credits and debits for each customer's account. To make it easier for the customers, this statement also includes a line that indicates whether they owe the company money, or if they have over paid.

Since a customer can't at the same time owe and have overpaid, these statements must be placed in mutually exclusive report areas in the customer summary.

When the Customer group was created, CodeReporter automatically created a single group footer area. This area is going to be used to output the customer's total credits/debits. Two additional footer areas (for the "You Owe Us" and "We Owe You" lines) may be created by using the **AREA | NEW FOOTER AREA** menu option twice while the Customer group is the selected group.

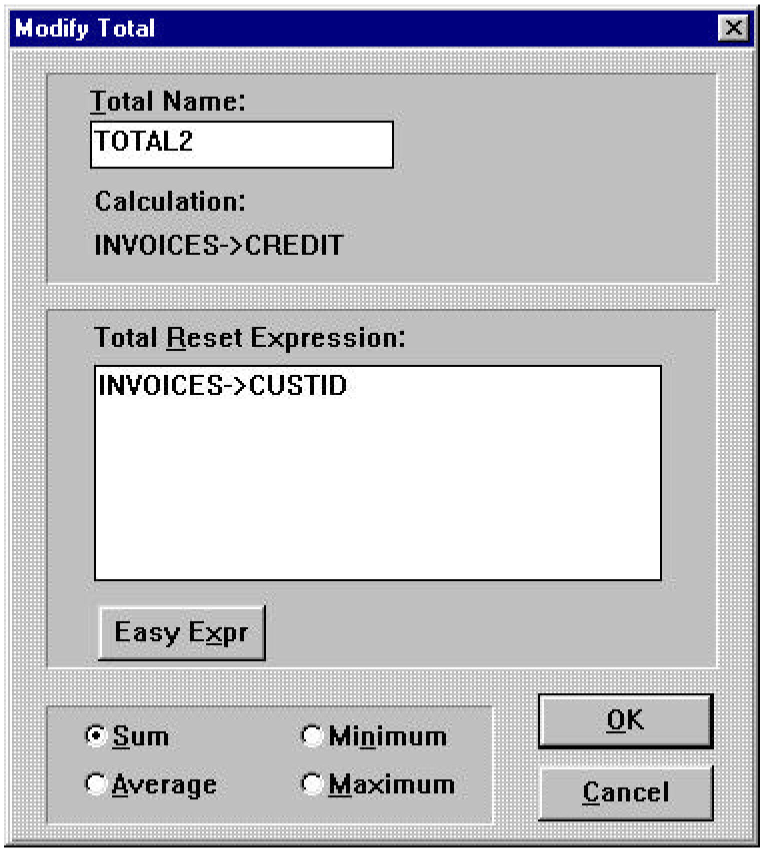
Add the Text Objects

Add a text output object to each of the Customer group's footer areas (See Figure 7)

- Total Credits / Debits
- We Owe You
- You Owe Us

Add the Totals

In order to determine who owes money to whom, the total amount credited and debited to the customer's account must be totaled and those totals compared. Totals are created through the "Total Calculations" floating list box which is invoked by selecting the "Total" button on the button bar. This list box contains the names of all numeric fields within the composite data file as well as all numeric calculations. A total output object for the CREDIT field may be placed by selecting the CREDIT entry in the list box and placing it within the first footer area of the Customer group. Doing so invokes the "Modify Total" dialog box (Figure 11) which contains the total's default values.



Total Name: TOTAL2

Calculation: INVOICES->CREDIT

Total Reset Expression: INVOICES->CUSTID

[✓ Sum] Minimum Average Maximum

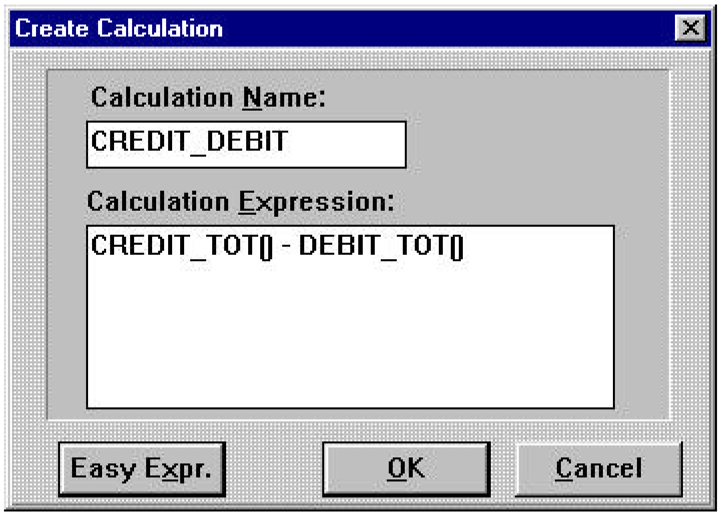
Figure 11 Modify total dialog

The "Modify Total" has all the appropriate settings (reset expression, total type) necessary to sum the CREDIT field. The TOTAL0 name, however, does not truly describe what is being totaled. Change the name TOTAL0 within the "Total Name" edit control to be CREDIT_TOT and select the "OK" button.

Repeat the above steps to create a total output object in Customer's first group footer for the DEBIT field, and use DEBIT_TOT as the name for the total.

Create the Calculations

The amount owed can be determined (and outputted) using a calculation output object that calculates the difference between the CREDIT_TOT total and the DEBIT_TOT total. This calculation which can (and will) be used in other dBASE expressions. A calculation output object is created through the "Calculation Object" dialog box which is invoked using the "Calculation" button on the button bar. Use the "New Calc" button to invoke the "Create Calculation" dialog (Figure 12).



CREDIT_DEBIT

CREDIT_TOT() - DEBIT_TOT()

Figure 12 Calculation object dialog

The calculation name is used to identify the calculation in other dBASE expressions. Enter CREDIT_DEBIT in the "Calculation Name" edit control and the following expression into the "Calculation Expression" edit control:

CREDIT_TOT()-DEBIT_TOT()

Select "OK" to close the "Create Calculation" edit control and return to the "Calculation Object" dialog. Select the new CREDIT_DEBIT() calculation in the list box and place it in the second ("We Owe You") Customer group footer.

When the customer owes the company money, the CREDIT_DEBIT() calculation contains a negative number. To properly output the third report area ("You Owe Us") however, this value should be positive. Create an expression output object in the third ("You Owe Us") Customer footer area (like the page number above) and enter the following expression:

CREDIT_DEBIT() * -1

This expression evaluates to a positive number in the case where the customer owes the company money.

Suppressing the footer areas

As it stands now, once the report has completed outputting all the credits and debits for a customer, the totals for the debits and credits are outputted as well as a line saying that the company owes them money and that the customer owes the company money.

Only one of these situations can be correct. The last two areas must be conditionally suppressed -- a condition based on the value of the CREDIT_DEBIT()calculation.

A click of the right mouse button on an empty spot in the second footer area of the Customer group ("We Owe You") invokes the "Modify Area" dialog box. In the "Suppression Condition" edit control, enter the following expression and select the "OK" button:

```
CREDIT_DEBIT() <= 0
```

This indicates that if the credits minus the debits is less than or equal to zero (the customer owes the company money) the second area ("We Owe You") should not be outputted. When the company owes the customer money the CREDIT_DEBIT() calculation returns a positive number and so the report area is outputted (i.e. not suppressed).

The third area should use the following suppression condition:

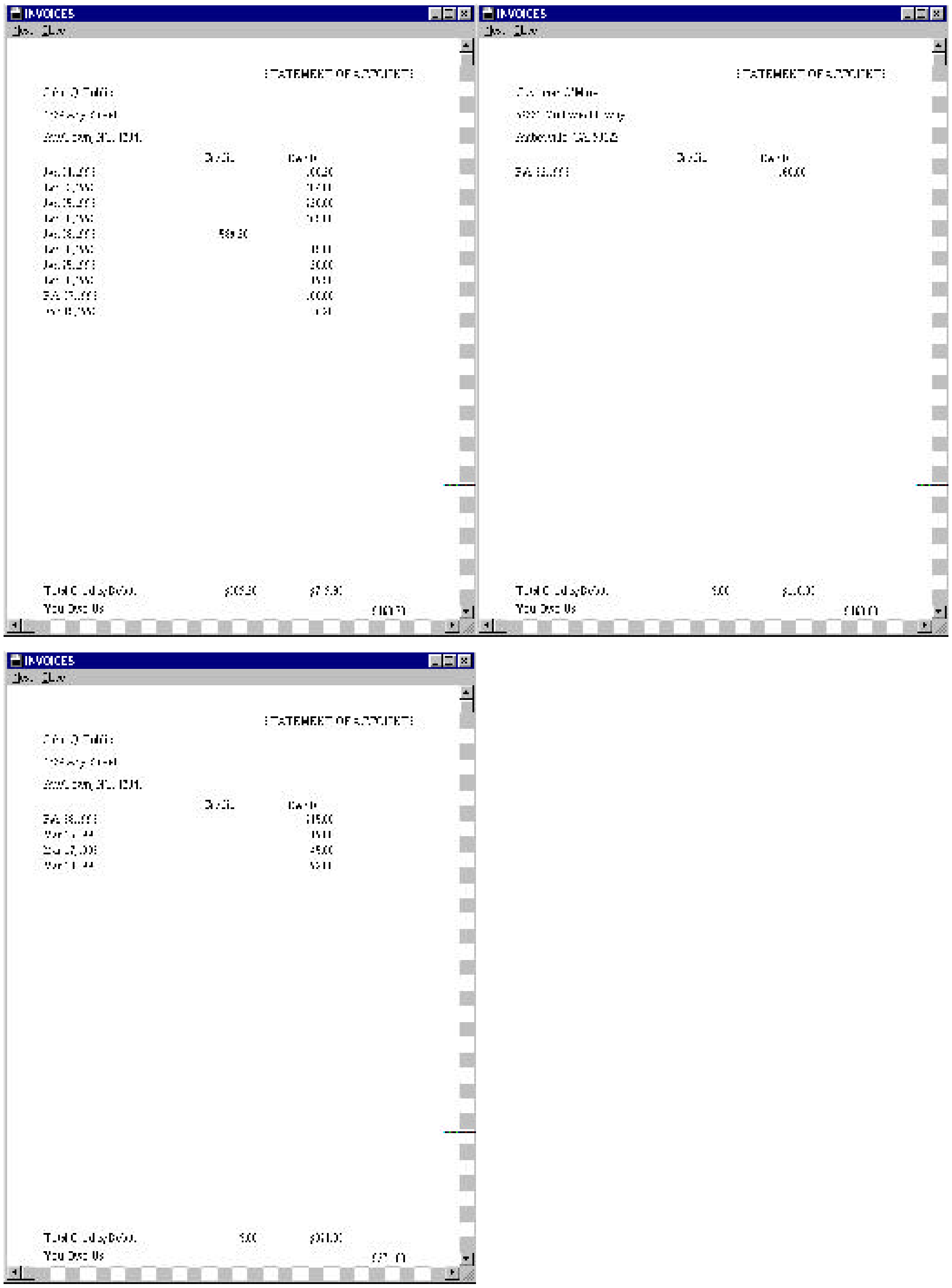
```
CREDIT_DEBIT() > 0
```

The final step in this tutorial report is to modify the numeric format of the numeric output objects in the Customer group's footer areas (two totals, a calculation, and an expression). For each output object, invoke its Object Menu (right mouse click on the object or pressing the Enter key while selected) and choose the **OBJECT SETTINGS** menu option. Change the following settings:

1. "Number of Decimals" edit control. Change this from the default setting of zero (0), to two (2).
2. "Numeric Type" radio buttons. Set the output object to be displayed as a currency.

View the Report

Use the **FILE | PRINT PREVIEW** menu option to view the pages of the report. The first three should appear like those in Figure 13.



```
INVOICES
                                   STATEMENT OF ACCOUNTS
  [illegible]
  [illegible]
  [illegible]
                          [illegible]      [illegible]
  [illegible]                                    [illegible]
  ...
  Total [illegible]        [illegible]      [illegible]
  You Owe Us                                          [illegible]
```

Figure 13 Preliminary output

Specifying a Sort Expression

As it can be seen in the first three pages of the report (Figure 13), the debits and credits for the first customer, John Q. Public, are broken up by those of Customer O'Mine. This is because CodeReporter is retrieving the records from the composite data file in natural order (i.e. the physical order of the composite data file).

The final step in creating this report is to order the records in the composite data file according to the customer's id number which is stored in the CUSTID field.

Use the **REPORT | SORT EXPRESSION** menu option to invoke the "Easy Expression" dialog for the sort expression. Enter the following expression and select the "OK" button:

```
INVOICES->CUSTID
```

When the report is displayed a second time, this new sort expression is taken into account and all of the credits and debits for John Q. Public are retrieved together before those of Customer O'Mine.

1. Designing a Report

Designing a report is a three phase cycle. The first phase is figuring out what the report should contain and look like. The second and third are laying out the report and obtaining a sample report. If the end product after the third cycle is the same as what was designed in the first phase, the report cycle is done. It is more often the case that the implementation cycles of the report need to be repeated many times until the report is "perfected".

It is even more often the case that the initial concept of what should be in the report or how it should be organized radically changes once the report is implemented. This usually occurs as a result of insufficient consultation and planning prior to the attempt to implement the report.

This chapter discusses an organized approach to the design phase of a report.

Report Purpose

It is important in any report to decide exactly why the report is being designed. Without a clear idea of why the report is necessary in the first place, it is quite easy to come up with a nice, new, crisp report that is totally useless.

If, for example, the Inventory Control department wants to know how many widgets they need to reorder, and they get a report that tells them how many widgets were used last year, they'll be quite upset, and the report will need to be redone.

A Statement of Purpose is a brief description of the requirements of a report. This is generally a line or two that specifies:

- Who is requesting the report,
- What information is required,
- What are the limitations of the report (eg. between which dates, for which products, for which location, etc.)
- Any special formatting

It can be as simple as "*Personnel needs a list of all people in the EMPLOYEE data file*", or as complex as "*The Board of Directors needs a two year summary of the revenue and expenses for each store of all the companies in the system -- sorting alphabetically by company and store name.*"

Creating this statement of purpose helps clarify the requirements of the report, and can be used later to verify that the report shows what it was required to show.

Prototype on paper

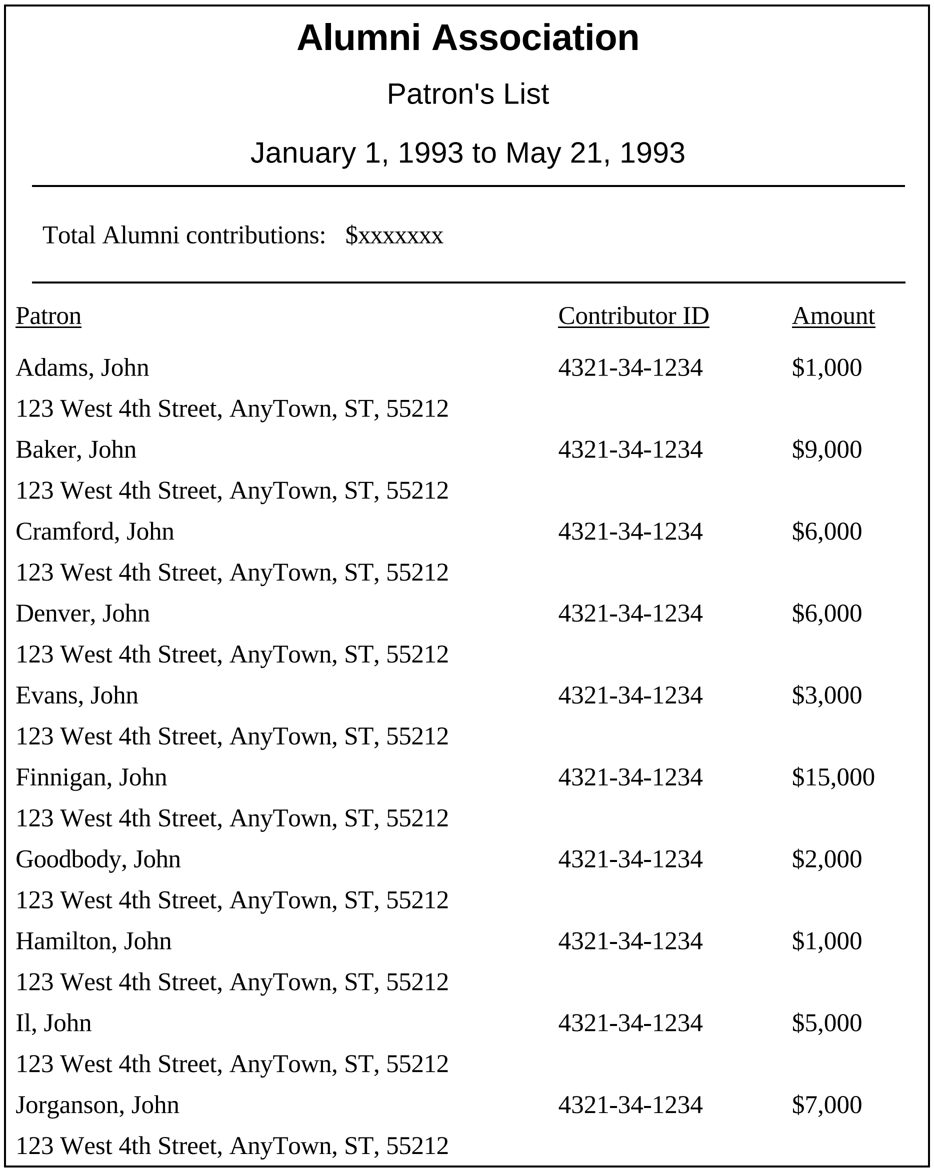
Determining how the report should physically look is the second most often modified aspect of a report after its content. Deciding early how the report should look will save countless comments like, "That's *ok*, but I really *think* this should be *over here*" and will greatly accelerate the actual implementation of the report.

The simplest way to obtain the format of the report is to prototype it quickly on a piece of paper, filling in sample information. In the prototype, the placement of information is more important than the accuracy.

Exactly where any piece of information is placed is often a combination of company policy and personal taste -- so there are no rules. Some companies like page numbers at the top, while others like them at the bottom. Try out a couple different types of designs until a good one is found.

The most important aspect of a prototype is that it fully satisfies the requirements of the Statement of Purpose. If it does not, the design needs to be revised.

Figure 1.1 shows a prototype report for the following Statement of Purpose: *The Alumni Association needs a report that shows the total amount of money donated this year and an alphabetical list of the names, addresses, contributor identity number and monetary amounts of all people who have contributed $1,000 or more so far this year.*



Alumni Association

Patron's List

January 1, 1993 to May 21, 1993

Total Alumni contributions: $xxxxxxx

Patron	Contributor ID	Amount
Adams, John	4321-34-1234	$1,000
123 West 4th Street, AnyTown, ST, 55212		
Baker, John	4321-34-1234	$9,000
123 West 4th Street, AnyTown, ST, 55212		
Cramford, John	4321-34-1234	$6,000
123 West 4th Street, AnyTown, ST, 55212		
Denver, John	4321-34-1234	$6,000
123 West 4th Street, AnyTown, ST, 55212		
Evans, John	4321-34-1234	$3,000
123 West 4th Street, AnyTown, ST, 55212		
Finnigan, John	4321-34-1234	$15,000
123 West 4th Street, AnyTown, ST, 55212		
Goodbody, John	4321-34-1234	$2,000
123 West 4th Street, AnyTown, ST, 55212		
Hamilton, John	4321-34-1234	$1,000
123 West 4th Street, AnyTown, ST, 55212		
Il, John	4321-34-1234	$5,000
123 West 4th Street, AnyTown, ST, 55212		
Jorganson, John	4321-34-1234	$7,000
123 West 4th Street, AnyTown, ST, 55212		

Figure 1.1 Prototype (Sketch) Report

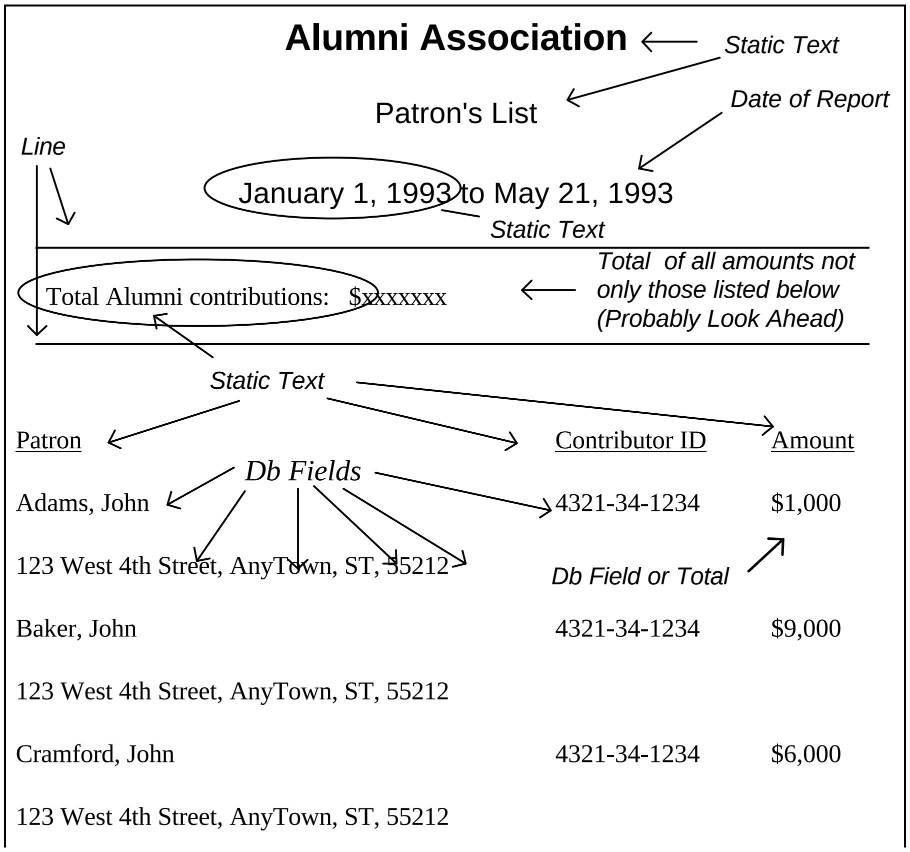
The prototype report in Figure 1.1 completely satisfies the statement of purpose. It lists the originator of the report at the top of the report, the total amount contributed by the alumni in the "Total contributions" line, and the names, addresses, and contributor identity numbers in alphabetical order. Once this prototype report has been approved by the originator of the report, in this case the Alumni Association, the building of the report may begin.

Analyze the prototype

The completed prototype lists all the essential pieces of information that are required in the report. It would be tempting at this point to move directly into CodeReporter to implement the report. It is important, however, that additional planning take place in order to be the most productive when using CodeReporter.

The first step is to analyze the prototype and determine the type of each element of the report. Does this piece of the report change during the report? Can that piece change each time the report is run? Is this amount calculated? On the prototype report, figure out what the pieces of the report are, and label them (as in Figure 1.2) with terms similar to:

- Db field. This information comes from a database
- Graphic. This is a picture stored on file. Don't include letter head, etc. that are not actually outputted by your printer.
- Static text. This is text that does not change within the report.
- Calculated Text/amount. This information combines two or more report elements total. This information summaries numeric data. lines.
- Special. Elements like the page number, or the current date/time which don't fit in any other category.



Alumni Association

Patron's List

January 1, 1993 to May 21, 1993

Total Alumni contributions: $xxxxxxx

Patron	Contributor ID	Amount
Adams, John	4321-34-1234	$1,000
123 West 4th Street, AnyTown, ST, 55212		
Baker, John	4321-34-1234	$9,000
123 West 4th Street, AnyTown, ST, 55212		
Cramford, John	4321-34-1234	$6,000
123 West 4th Street, AnyTown, ST, 55212		

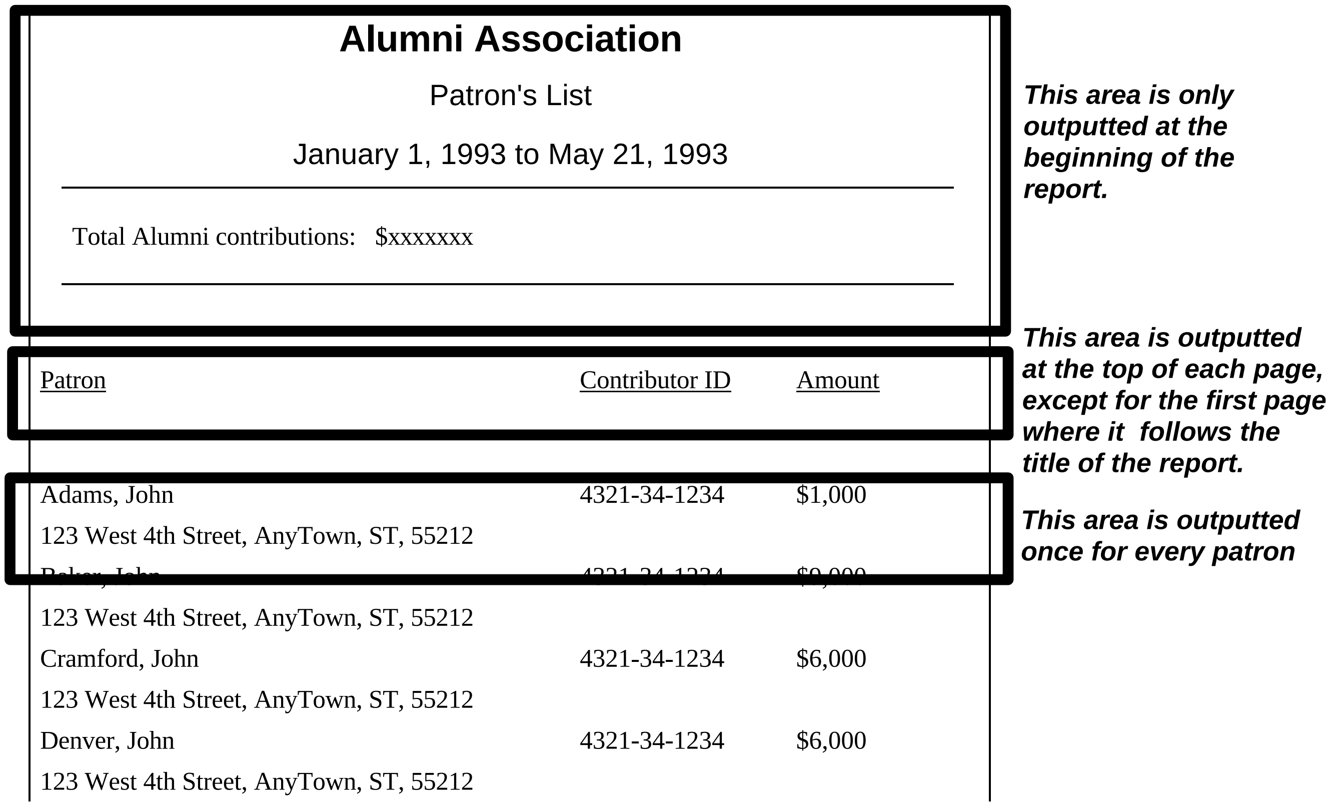
When the report is actually created with CodeReporter, these labels will accelerate the placement of the report elements.

Finding common report areas

On a separate copy of the report prototype, draw rectangles around the parts of the report that form a distinctive unit. A distinctive unit could be a part of the report that only prints at the beginning or end, a section that repeats on each page, a section that repeats occasionally throughout the report, or a section that repeats continually throughout the report.

These sections, called areas, determine when and why common parts of the report are outputted together. Deciding on the areas of a report early on in the design helps clarify the design of the report -- assisting in the final layout of the report.

Once the rectangles are drawn, write a brief note describing when the area of the report is outputted. Figure 1.3 shows the areas necessary for the example report, and a brief explanation why each area is as it is.



Alumni Association

Patron's List

January 1, 1993 to May 21, 1993

Total Alumni contributions: $xxxxxxx

This area is only outputted at the beginning of the report.

Patron	Contributor ID	Amount
Adams, John	4321-34-1234	$1,000
123 West 4th Street, AnyTown, ST, 55212		
Baker, John	4321-34-1234	$9,000
123 West 4th Street, AnyTown, ST, 55212		
Cramford, John	4321-34-1234	$6,000
123 West 4th Street, AnyTown, ST, 55212		
Denver, John	4321-34-1234	$6,000
123 West 4th Street, AnyTown, ST, 55212		

This area is outputted at the top of each page, except for the first page where it follows the title of the report.

This area is outputted once for every patron

Locating report elements

Having a report prototyped may be the easiest part of obtaining a report. Figuring out how to obtain the information in the report can often be formidable. It requires a knowledge (or a list) of all the data files that might possibly contain information in the report.

The static text elements of a report may be ignored, since they are entered directly into CodeReporter when the report is laid out. Lines, frames, and "special" elements may also be ignored at this point, since they, too, are created with CodeReporter. That leaves the elements that are from data files and graphics.

Creating a list of these extra report elements, and locating the files in which they may be found, defines which data files and graphic files need to be included in the report. Doing this ensures that essential files for the report are found, while making sure unnecessary files are not included.

The list of report elements that use information from a data file for Figure 1.1 are:

- <u>$xxxxxxx</u> - the total amount of money contributed by alumni
- <u>John Q. Public</u> - the patron's name

- 123 West 4th Street, AnyTown, ST, 55212 - the patron's address which is made up of the street address, city, state, and zip code
- 4321-34-1234 - the patron's contributor identity number, and
- $******** - the total amount of money the patron donated

Depending upon the way the databases are laid out, this could be a straight forward report or a more complex report. In the simple case, the data file might look like PATRONS.DBF Figure 1.4. All of the data file fields necessary in the report are included in this one data file, so the report would in essence be a listing of this one data file.

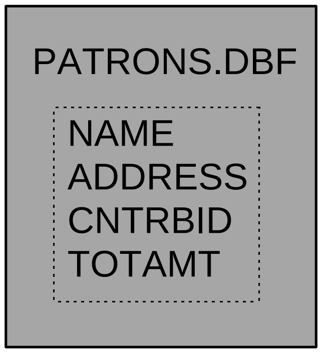


Figure 1.4

Usually, the information for a report is not as nicely laid out. It is often the case that information is spread out through several related data files. Even for this simple case, the data files necessary might look like those in Figure 1.5.

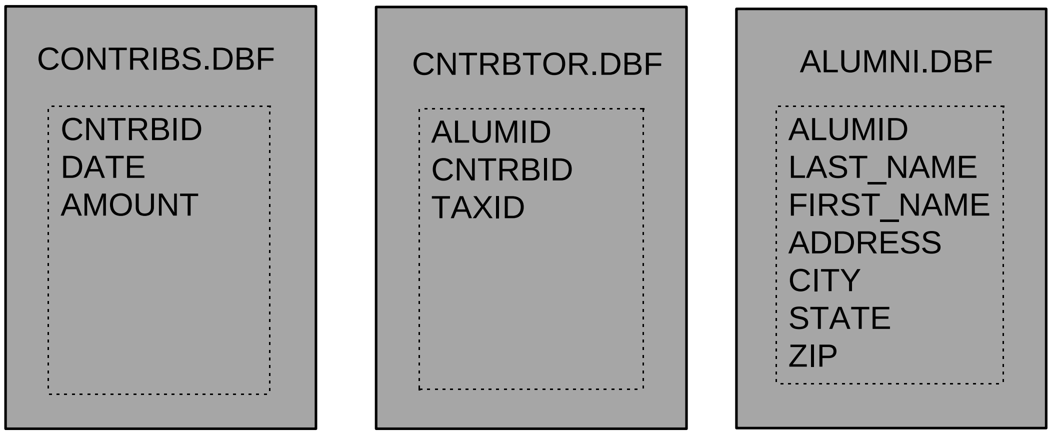


Figure 1.5

In most cases, data will be found in more than one data file, so for the rest of this chapter, Figure 1.5 will be used as the example data file.

The list of report elements that depend upon data files is then linked up with the data files that contain that information. Table 1.1 shows such a list. Notice that some elements, such as the patron's name, are built using more than one field, while others, such as the patron's contributor identity number, are found in more than one data file. List them all, because they may all be required during the layout of the report.

Element	Data file fields for element
$xxxxxxx	CONTRIBS->AMOUNT
Adams, John	ALUMNI->LAST_NAME, ALUMNI->FIRST_NAME
123 West 4th Street, AnyTown, ST, 55212	ALUMNI->ADDRESS ALUMNI->CITY, ALUMNI->STATE, ALUMNI->ZIP
4321-34-1234	CNTRBTOR->CNTRBID, CONTRIBS->CNTRBID
$*******	CONTRIBS->AMOUNT

Table 1.1

Designing the Relation

The data file fields listed are the ones necessary for the report. If they all are found in the same data file, this section may be skipped, and the report may be laid out using CodeReporter.

As stated earlier, most reports require information from several data files. The process of tying, or linking, these separate data files together is the heart of relational databases. The relational database model and the process of creating a relation using CodeReporter is discussed in-depth in Chapter 2.

Using the list of data files and their fields, draw lines between the common fields. This visually illustrates the necessary linkages between the data files, helps determine the type of relationship that exists between the data files, and may help identify the report's top master data file.

Notice in Figure 1.6, the identification of the left to right relation. There are (or may be) more than one contribution recorded in the CONTRIBS.DBF data file for each person contributing.   Each alumnus only has one record in the CNTRBTOR.DBF data file, that is, all contributions made by an alumnus are recorded under only one number.

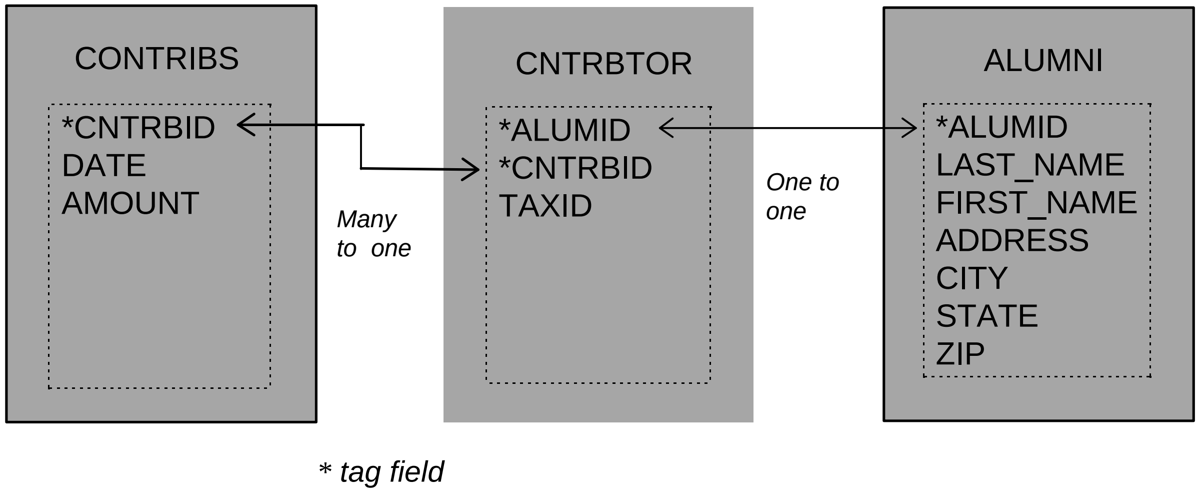


Figure 1.6 Designing a Relation

Identify the Tags

CodeReporter can link data files only when there are index tags available. A tag defines a quick method of locating the records. When two data files are linked, the first data file uses a tag to quickly locate the appropriate information in the second data file. The tags are generally created at the same time as the data files.

Make a special note of each data file's tags in the relation list. If the data file has a compound tag (containing more than one field), write the tag name at the bottom of the field list and mark it as a tag. Change the lines to arrows if the relation points to a tag field.

If a line in the relation diagram points to a field that does not have a tag built upon it, it may be necessary to erase the relation line or to build a tag on the field using another database tool. There only needs to be one link to a data file for it to be included in the relation.

Determining Top Master Data File

Once this diagram is completed, determine the best flow for the relation. Place the data files in a tree diagram with one data file at the top of the diagram, with the rest of the data files below in the order dictated by the linkage lines. In many relations, the data files may fit together in many different ways. The following guidelines may be used to determine best flow for a relation:

- Each lower data file must have a tag pointed to by the higher data file.
- Many-to-one and one-to-one relations are of better design than one-to-many or many-to-many relations

Figure 1.7 shows the different tree diagrams possible using the relation diagram in Figure 1.6. Note the that the first relation has no one-to-many relations and so technically may flow better than the other two. Depending upon the nature of the report, however, it may be necessary to use one of the other relation trees.

For example, if the report requires a list of all alumni, not only the ones that contributed, it would be necessary to use the relation that has ALUMNI on the top. See the Relational Reporting chapter for more information on the implications of a relation upon the outcome of a report.

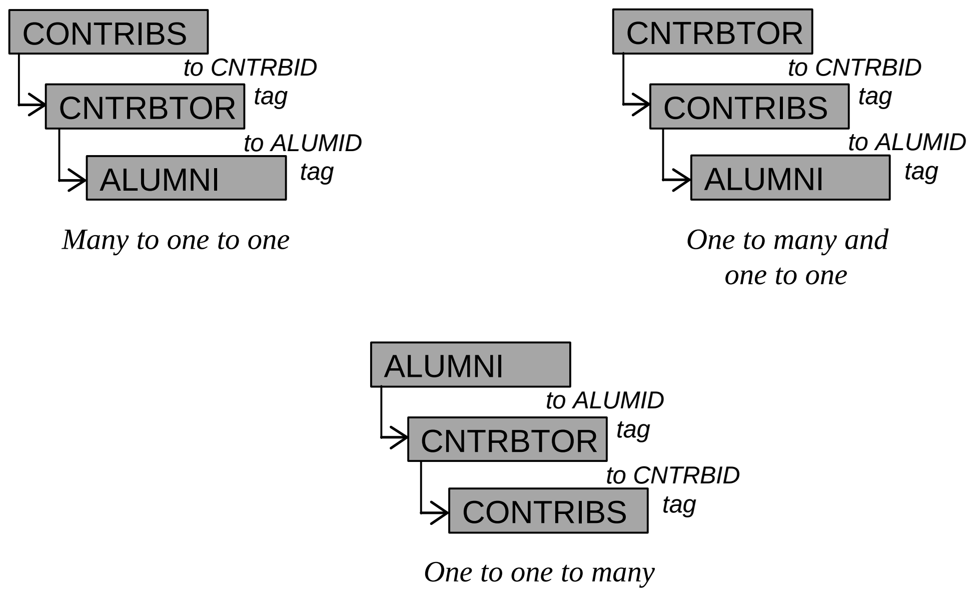


Figure 1.7 Relation Trees

Layout the report

Once this type of diagram is created -- assuming all the data files necessary for the report can be tied together -- the report may be laid out. The rest of this manual discusses the actual implementation of a relational report.

Validate the report

Once the report has been designed using the areas, relations, and report elements discussed in this chapter, check the report against the original Statement of Purpose to ensure that the final report does indeed fulfill the original requirements. If the report passes this final check, the report cycle is finished and the report design is complete.

2. Relational Reporting

Relational reporting is the act of creating a report which uses more than one data file. These diverse data files are integrated in the report to produce a cohesive whole "composite data file" upon which the report is based. Relational reports generally take information from several affiliated data files and output them in a more readable fashion. Relational reporting, put simply, is a process of creating a report which integrates two or more data files.

Relations

When creating a relational report using more than one data file, it is necessary to locate accurate and appropriate information in all the data files. That is, information found in a record in one data file must be logically consistent with the records found in the other data files. For example, the bill for customer #2345, Jane Smith should not be sent to the address of customer #4321, Peter Rodriguez.

In order to ensure that appropriate information is retrieved from all the data files during the report, the manner in which information is obtained must be explicitly described when the report is designed. When this description is created, the data files are said to be linked, or related. A relation says, "When the report uses a record in this data file, use this information from it to locate a record in *that* data file that contains the same *information*."

The data files are related in a hierarchical Master-Slave relationship. The controlling data file is called the master data file, and the controlled data file (the one used for lookups) is called a slave data file.

Breaking the relation statement down helps define the necessary components of a relation.

- "When a report uses a record in this data file" -- This defines the master data file.
- "use this information from it" -- this is a dBASE expression that is evaluated for each record of the master data file. This evaluated expression is used as a search key into the slave tag. This expression is called the Master Expression.
- "to locate a record in that data file that contains the same information" -- this defines the slave data file used for the lookup, and the slave data file's index ordering that is used to locate the corresponding record. The index ordering used in the relation is called the slave tag.

A relation between two data files only operates one way. One data file is used to look up information for another data file. The one data file is controlled by the other, whose records are only retrieved as dictated by the other data file.

In other words, one data file has some information in it that contains a reference to information in another data file. If the two data files are related, the master (first data file) tells the slave data file to locate a record in the slave data file that contains the same information. The master data file, instead of only having a reference to the information in the slave data file, then can use the actual information located in the slave data file and output it.

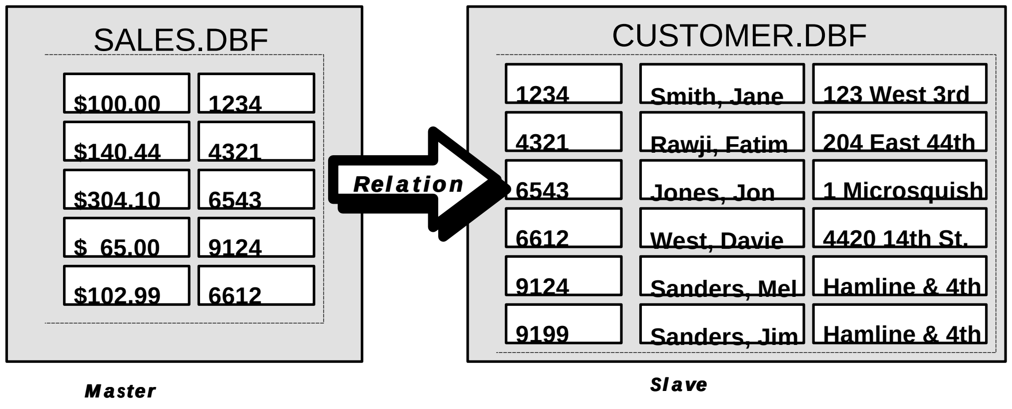


Figure 2.1 Simple Relation

Without a relation, or linkage between SALES.DBF and CUSTOMER.DBF in Figure 2.1, how would a report of sales list the names of the people to whom the sale was made?

With a relation, all the fields of all the data files may be considered a part of a single data file called the composite data file. The composite data file does not actually exist on disk -- CodeReporter does not physically create a file containing all the information of all the related data files -- but is a high level way of describing the intricate way CodeReporter maintains the positioning of the individual data files.

Once a relation is made, the composite data file contains all the fields of all the data files in the relation. The information in the fields of the composite data file is then kept accurate by CodeReporter during the output of the report.

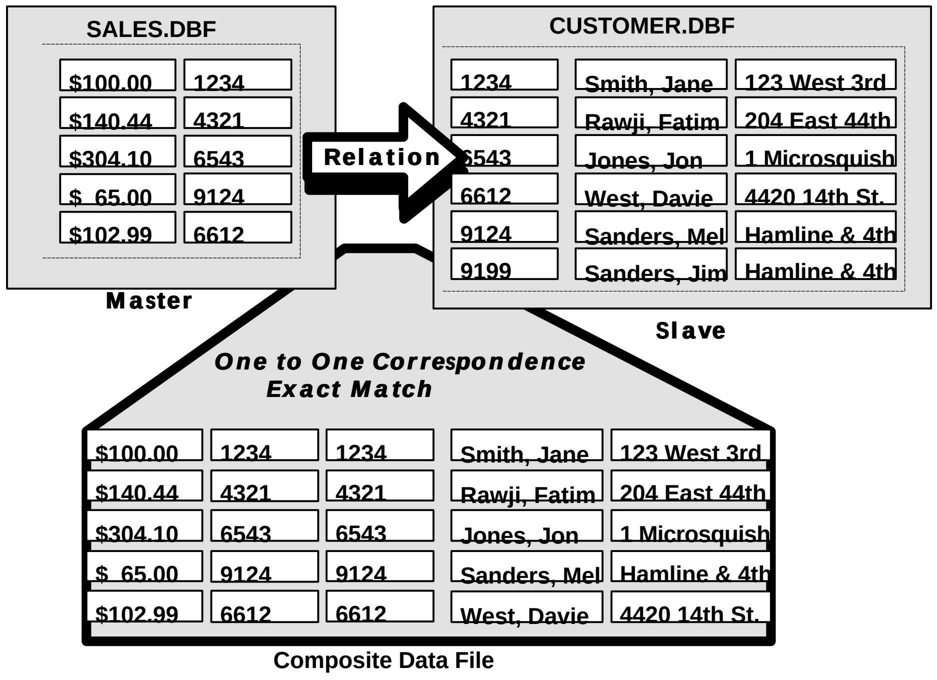

SALES.DBF
$100.00 1234
$140.44 4321
$304.10 6543
$ 65.00 9124
$102.99 6612
Master
Relation
CUSTOMER.DBF
1234 Smith, Jane 123 West 3rd
4321 Rawji, Fatim 204 East 44th
6543 Jones, Jon 1 Microsquish
6612 West, Davie 4420 14th St.
9124 Sanders, Mel Hamline & 4th
9199 Sanders, Jim Hamline & 4th
Slave
One to One Correspondence
Exact Match
$100.00 1234 1234 Smith, Jane 123 West 3rd
$140.44 4321 4321 Rawji, Fatim 204 East 44th
$304.10 6543 6543 Jones, Jon 1 Microsquish
$ 65.00 9124 9124 Sanders, Mel Hamline & 4th
$102.99 6612 6612 West, Davie 4420 14th St.
Composite Data File

Figure 2.2 Composite Data File

While the composite data file contains all the records of the master data file, it does not necessarily contain all the records of the slave data file. This is a direct result of the Master-Slave relationship. When the relation is used, the records of the master data file are used to locate corresponding records in the slave data file. However, the master data file is not required to have references to all the records in the slave. The slave is only used as a lookup.

The steps taken internally in CodeReporter are:

1. Position to a record in the master data file,
2. Take the common information in the master data file (defined by the master expression)
3. Search for a record in the slave data file with the same information (this is now the composite record),
4. Repeat steps 1 - 3 until there are no more records in the master data file.

It becomes apparent that if the master doesn't reference a particular record for the slave, that slave record is not included in the composite data file. This is why it is important during the design of the relation, that the statement of purpose for the report be take into account. If the wrong top master data file is chosen, the results of the report can be quite different than what is desired.

This occurs in the composite data file of Figure 2.2. Notice that in the CUSTOMER.DBF, there is an entry for Jim Sanders (Customer number 9199). However, since the master data file, SALES.DBF does not reference customer 9199, Jim Sanders is not included in the composite data file.

It can also be the case that the slave data file does not have a corresponding record for the master. The master data file requests information from the slave, but the slave hasn't got a record containing that information.

When this occurs, CodeReporter can do one of three things -- depending upon the way the relation was set up.

1. Blank Fields. CodeReporter fills in the fields of the slave data file with spaces. Numeric values contain a zero value. This is the default action.
2. Skip Record. The composite record is ignored. Both the master and slave records are skipped by the CodeReporter as if neither one existed.
3. Stop with Error. The report is stopped and an error message is displayed. This may the report should be aborted.

Complex Relations

Using just two data files is fine if they contain all the information needed for the report. If, however, two data files do not suffice -- if there are three or more data files necessary for the report -- a complex relation is needed.

A complex relation is a relation that contains sub-relations. That is, a slave of one data file must act as a master data file for a lower level slave.

A master data file can be linked to more than one slave data file, and a slave data file can in turn be used as a master data file to link in yet another slave data file for further relations. This allows the creation of a relation "tree", with all the relations descending from a single top level master data file.

Terminology at this point can get somewhat confusing. As a convention the term "top master data file" refers to the top data file in the relation tree, which is the main data file of the report. "Master data file" refers to the controlling data file in the relation currently being discussed.

All the relations descending from the top master data file as a unit are called the relation set.

The diagram 2.3 in the CodeReporter manual below shows a complex relation where the STUDENT.DBF data file is the master of the ENROLL.DBF data file, which is then a master of the COURSE.DBF data file.

Relation Types

CodeReporter supports three different types of relations. Each different relation type accesses data from the slave data file in a different manner. The three relation types available in CodeReporter are: exact match, scan, and approximate match.

Exact Match Relations

An exact match relation defines a one-to-one correspondence between the master and slave data files. Each record from the master data file can have only one corresponding record in the slave data file. An exact match relation will always return the first corresponding record in the slave data file, even if more than one record matches the evaluated master expression. This is a one-to-one relation. One record in the master data file locates one record in the slave data file.

In other words, for each record in the master data file, CodeReporter searches the slave data file index until it finds a tag entry with the exact contents as the evaluated master expression. The search stops when the first record match is found, or until the entire slave tag has been searched.

In general, an exact match is best used in conjunction with a slave tag that is unique.

The diagram shows an exact match relation between the SALES.DBF and the CUSTOMER.DBF.

Approximate Match Relations

The second type of relation is the approximate match relation. This is similar to the exact match relation in that it only permits one match for a master record. The only difference is the way it behaves when an exact match is not found in the slave data file. If the match fails, the slave record whose index key appears next in the slave tag is used instead.

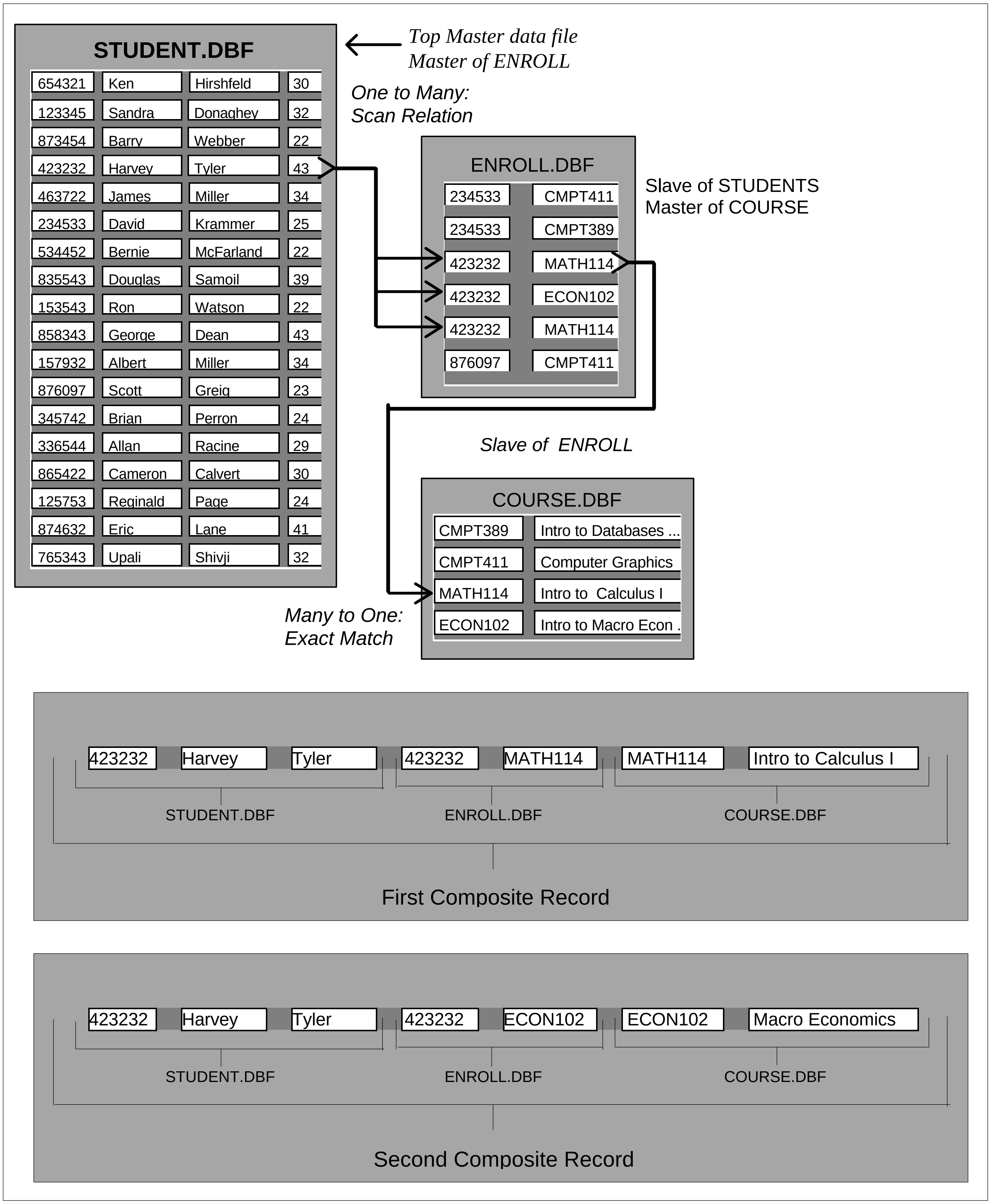


Figure 2.3 Complex Relation

Approximate match relations are generally quite rare and are usually used only when a range of values in are represented in the data file by a single high value.

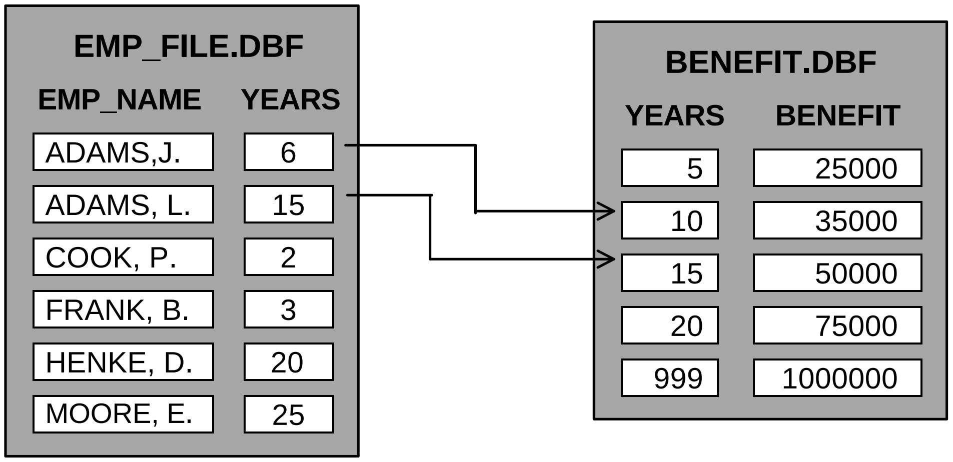


Figure 2.4 Approximate Match Relation

An approximate match relation is shown in Figure 2.4. In this case, the employees' retirement benefits are determined by the number of years that they put into the company. Instead of making an entry for each possible year served, the BENEFIT.DBF only lists the upper limit for each pay out level. The first pay out level is from zero to five years, the second is six to ten, et cetera until the maximum entry of twenty-one years and above pays out 100,000.

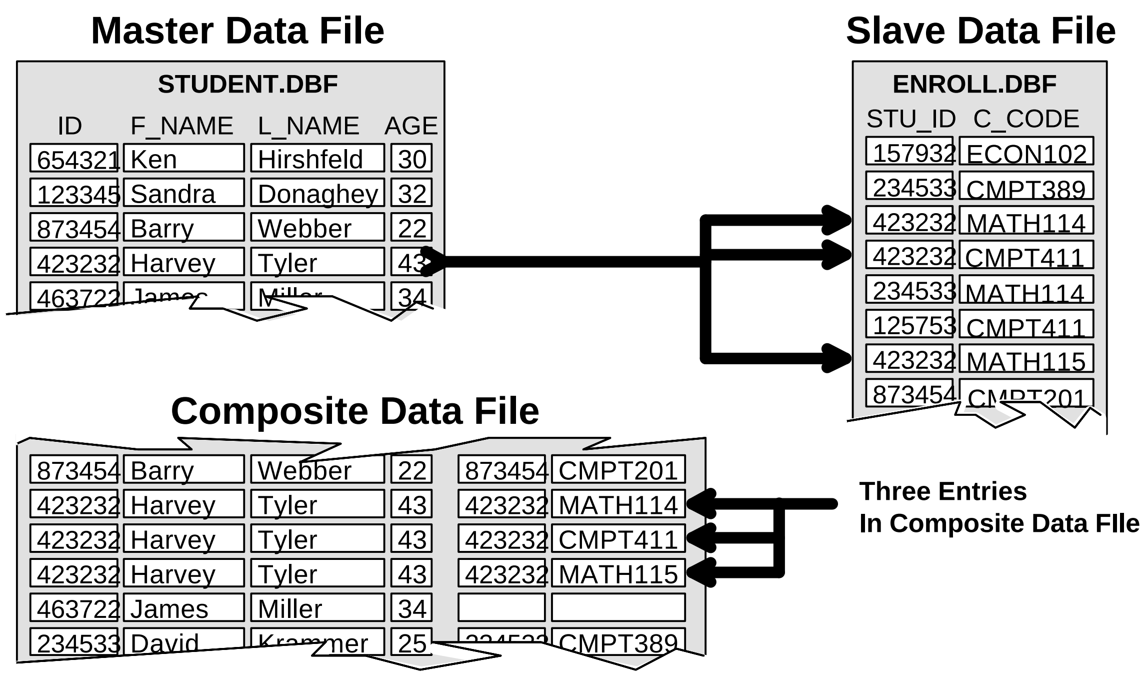
If an exact match is not found using an approximate match relation, the record used from the slave data file is not necessarily the record closest to that of the master expression, but the first record with a key value greater than the master expression.

In Figure 2.4, a master expression value of six looks up the tag entry for ten, even though six is numerically closer to five.

Scan Relations

Scan relations define a one-to-many correspondence between the master data file and the slave data file. This means that for each record in the master data file, CodeReporter finds all the matching records in the slave data file, not just the first.

Figure 2.5 uses a scan relation to relate the master data file STUDENT.DBF with ENROLL.DBF. This relation would include every class in which a student is enrolled. In this example, the scan relation produces three composite records using a single record in the master data file.



STUDENT.DBF

ID	F_NAME	L_NAME	AGE
654321	Ken	Hirshfeld	30
123345	Sandra	Donaghey	32
873454	Barry	Webber	22
423232	Harvey	Tyler	43
463722	James	Miller	34

ENROLL.DBF

STU_ID	C_CODE
157932	ECON102
234533	CMPT389
423232	MATH114
423232	CMPT411
234533	MATH114
125753	CMPT411
423232	MATH115
873454	CMPT201

873454	Barry	Webber	22	873454	CMPT201
423232	Harvey	Tyler	43	423232	MATH114
423232	Harvey	Tyler	43	423232	CMPT411
423232	Harvey	Tyler	43	423232	MATH115
463722	James	Miller	34		
234533	David	Krammer	25	234533	CMPT389

Figure 2.5 Scan Relation

Master of multiple slaves

A complex relation may also include one master data file and two or more slave data files. That is, two slave data files may be related to the same master data file. In most ways, this configuration is exactly the same as any other -- the master expression for each relation is evaluated for the current record in the master data file, and each result is used as a lookup key into each slave tag.

When a scan relation is not involved, the relation behaves in the standard manner -- the composite record includes the information from the master data file's record and from each slave data file. It does not matter whether the relations are exact match or approximate match. See Figure 2.6.

However, when a single master data file has two or more slave data files each with scan relations, CodeReporter performs the relation in a slightly different manner. The error action for each of the master's scan relations must be set to blank fields because the complex relation is performed on each scan relation individually, leaving the other scan relations blank.

Figure 2.6 illustrates a single master with two data files, first with exact match relations and then with scan relations.

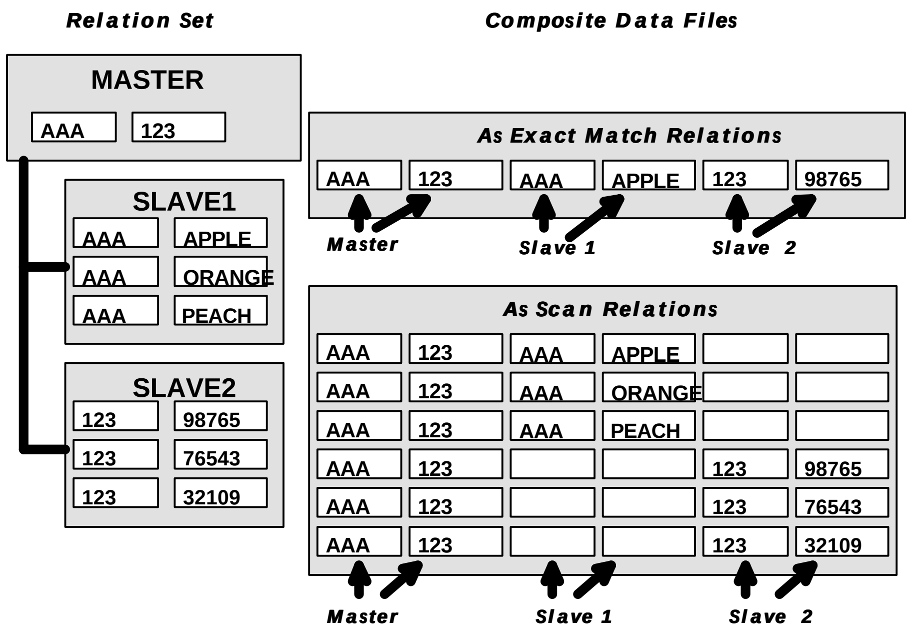


Figure 2.6 Complex relation with one master and two slaves

A scan relation may not be mixed with exact or approximate match relations when a single master data file has two or more slave data files.

Creating Relations

The first step in creating a report is to create the backbone relations for the report. It is suggested for any report that may involve several data files that the procedures described in the Report Design chapter be followed and that the relation be sketched out on paper before creating them in CodeReporter.

Selecting Top Master Data File

A new relation may be created by selecting the **FILE | NEW** menu option. The "Select Data File" dialog box is invoked (a Windows 3.1 common "File Open" dialog), prompting for the top master data file. Use the directories and file name list boxes to locate the top master data file for the report and open it by selecting the "OK" button.

Path Names

CodeReporter, by default, saves the full path names to the data files used in a report within the report file. As the report is loaded, the data files for the report are also opened. If the data files have been moved or deleted, CodeReporter is unable to locate the non-existent files. When this situation occurs, use the **FILE | OPEN WITH PATH** menu option. After a report file has been selected, CodeReporter pauses and prompts for the directory in which all the data files of the report may be found. This new directory is then saved with the report.

Once selected, the top master data file for the report may not be changed. If another top master data file is desired, the report must be re-created from the beginning.

Bit Optimized Query Technology

CodeReporter uses Sequiter's Bit Optimized Query Technology (BOT) to perform high-speed querying of the composite data file. This is done by comparing the query expression with the tag sort orderings of the top master data file. If the query matches the tag expression, the tag itself is used to filter out records that do not match the query.

This results in lightning quick performance, even on the largest of composite data files, since only necessary records are actually physically read from disk.

See , below, for information on how to open tags for the top master data file.

In order to maximize the chances BOT can be used, it is suggested that all the possible tags for the top master data file be opened.

The Relation Dialog

Slave data files may be added to the relation set by using the "Relation" dialog (Figure 2.7) which is invoked from the **RELATION | MODIFY** menu option. This dialog visually shows the relation as it is assembled -- with lines showing where the linkages occur. The data file at the upper left of the "Relation" dialog is the top master. As data files are added to the relation set, a button is added below and to the right of its master -- visually creating the relation "tree." When more slaves are added to the same master, their buttons are added directly below those of previously added slaves

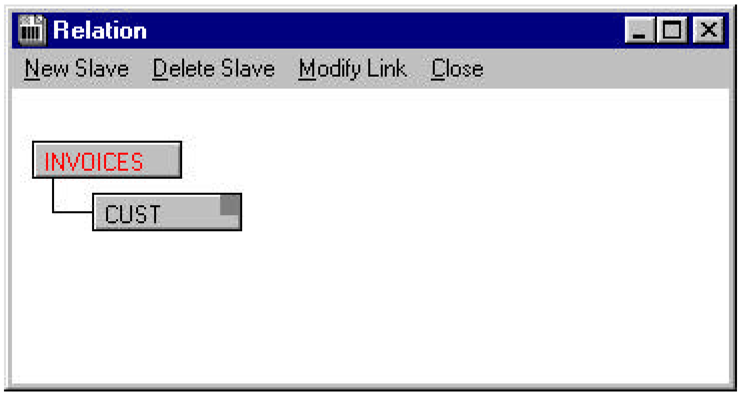


Figure 2.7 Modify Relation Dialog

Adding a Slave Data File

A slave data file may be added to the relation set by selecting its master and choosing the **NEW SLAVE** menu option, or double clicking the master's button. It makes no difference whether or not the master data file is already a slave of another data file. The "Select Data File" dialog is invoked to locate and open the slave data file. Again, use the file name and directory list boxes to locate the slave data file.

Modifying a Relation

The "Data File Link" dialog (Figure 2.8) is used to define and modify a relation between a master and a slave data file. This dialog is automatically invoked when a new slave is created. It may be invoked at a later point for modification of the relation by selecting the slave data file's button and choosing the **MODIFY LINK** menu option, or by clicking on the slave data file's button with the right mouse button.

Enabling BOT

If the selected data file button is the top master data file for the relation, and **MODIFY LINK** (or the right mouse button) is used, the "Master Index Files" dialog is invoked. This dialog may be used to open and/or close index files for the top master data file -- thus enabling the report to take advantage of BOT.

If the relation is new, CodeReporter fills the "Data File Link" dialog with default information, as shown in Figure 2.8. Before the new relation is accepted, both the "Master Expression" edit control and an existing tag must be selected.

Master Expression

The master expression is a dBASE expression, based on the master data file (or data files higher in the relation tree), that is used as a lookup into the slave tag. When the report is run, this expression is evaluated for each record retrieved for the master data file, and its results are used to locate a record in the slave data file via the specified tag for the slave.

This expression is generally as simple as a field from the master data file, but may be more complex -- involving fields from higher data files in the relation tree and/or dBASE functions.

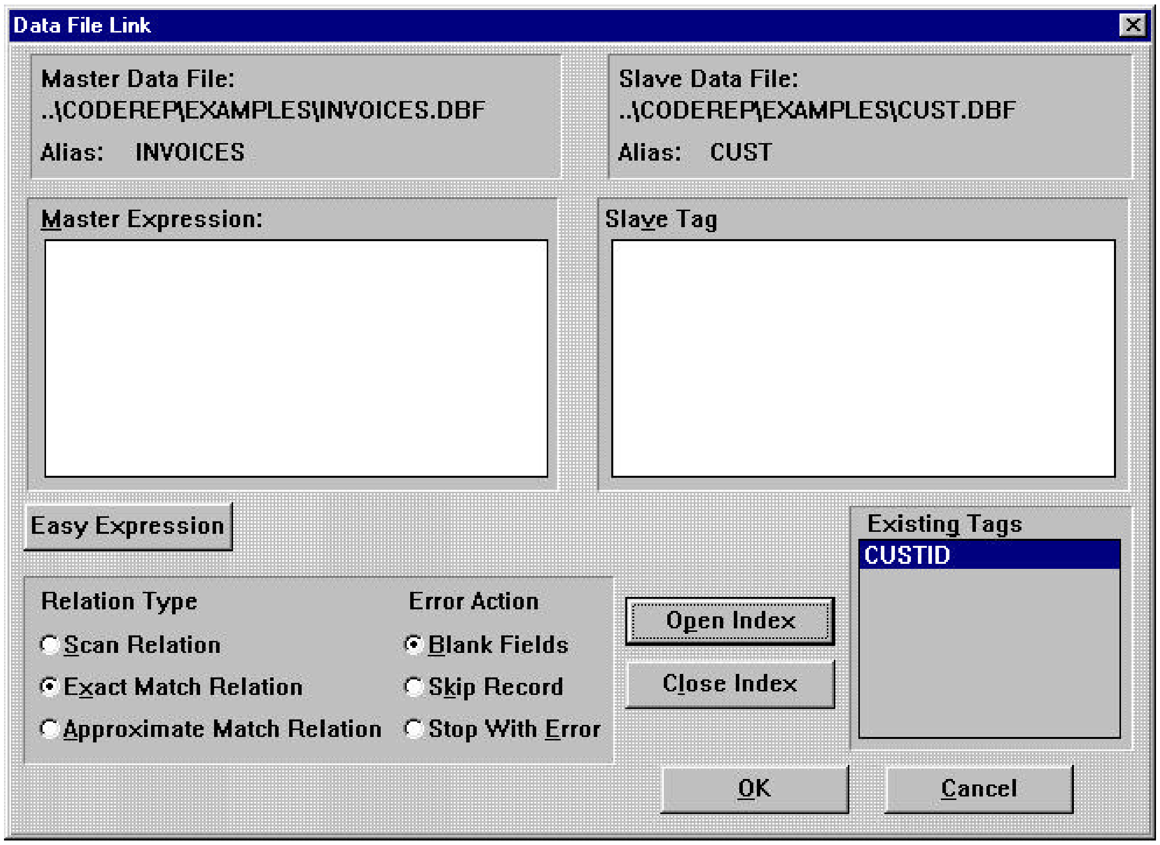


Figure 2.8 Data File Link Dialog

A master expression may either be typed manually into the "Master Expression" edit control, or the "Easy Expression" button may be used to simplify the expression entry.

For more information on dBASE expressions and the "Expression Entry" dialog box, see the Expressions chapter.

Selecting a Tag

The tags for the slave production index file (if there is one) are listed in the "Existing Tags" list box. When a tag in the list is selected, information about it, including its sort ordering, is displayed in the "Slave Tag" edit control. If used in the relation, the dBASE expression listed for the tag should correspond to the master expression for the lookup to function correctly.

Opening An Index

If there isn't a tag that corresponds to the master expression, or if the slave data file does not have a production index file, the "Existing Tags" list box only contains the 'None' entry.

Tags from index files other than the production index file may be opened and used in the relation by selecting the "Open Index" button. This button invokes the "Select Index File" dialog box which may be used to locate and select the new index file.

The CodeReporter executable is index file specific and may only open index files which are compatible with its index format. Reports may not mix index file

formats. Other index file formats require a different index specific CodeReporter executable. See the Getting Started section of this manual for using the appropriate CodeReporter DLLs.

Once an index file is opened, its tags may be selected for the relation from the "Existing Tags" list box.

Index files containing unused tags may be closed using the "Close Index" button. When selected, the "Close Index" button closes the index file associated with the selected slave tag, and removes all other tags for that index file. However, since having the added open index files does not degrade performance, it is unnecessary to close them.

Slaves Without Tags

CodeReporter allows an alternate method of performing lookups that does not use tags from the slave data file. Instead of having the master expression evaluate to a lookup key, it may evaluate to a record number. This record number specifies the physical record number of the slave record retrieved from the slave data file.

This method is mainly useful on static unchanging slave data files that are never packed. This method has the advantage of being faster and more efficient than performing seeks on a tag.

To use this method, simply select the "None" option in the "Existing Tags" list box. No tag is selected, and the master expression is then taken as a record number.

It is the responsibility of the report designer to ensure that the record number for the evaluated master expression references the appropriate slave record number for the master data file record.

If the evaluated master expression contains a reference to a non-existent record number (<= 0 or > the number of records in the slave data file), CodeReporter generates a -70 error as the report is generated.

Setting the Relation Type

The "Data File Link" dialog is used to set the type of relation between the master and slave data files. The different relation types specify how records are retrieved from the slave data file. See Relation Types, above, for information on the different types of relations.

The default relation type, an exact relation, may be modified by selecting either the "Scan Relation" or "Approximate Match Relation" radio buttons.

The Error Action

The "Error Action" radio buttons control how CodeReporter acts when a lookup into a slave data file fails to locate a record. For more information on error actions, see Error Actions, above.

The default error action, blank fields, may be modified by selecting either the "Skip Record" or "Stop with Error" radio buttons.

When an approximate match relation is defined, the only error action available is "Blank Fields".

Moving a Slave Data File

As shown in "Figure 2.7", the buttons representing the data files in the "Relation" dialog contain movement handles. The dark gray square in the upper right corner of the data file button may be used by the mouse to move a slave data file above or below a slave of the same master.

The only advantage to moving a slave higher or lower in relation to its master data file is that lower level slaves may use the fields of higher level slaves in the master expression for its relation.

In Figure 2.7, the STORES - EXPENSES relation may use all the fields of the COMPANY.DBF, STORES.DBF, and SALES.DBF data files to define its relation. The STORES - SALES relation, on the other hand, may only use fields from the COMPANY.DBF and STORES.DBF data files in its master expression. A data file may be moved in relation to its master data file by pressing and holding its movement handle with the left mouse button and dragging it higher or lower. If a data file may not be moved, dragging it to a new location has no effect.

Sorting the Composite Data File

When information is entered into data files, it is usually done in a random manner. All the sales for customer #1 are not always entered before the sales for customer #300. However, reports generally require the information be outputted in a logical order -- alphabetic, numeric, by date, etc.

CodeReporter provides a way to sort the composite data file, via the sort expression. This dBASE expression is evaluated for each composite record, and the results (and the composite records) are retrieved in the new logical order. This process is called sorting the composite data file.

Sort Expression

The sort expression is simply a dBASE expression that can incorporate any field or combination of fields in the composite data file. Figure 2.9 illustrates the affects of a sort expression. Notice that the sort expression incorporates a field from the master data file as well as the slave data file.

Entering a Sort

The sort expression is entered using the "Easy Expression" dialog. This dialog may be invoked from the **REPORT | SORT EXPRESSION** main menu option.

For more information on dBASE expressions and the "Easy Expression" dialog box, see the expressions subsection of the Objects chapter.

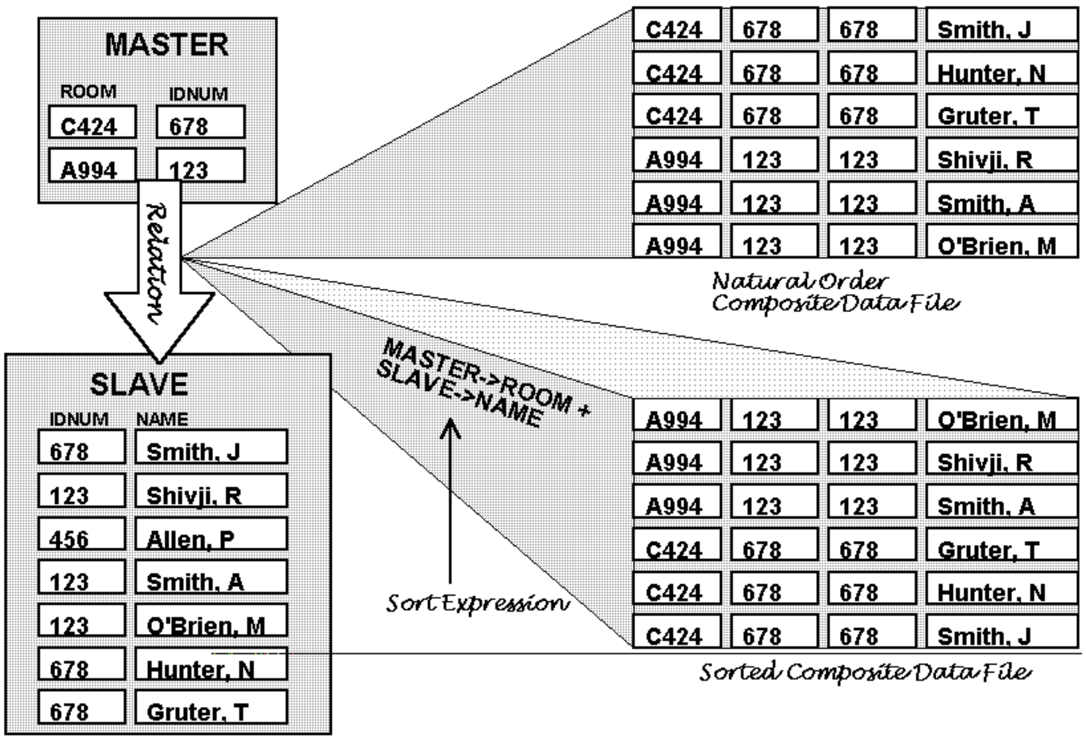


Figure 2.9 Sorting a Relation Set

Query the Composite Data File

A query is a logical expression which is used to create a subset of the composite data file. This query expression creates a filter through which composite records of the relation must pass. Only the records that meet this filter criterion are included in the report.

That is, this expression is evaluated for each composite record of the composite data file and if it results in a .TRUE. value, the record is included in the report, otherwise it is ignored. For example, if it was required that the relation described in Figure 2.9 only include the people in room A994 whose name began with an 'S', the queried composite data file (in natural order) would be that of Figure 2.10.

Entering a Query

The query expression is entered using the "Easy Expression" dialog. This dialog may be invoked from the **REPORT | QUERY EXPRESSION** main menu option.

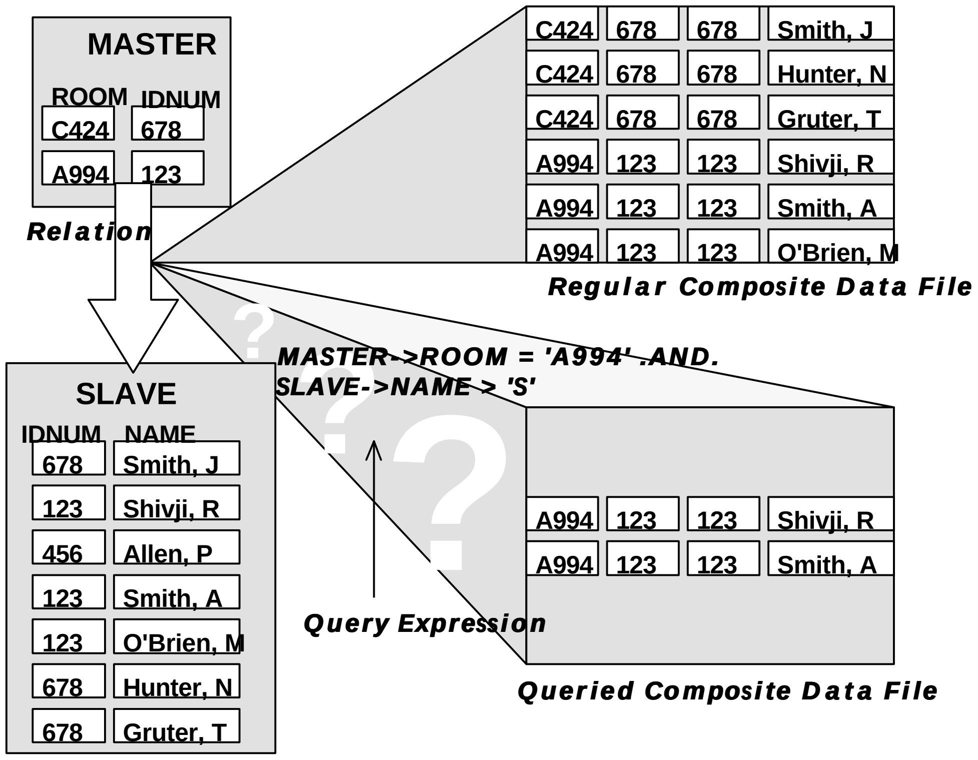


Figure 2.10 Queried Data File

Relations on Disk

Many times the relation set of one report is the same, or similar to that of other reports. Relation sets created with CodeReporter may be saved to disk and retrieved into new reports using the **FILE | SAVE RELATION** and **FILE | LOAD RELATION** main menu options. These menu options prompt for the file name of the relation file and save/load the relation.

Relations are saved into special CodeReporter relation files which have a .REL extension. If a relation is not needed for another report, it is not necessary to save it, since a saved CodeReporter report file (.REP extension) contains the report's relation.

When loading a relation file from disk, the current report's relation, and all objects, totals, and calculations, are deleted.

Saving as Code

CodeReporter may also save the relation as source code used with the CodeReporter API function calls. Use the **RELATION | SAVE RELATION AS SOURCE CODE** menu option. After selecting a destination file type, select a file name, and directory using "Specify A Source File" dialog. This file may then be used with the CodeReporter API. See the CodeReporter API for more information on using the generated source code file.

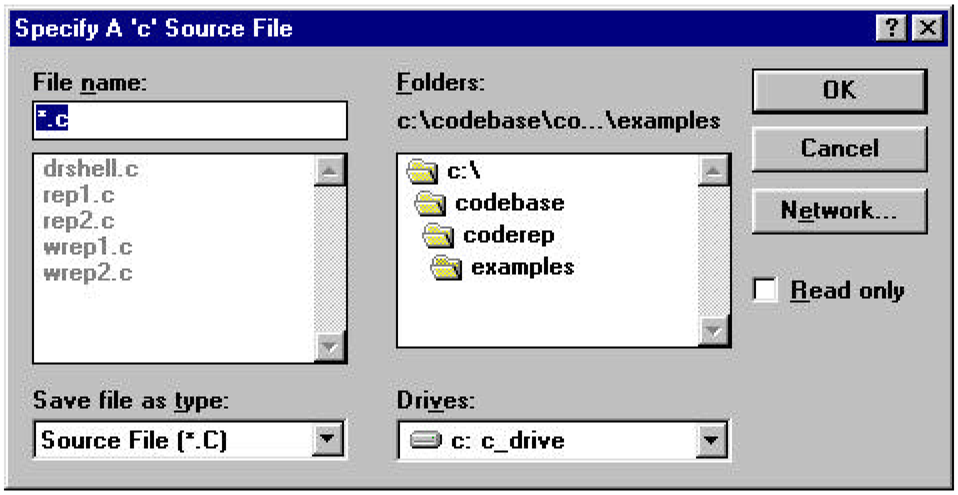


Figure 2.11 Save As Code Dialog

Example

As an example of building a relation, this section will describe the steps necessary for building the relation shown in Figure 2.7.

Top Master Data File

The top master data file for any relation, COMPANY.DBF in this case, is set when a new report file is created. Select **FILE | NEW** to create a new report file.

When the "Select Data File" dialog appears, locate the COMPANY.DBF data file (in the CodeReporter examples directory: .\EXAMPLES), and select the "OK" button.

The report definition screen is displayed. Select **RELATION | MODIFY** to add the slave data files to the relation set. The "Relation" dialog will look like Figure 2.12.

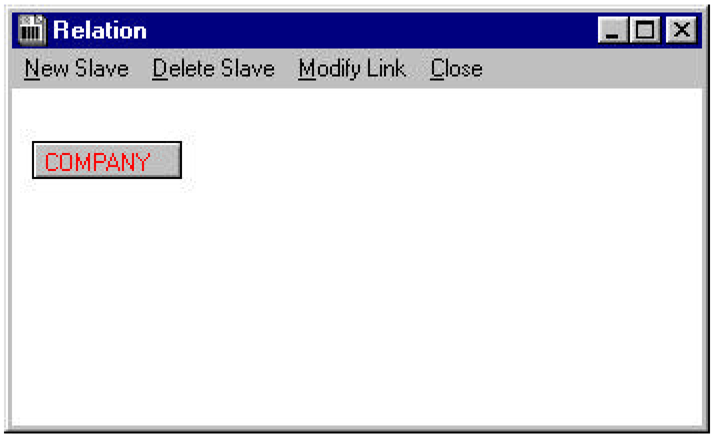


Figure 2.12 Relation dialog for example

The top master data file is displayed as a button with the data file's name upon it. The next related data file to add is the STORES.DBF data file. Choose the **NEW SLAVE** menu option and, when the "Select Data File" dialog appears, select the STORES.DBF data file.

Modifying the Relation

Since this is a new relation, the "Data File Link" dialog box automatically appears. STORES.DBF has two tags in the "Existing Tags" list box: COMPID (keyed on STORE->COMPID) and COMP_STORE (keyed on STORE->COMPID+STORE->STOREID). The master data file COMPANY has a COMPID field, but no STOREID field, so the STORES.DBF tag COMPID should be selected.

Use the keyboard to tab to the "Existing Tags" list box and select COMPID, or click on it with the left mouse button. Notice that the "Slave Tag" edit control automatically displays information on the COMPID tag.

A master expression that corresponds to the selected tag should be entered into the "Master Expression" edit control. In this case, the master expression is merely a field of the master data file. Enter the following text (but do not hit the enter key).

```
COMPANY->COMPID
```

dBASE III and Clipper users must manually open the index files containing the appropriate index ordering using the "Modify Link" dialog's "Open Index" button. The files used in this example are found in the CodeReporter examples directory (.\EXAMPLES) and are named:

- **STCOMPID.NTX (.NDX) (for STORES.DBF)**
- **SACMPSTO.NTX (.NDX) (for SALES.DBF)**
- **EXCMPSTO.NTX (.NDX) (for EXPENSES.DBF)**

The type of relation between COMPANY and STORES may depend upon the report. Since each company likely has more than one store, however, the relation type is probably a scan relation. Use the mouse to select the "Scan Relation" radio button.

Change the default error action, Blank Fields, to Skip Record and the "Modify Link" dialog looks like Figure 2.13. Press the "OK" button.

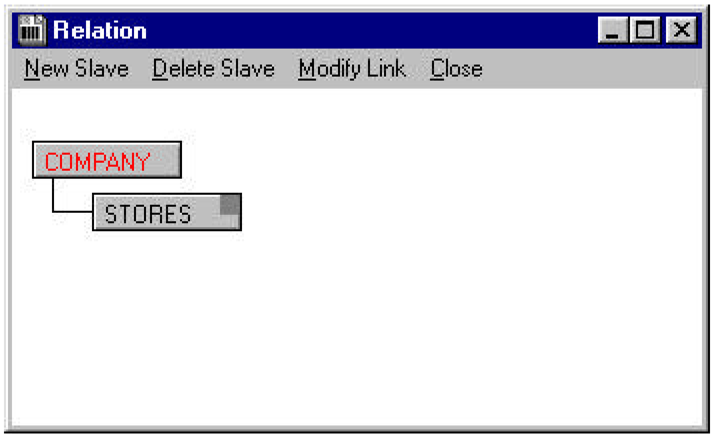


Figure 2.13 Partially completed sample relation

Adding a Slave to a Slave

The STORES.DBF data file is a slave of the COMPANY.DBF data file. In order for the relation to properly retrieve information for each store's sales, an additional data file, SALES.DBF, is needed. The SALES.DBF data file contains the following fields: COMPID, STOREID, AMOUNT, and PRODCODE (a reference to a products data file). As it can be seen each sale in the data file is associated with a company and a store. As a result, in order for the sales to be attributed to exactly the right store, information about the company and its store is necessary.

This indicates that the SALES.DBF data file must be placed within the relation in a place where it has access to both the COMPANY.DBF and STORES.DBF data files. The SALES.DBF data file is properly placed as a slave of the STORES.DBF data file. STORES.DBF, then, is acting as both a slave data file of COMPANY.DBF and a master data file of SALES.DBF.

Use the Tab key to move the selected data file (currently COMPANY.DBF) to STORES.DBF or select STORES.DBF with the left mouse button.

To add the SALES.DBF data file to the relation, select the **NEW SLAVE** menu option or double click on the STORES.DBF data file's button. Either one invokes the "Select Data File" dialog box in which the SALES.DBF data file may be located and selected.

Once the data file is selected, the "Modify Link" dialog box is invoked so that the relation may be properly set up. Enter the following settings and select the "OK" button:

- Select the COMP_STORE tag entry,
- Enter COMPANY->COMPID+STORE->STOREID for the master expression,
- Set a scan relation by selecting the "Scan Relation" radio button,
- Ensure the default error action, blank fields, is selected using the "Blank Fields" radio button.

Once these settings are made, choose the "OK" button to exit the "Modify Link" dialog. The SALES.DBF data file is added below STORES.DBF.

The reasoning behind placing the SALES.DBF data file in its position is exactly the same for the EXPENSES.DBF data file. EXPENSES.DBF requires information from both COMPANY.DBF and STORES.DBF, but not SALES.DBF. As a result, it should be placed on the same level as SALES.DBF.

While the STORES.DBF data file's button is selected, choose the **NEW SLAVE** menu option, select the EXPENSES.DBF data file, and enter the following settings in the "Modify Link" dialog:

- Select the COMP_STORE tag entry,
- Enter COMPANY->COMPID+STORE->STOREID for the master expression,
- Set a scan relation by selecting the "Scan Relation" radio button,
- Ensure the default error action, blank fields, is selected using the "Blank Fields" radio button.

Select the "OK" button to close the "Modify Link" dialog and return to the "Relation" dialog.

This complex relation may now be used as the basis for a complex report involving information from all four data files.

Since this relation involves two scan relations using the same master data file, the information retrieved may be slightly different than expected. See the Master of Multiple Slaves section, above for more information.

3. Groups

What is a Group?

It is the purpose of all reports to put randomly entered data into an organized format that is easily understandable. Reports break up this data into logical sections and/or summarizes it to put the data into a useful format.

By sorting the composite data file, records with common information are placed next to each other. Records may be sorted upon one or many fields. That is, the composite records are accessed in a new order depending upon the fields located in the sort expression.

As a result of being sorted, these records can be thought of as being in logical subsets.

For example, a data file that is sorted on department code creates a composite data file that contains subsets of different departments. All records containing department 'A' will be placed before the records containing department 'B'. All records containing department 'A' may be thought of as a subset of the composite data file.

When the sort expression contains more than one field from the composite data file, the resulting composite data file may be thought of as having major subsets which contain their own minor subsets.

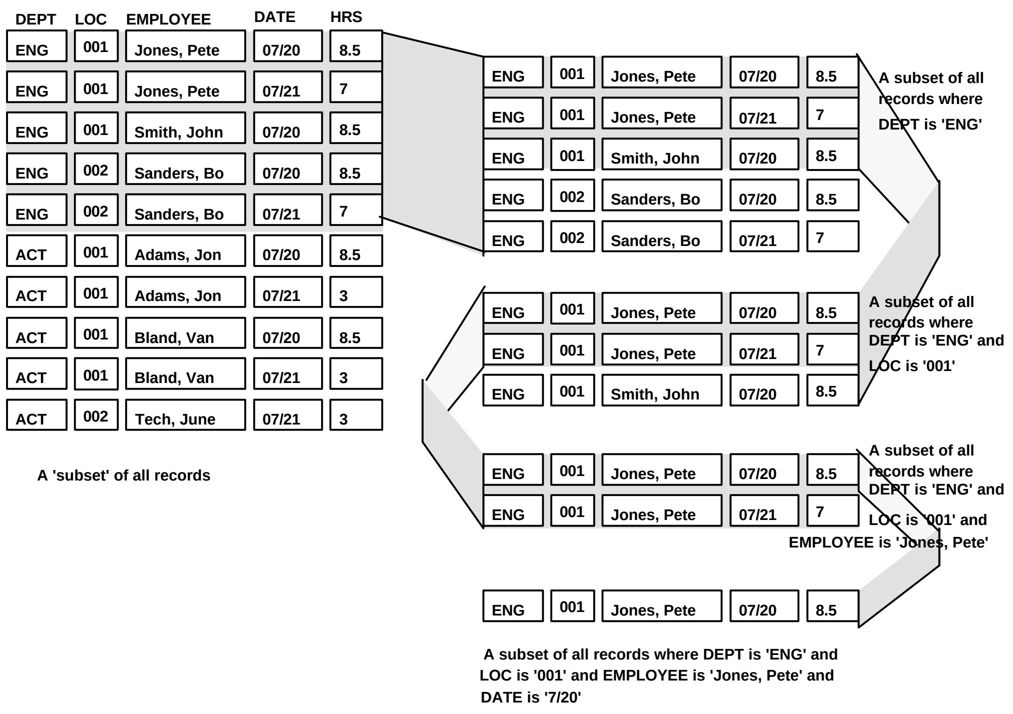
For example, a sort expression containing both the department code and the location code will have individual department subsets which contain their own location subsets. See Figure 3.1 for an illustration of major and minor subsets.

These inherent subsets, then, also describe the overall flow of the report. The report logically progresses from one record to the next, stepping through each major and subsequent minor subsets of the composite data file, until all records have been processed.

When each new subset of the composite data file is processed for the report, certain actions may be performed, such as accumulating/resetting totals, outputting fields from the composite data file, outputting titles, performing calculations, etc. Which actions are performed for a given composite record depends upon the subset in which the record may be found.

These subset-specific actions are performed by CodeReporter in constructs called groups.

The best way to illustrate the concept of subsets and groups are by way of an example. Figure 3.1 shows a composite data file which is sorted on DEPT, LOC, EMPLOYEE, and DATE. As shown on the right, sorting the data file in this manner produces four distinct subsets -- one for each part of the sort expression.



DEPT	LOC	EMPLOYEE	DATE	HRS
ENG	001	Jones, Pete	07/20	8.5
ENG	001	Jones, Pete	07/21	7
ENG	001	Smith, John	07/20	8.5
ENG	002	Sanders, Bo	07/20	8.5
ENG	002	Sanders, Bo	07/21	7
ACT	001	Adams, Jon	07/20	8.5
ACT	001	Adams, Jon	07/21	3
ACT	001	Bland, Van	07/20	8.5
ACT	001	Bland, Van	07/21	3
ACT	002	Tech, June	07/21	3

A 'subset' of all records

ENG	001	Jones, Pete	07/20	8.5
ENG	001	Jones, Pete	07/21	7
ENG	001	Smith, John	07/20	8.5
ENG	002	Sanders, Bo	07/20	8.5
ENG	002	Sanders, Bo	07/21	7

A subset of all records where DEPT is 'ENG'

ENG	001	Jones, Pete	07/20	8.5
ENG	001	Jones, Pete	07/21	7
ENG	001	Smith, John	07/20	8.5

A subset of all records where DEPT is 'ENG' and LOC is '001'

ENG	001	Jones, Pete	07/20	8.5
ENG	001	Jones, Pete	07/21	7

A subset of all records where DEPT is 'ENG' and LOC is '001' and EMPLOYEE is 'Jones, Pete'

ENG	001	Jones, Pete	07/20	8.5

A subset of all records where DEPT is 'ENG' and LOC is '001' and EMPLOYEE is 'Jones, Pete' and DATE is '7/20'

Figure 3.1 A composite data file and its subsets

Each of these subsets will most likely have an action associated with it; probably providing a subtotal of hours for each subset; a total number of hours worked by each employee, a total number of hours worked at each location, etc.

As a result, each subset may be represented by a group which will perform this action. The full composite data file itself may also be considered a group in this regard, since the report will likely contain a grand total of all departments. The sorted composite data file in Figure 3.1 suggests a five group report similar to the one found in .

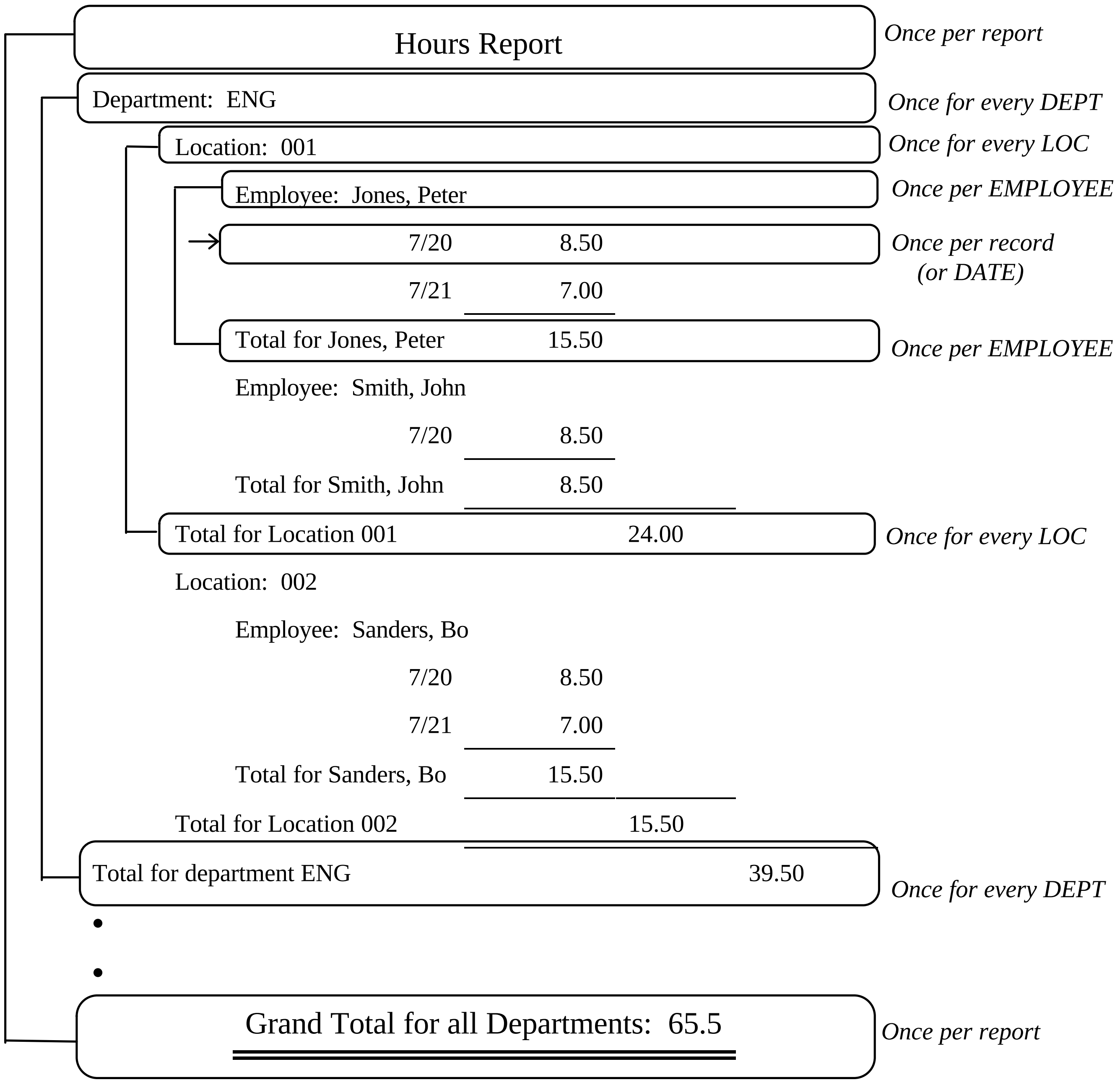
Group Expression

The group expression is a dBASE expression that describes the subset with which the group is associated. This expression is evaluated for each composite record in the data file, and when its value changes, the group's actions are performed (mostly outputting the group's header and footer for the composite record). This change of the evaluated expression is called a reset condition -- also known as a control break.

If a group does not have a group expression (the "Group Expression" edit control is left blank), a reset condition occurs for every record of the composite data file. This is generally used for producing the detail lines of a report.

There is usually only one group in a report that has no group expression. Having more than one group without a group expression can cause undesirable results when outputting the report.

In "Figure 3.1" on page 70, the first subset is based on all records that have 'ENG' in the DEPT field. The DEPT field determines the difference between that subset and the one that contains 'ACT'.



```
                         Hours Report
Department:  ENG
   Location:  001
      Employee:  Jones, Peter
                    7/20          8.50
                    7/21          7.00
      Total for Jones, Peter     15.50
      Employee:  Smith, John
                    7/20          8.50
      Total for Smith, John       8.50
   Total for Location 001                 24.00
   Location:  002
      Employee:  Sanders, Bo
                    7/20          8.50
                    7/21          7.00
      Total for Sanders, Bo      15.50
   Total for Location 002                 15.50
Total for department ENG                             39.50
•
•
          Grand Total for all Departments:  65.5
```

Figure 3.2 Report Groups

The group expression for that group would then be "DBF->DEPT" (assuming the data file name was DBF). Notice that the group expression was not "DBF->DEPT='ENG'". This is because the group expression does not describe a specific subset, but the basis of the subset. When the evaluated group expression changes, a new subset has been reached and the actions of the group are performed.

The group is "grouped on" its group expression.

Header and Footer

A group can be delimited at the beginning and ending of the subset by outputting a section of text. These sections, or report areas (See), are associated with the group and are an integrated part of the action of the group.

The area at the beginning of the group (eg. "Department: ENG" in Figure 3.2) is called the group header, and the area at the end (eg. "Total for department ENG") is called the group footer.

A group's header and/or footer may contain zero, one, or more report areas -- which are sections of the report in which output objects may be placed (see and). The group header and footer are used to perform the actions of the group, that is, when a group reset condition occurs, the report areas associated with the header and footer are outputted.

The main difference between the report areas associated with the header and the footer is that:

- the values of the objects outputted in the header area are based on the composite record that caused the reset condition, and
- the values of the objects outputted the footer area are based on the composite record immediately prior to the composite record that caused the reset condition.

See Figure 3.3 for a visual representation of this concept. When a group is created, it automatically contains one header and one footer area.

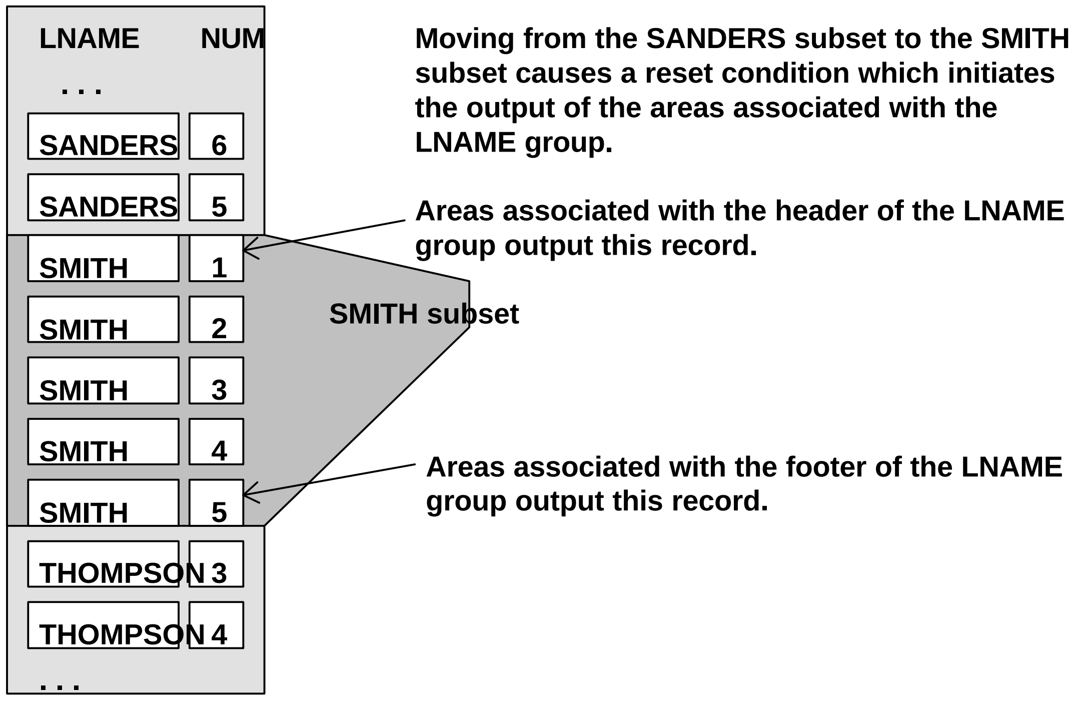

LNAME NUM
. . .
SANDERS 6
SANDERS 5
SMITH 1
SMITH 2
SMITH 3
SMITH 4
SMITH 5
THOMPSON 3
THOMPSON 4
. . .
Moving from the SANDERS subset to the SMITH subset causes a reset condition which initiates the output of the areas associated with the LNAME group.
Areas associated with the header of the LNAME group output this record.
SMITH subset
Areas associated with the footer of the LNAME group output this record.

Figure 3.3 Headers and footers

Creating Groups

A group is created by using the **GROUP | NEW** menu option. Each new group, by default, is created so that all of the previously created groups are between the new group's header and footer area(s). This default action creates new groups at the highest (outermost) level.

When a group is placed within another group, it is said to be nested.

As groups are created, new information windows are created to identify the groups and their header(s) and footer(s). The information windows may be hidden by de-selecting the **VIEW | INFO** Windows menu option. See the Report Design Screen in the Getting Started section and for information on the information windows.

Using the **GROUP | NEW** menu option invokes the "Group Settings" dialog (As shown below in Figure 3.4).

Name

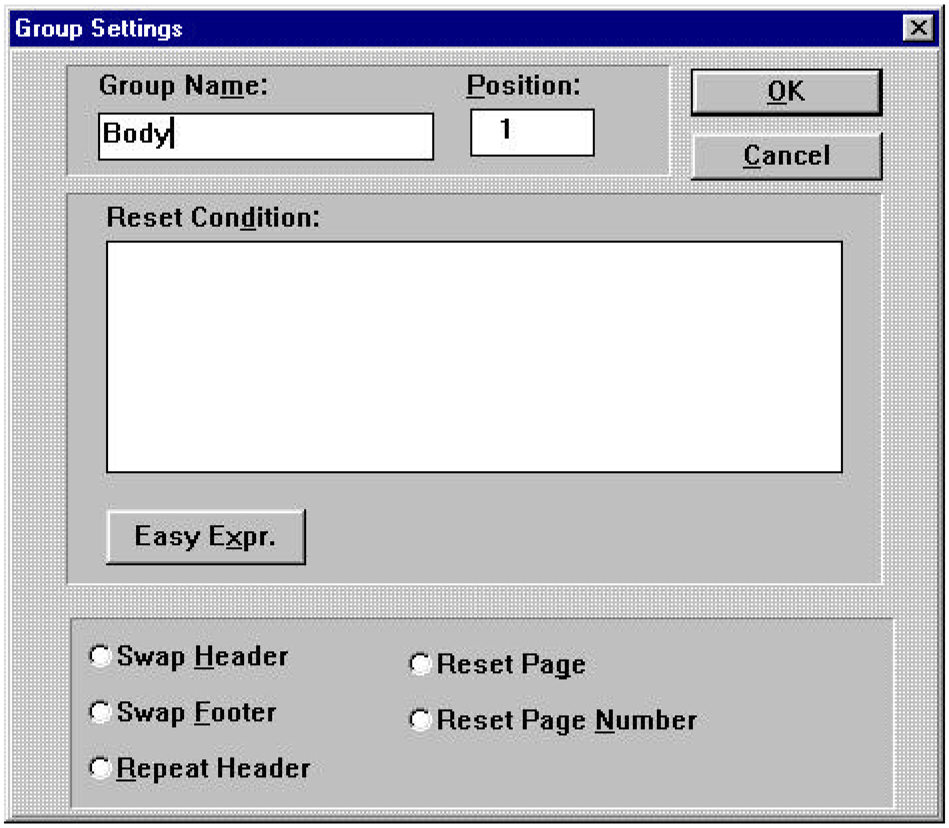
A group may have a descriptive name associated with it which helps the report designer identify the contents of the group. By default, CodeReporter uses "Body" for the first group, "Group 2" for the second group, "Group 3" for the third, etc.

This name is only a descriptive way of referring to the group, and may be changed or modified as desired. Usually this descriptive name reflects the group expression in some way.

Position

The "Position" setting determines where the group is placed in relation to other groups. The innermost group in the report (the smallest subset) is group 1. New groups are by default made the outermost group, and have the highest group number.

Changing this number moves the group into the desired position and shuffles the other groups appropriately.


Group Settings

Group Name: Body

Position: 1

Figure 3.4 Group Settings Dialog

Group Options

CodeReporter provides some special group handling that provides ways to customize the look of the reports.

Swap Header and Swap Footer

When a group encounters a reset condition and the "Swap Header" radio button is set for the group, a new page is generated and the group's header is outputted in the position of, and instead of, the page header area. The page header is not outputted on that page.

This is an advanced concept that may be difficult to understand initially. It may be necessary to take some time to examine Figure 3.5 and the text

listed below before the concept is understood. Also see the Statement of Accounts tutorial in the CodeReporter printed documentation.

In a similar manner, when a group encounters a reset condition and the "Swap Footer" check box is set for the group, the rest of the current page is skipped and the group footer is used in the position of, and instead of, the page footer area. The page footer is not outputted for that page.

A swapped header is generally used to display different information or information in a different format than the page header area.

The main purpose of a swapped footer or header is to suppress the page header/footer, generate a new page, and output the group's area whenever a group reset condition occurs.   This is different than the area suppression condition available for the page header/footer areas (See), because the area is suppressed on a change of value instead of a logical condition.

Figure 3.5 shows a sample report that uses a swapped header and a swapped footer. The page header used on page 2 and 3 is not appropriate for page 1, since the summary provided in the page header is already found in the group header. Every new customer invoice begins upon a new page, and the customer's full information is listed.

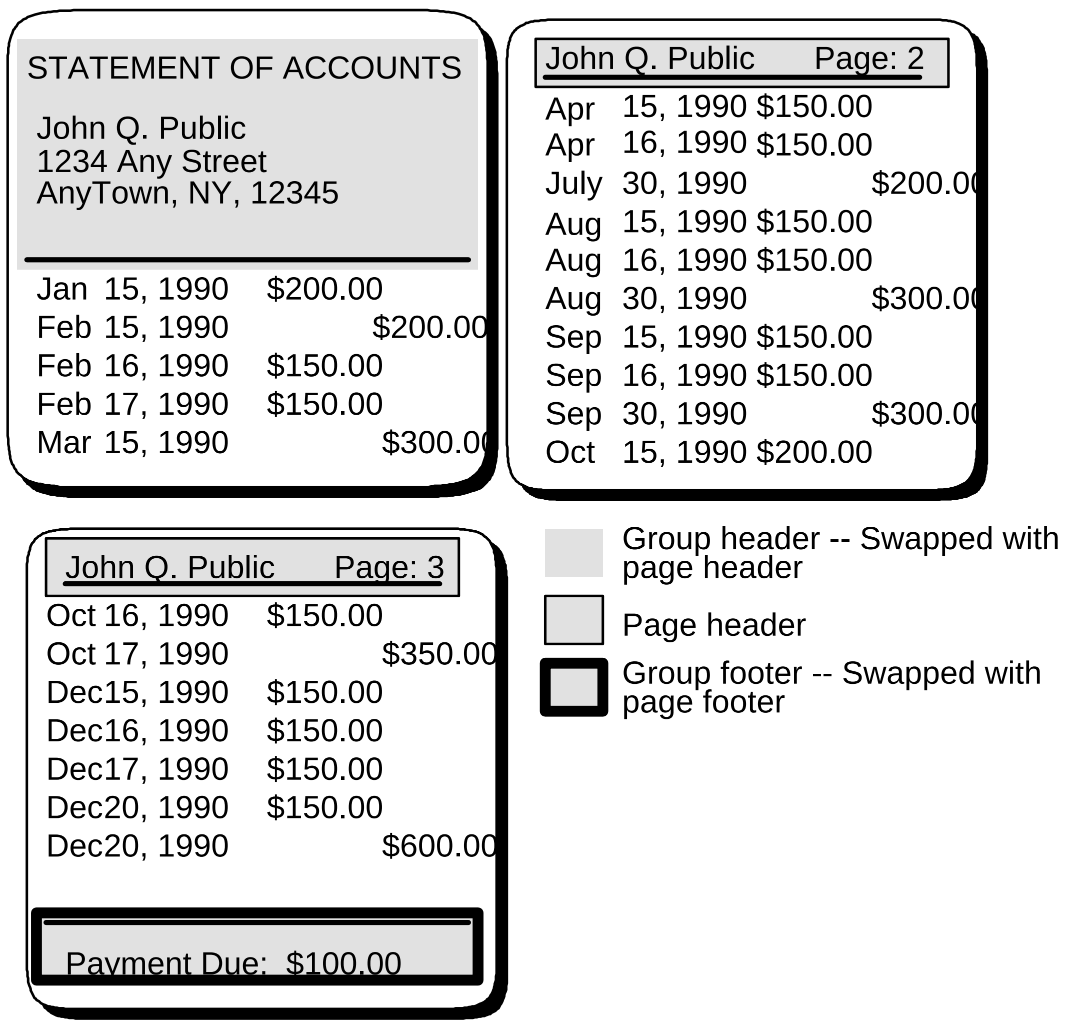


Figure 3.5 Swap Header/Footer Example

If a simple page header were used instead of a swapped group header, the unbordered grayed portion (as page header) would be printed on each page of the invoice -- giving very little differentiation between the cover page and the remaining pages.

Notice that Figure 3.5 does not have a page footer area. In the case of a non-existent page header/footer, the group header/footer is outputted where the page header/footer would have been.

Repeat Header

When a subset of the composite data file (represented by a group) cannot fit on the rest of the current page, having the "Repeat Header" radio button selected causes CodeReporter to output the group's header area at the top of the page (below the page header), even though a reset condition has not occurred.

This is useful for maintaining column titles across page boundaries.

Reset Page

The "Reset Page" radio button generates a new page when a group reset condition occurs. The actual sequence of events are: the group's footer, as well as the footers of all inner groups, are outputted and a new page is generated. The next page's header and the group's header are both processed at the beginning of the new page.

Reset Page Number

The "Reset Page Number" functions exactly the same as the "Reset Page" option, except that in addition, the new page's page number is reset to '1'.

Hard Reset Page

The behavior of reset page (including the page reset for swap header and reset page number) is affected by the Hard Reset Page setting in the "Report Preferences" dialog. See .

Modifying a Group

The "Group Settings" dialog box is used to change the settings for an existing group. Once a group is created, this dialog may be invoked either by selecting **GROUP | MODIFY** or by right clicking on the group's information window.   See   for information on the settings in the "Group Settings" dialog.

Deleting a Group

The selected group, and all areas and output objects associated with it may be deleted by selecting the **GROUP | DELETE** menu option. Deleting a group does not affect other groups in the report.

Selecting a Group

Clicking the left mouse button anywhere within an area of the group -- or clicking upon an object within an area -- selects that group.

Alternately, the Ctrl- Up and Down Arrow keys may be used to select another group.

Reset Conditions and Group Printing

When a group expression changes, a reset condition occurs. This reset causes that group's footer to be processed, its totals to be reset, and finally the group's header to be processed with the next composite record. In a multi-group report, a reset condition resets the group whose expression changed, as well as all lower level groups (groups between that group's header and footer).

Figure 3.6, in your CodeReporter manual shows the output of a report with groups on Year, Month, and Day. When the 'Month' group is reset the 'Day' group is automatically reset.

Groups are only outputted when they are reset. In Figure 3.6, in your CodeReporter manual, the second record is exactly the same as the first. Since each group has a reset condition -- and none of them were reset -- the second record is not outputted. However, when the third *record (FEB 20, 1992*) is encountered, the 'Month' group is reset. This automatically resets any inner group ('Day', in this case).

When a reset condition occurs, the footers are outputted, in order beginning with the innermost group, until the footer of the original group that caused the reset condition is outputted.

Figure 3.6, in your CodeReporter manual, demonstrates this by first outputting the footer of the 'Day' group and then the footer of the 'Month' group. Notice that the 'Year' group footer is not outputted at this point, because the 'Year' group has not been reset Also notice that the date value displayed in the 'Day' and 'Month' footers is that of *JAN 20, 1992*, the second record, and not *FEB 20, 1992*, the record that caused the reset condition. This illustrates the second rule mentioned in Group Header and Footers, above: footers are outputted with the values from the record immediately prior to the record that caused the reset condition.

Once the innermost group is processed, the next record is used. In Figure 3.6 in your CodeReporter manual, the *FEB 21, 1992* record resets the 'Day' group. Since only the 'Day' group is reset, only the footer for the 'Day' group is outputted -- again with the values of the previous record -- and then its header is outputted with the values of the new record. The processing of all the records is complete, so the report is ended by outputting the footers of all the report's groups. Therefore, in brief, when a group is reset:

- The footers of all the groups are outputted first, before any of the headers are outputted. The footers are outputted in order, beginning with the innermost group, until the original group that caused the reset condition is output. The value of the record immediately prior to the record that caused the reset condition is used to output the group footers.
- If any of the groups reset have the "Reset Page" option set, the output of the report begins on a new page. (See , above and)
- Once the footers have been processed, the headers are processed inwards beginning with the group that was reset using the value of the record that caused the reset condition.

Groups outside a group that encounters a reset condition are not processed. In this example, the 'Year' group is not reset until its own group expression has changed.

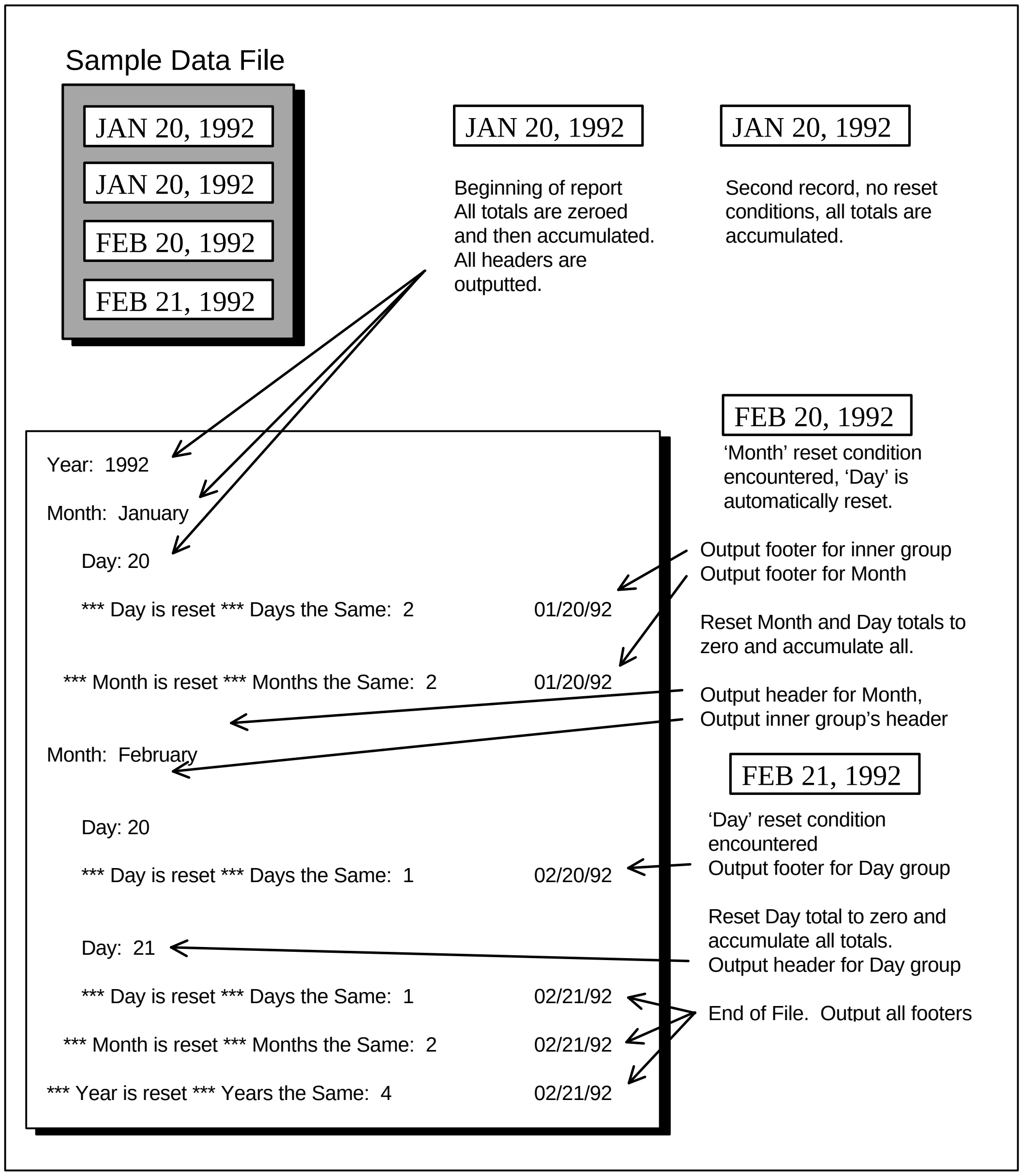
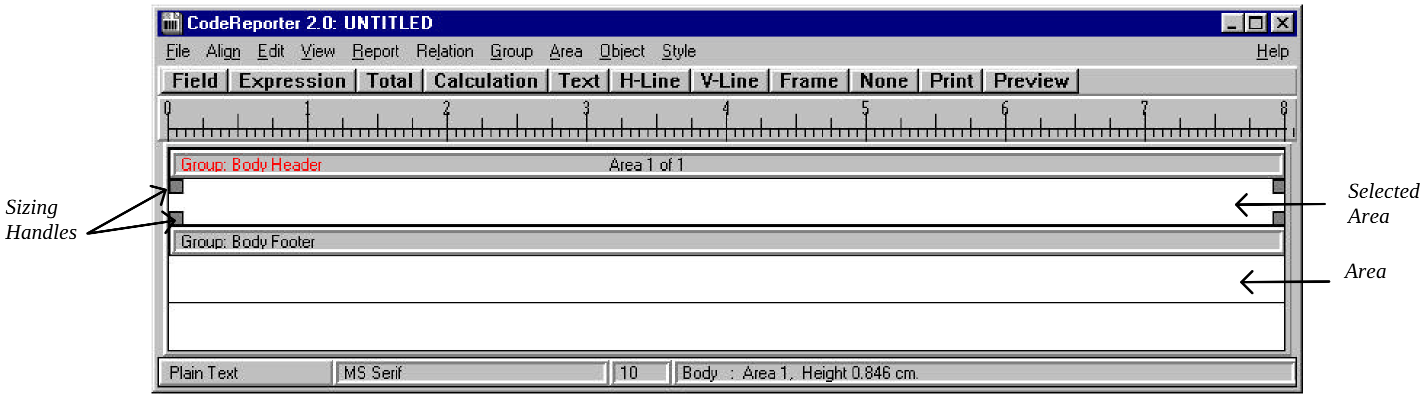

Sample Data File
JAN 20, 1992
JAN 20, 1992
FEB 20, 1992
FEB 21, 1992
JAN 20, 1992
Beginning of report All totals are zeroed and then accumulated. All headers are outputted.
JAN 20, 1992
Second record, no reset conditions, all totals are accumulated.
FEB 20, 1992
'Month' reset condition encountered, 'Day' is automatically reset.
Output footer for inner group
Output footer for Month
Reset Month and Day totals to zero and accumulate all.
Output header for Month,
Output inner group's header
FEB 21, 1992
'Day' reset condition encountered
Output footer for Day group
Reset Day total to zero and accumulate all totals.
Output header for Day group
End of File. Output all footers
Year: 1992
Month: January
Day: 20
*** Day is reset *** Days the Same: 2 01/20/92
*** Month is reset *** Months the Same: 2 01/20/92
Month: February
Day: 20
*** Day is reset *** Days the Same: 1 02/20/92
Day: 21
*** Day is reset *** Days the Same: 1 02/21/92
*** Month is reset *** Months the Same: 2 02/21/92
*** Year is reset *** Years the Same: 4 02/21/92

Figure 3.6 Group Reset Conditions

4. Areas

An area is a spot in the report where output objects may be placed. The page header, page footer, and the main body of the report all are considered areas where output objects may be placed. Except for special areas (page header/footer, title/summary), areas are associated with groups, which dictate when the output objects within the area may be outputted. When a group encounters a reset condition, the areas associated with the group are outputted.

When a group is created, it has two default areas associated with it: the group header and the group footer -- except for the default "BODY" group which only has a group header. These areas may be used as defaults for the group's report areas, or they may be sized, deleted, or suppressed as needed by the report.


CodeReporter 2.0: UNTITLED
File Align Edit View Report Relation Group Area Object Style Help
Field Expression Total Calculation Text H-Line V-Line Frame None Print Preview
0 1 2 3 4 5 6 7 8
Group: Body Header
Area 1 of 1
Group: Body Footer
Sizing Handles
Selected Area
Area

Plain Text | MS Serif | 10 | Body : Area 1, Height 0.846 cm.

Figure 4.1 Area and Sizing Handles

Areas are very simple to use, yet very flexible. In addition to grouping output objects, a set of mutually exclusive suppressed areas may serve to change the layout of the report. A group may have several header areas and/or several footer areas which may be outputted at different points within the report -- depending upon the contents of the report. In addition, an area can be configured so that it may span a page break.

Selecting an Area

Clicking the left mouse button anywhere within an area -- or clicking upon an object within an area -- selects that area. Alternately, the Ctrl- Up and Down Arrow keys may be used to select another area. Selecting an area in this manner also selects the first output object added to the area. Once an area is selected, it may be modified, deleted, or an additional area for the current group may be created.

Creating an Area

A header or footer area may be created for the selected group by using the **AREA | NEW HEADER**, or **AREA | NEW FOOTER** menu options. To create a Page Header, Page Footer, Title or Summary area, choose the appropriate menu option from the Area menu. For more information on these areas, see the appropriate sections below.

Deleting an Area

The selected area may be deleted by choosing the **AREA | DELETE AREA** menu option. Deleting an area also deletes all output objects within the area.

All areas in a group header or footer may be deleted, however the information window for the last group area will remain displayed if the **VIEW | INFO WINDOWS** menu option is set.

Modifying an Area

An area has three characteristics that may be modified: its size, whether it is suppressed, and whether it spans a page break.

These options may be set through the "Modify Area" dialog box , which is invoked for the selected area with the **AREA | MODIFY AREA** menu option, or by right clicking within the area.

Sizing an Area

The vertical size of an area may be changed by using the mouse or the "Modify Area" dialog box (Figure 4.2). The horizontal size of areas may not be individually changed. The horizontal size of the report is changed by modifying the entire report's page size or margins.

Expanding the vertical size of an area does not affect the position of the output objects within the group. Reducing the size of an area, however, may cause some output objects to be deleted if, at their current position, they no longer completely fit within the new, smaller-sized area.

The desired size for the area may be set manually by entering the new height in the "Modify Area" dialog's "Height" edit control. The units of measurement are those set with **REPORT | PREFERENCES**.

Using the mouse

The mouse may also be used to change the vertical size of a selected area by dragging one of the selected area's size handles to the desired position with the left mouse button. See Figure 4.1.

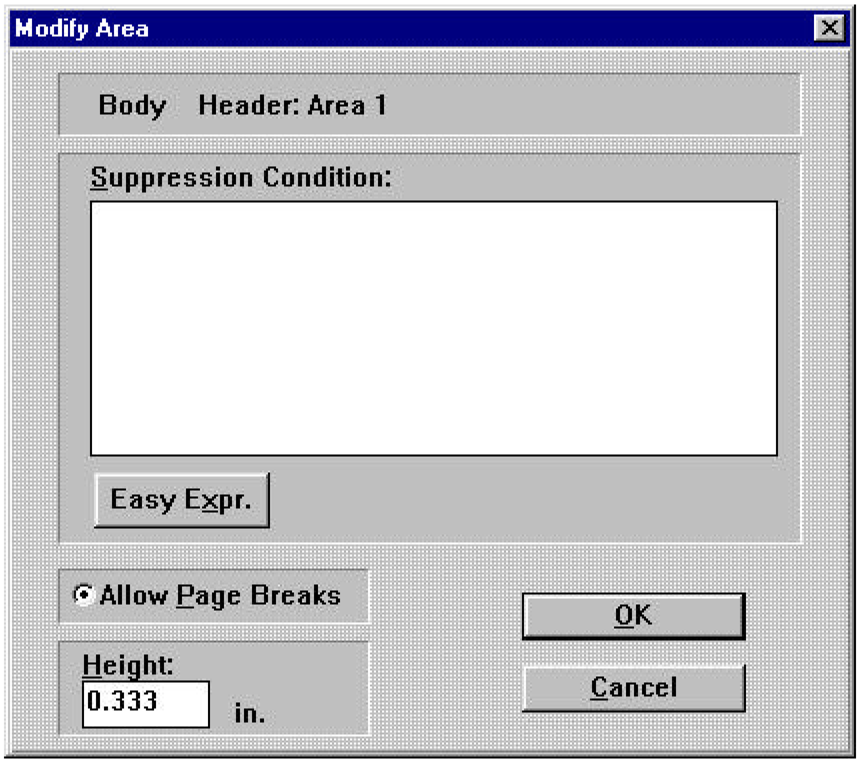


Figure 4.2 Modify Area Dialog

Allow Page Breaks

When the "Allow Page Breaks" option is enabled (the default) for an area, the area may span a page break. If a page break (bottom of the page minus the page footer) would fall within an area, the area is divided between the two pages. CodeReporter outputs as much of the area as it can fit on the page -- without dividing any object(s).

Frames, lines, and word wrapped objects that span most of the height of an area may make this setting useless. CodeReporter will not divide an object between two pages, so even if the "Allow Page Breaks" radio button is set, the whole area may be placed on a following page regardless.

If an output area must not be divided between two pages, de-select the "Allow Page Breaks" radio button.

Suppressing an Area

A logical dBASE expression may be associated with an area to determine whether or not the area should be outputted. If the dBASE expression entered in the "Suppression Condition" edit control evaluates to a true value, the area (and all the objects within it) is ignored.

This feature may be used to vary the outputted area's appearance or contents depending upon the data within the report. This is done by creating two or more areas within the same part of the group (header or footer), adding the different information to the different area(s), and deciding when to suppress which area(s).

For example, if a numeric field value is negative, it may be appropriate to display it in a red font, instead of a black font. The objects in the two versions of the area would be identical except for that in one case the red font would be used. Each area would contain a suppression condition.

Black font (Suppress for negative values):	DBF->FIELD < 0
Red font (Suppress for positive values):	DBF->FIELD >= 0

An example of suppressing an area may be found at the "end of this chapter" on page 85.

Page Header and Page Footer Areas

The Page Header area(s) are outputted at the top of every page in the report and are generally used for text objects containing a brief name of the report, a date, and/or a page number.

The Page Footer area(s) are outputted at the bottom of every page in the report and are generally used for text objects, running totals, page totals, and page numbers.

If used, these areas appear on every page of the report unless they are suppressed, or if one of the groups in the report has the Swap Header or Swap Footer option enabled. These areas are created with **THE AREA | NEW PAGE FOOTER** and **AREA | NEW PAGE HEADER** menu options.

Title and Summary Areas

The Title area(s) are the first report areas to be displayed on the first page of the report. The Title area(s) are even outputted above the page header area(s). Only one title area is outputted per report. As such, the Title area(s) are generally used to display descriptive information unique to the report, such as the report name. The title area may be thought of as a cover page to the report.

Page Break After

If a page break is desired after the Title area, select the "Page Break after Title" check box, found in the "Report Preferences" dialog (See the).

The Summary area(s) are the last report areas to be displayed before the page footer on the last page of the report. Only one summary area is outputted per report. The Summary is usually used to present final comments, and/or numerical data which summarizes the entire report.

These areas are created with the **AREA | NEW TITLE AREA** and **AREA | NEW SUMMARY AREA** menu options.

Example

This example displays the contents of the NUMBERS data file which contains both positive and negative numbers. By using two groups, suppression conditions, and a different font, the negative values found in the NUMBERS data file will be displayed in red, while the positive numbers will be displayed in black.

This example illustrates some of the skills discussed in this chapter, and incorporates some skills found in the Objects and Styles chapters. For further information on objects and styles, see their respective chapters.

Open a New File

Once CodeReporter is running, choose the FILE | NEW menu option and select the NUMBERS.DBF data file from the .\EXAMPLES subdirectory. CodeReporter creates one group, "Body", with one header area.

Second Header

A second group header to display the negative numbers is created by selecting the **AREA | NEW HEADER AREA**. Once the second area is created (and automatically selected), choose the **AREA | MODIFY AREA** menu option to invoke the "Modify Area" dialog and type the following expression into the "Suppression Expression" edit control:

```
            NUMBERS->NUMERIC >= 0
```

Whenever the value of the NUMERIC field is greater than or equal to zero (a positive number) the second area will not be outputted. This is necessary because the second area is used to output only the negative numbers -- positive numbers are outputted using the first area. Select the "OK" button when finished.

Modifying the Area

The first area, by default, has no suppression condition and would display for every value -- positive or negative -- in the NUMBERS data file. To set a suppression condition for the first area, invoke the "Modify Area" dialog while the first area is selected, or right click while the mouse pointer is above the first area.

Type the following expression into the "Suppression Expression" edit control:

```
            NUMBERS->NUMERIC < 0
```

Whenever NUMERIC is a negative number, the first group -- the one used to output positive numbers -- is suppressed.

Adding the Fields

Use the mouse to select the "Fields" button on the button bar. Single click on the NUMERIC field in the popup list to select it.

Position the mouse over the first area (the mouse cursor changes to the Field cursor) and press the left mouse button to position the field (See the for more information on placing and moving output objects).

Select the NUMERIC field again in the popup list and place it in the second area. When finished, select the "Done" button on the "Field Objects" list box.

Load a Style Sheet

Use the **STYLE | LOAD STYLE SHEET** menu option and select the TUTORIAL.CRS style sheet. When CodeReporter prompts to override the current style, choose the "Yes" button. For more information on creating styles, saving and retrieving a style sheet, see the Styles chapter.

Selecting a Style

Select the NUMERIC field in the "negative" area by left clicking on it. Select the "Style" button from the status bar and double click on the "Red" style. This changes the style for all selected output objects. Since the NUMERIC field in the negative area is selected, its style is set to Red.

Preview the Report

View the completed report by selecting the **FILE | PRINT PREVIEW** menu option. Notice that the report displays all of the values in the NUMBERS data file, but that now, since two areas and mutually exclusive suppression conditions are used, the negative numbers are outputted in a red typeface.

5. Output Objects

The term "output object" describes the collection of report elements that are used to convey the information of the report to its reader. Output objects are the "guts" of the report -- the text, fields, totals, graphics, lines, frames, etc. -- that are actually put on paper when the report is printed. Everything outputted in the report must be done through an output object.

Static vs. Dynamic

Some output objects output the same information throughout the life of the report, while others change with every composite record. The output objects that stay the same -- such as lines, descriptive text, company logos, etc. -- are called static output objects. The value and settings for these report elements are set at report design time and do not change when the report is run.

The values for other output objects -- such as fields and totals -- can change from each different run of the report or indeed from one composite record to another. These constantly changing report elements are called dynamic output objects. The values for dynamic output objects reflect the information in the composite data file(s) -- which can change at any time.

The different types of static and dynamic output objects are discussed in-depth later in this chapter.

Creating Output Objects

An object has a specific type associated with it; a field, a line, some text, etc. The type of an object is specified as it is created and stays with it throughout its life. Output objects may be placed in any report area, including the page header/footer, the title/summary, and a group's header/footer. (See the Groups and Areas chapters.)

Objects created in an area remain in the area unless they are moved using the cut and paste procedure described under “Moving Objects” on page 90, below.

Insert Mode

When an object type has been selected for addition to the report, CodeReporter is put into insert mode. The status bar indicates this by updating the status bar with the words "Insertion Mode:" followed by the object type being inserted. The mouse cursor, which is changed to reflect the appropriate object type (see Appendix C), may be used to indicate the initial position of the new object. Clicking with the left mouse button places an object of the selected type.

The exact process of creating an object varies from type to type. Some objects require an initial value, while others use default values. Listed below are the general procedures for creating an output object. For a detailed description of creating a specific object, see the object explanation below.

Using the Button Bar

The button bar is the easiest way to add different types of output objects. Simply click on the object type's button and CodeReporter is put into insertion mode for the specified type of object. All object types except graphic objects are included in the button bar. Graphics may only be added using the menu.

Using the Menu

The **OBJECT** menu option contains a list of all the output objects. Select the appropriate menu option to put CodeReporter into insertion mode for the specified type of object.

Creating Multiple Objects

CodeReporter continues to be in insertion mode for the specific object type until another object type is selected. That is, once an object type is selected, multiple objects of the same type may be added without having to re-select the object type.

CodeReporter is moved out of insertion mode by using the "None" button, the **OBJECT | NONE** menu option, or pressing the Escape key.

Objects within Objects

An output object may be placed so that it completely surrounds another smaller output object. When this occurs, the smaller object is considered "within" the larger object, and it may be treated as if it is a part of the larger object.

An action performed with the larger container object affects all the object(s) within it as well. For example, a frame object may be placed around several field objects to provide a unique look. If a new style is selected for the frame object, all objects within it will also use the new style. If the container object is moved, all of the inner contained objects are also moved. If the frame object is then deleted, all the field objects within it are also deleted.

Selecting Objects

A "selected" object is one for which modifications, deletions, object movement, etc. occur. In order to perform these actions an object must be selected using one of the procedures described below. When an object is selected, it is displayed in the report design screen in a red font and having sizing handles.

Selecting an output object that contains other objects multiply selects all objects within the container object.

Mouse

An object may be selected with the mouse by clicking upon it once. In addition, the group and area for the object are also selected.

Keyboard

The Tab key may be used to select output objects in the selected area. Repeatedly pressing the Tab key cycles through all objects added to the area. The Shift-Tab key cycles backwards through the objects in the area.

Multiple Selection

It is often necessary to perform an action (such as changing a style) upon several output objects. Multiple output objects may be "selected" and when an action is performed once, it is applied to all selected output objects.

When more than one object is selected, the additional objects are displayed in a red font to indicate that they are selected. Only the first object selected, however, retains its given sizing handles.

The actions for which multiple selection apply are:

- Moving objects,
- Deleting objects,
- Cut, Copy, and Pasting objects,
- Selecting Styles, and
- Using any Alignment menu option.

Mouse

Multiple output objects may be selected by holding down the Shift key and clicking on the objects to be selected. Any object, in any area, may be selected in this manner. Objects may be de-selected by clicking on them a second time while the shift key is held down.

Keyboard

Multiple output objects are selected using the keyboard by holding the Ctrl key down and pressing the Tab key. Multiple objects in different areas may not be selected using the keyboard.

Deleting Objects

The selected output object or multiply selected objects may be permanently removed from the report by pressing the Delete key, or by choosing the **OBJECT | DELETE** menu option.

When an output object which contains other objects is deleted, all contained objects are also deleted.

Once an object is deleted it cannot be recovered. If an object is deleted by accident, it must be recreated from scratch.

Moving Objects

An output object may be moved to a new location within its area by selecting the object with the mouse and dragging it to the desired position. Multiply selected output objects in one or more report areas may also be positioned by pressing the Shift key while dragging the objects.

Precise positioning may also be obtained by using the X and Y edit controls of the "Object Settings" dialog.

Output objects may not be dragged or positioned outside of their area. Multiply selected objects in two or more report areas have their movement constrained by the smallest report area.

Sensitivity

CodeReporter can precisely position output objects to a single Windows Device Unit. This is a very small and very precise unit of measurement -- often too small to line objects up concisely with the mouse.

The **ALIGN | SENSITIVITY** menu option invokes the "Grid Sensitivity" dialog. In this dialog, the Horizontal Sensitivity and Vertical Sensitivity may be set to a movement distance (in the currently selected unit of measurement). This defines an increment with which objects are moved. The larger the increment, the fewer number of possible coordinates an object can occupy. With a larger sensitivity setting, it is easier to quickly position objects.

The Sensitivity setting only affects new objects being placed and objects being moved. Output objects that are already placed in the report are not affected by this setting.

Alignment

Multiple objects may be lined up with one another quickly using the **ALIGN** menu options. The first selected output object is used as the guide for the movement of all subsequently selected output objects.

Left and Right

Alignment along the left or right edge of the first selected object is done by choosing the **ALIGN | LEFT OR ALIGN | RIGHT** menu option. Multiply selected objects in one or more report areas may be aligned to the right or left of the first selected object.

Center

The **ALIGN | CENTER** menu option horizontally centers the currently selected object within the report area. If multiple objects are selected, the center of the first selected object is used as the center point upon which the other objects are centered. The first object is not moved.

Top and Bottom

Alignment along the top or bottom edge of the first selected object is done by choosing the **ALIGN | TOP OR ALIGN | BOTTOM** menu option. Only objects selected in the same report area may be aligned along the top or bottom.

Space Horizontal - Vertical

Three or more objects may be moved so that there is an equal amount of space between all objects using the **ALIGN | SPACE HORIZONTALLY** and **ALIGN | SPACE VERTICALLY** menu options. Using the first and last selected output objects as the end points, CodeReporter moves all of the interior objects horizontally or vertically depending upon the option selected.

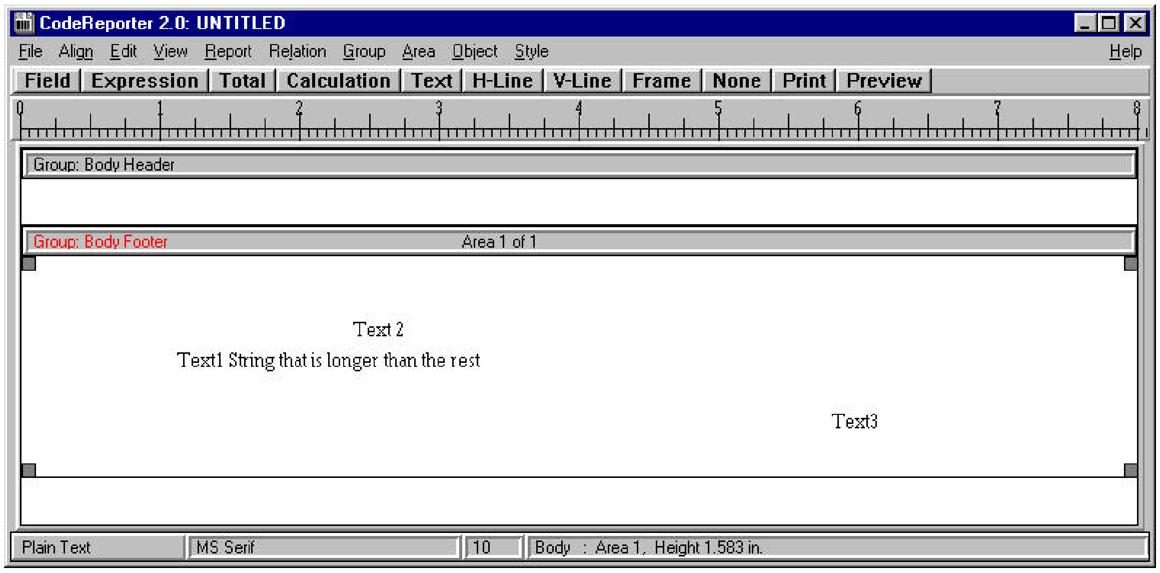
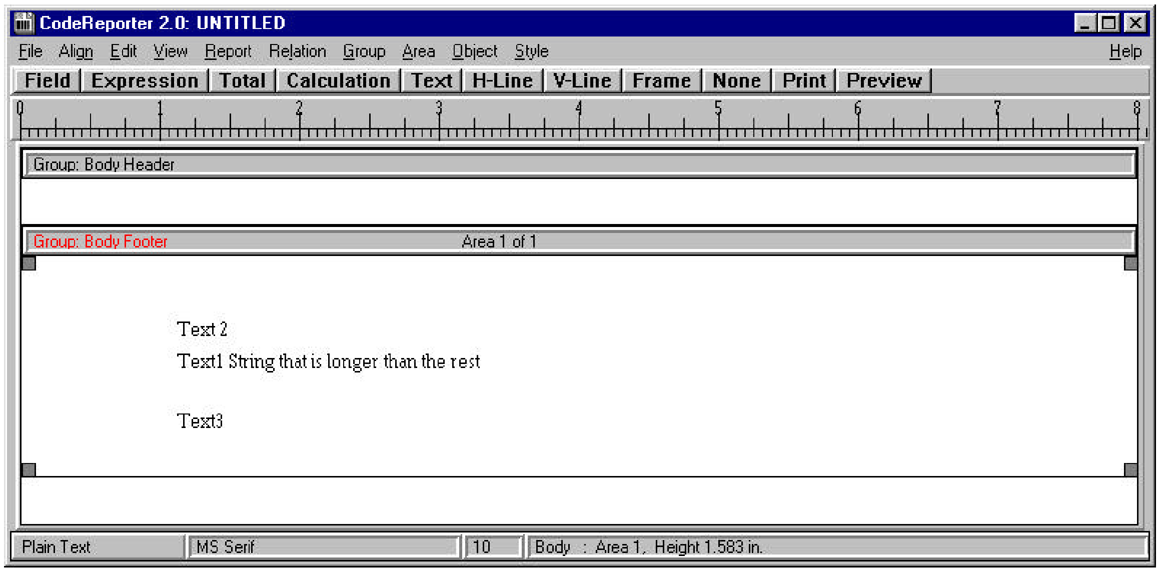

CodeReporter 2.0: UNTITLED
File Align Edit View Report Relation Group Area Object Style Help
Field Expression Total Calculation Text H-Line V-Line Frame None Print Preview
0 1 2 3 4 5 6 7 8
Group: Body Header
Group: Body Footer
Area 1 of 1
Text 2
Text1 String that is longer than the rest
Text3
Plain Text
MS Serif
10
Body : Area 1, Height 1.583 in.

Figure 5.1 Initial Layout

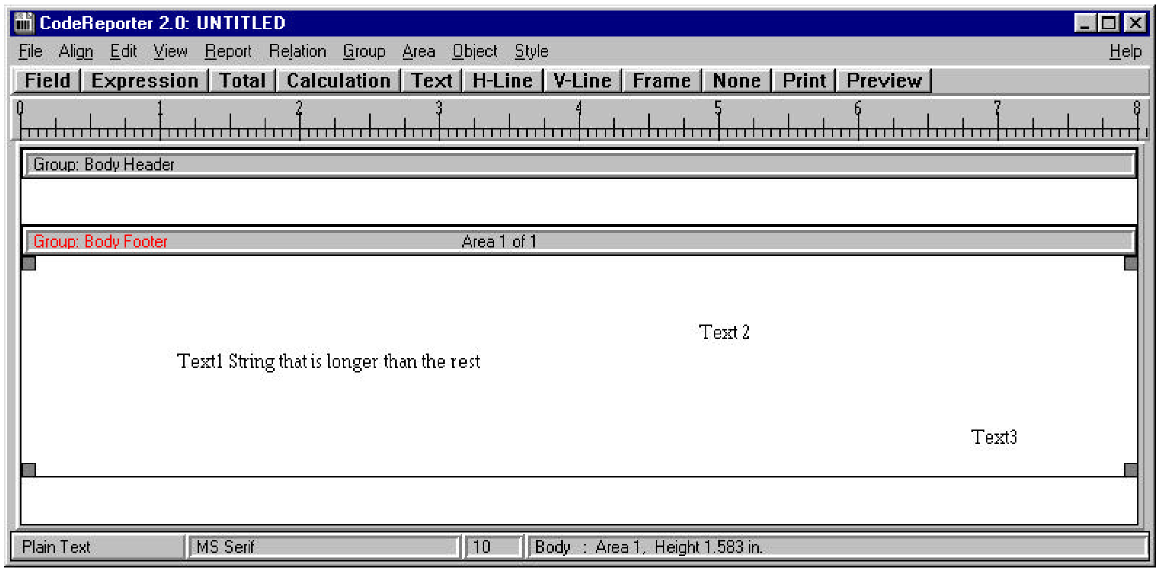
Figure 5.1 shows some randomly placed output objects. Given that they are then selected in order (i.e. Text1, Text2 and then Text3), **ALIGN | LEFT** moves the output object to the positions shown in figure 5.2.


CodeReporter 2.0: UNTITLED
File Align Edit View Report Relation Group Area Object Style Help
Field Expression Total Calculation Text H-Line V-Line Frame None Print Preview
Group: Body Header
Group: Body Footer
Area 1 of 1
Plain Text
MS Serif
10
Body : Area 1, Height 1.583 in.

Text 2
Text1 String that is longer than the rest
Text3

Figure 5.2 Align Left

Using the positions in Figure 5.1 again, selecting the objects in order, and choosing **Align | Space Horizontally**, the output objects are moved to the positions seen in Figure 5.3. Notice that the first and last selected objects (Text1 and Text3 respectively), do not move—but the middle object (Text2) is spaced horizontally, equidistant from both ends.


CodeReporter 2.0: UNTITLED
File Align Edit View Report Relation Group Area Object Style Help
Field Expression Total Calculation Text H-Line V-Line Frame None Print Preview
Group: Body Header
Group: Body Footer
Area 1 of 1
Plain Text
MS Serif
10
Body : Area 1, Height 1.583 in.

Text 2
Text1 String that is longer than the rest
Text3

Figure 5.3 Align Horizontal

To Top - Bottom

When two output objects occupy the same space, one object is displayed and outputted over the top of the other. CodeReporter determines which object belongs on top of the other by the order in which the objects were created.

That is, older objects are always placed below newer objects. The **OBJECT | TO TOP AND OBJECT | TO BOTTOM** menu options change the ordering for a selected object so that it may be placed on top of or beneath another object.

Cut, Copy and Paste

Objects may be moved or copied to other report areas using the **EDIT | CUT, EDIT | COPY**, and **EDIT | PASTE** menu options. Objects that are cut or copied are placed in the Windows clipboard until such time as they are needed again.

Invoking the **EDIT | PASTE** menu option changes the mouse cursor to a paste icon. Position the cursor to the new position for the output object and click the left mouse button.

Multiple objects may be cut, copied and pasted to and from the clipboard, however, if the destination report area is too small to contain the output objects, only the ones that can fit will be pasted.

Pasting from Other Applications

CodeReporter uses the Windows clipboard in the cut, copy and paste process, so if another application has placed something in the clipboard, it may be pasted into a report as a text object.

Graphic images from other applications may be pasted into a CodeReporter report area as a static graphic objects.

Other applications, through their paste operation, may retrieve the text and static graphics for CodeReporter output objects. Since CodeReporter output objects are unique to CodeReporter, other applications may only access the text and static graphic objects for output objects and not the actual objects.

Modifying Objects

Often it may be desirable to change some aspect of an output object, such as its size, its style, its justification, the number of decimals used, etc. These changes may be made for the selected object through the "Object Settings" dialog box.

This dialog is invoked from the selected object's Object Menu.

Object Menu

Modifications specific to a particular output object are invoked from a popup menu tied to the object. This menu contains three to four menu options, depending upon the object's type. An Object Menu may be invoked in two ways:

- Click and hold on an object with the right mouse button, or
- Select an object and press the Enter key.

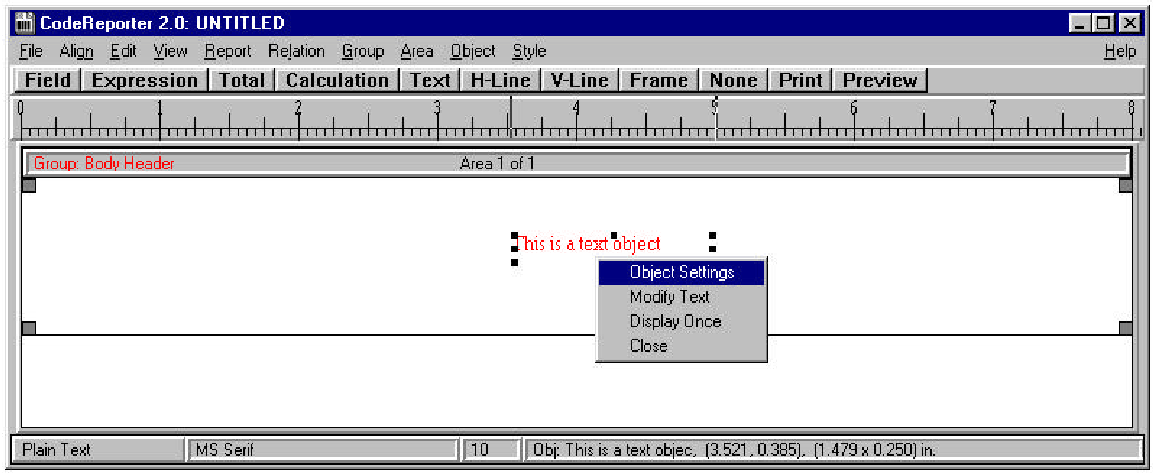

CodeReporter 2.0: UNTITLED
File Align Edit View Report Relation Group Area Object Style Help
Field Expression Total Calculation Text H-Line V-Line Frame None Print Preview
Group: Body Header
Area 1 of 1
This is a text object
Object Settings
Modify Text
Display Once
Close
Plain Text
MS Serif
10
Obj: This is a text objec, (3.521, 0.385), (1.479 x 0.250) in.

Figure 5.4 Object Menu

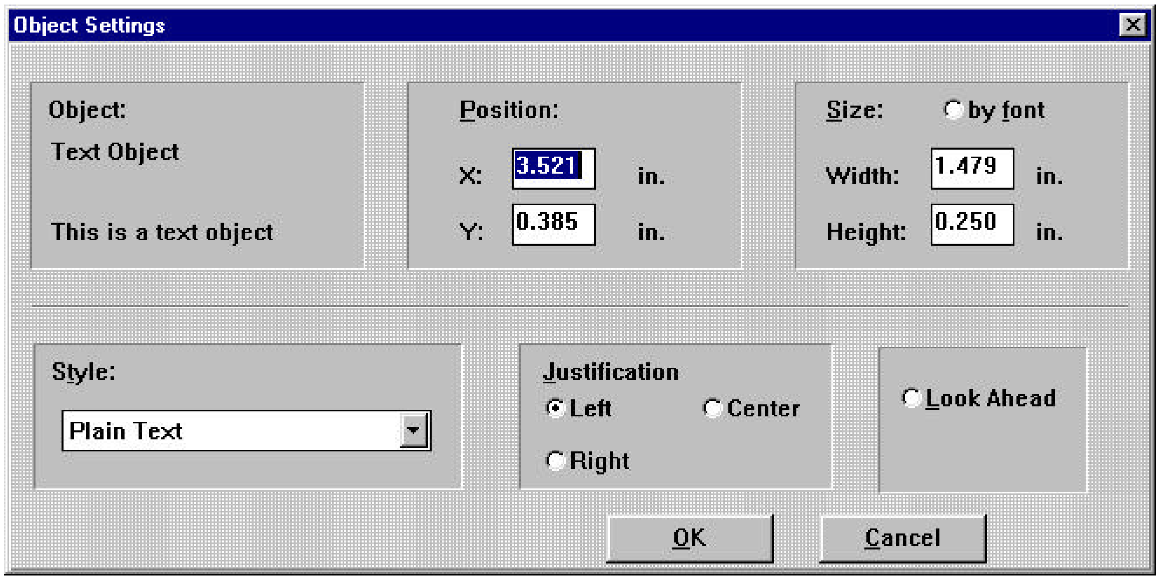
Object Settings

The "Object Settings" dialog is the primary tool for changing the attributes of an output object. This dialog is invoked from the **Object Settings** object menu item.

The "Object Settings" dialog is divided into sections that provide options for modifying the output object. Since some options are unique to certain output object types, some sections may not appear for certain output objects.

Description

The upper left section of the "Object Settings" dialog contains a description of the object being modified. It lists the object type as well as the name, expression, or text used by the object.


Object Settings

Object:
Text Object
This is a text object

Position:
X: 3.521 in.
Y: 0.385 in.

Size: by font
Width: 1.479 in.
Height: 0.250 in.

Style: Plain Text

Justification
[✓ Left] Center
Right

Look Ahead

Figure 5.5 Object Settings Dialog

Position

The "Position" section of the dialog contains the current Horizontal ("X") and Vertical ("Y") coordinates of the object's upper left corner. These coordinates are in the units of measurement selected in the "Report Preferences" dialog. Changing the X and Y coordinates of an object changes the position of the output object in the design screen. See Moving Objects, above, for information on moving an object with the mouse.

An object may not be positioned so that it is completely outside of its area.

Size

The "Size" section of the dialog lists the current width and height of the selected output object. These values are, by default, in the units of measurement selected in the "Report Preferences" dialog. These values may be changed to select a precise size for the output object. See Sizing, below, for information on sizing an output object with the mouse.

When the "By Font" radio button is selected, the values in the "Height" and "Width" edit controls are converted to approximate character units. The size of the edit control can be changed so that it is *n* characters wide, and *m* lines tall. These units are in average character widths, so setting the size of the object when proportionally spaced fonts are used may not be completely accurate.

Style

The style (typeface, color, etc.) of the output object may be changed using the "Style" drop-down list box. Only styles that have previously been created are in this list. For more information on styles and creating styles, see the Styles chapter.

Justification

The text for output objects may be justified. That is, the text outputted for the object may be placed on the left, right or center of the object.

Left	Right	Center
Justified	Justified	Justified
Object	Object	Object

By default, all output objects are left justified, except for numeric output objects, which are right justified. The justification setting affects the following object types: Fields, Static Text, Expressions, Totals, and Calculations.

The TRIM() function is very important when justifying dBASE expressions containing data file fields. Since data file fields contain a fixed length (padded with spaces if it is not filled with data), justification produces little if any results. An output object of 10 characters, for example, still contains

10 characters no matter how it is justified -- printing 10 characters in a 10 character space is simultaneously left, right, and center justified.

If an expression object, which evaluated to a 10 character result: "SHOES ", were center justified, it would appear almost the same as if it were left justified, since the trailing spaces (even if they are proportional) are taken into account. If the field were trimmed, "SHOES " would be converted to "SHOES" and then the center justification would be visually correct.

Field output objects are automatically trimmed.

Proportionally spaced fonts may be justified without using the TRIM() function. However, the outputted text will not visually be justified correctly, since the trailing spaces, no matter how proportional they are, still take up space in the string.

It is recommended that the TRIM() function be used liberally whenever right or center justification of character data is desired.

Sizing

An output object may be sized in two ways: by manually changing the value in the "Size" edit control in the "Object Settings" dialog, or by dragging the sizing handles on the object with the mouse.

The "Size" edit control provides precise control over the height and width of an output object, however, it is not visual, and is time consuming.

Sizing Handles

The sizing handles are a set of black squares placed on an object's display text that may be used to change the size of an object (see Figure 5.6). By pressing the left mouse button on a size handle and dragging the mouse, the size of the object is changed.

When used in conjunction with the sensitivity settings, this method provides a rapid method for accurately changing the size of any output object.

Graphic objects may be sized either with the "Size" edit control in the "Object Settings" dialog, or with the sizing handles. Once the new size is set, however, the image is enlarged and/or reduced so that it fills the new space. The aspect ratio may not be maintained.

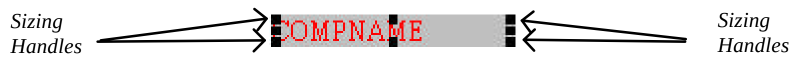


Figure 5.6 Object Sizing Handles

Word Wrap

All output objects -- except lines, frames, and graphics -- output their contents using the selected character set of the object's style. Most often the output of a single object is done on a single line. However, in some cases, especially long field objects (including memo fields) may not fit on a single line.

CodeReporter handles this by word wrapping objects within the size of the output object. That is, if the text for an output object cannot be outputted within the horizontal space allocated for the object, CodeReporter outputs as many words (separated by spaces) as it can before the edge of the object, and then -- vertical space permitting -- outputs the remainder in the second line of the object.

Word wrapping does not increase the size of an object. If the text cannot be outputted within the confines of the size of the object, the excess is ignored.

CodeReporter, by default, creates single line output objects. If a multi-line object is desired, change the size of the object through the "Object Settings" dialog, or by using the object's size handles.

Look Ahead

The "Look Ahead" feature allows an object to be outputted in a group header area with the value that it would have obtained if it had been in the group footer. Essentially, this lets the output object obtain its value from the last record before a group reset condition occurs. See the Header and Footer section of the Group chapter for information on the differences between the group header and footer.

For example, suppose an accounting report was to list the activity for the accounts sorted by account number and date, and for each account, it was important to list the actual range of dates covered in the account. Since the report is sorted by date, the first record in the group (the one used for the output of the group header) contains the first date the account was active. A simple field object would suffice for the beginning of the activity range. The last record before the group is reset would contain the ending activity date for the account. A simple field object placed in the group's header *set as a look ahead object* would output the ending activity date for the account.

Look Aheads and Totals

A look ahead total object placed in a group's header contains the same value there as it would if a simple total object were placed in the group footer.

The practical implications of this, however, is that the summation of the group can occur before the output of the records, and the look ahead total can be used in an interior group to output detail lines that contain a percentage of the total.

For example, a report of the sales sorted by salesperson can display each salesperson's percentage of total sales, simply by creating a look ahead total object in an outer group, and in the salesperson's group footer creating a calculation that contains a total of the salesperson's sales divided by the look ahead total.

The following output object types can be set as look ahead objects: field, total, calculation, expression, and dynamic graphic.

Example

As an example of look ahead output objects, this section documents the steps necessary to create the sales report mentioned above.

```
                  XYZ Sales Inc.
                  Sales Summary
                  Total Sales:       $xx,xxx.xx
                                                    Percentage of
                                                    Total Sales
Salesperson Name                      Sales
  Sanders, John                      $xxx.xx          xx.xx%
  Smith, John                        $xxx.xx          xx.xx%
  Thompson, John                     $xxx.xx          xx.xx%
```

Figure 5.7 Look Ahead Sketch Report

Open the Data Files

Select the **FILE | NEW** menu option and choose LOOKAHD.DBF for the top master data file. This file is located in the .\EXAMPLES directory. For simplicity, this file contains the salespersons' names and total sales.

A more complex configuration of the data files would probably have an individual sales data file and a salesperson data file. In this configuration, a relation would need to be established and a total output object for the salesperson would be used in the "Body" footer instead of simply listing the total sales field in its header.

Creating the Report Areas

Since the report description indicates that the look ahead total object sums the entire report, it should be placed in a header area that is only outputted at the beginning of the report -- such as the title area. Use **AREA | NEW TITLE AREA** to create the title area. Size it to about 1 inch tall.

Adding the Field Objects

In the "Body" header area, place the two fields of LOOKAHD.DBF. This is done by selecting the "Fields" button in the design screen, select the NAME field, move the cursor to the "Body" header area and click the mouse to place the field. Repeat these steps for the TOTSALES field. For information on placing data file fields in a report, see the Fields section below.

Adding the Labels

For information purposes, the title of the report and the column titles for the salespeople should be added in the title area. Select the "Text" button in the design screen, position the cursor in the title area and click the left mouse button. The "Enter Text for Text Object" dialog is invoked to prompt for the text used in the text object. Enter `XYZ Sales` and select the "OK" button. Repeat this process for "Sales Summary", "Total Sales", "Salesperson Name", "Sales", and "Percentage of Total Sales".

Adding the Totals

Totals are based upon numeric calculations and numeric data file fields. Since the total sales are already calculated in the TOTSALES field, no calculation is needed.

Select **OBJECT | TOTAL** to invoke the "Total Calculations" dialog box. Choose the TOTSALES field and place the total output object in the title area (as seen in Figure 5.7).

The "Total Settings" dialog is invoked to prompt for the name and reset condition for the total object. Use TITLETOTAL for the name and, since the total summarizes the entire report, leave the "Total Expression" edit control blank. Select "OK". Select the "Done" button on the "Total Calculations" dialog to remove it from view.

All objects, including total objects, are not originally set to be look ahead objects. To make the new TITLETOTAL total object function as expected, it is necessary to modify its object settings. Invoke the Object Menu, and choose **OBJECT SETTINGS**.

A dialog similar to Figure 5.5 appears. Select the "Look Ahead" radio button. According to the sketch report in Figure 5.7, the total sales is displayed to two decimal places. Modify the "Number of Decimals" edit control from '0' to '2', and select the "OK" button to save the changes.

Add the percentage

The original sketch report lists the sales personnel, their sales, and their percentage of the total sales. The sales percentage is obtained by dividing a salesperson's sales by the total sales. Since the total sales were calculated in the title area, it may be used in the lower level part of the report.

A calculation object is needed to determine the sales percentage. Invoke the "Calculation Object" dialog by selecting the "Calculation" button on the button bar. Select the "New Calc" button to create a new calculation. Enter PERCENTAGE for the calculation name and LOOKAHD->TOTSALES/TITLETOTAL() in the "Calculation Expression" edit control. Select the "OK" button.

The new PERCENTAGE calculation is added to the "Calculation Object" list. Select it with the left mouse button and place the calculation in the "Body" group. To remove the "Calculation Object" dialog select the "Done" button.

Add Object Formatting

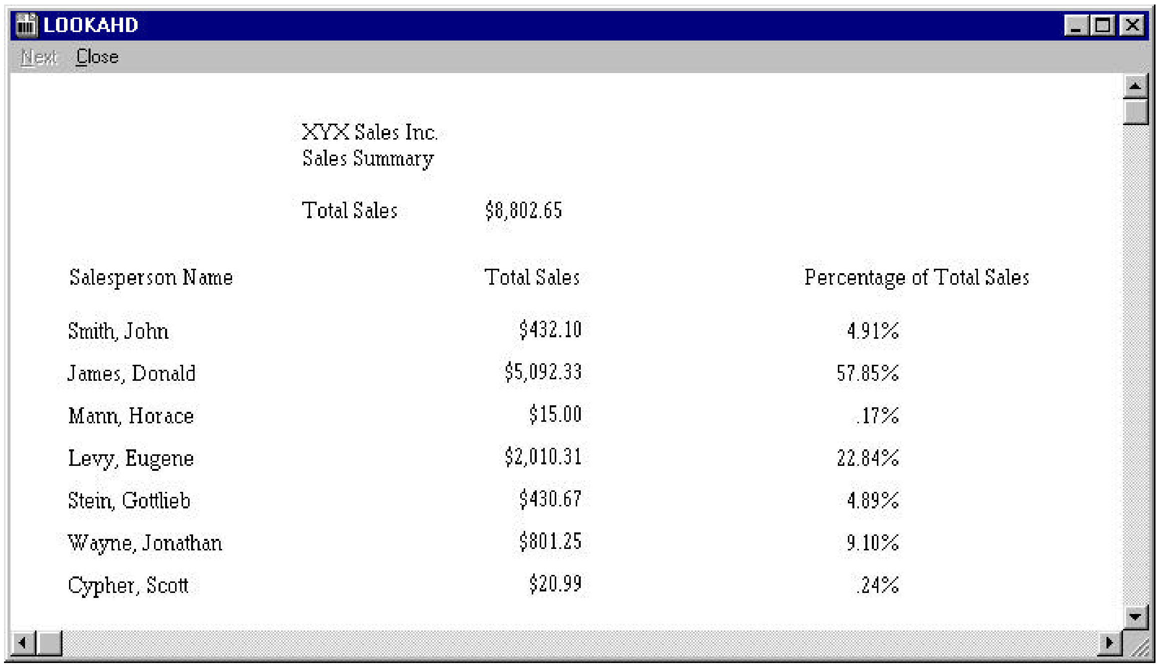
Select the newly created calculation output object, invoke its Object Menu (right click on the object or press the Enter key), and choose the "Object Settings" menu option. Since the percentage displayed in the Figure 5.7 sketch report has two decimal places and is a percentage, these attributes must be set.

Choose the "Percentage" radio button in the Numeric Type section and change the number of decimals for the object to two. Select "OK" to close the dialog.

If the LOOKAHD->TOTSALES output object is to be displayed as a currency value, its settings also must be modified. Invoke the "Object Settings" dialog and select the "Currency" radio button in the Numeric Type section.

Viewing the Report

The report design is complete. Use **FILE | PRINT PREVIEW** to view the completed report. Its contents should appear similar to the sketch report and Figure 5.8, below.



```
                    XYX Sales Inc.
                    Sales Summary

                    Total Sales        $8,802.65

Salesperson Name                      Total Sales          Percentage of Total Sales

Smith, John                              $432.10              4.91%
James, Donald                          $5,092.33             57.85%
Mann, Horace                              $15.00               .17%
Levy, Eugene                           $2,010.31             22.84%
Stein, Gottlieb                          $430.67              4.89%
Wayne, Jonathan                          $801.25              9.10%
Cypher, Scott                             $20.99               .24%
```

Figure 5.8 Look Ahead Output

Numbers

When an output object evaluates to a numeric value -- whether it is a numeric field object, an arithmetic calculation, a total, or a numeric dBASE expression -- the output of the object can be specially formatted.

Numeric Types

CodeReporter can output numeric values in four different ways, as:

- a straight number (no additional formatting),
- a currency value,
- a percentage, and
- an exponent.

Each different type displays the same value differently.

Number

All numeric values, if necessary, are outputted using thousand separators and/or a decimal point. The characters used for these items are under the control of the report and are set in the "Report Preferences" dialog. See Preferences in the Customizing Reports chapter for information on changing these values.

Currency

A numeric value may be formatted as a currency value by selecting the "Currency" radio button in the "Object Settings" dialog. Doing so causes CodeReporter to include the currency symbol (default is '$') before the actual number. The currency symbol is under the control of the report and is set in the "Report Preferences" dialog box. See Preferences in the Customizing Reports chapter for more information.

Percent

A number formatted as a percentage is multiplied by one hundred, and the percent symbol ('%') is placed immediately following the number.

Exponent

A number may be converted into scientific notation and outputted in the format *n.nnnnn* e *xx*, where *n* is the numeric value, and *x* is the exponential value. When using this format, it is very important to set the appropriate decimals setting (see below).

Negative Numbers

By default, CodeReporter displays negative numeric values with a minus sign (-) preceding the value. Certain reports may require negative values to be outputted within brackets. This is accomplished by selecting the "Use Brackets" radio button in the "Object Settings" dialog.

Without Brackets	With Brackets
-1234.56	(1234.56)

Leading Zero

Fractional numbers (those between 1 and -1) are represented by the decimal point character and the fractional number. For some objects, it may be desirable to have a zero placed before these types of numbers. Selecting the "Leading Zero" radio button in the "Object Settings" dialog causes CodeReporter to place a zero before the decimal point character.

Leading Zero	No Leading Zero
0.3	.3
33.3	33.3
-0.3	-.3

Table 5.1 Leading zero

Display Zero

By default, CodeReporter displays numeric output objects that have a zero value. However, deselecting the "Display Zero" radio button in the "Object

Settings" dialog box causes the numeric output object to be omitted from the report if it has a zero value.

Decimals

The number of decimal places outputted for numeric objects is controlled by the "Number of Decimals" edit control in the "Object Settings" dialog box. The default number of decimals is zero (except for numeric field objects, which use the field's number of decimals).

If the number to be outputted has a greater precision (more decimal places) than allowed by the decimals setting, the outputted number is rounded.

Rounding does not affect the actual value of the number with regards to totals. When rounding occurs, it is possible to have a column of numbers that, due to rounding, do not add up to a total output object for that column.

Dates

Date fields in a database are stored in a way that makes them easy to sort: January 2, 1992 is '19920102' in the database, January 3, 1992 is '19920103' in the database and so on. However, in printed reports, it is more intuitive to read 'January 2, 1992' or '02 Jan 1992' than '19920102'.

CodeReporter provides the flexibility of determining which format should be used to output date objects.

Date Pictures

The output format of a date value is represented by a date picture string. This string can contain several formatting characters and/or 'other characters'. The formatting characters are:

- **C** Century. A 'C' represents the first digit of the century. If two 'C's appear together, then both digits of the century are represented. Additional 'C's are not used as formatting characters.
- **Y** Year. A 'Y' represents the first digit of the year. If two 'Y's appear together, both digits of the year are outputted. Additional 'Y's are not used as formatting characters.
- **M** Month. One or two 'M's represent the numeric digits of the month. If there are more than two consecutive 'M's, a character representation of the month is returned.
- **D** Day. One or two 'D's represent the numeric digits of day of the month. Additional 'D's are not used as formatting characters.
- **Other Characters**. Any character which is not mentioned above is placed in the date string when it is outputted.

July 4, 1776, for example, would be outputted differently using different picture formatting:

FORMAT	OUTPUT
MMMMMMMM DD, CCYY	July 04, 1776
YY/MMMMMMMM/DD	76/July /04
MM DD YY CC	07 04 76 17

Default Date Format

CodeReporter uses a default picture format of MM/YY/DD for all objects outputting a date value. The report's default date format may be modified by changing the "Default Date Format" edit control in the "Report Preferences" dialog box. The default date format may also be overridden on an object-by-object basis, by changing the "Date Format" edit control (within the "Object Settings" dialog -- not shown in Figure 5.5.)

Using the Default Date Format

The default date format is automatically used when a date output object is created. The object retains the default date format throughout its life (unless the object's "Date Format" edit control is changed) even if the default date format for the report is changed at a later time.

If an object's date format is completely deleted and the "OK" button is selected, the date format reverts to the current default date format.

Display Once

In some cases it may be desirable to output an object occasionally -- when its value changes -- instead of outputting it every time its group resets. For example, Figure 5.9 displays a simple report displaying the contents of a data file. Instead of outputting the same month for each line of the report, the month is only outputted when it changes.

This type of selective output can be accomplished by using the **DISPLAY ONCE** Object Menu item. This invokes the "Object Display Suppression." Select the "Display Once" check box and enter an expression to be used to suppress the output of the object. This expression is evaluated when the group reset condition occurs, and if its value has changed from the previous reset condition, it is outputted.

To vary the output using a true/false condition, see the Suppressing an Area section in the Areas chapter.

The DBF->MONTH output object in Figure 5.9 uses the 'Display Once' option with a suppression expression containing the same value as outputted: DBF->MONTH. When the first 'January' entry is outputted, the month is outputted. The second entry, however, does not change the value of DBF->MONTH's display once condition, and so the month is not outputted.

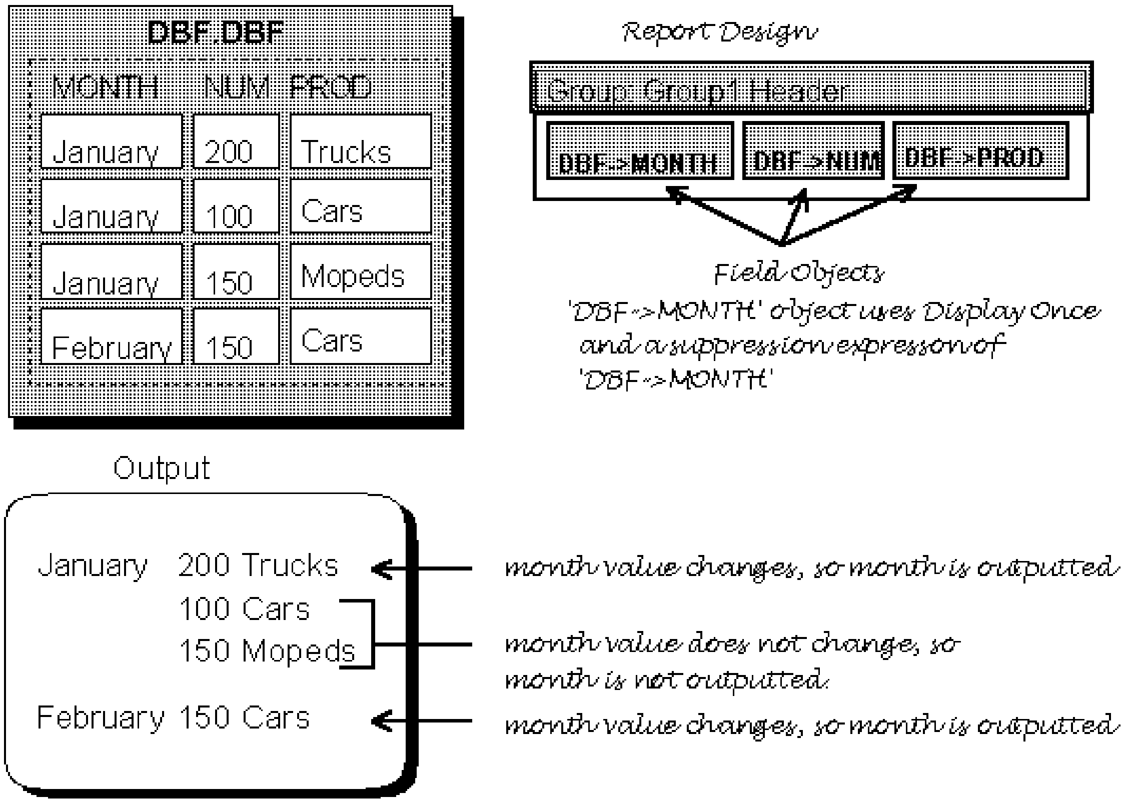


Figure 5.9 Display Once Example

To change the display once condition for an object or to remove it, select the **DISPLAY ONCE** Object Menu option to invoke the "Object Display Suppression" dialog box. Make the desired changes to the suppression condition, or deselect the "Display Once" check box to display the output object every time the group resets.

All output object types may be set to display once.

Text Objects

The simplest type of output object is the static text object. A static text object consists of a string of alphanumeric text which is reproduced verbatim in the report.

As such, they are often used to identify the report (such as a title), to identify report elements that may not be totally clear, or to bring attention to a part of the report.

CodeReporter is put into insertion mode for Text objects by selecting the **OBJECT | TEXT** menu item, or the "Text" button. When a text object is placed, the "Enter Text for Text Object" dialog is invoked.

Text objects are also created when text is placed in the Windows clipboard from another application, such as a word processor, and CodeReporter's **EDIT | PASTE** menu option is selected.

Lines and Frames

Another way to bring attention to a section of the report or to set it apart from other sections is to use a static line or frame. Lines can either be horizontal or vertical. Frames are simply rectangles which may be filled, or may have rounded corners. See Figure 5.10 for an illustration of the different types of lines and frames.

Lines

The creation of a horizontal line is accomplished by selecting the "H-Line" button (or the **OBJECT | HORIZONTAL LINE** menu option) and clicking the mouse where the line is to appear. In the same manner, a vertical line is created using the "V-Line" button (or the **OBJECT | VERTICAL LINE** menu option).

Lines have a default length and width which may be modified by using the **OBJECT SETTINGS** Object Menu selection to invoke the "Object Settings" dialog box. This dialog is used to increase the thickness of the line, and to change its color. (See Line Thickness, below, and the Color sub-section, also below.)

Line Thickness

The thickness of a line is set in the "Thickness" edit control of the "Object Settings" dialog. The thickness of lines are set in pixels -- the smallest unit available on a computer screen.

Line Length

The length of a line may by changed by using the line's sizing handles to drag it to the new length, or by changing the value in the "Length" edit control of the "Object Settings" dialog.

Frames

Frames are a special type of line object. Frames are rectangles which may be used to offset a special piece of information in a report.

A frame object may be created by selecting the "Frame" button from the button bar, or by selecting the **OBJECT | FRAME** menu option. When a frame is created, it has a default height and width, has square corners and is hollow. The frame's sizing handles may be used to change the height and width.

The thickness of the line used to draw the rectangle may be modified by changing the "Thickness" edit control in the "Object Settings" dialog.

Corners

The corners of frame objects are square by default. If rounded corners are desired, select the "Rounded Corners" radio button in the "Object Settings" dialog.

Filled

A filled frame is much like a very thick line. The inside of the rectangle is filled with the color of the style used when the frame is created. The difference between a filled frame and a very thick line is that a frame may be sized vertically and horizontally using the sizing handles, while a line may only be sized in one direction.

A frame may be filled by selecting the "Filled" radio button in the "Object Settings" dialog box.

De-selecting the "Filled" or "Rounded Corners" radio button makes the frame hollow or have square corners, respectively.

Color

Lines and frames may have a color associated with them in the same manner as other output objects, by selecting a style that contains the desired color. Lines and frames only use the color portion of the style and ignore the typeface, the point size, etc. See the Styles chapter for more information on creating and modifying styles.

Objects Within

Frames (and thick lines) are often used to visually group output objects together. As mentioned in Objects Within Objects in the Creating Objects section, above, an object that completely surrounds another object is said to contain the second object.

Frames often do this, since they are usually placed around other output objects. When using filled frames, two items should be noted:

1. The white rectangle that is visible around an output object placed within the frame (in design mode) is not outputted when the report is outputted. It merely shows the outline and sizing handles of the interior output objects.
2. Output objects within a filled frame should not use a style that has the same color as used to fill the frame. Black text on a filled black frame, for example, results in the text being "invisible." This often occurs when the Style popup menu is used to select a style for the frame -- since all interior objects are also selected when the frame is selected, their style is set to the same style as the frame.

Graphics

CodeReporter may include graphical elements into reports either statically or dynamically. These graphics may be anything from a company logo to a series of personnel photos.

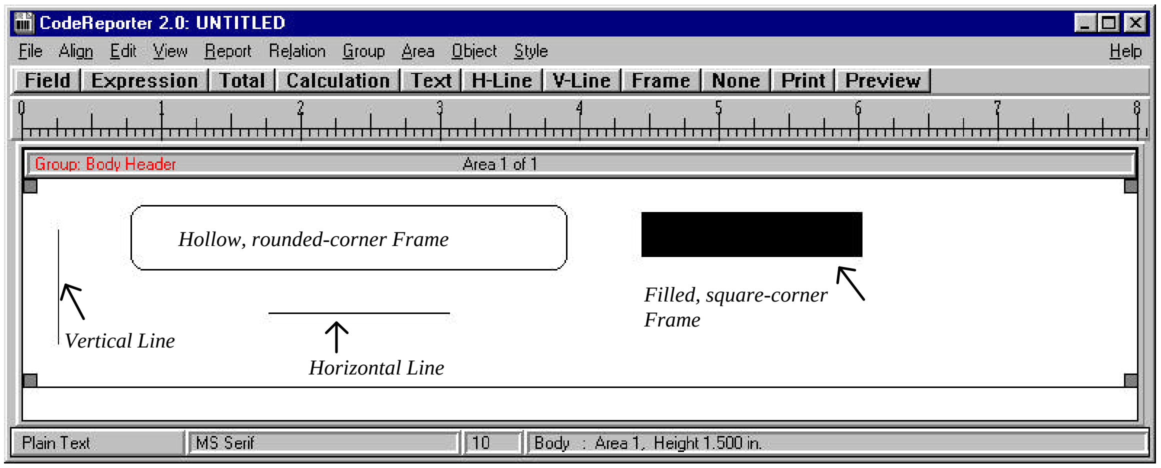

CodeReporter 2.0: UNTITLED
File Align Edit View Report Relation Group Area Object Style Help
Field Expression Total Calculation Text H-Line V-Line Frame None Print Preview
Group: Body Header
Area 1 of 1
Hollow, rounded-corner Frame
Filled, square-corner Frame
Vertical Line
Horizontal Line
Plain Text
MS Serif
10
Body : Area 1, Height 1.500 in.

Figure 5.10 Lines and Frames

Average (boring) reports can be made visually exciting by the inclusion of graphical elements. For example, through the use of suppressed areas and different graphical elements the "bottom line" of a financial report could display a "thumbs-up" or a "thumbs-down" depending upon the numeric outcome.

CodeReporter currently supports Windows bitmap graphics in three ways:

1. statically with bitmaps pasted in the report,
2. statically by referencing a file name, and
3. dynamically using a data file field which contains a file name.

Smaller bitmaps are generally outputted better than larger ones. Due to the scaling involved in shrinking high resolution bitmap images (such as scanned images), it is recommended that low resolution images be used.

Creating a Graphic Object

Graphic objects are created with the **OBJECT | GRAPHIC** menu option, or by pasting a graphic element from the clipboard. The former method is used to create graphic objects with a minimum of additional disk space necessary. The later creates a static graphic object that is actually stored within a CodeReporter-created bitmap file.

Pasting Graphics

Other Windows applications, such as Windows Paintbrush, allow users to create customized bitmap images. Once the image is designed, it may be placed in the Windows clipboard (usually with the application's cut or copy command) where it is accessible to CodeReporter. Once in the clipboard, this image may be pasted into a report as a static graphic object using the **EDIT | PASTE** menu option.

The bitmaps for pasted graphic objects are displayed in the report design screen.

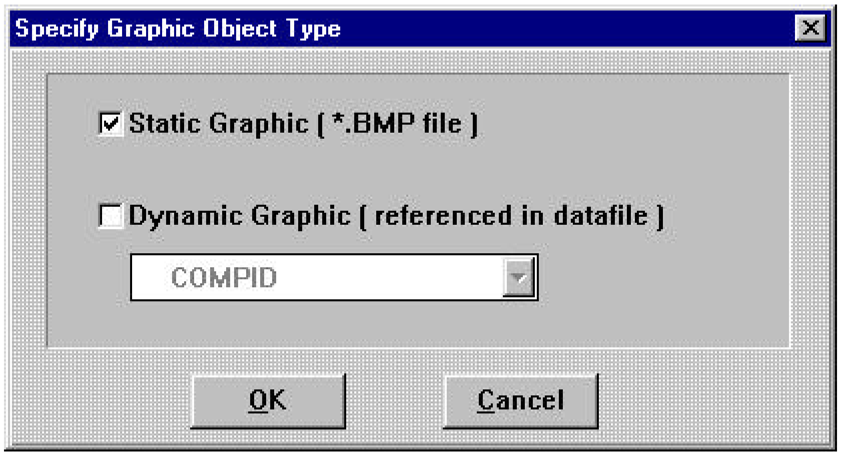
Bitmaps that are pasted into a report are stored within an external bitmap file. Since this bitmap file is created without user input, it is given a unique -- and generally obscure -- name. If the pasted bitmap is already saved in a file using another application, this process creates an unnecessary duplicate file.

It is usually more appropriate to use the source application to save the bitmap as a bitmap file (*.bmp) and use the OBJECT | GRAPHIC menu option to create a graphic output object that accesses the file.

Using the menu

The **OBJECT | GRAPHIC** menu option invokes the "Specify Graphic Object Type" dialog where the two types of objects may be selected.

When the "Static Graphic" check box is chosen and a graphic element is placed, CodeReporter prompts for the file name of the bitmap file and displays the bitmap within the CodeReporter design screen. CodeReporter only stores the specified file name within the report file, and not the actual bitmap image. This allows the report designer the flexibility of changing a graphic object in a report simply by altering the referenced bitmap file.



Static Graphic [*.BMP file] [✓]

Dynamic Graphic [referenced in datafile] [ ]

COMPID

Figure 5.11 Specify Graphic Object Type Dialog

The "Dynamic Graphic" check box, when chosen, enables the drop down combo box which contains all the field names in the current composite data file. Once a field name is chosen, the graphic object may be placed. Since the referenced field does not contain a value until the report is outputted (and then it may constantly change), CodeReporter displays the dynamic graphic object as a regular field in the design screen.

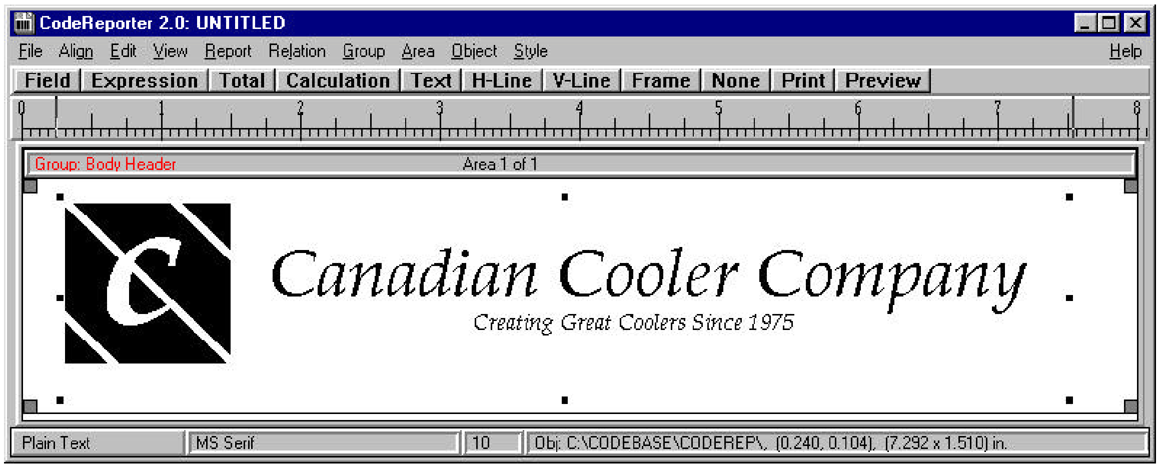
It is important to appropriately size a dynamic graphic output object, since CodeReporter stretches or reduces the bitmap so that it entirely fits within the specified size. In addition, ensure that all bitmaps referenced by the field are of the same size. Failure to do so can cause some graphic objects to be displayed "correctly" while others may be distorted.

Scaling Graphic Objects

Graphic images within CodeReporter are automatically scaled to the size of the output object. That is, bitmap images that are larger than the current size of the output object are reduced, while images that are smaller are enlarged.

This activity occurs on each axis. If an image that is tall and skinny is pasted into a square graphic object, the image is made short and fat. CodeReporter does not maintain an image's aspect ratio, but manipulates the image to fit the size of the graphic object.

The size and the scaling of a graphic object is controlled in the same manner as a regular output object: through the "Object Settings" dialog, or by the sizing handles.


CodeReporter 2.0: UNTITLED
File Align Edit View Report Relation Group Area Object Style Help
Field Expression Total Calculation Text H-Line V-Line Frame None Print Preview
Group: Body Header
Area 1 of 1
Canadian Cooler Company
Creating Great Coolers Since 1975
Plain Text
MS Serif
10

Obj: C:\CODEBASE\CODEREP\. (0.240, 0.104). (7.292 x 1.510) in.

Figure 5.12 Static Graphic Object

Fields

Field objects reflect the contents of the composite record at the time of the area's group reset condition. If one or more fields need to be combined or manipulated to arrive at the "appropriate" output -- such as combining first and last name fields -- an expression output object should be used.

Placement

Fields from the composite data file are added to the report from the "Field Objects" floating list box, which is invoked from the **OBJECT | FIELD** menu item or the "Field" button.

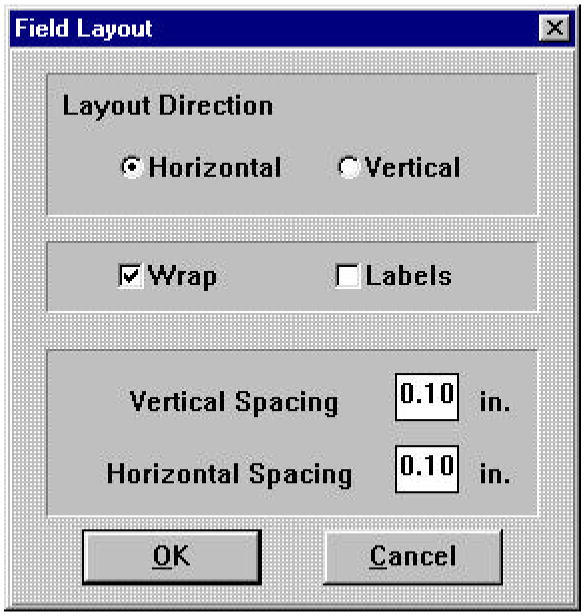
Field objects may be placed individually, or several field objects may be placed at the same time.

Single Selection

A single field may be added to the report by using the "Field Objects" floating list box to select the desired field, using the mouse to move the cursor to the desired spot, and clicking the left mouse button. The field, with an approximate size and default settings, is placed in the desired spot.

Multiple Selection

When two or more fields are selected in the "Field Objects" list box, the "Field Layout" dialog box is invoked as the objects are placed in the design screen. Shown below, this dialog provides several options as to the placement of the output objects.



Field Layout

Layout Direction

[● Horizontal] ○ Vertical

[✓ Wrap] ☐ Labels

Vertical Spacing 0.10 in.

Horizontal Spacing 0.10 in.

Figure 5.13 Field Layout Dialog

Layout Direction

CodeReporter places the output objects either left to right or top to bottom from the insertion point, depending upon the "Layout Direction" radio buttons. These settings also take into account the "Wrap" check box setting.

Wrap

As the output objects are being inserted, CodeReporter checks to see if any object extends beyond an edge of the report area in which they are placed. Normally in this case, the object that extends beyond the edge is sized so that it fits within the area, and insertion is terminated.

However, if the "Wrap" check box is set, CodeReporter changes the insertion point back to the beginning coordinates and moves down or to the right (depending upon the horizontal or vertical setting, respectively) and continues to insert the fields. If it continues to run out of room in the area (if it hits the lower right corner), CodeReporter may choose to vertically enlarge the area so that all output objects may fit.

CodeReporter vertically enlarges the area if the "Layout Direction" radio button is set to be vertical and the "Wrap" radio button is not selected, or if the "Layout Direction" is horizontal and the "Wrap" radio button is selected.

"Wrap" is set by default.

Labels

If the "Labels" check box is set, CodeReporter also inserts text objects containing the field names in a column to the left of the field objects.

Vertical and Horizontal Spacing

The space between the inserted field (and label) output objects is controlled by the "Vertical Spacing" and "Horizontal Spacing" edit controls. As a default, the spacing is set to .1 inch, but it may be changed to any value. This is a useful option for quickly placing output objects in reports which may be outputted in an environment other than Windows where precise positioning is important.

Memo Fields

Memo fields, which have their contents stored in a separate memo file, behave exactly the same as regular fields. It is important to note that:

- if the memo field is blank, the object is displayed with an empty value,
- if the memo does not fill the entire size of the output object, the excess space is wasted, and
- if the memo is larger than the size of the output object, the excess information is ignored.

Expressions

dBASE expressions which are needed only once or twice may be outputted within the report using an expression output object. Simply put, an expression object is used to output an evaluated expression in the report.

For example, if a data file has separate first name and last name fields, but within the report they are to appear in the format "Smith, John", an expression output object containing the following expression could be used.

```
TRIM(DBF->LAST_NAME)+', '+DBF->FIRST_NAME
```

Creating

CodeReporter is put into insertion mode for expression output objects by selecting the **OBJECT | EXPRESSION** menu option or by selecting the "Expression" button. The mouse cursor then changes to indicate insertion mode is active.

Position the mouse cursor within a report area and click the left mouse button to place an expression output object.

CodeReporter prompts for the initial expression for the expression output object using the "Easy Expression" dialog box (see the Expressions chapter for information on the "Easy Expression" dialog). Once an expression is entered, select the "OK" button to complete the creation.

Calculations

A calculation performs a numerical or character-based computation that is used in the report. These computations take the form of dBASE expressions. In many cases a computation is used simply to add one or more data file fields together, but it may be more complex involving composite data file fields and/or dBASE functions. The calculation may be thought of as a "short hand" way of referring to the computation it contains.

A calculation may be used in zero, one or multiple calculation output objects. In addition once a calculation is defined, it may be used in any expression within the entire report -- including sort expressions, query expressions, expression output objects, relation expressions, etc.

For example, a calculation could figure out an employee's total pay using the following formula:

```
PAY->REG_HRS*EMP->RATE + PAY->OT_HRS*EMP->RATE*1.5
```

or format an employee's first and last name using the following computation:

```
TRIM(EMP->F_NAME)+'  '+EMP->L_NAME
```

If a report was simply a payroll report that listed the employees' names and gross pay, it might not be necessary to define a calculation for the formula -- the expression might just as well be put in its own expression output object. However, if the report also needed to display the total of all salaries paid during the report or do other computations on the employee's total pay, a calculation is suggested.

By using a calculation instead of retyping the computation in separate expression objects, the report design time is shorter, and the report is easier to modify and maintain.

If an expression is used more than once in a report, it is recommended that a calculation be created.

A calculation may also include one or more calculations. That is, you may nest one calculation within the definition of another calculation.

For example net pay is calculated from the total pay minus any deductions. The calculation for net pay might then be:

```
TOT_EMP_PAY() - DEDUCT()
```

where TOT_EMP_PAY() and DEDUCT() are previously defined calculations.

Creating Calculations

A calculation is created by invoking the "Calculation Object" dialog box and selecting the "New Calc" button. The "Create Calculations" dialog is invoked.

A calculation has two elements, the calculation name and the calculation expression. The calculation name is a character string that is used to represent the computation.

It is this name that appears within the "Calculations" list box in the "Easy Expression" dialog. The calculation expression, as described above, may be entered into the "Calculation Expression" edit control.

The name of a calculation may not have spaces within it. In the event that the calculation name is entered with spaces, CodeReporter uses the characters up to the first space as the name of the calculation.

Even though calculations are created through the **OBJECT** menu, it is not necessary to have a calculation output object for every calculation. In fact, it is very common to have calculations that are used only within other calculations, expressions, and totals.

Deleting Calculations

Calculations are deleted from the "Calculation Object" dialog box. To delete a calculation, select its name within the dialog's list box and select the "Delete Calc" button. This removes the calculation, all calculations/totals that use the calculation, and all output objects that use any of the removed calculations and totals.

Deleting a calculation can quickly remove many related elements in a report. Delete with care.

Calculation Objects

Once a calculation is created, a calculation output object may be placed by:

1. selecting the desired calculation name within the "Calculation Object" dialog box,
2. positioning the mouse cursor to the desired spot within a report area, and
3. single clicking the left mouse.

CodeReporter is automatically removed from insertion mode once the calculation object is placed.

Deleting a calculation output object does not delete the calculation upon which it was based.

Totals

A total gives the report designer the ability to summarize the numerical data obtained from a numeric calculation or numeric data file field and output it within the report. Essentially, a total is an output object that retains its value from one composite record to another -- using each composite record to update its value.

Totals are based upon previously created numeric calculations and numeric data file fields. As a result, multiple totals which are based upon the same information may use the same calculation.

Unlike calculations, a total may not exist without an object having been created for it. This is a direct result of the nature of totals: a total achieves its value at a certain place and time within the report. For example, a total of sales for each employee changes its value from one employee to the next. A reference to this total within the report title area, for example, would display a non-existent value since at the beginning of the report the total has not yet obtained a value.

A total output object within report wide expressions, such as the query expression, makes no sense, since the value of the total is not calculated until the report is outputted. In this case, since a report is based on the composite data file, an attempt to limit the composite data file based upon the contents of the report, creates a logical contradiction.

Totals are properly placed within the group footer for the subset of data it is totaling -- unless it is a look ahead total, in which case it may be placed within the group header area.

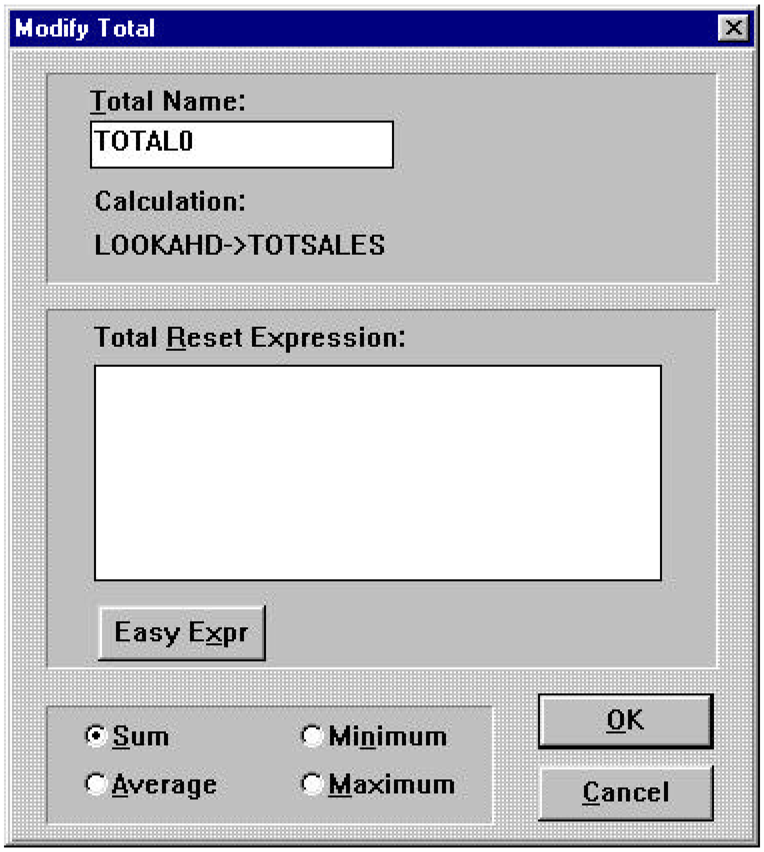
Creating a Total

A total output object is created through the "Total Calculations" dialog box which is invoked by selecting the **OBJECT | TOTAL** menu option or the "Total" button in the report design screen.

The "Total Calculations" dialog box contains a list box which is filled with previously created numeric calculations and previously created totals. Total output objects are placed within the report in the same manner as calculations:

1. select the desired calculation, total name, or numeric data file filed within the list box,
2. position the mouse cursor within the desired destination report area, and
3. click the left mouse button.

As an output object is placed, the "Total Settings" dialog (Figure 5.14) is invoked.


Modify Total

Total Name: TOTAL0

Calculation:

LOOKAHD->TOTSALES

Total Reset Expression:

[✓ Sum] Minimum Average Maximum

Figure 5.14 Total Settings Dialog

Name

Totals, like calculations, may be used in any dBASE expression within CodeReporter; and thus each new total requires a unique name. This name defaults to "TOTAL*n*", where *n* is an ascending number used to keep the total name unique. This name may be modified to be more descriptive of what the total is actually totaling.

When the total is used within other expressions, it is this name, followed by a set of parentheses (), that identifies which total is to be used.

If the value of a total is needed in a dBASE expression, but does not need to be outputted within the report, the total output object may be "hidden" by changing its size so that it is too small to display any of its contents.

Types

The actual process involved in updating the total's value from composite record to composite record depends upon the use of the total. A total may maintain an arithmetic sum, an average, a maximum value, or a minimum value.

Sum

A sum total adds the number returned from the evaluated calculation, total or numeric field for each record to its preceding value.

If a total is used to simply count the number of records between total reset conditions, base the total output object upon a calculation with a constant value of one (1).

Average

An average total stores the arithmetic mean for the evaluated calculation, total, or numeric field. This mean (or average) is obtained by taking the sum of the values for the records and dividing it by the number of records encountered.

Maximum

A maximum total stores the largest number encountered for the evaluated calculation, total, or numeric field.

Minimum

A minimum total stores the smallest number encountered for the evaluated calculation, total or numeric field.

Reset Expression

The total reset expression is used to create totals that summarize subsets of information within the composite data file. For example, the total that describes the amount of sales made by a certain salesperson is a total for the salesperson's subset of sales.

Usually, the total reset expression is the same as used in the reset expression for the group with which the total is logically related. For example, a report that lists sales may be sorted and grouped by months. A total of the sales for a month would have a total reset expression of DBF->MONTH -- which would be the same as the month group's expression. See Figure 5.15.

The total reset expression need not be the same as the group reset expression for the group it is within and may reset in an entirely different manner.

Subtotals

The total reset expression dictates when a total is reset to its initial value. The reset expression is evaluated for each composite record in the composite data file, and when the value of the evaluated reset expression changes, the total is reset so that a new accumulation may begin.

The value with which the total is reset varies, depending upon the type of total. Sum and average totals are reset to zero, while the maximum and minimum totals are reset to the smallest and largest numbers possible (respectively).

Report-wide Totals

A report wide total, or grand total, is achieved by using an empty total reset expression. That is, the total is only reset when the report begins, so when the total object is outputted in the report summary (or title if it is a look ahead total), it totals the entire report.

Running Totals

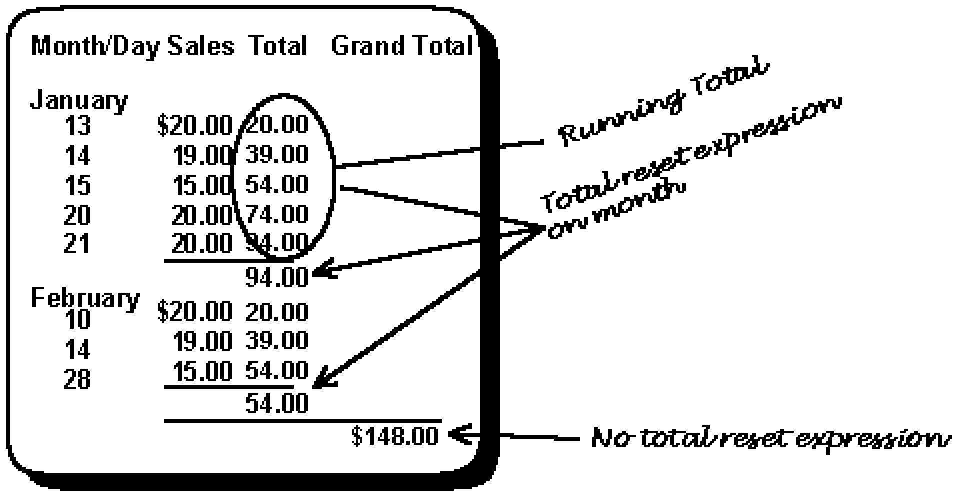
A total output object that is placed in the same area as the information that it is totaling is called a running total. Since the total is constantly being updated, each time the group resets (and the output objects within the group's areas are outputted) the total output object contains a new value.

Deleting a Total

Deleting a total output object deletes the total upon which the object is based. Doing so also removes all other totals, calculations, expressions, and objects that contain the total. This can cause further deletions, and cause a cascading effect which might result in the ruination of a report. Delete total objects with care.

Look Ahead Totals

Look ahead totals are used to output the summary of a subset before the detail lines of the subset are actually outputted. This important concept, along with an example of using look aheads, is illustrated in the Look Ahead section of this chapter (above).



Month/Day	Sales	Total	Grand Total
January			
13	$20.00	20.00	
14	19.00	39.00	
15	15.00	54.00	
20	20.00	74.00	
21	20.00	94.00	
		94.00	
February			
10	$20.00	20.00	
14	19.00	39.00	
28	15.00	54.00	
		54.00	
			$148.00

Running Total
Total reset expression on month
No total reset expression

Figure 5.15 Total Sample Report

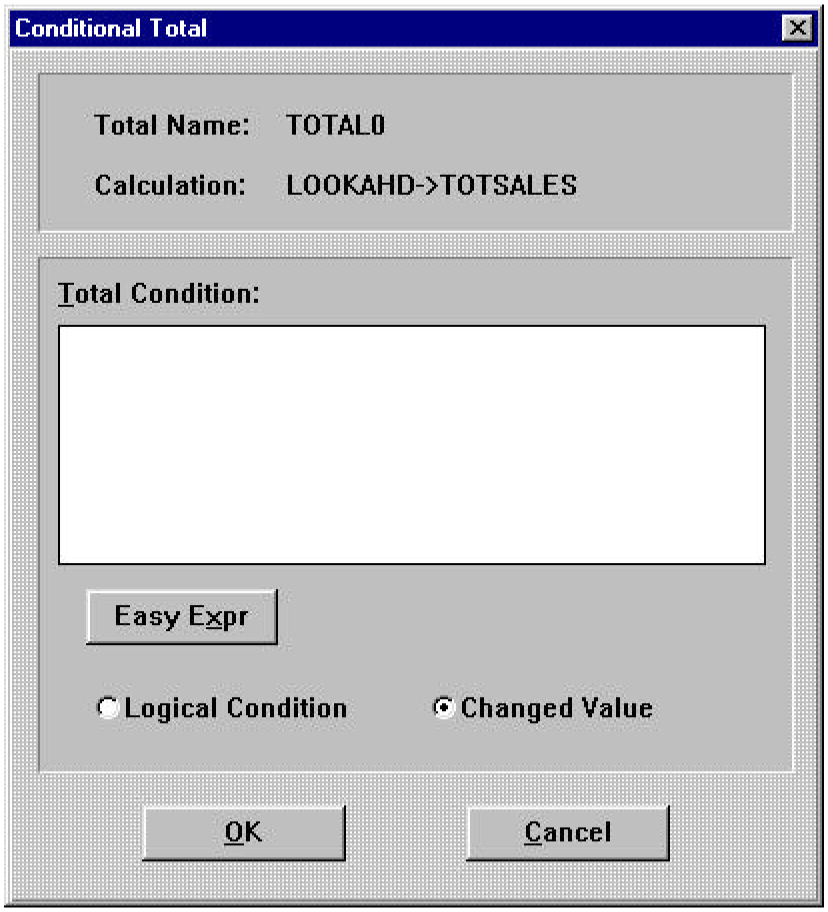
Conditional Totals

Totals may be associated with a dBASE expression that determines under which conditions the total is to be accumulated. By setting up a conditional accumulation condition, the total output object may update its value on some composite records in the composite data file, while ignoring others. Perhaps it is important to only accumulate the total when a field contains a certain value, or only accumulate the total for a master record related to several scanned records. Using a conditional total makes this possible.

A conditional accumulation may be established by invoking a total output object's Object Menu and selecting the **CONDITIONAL TOTAL** menu option. This action invokes the "Conditional Total" dialog, in which the total's condition may be entered.

Total Condition

The "Total Condition" entry window is used to enter the dBASE expression which is used to determine when the total is accumulated. This expression may evaluate to any dBASE type, excluding memo.



Total Name: TOTAL0

Calculation: LOOKAHD->TOTSALES

Total Condition:

[Logical Condition] [✓ Changed Value]

Figure 5.16 Conditional Total Dialog Box

The "Logical Condition" and "Changed Value" radio buttons determine how the total condition is used to selectively update the total's value.

Logical Condition

If the "Logical Condition" radio button is selected and the total condition (which must evaluate to a logical value) evaluates to a .TRUE. value, the total is updated with the contents of the current composite record. If the expression evaluates to a `.FALSE.` value, the current composite record is ignored.

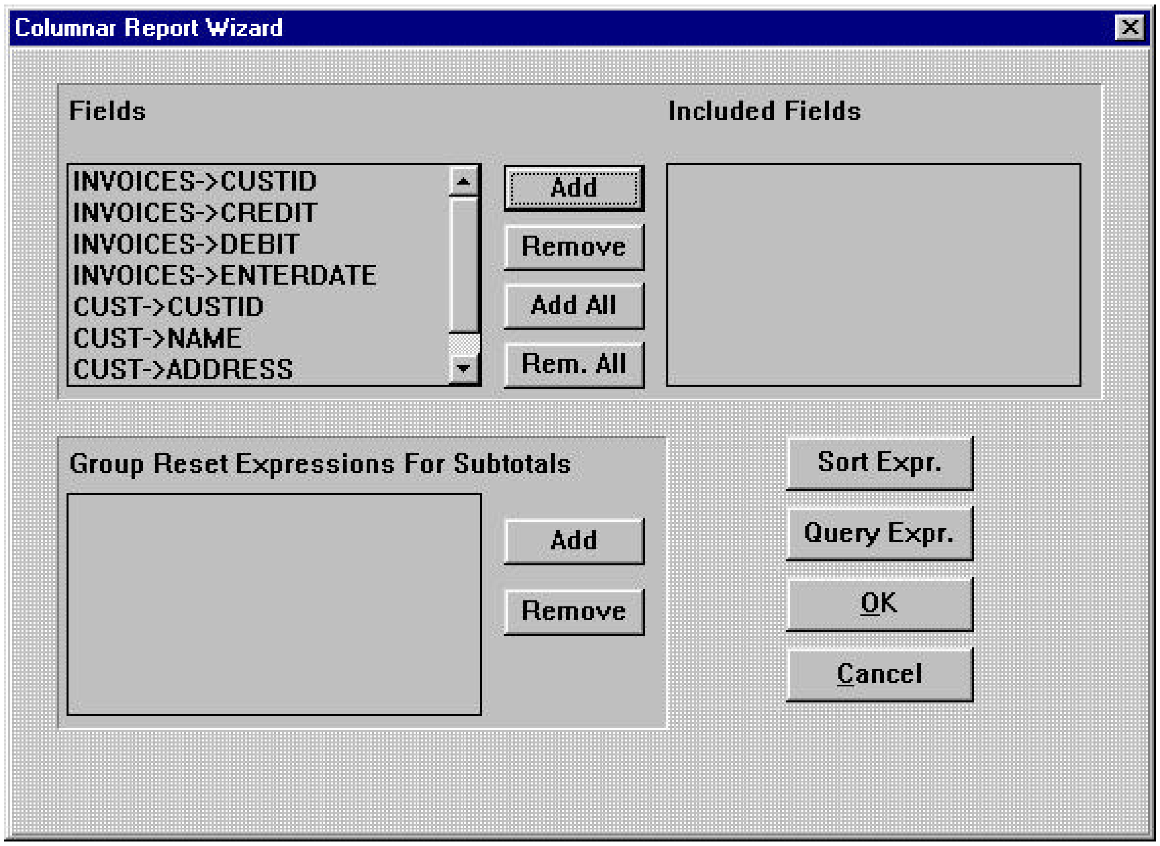
Changed Value

If the "Changed Value" radio button is selected, the total is updated only when the evaluated total condition changes. For example, if there is a scan relation, the contents of the master data file are repeated in the composite data file for each record in the related slave data file. A total output object that depends upon a field within the master data file would be accumulated "incorrectly" since the master data file field would be repeated several times.

Setting the "Changed Value" radio button and using the total condition that corresponds to the master data file's record change (or the master expression in the scan relation), the total would be accumulated "correctly."

6. Columnar Report Wizard

CodeReporter provides a quick and easy automatic report design option through the "Columnar Report Wizard" dialog. This dialog provides a means of rapid report creation -- including the insertion of fields in the relation set, the creation of group headers and footers, and automatic totaling and subtotaling of numeric fields.



INVOICES->CUSTID
INVOICES->CREDIT
INVOICES->DEBIT
INVOICES->ENTERDATE
CUST->CUSTID
CUST->NAME
CUST->ADDRESS

Figure 6.1 Columnar Report Wizard

Invoking Report Wizard

The "Columnar Report Wizard" dialog box is invoked from the **REPORT | COLUMNAR REPORT WIZARD** menu option. If the current report already contains some output objects, they are destroyed when the "Columnar Report Wizard" dialog box is invoked. Before doing so, however, CodeReporter prompts the user to save the file or cancel the columnar report.

Creating a Report

The "Columnar Report Wizard" dialog is used to add selected fields to the report, to create subtotals of numeric fields, and to query and sort the composite data file. Once the report is defined within the "Columnar Report Wizard" dialog, use the "OK" button to create the report. The "Cancel" button may be used to exit the dialog without creating a report.

Adding Fields

All the fields of the composite data file are listed in the "Fields" list box. Fields can be multiply selected in the "Fields" list box and added to the report by selecting the "Add" button. "Add All" automatically adds every field in the composite data file to the report.

The fields are placed within the report, from left to right, in the order they appear in the "Included Fields" list box. If a field is added to the "Included Fields" list box out of the desired order, or was added in error, the "Remove" and "Rem. All" may be used to remove the field (or all fields) from the report.

If more fields are inserted within the report than may fit upon one line, the field that exceeds the page margin is truncated, and the excess fields are placed on a second line.

Subtotals

The report created by the "Columnar Report Wizard" dialog automatically creates report-wide totals for all numeric fields and places them in the report summary area. If subtotals are desired for subsets of the composite data file, they may be created by defining the subset for which they are associated.

This is done by using the "Add" button in the "Group Reset Expressions for Subtotals" section of the dialog. The "Add" button invokes the "Easy Expression" dialog in which a group expression may be entered to describe the subset for which the subtotal is desired. The "Columnar Report Wizard" automatically creates a group (named "Group*n*" where *n* is the number of the group) for this expression and places the subtotals within this group's footer area.

The subtotal groups are added to the top of the "Subtotal On" list box. The groups, therefore, are added from the innermost to the outermost.

If a group is added in the wrong place, either use the "Remove" button and "Add" button to correct the mistake, or, once the report is created, modify the group's position.

Sorting and Querying

The sort expression and query expression for the report may be specified within the "Columnar Report Wizard" dialog using the "Sort Expr." and "Query Expr." buttons. These buttons invoke the "Easy Expression" dialog in which the expressions may be entered. For information on sorting and querying the composite data file, see .

The sort and query expressions previously defined are reflected in the "Easy Expression" dialogs. If these expressions are no longer appropriate, they may be removed by deleting the expression in the "Easy Expression" dialogs.

Example

As an example of a simple columnar report, the following demonstrates the steps necessary to create a sales report for the 'ATHA' company, including subtotals for every store.

Locate SALES.DBF

The data file necessary for this sales report is SALES.DBF, which is located in the .\EXAMPLES directory. Use the **FILE | NEW** menu option to create a new report, and the "Select Data File" dialog to locate SALES.DBF.

For simplicity sake, this report does not include relations to the COMPANY.DBF and STORES.DBF data files. As a result, the data for the company and store that generated the sale are outputted as stored in SALES.DBF -- as codes.

Report Wizard Dialog

Begin the report wizard process by selecting the **REPORT | COLUMNAR REPORT WIZARD** menu option to invoke the "Columnar Report Wizard" dialog.

Add Fields

Most of the fields of the SALES.DBF data file are to be included within the report. Since the report is primarily interested in the financial aspects of the stores and not the actual products sold, PRODCODE is not to be included.

The fields of SALES.DBF (COMPID, STOREID, AMOUNT, and PRODCODE) are all listed within the "Fields" list box. Single click on the "Add All" button to add all of the fields to the "Included Fields" list box. PRODCODE, which was also added, should be removed by selecting it from within the "Included Fields" list box, and single clicking the "Remove" button. This removes the field from the "Included Fields" list box and adds it again to the "Fields" list box.

Add Subtotals

The report specification indicates that subtotals are desired for every store. A logical subset of the composite data file is created by the STOREID field in the same manner that the LOC created a subset within Figure 3.1 in the Groups chapter.

The expression that describes this subset is:

SALES->STOREID

(Normally, the expression would also include the major subset of COMPID, but since a query is going to be used to limit the company to 'ATHA', the above expression is sufficient).

Use the "Add" button within the "Group Reset Expressions for Subtotals" area to invoke the "Easy Expression" dialog and enter this expression. The expression should appear within the "Group Reset Expressions for Subtotals" list box.

Sorting

The records within SALES.DBF most likely are not entered in a sorted order for every store. It is therefore necessary to sort the data file. This is accomplished by selecting the "Sort Expr." button and entering the sort expression:

SALES->STOREID

The sort expression may safely ignore the COMPID, the major subset of the sales data file, since the specification of the report indicated only the 'ATHA' company sales were to be included within the report.

Query

The limiting of the composite data file is done with a query expression SALES->COMPID='ATHA'. Select the "Query Expr." button and enter the expression in the "Easy Expression" dialog.

Viewing the Report

After the sort and query expressions are entered, the columnar report is created by selecting the "OK" button. Display the report by selecting the **FILE | PRINT PREVIEW** menu option. The fields and totals correctly appear within the output window.

Polish

The instant report is by no means ready to be published. There are several things that may be done to "polish" the report -- including altering the styles and typefaces, adding descriptive labels to the totals, adding report titles, etc. The instant report may be modified in the same manner as any report created in the standard manner.

7. Expressions

Much of the interaction between CodeReporter and the report designer is done through the use of dBASE expressions. dBASE expressions are, conceptually, a macro language used to describe some operation or identify some information.

dBASE expressions are much like arithmetic equations. Values may be added together to get a result, equations can be tested for validity (true or false), etc. Like arithmetic, dBASE expressions follow several rules. Unlike arithmetic which only deals with numeric values, dBASE expressions have several different types of values, including characters and dates.

Those familiar with dBASE or dBASE expressions may skip ahead to the Section.

General dBASE Expression Information

All dBASE expressions return a value of a specific type. This type can be Numeric, Character, Date or Logical.

A common form of a dBASE expression is the name of a field. In this case, the type of the dBASE expression is the type of the field.

Field names, constants, and functions may all be used as parts of a dBASE expression. These parts can be combined with other functions or with operators.

Example dBASE Expression:

```
UPPER(DBF->FIELD_NAME)
```

Returns

In arithmetic, 1+2 is considered a statement that represents the same value as 3. 4x3ö6 is the same as 2. In dBASE terminology, these arithmetic statements are said to return a value. The numbers are combined in an arithmetically consistent fashion and the result is obtained.

Equations

If a student were given a true or false test and he/she were told to evaluate 1+2 = 5, he/she would correctly mark it false. 1+2=3 should be marked as true. Both sides of the equals sign are evaluated separately and their return values are compared. In the first case 1+2 evaluates to 3 and 5 evaluates to 5. The statement that 3 is the same as 5 is incorrect, so it is considered false. With 1+2=3, however, both sides evaluate to 3. dBASE expressions where both sides of the equation are provided are called logical expressions. Logical expressions use the relational operators listed below.

Field Name Qualifier

Since most reports have a multitude of data files, it is necessary to qualify a field name in a dBASE expression by specifying the data file. Observe above that the first part of a field name, the qualifier, specifies a data file alias. The data file alias is usually just the name of the data file. The "->" terminates the data file name, and marks the beginning of the actual field name.

dBASE Expression Constants

dBASE Expressions can consist of Numeric, Character or Logical constants. However, dBASE expressions which are only made up of constants are usually not very useful. Constants are usually used within a more complicated dBASE expression.

A Numeric constant is a number. For example, 5, 7.3, and 18 are all valid dBASE expressions containing Numeric constants.

Character constants are characters with quote marks around them. 'This is data', 'John Smith', and ' "John Smith" ' are all examples of dBASE expressions containing Character constants. If you wish to specify a character constant with a single quote or a double quote contained inside it, use the other type of quote to mark the Character constant. For example, "Man's" and ' "Ok" ' are both legitimate Character constants.

Unless otherwise specified, all dBASE Character constants in this manual are denoted by single quote characters.

Constants .TRUE. and .FALSE. are the only legitimate Logical constants. Constants .T. and .F. are legitimate abbreviations.

A date constant may be obtained using the STOD() dBASE function with a character constant as a parameter.

dBASE Expression Operators

Operators like '+' , ' * ', and '<' are used to manipulate constants and fields. For example, 3+8 is an example of a dBASE expression in which the Add operator acts on two numeric constants to return the numeric value 11.

The values upon which an operator acts must have a type appropriate for the operator. For example, the divide '/' operator acts upon two numeric values.

Precedence

Operators have a precedence which specifies operator evaluation order. The precedence of each operator is specified in the following tables which describe the various operators. The higher the precedence, the earlier the operation will be performed. For example, 'divide' has a precedence of 6 and 'plus' has a precedence of 5 which means 'divide' is evaluated before 'plus'. Consequently, 1+4/2 is 3.

Evaluation order can be made explicit by using brackets. For example, 1+2 * 3 returns 7 and (1+2) * 3 returns 9.

Operator Name	Symbol	Precedence
Add	+	5
Subtract	-	5
Multiply	*	6
Divide	/	6
Exponentiation	** or ^	7

Table 7.1

Character Operators

There are two character operators, named "Concatenate I" and "Concatenate II", which combine two character values into one. They are distinguished from the Add and Subtract operators by the types of the values they operate on.

Operator Name	Symbol	Precedence
Concatenate I	+	5
Concatenate II	-	5

Table 7.2

Examples:

```
'John  ' +   'Smith'      becomes  'John  Smith"
ABC' + 'DEF'             becomes 'ABCDEF'
```

Concatenate II is slightly different as any spaces at the end of the first Character value are moved to the end of the result.

```
'John'  -  'Smith   '        becomes   'JohnSmith   ‘
"ABC'  -  'DEF'              becomes    'ABCDEF’
"A  '  -  'D   '             becomes    'AD   '
```

Relational Operators

Relational Operators are operators which return a Logical result (true or false). All operators, except Contained Within, operate on Numeric, Character or Date values. Contain Within operates on two character values and returns true if the first is contained within in the second.

Operator Name	Symbol	Precedence
Equal To	=	4
Not Equal To	<> or #	4
Less Than	<	4
Greater Than	>	4
Less Than or Equal To	< =	4
Greater Than or Equal To	> =	4
Contained Within	$	4

Table 7.3

Examples:

'CD' $ 'ABCD'	returns .T.
8<7	returns .F.
5 = 4 + 1	returns .T.

Logical Operators

Logical Operators return a Logical Result and operate on two Logical values.

Operator Name	Symbol	Precedence
Not	.NOT.	3
And	.AND.	2
Or	.OR.	1

Table 7.4

Examples

.NOT. .T.	returns .F.
.T. .AND. .F.	returns .F.
.NOT.(1+2=3)	returns .F.

Easy Expression Entry

In many parts of CodeReporter, the report designer is prompted for a dBASE expression. For example: sorting, querying, and relating data files all require one or more dBASE expressions.

In the instances where an expression is necessary, CodeReporter generally provides two ways this expression may be obtained: direct entry or through the "Easy Expression" dialog.

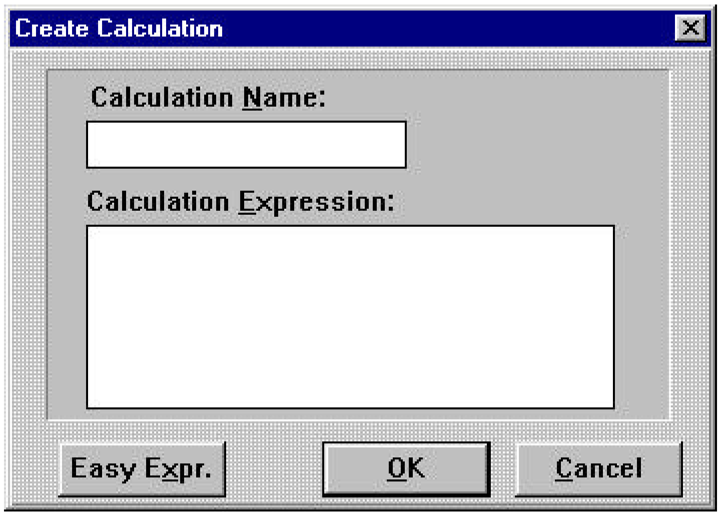


Figure 7.1 Create Calculation Dialog

Figure 7.1 above, illustrates a typical request for a dBASE expression. The "Create Calculation" dialog includes an area ("Calculation Expression" edit control) in which the expression may be directly typed. To use this area, simply position the cursor within the area and enter the appropriate dBASE expression.

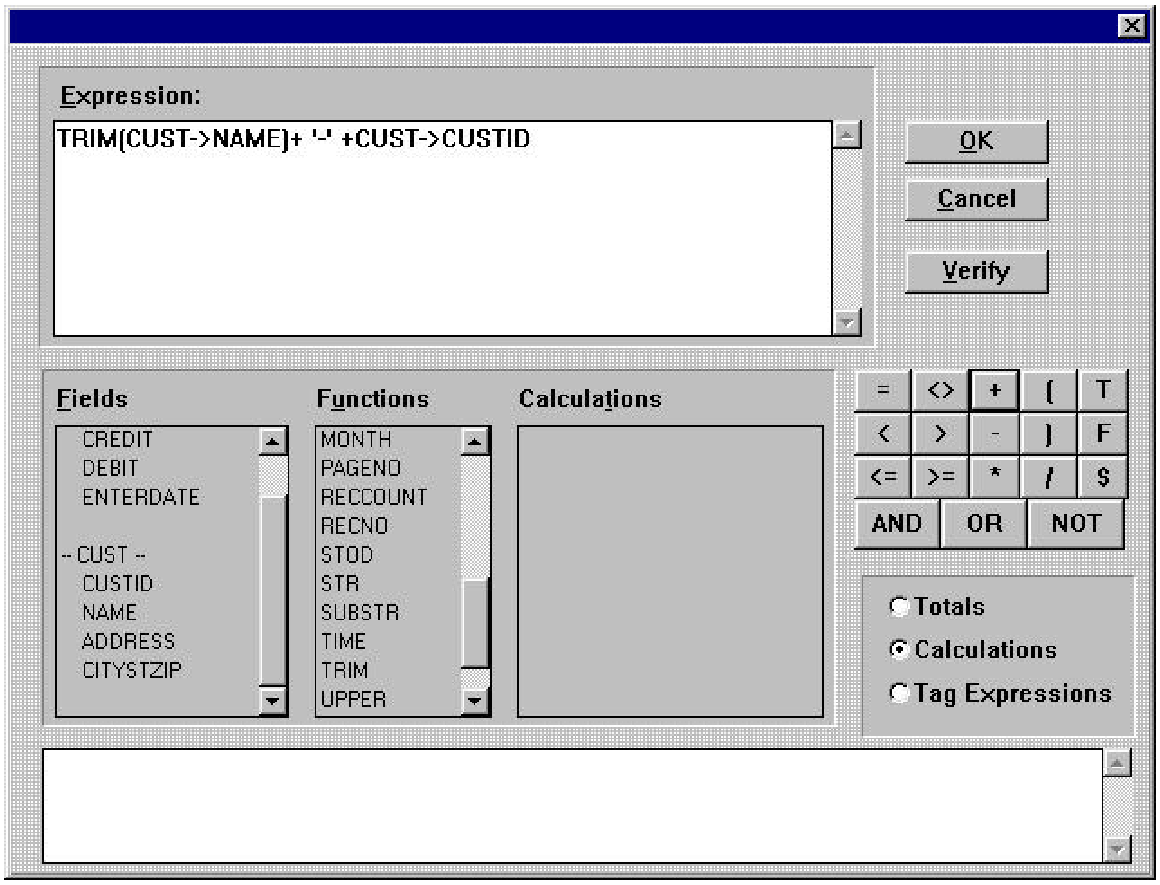
This dialog also shows an entry point for the "Easy Expression" dialog -- the "Easy Expr." button. Pressing this button invokes the "Easy Expression" dialog for point-and-click entry of a dBASE expression. Upon completion, the expression entered in the "Easy Expression" dialog is placed within the "Calculation Expression" edit control.

Easy Expression Entry Dialog

The "Easy Expression" dialog provides the report designer with a point-and-click way to build a dBASE expression. The "Expression" edit control contains the expression as it is constructed. Double clicking upon any of the dialog box's list boxes inserts that item into the expression at the current position of the insertion caret. dBASE operators may be inserted by single clicking the button containing the desired operator.

The advantage of using the "Easy Expression" dialog lies in the speed in which an expression may be constructed. Fields are automatically added with their data file qualifiers, functions are automatically added with their parentheses and commas, etc. The list boxes contain a list of every field in the composite data file, every supported dBASE function, and all the created calculations and totals.

In addition, descriptive information can be obtained about any field or function by single clicking on the item. The descriptive information is displayed within the bottom edit control.



Expression: TRIM(CUST->NAME)+ '-' +CUST->CUSTID

Fields: CREDIT, DEBIT, ENTERDATE, -- CUST --, CUSTID, NAME, ADDRESS, CITYSTZIP

Functions: MONTH, PAGENO, RECCOUNT, RECNO, STOD, STR, SUBSTR, TIME, TRIM, UPPER

( ) Totals [✓ (•) Calculations] ( ) Tag Expressions

Figure 7.2 Easy Expression Dialog

Using Easy Expression

The "Easy Expression" dialog box is simple to use. The controls work together to provide a complete point-and-click environment.

Expression

The "Expression" edit control may be used to manually type in or edit an expression or part of an expression. The dialog's other controls perform their insertions in this edit control at the current caret position. The caret is the flashing vertical line that appears within the control when it has focus.

Fields

The "Fields" list box contains all of the fields of the composite data file. The fields from the individual data files are separated by a "-- *xxxxxxxx* --", where *xxxxxxxx* is the name of the data file.

When a field is selected with a single click of the mouse (or when focus is shifted to the list box), the information window displays information about the field.

A double click of the mouse on a field inserts the field name (with its field name qualifier) within the "Expression" edit control.

Functions

The "Functions" list box contains all of the supported dBASE functions that may be placed within an expression.

When a function is selected, the information window displays a short description of the function -- including parameters (if any) and the function's return type.

A double click on one of the functions inserts that function within the "Expression" edit control and moves the insertion caret between the function's parentheses.

Calculations

The "Calculations" list box contains all of the previously defined report calculations. The calculation entered into the expression is evaluated anew when the expression is evaluated.   It is possible to include a calculation within the expression of other calculations or totals. This "nesting" can provide very flexible report design.

This list box also doubles for the "Tag Expressions" and "Totals" list boxes. These list boxes are made available when the appropriate radio button is selected.

Tag expressions do not have the necessary field name qualifiers required for CodeReporter expressions. These qualifiers must be entered manually if a tag expression is to be used.

Verifying an Expression

An expression entered in the "Expression" edit control can be verified by using the "Verify" button. "Verify" attempts to evaluate the expression. If it fails for any reason -- including incompatible types, incorrect number of parameters for a function, or unrecognized symbols -- CodeReporter reports the errors. If, on the other hand, an expression is correct, CodeReporter indicates that there were no errors in the expression.

Exiting Easy Expression

Once an expression is complete and verified, the "OK" button may be used to close the "Easy Expression" dialog. "Cancel" may also be used, but any changes to the expression are lost.

8. Styles

A professionally designed report not only has all of the necessary information, but also formats it in an easily comprehensible manner. One aspect of a visually appealing report is the typeface, size and color of the text within the report. By using different typefaces, sizes, and colors, important output objects (such as column titles, totals, negative numbers, etc.) can be emphasized.

CodeReporter provides this functionality through the use of styles. A style simply identifies a typeface, size, color, and special characteristics (underlining, bold, italics, etc) by associating them with a name. Whenever the same appearance is desired for an output object, it isn't necessary to re-create the look, simply select a style.

As a report is designed, new styles may also be defined to provide a custom look.

Why Styles

CodeReporter makes use of styles and style sheets for a number of reasons. Foremost is to make customizing a report a simple task. A particular look for the report (or series of reports) need only be defined once. When completed, through the use of styles and style sheets, the process of selecting the customized fonts need not be repeated.

Consistency

In addition to making the design of the report easier, it also promotes report consistency. Instead of having to select a font, size and color for each and every output object within a report (and possibly making a mistake), objects of the same importance only need the style defined once. For example, if all total objects are to be outputted in a special color, it is only necessary to select the color in a style and associate that style with the total objects.

Through the use of style sheets, common styles can be used by multiple reports. CodeReporter can save the style information into a special file that can be loaded into other reports. This provides a quick way to have a consistent look between reports without having to recreate the styles for each new report.

Flexibility

The flexibility inherent in styles and style sheets only becomes apparent when a change needs to be made to the look of a report.   When a font or color must be changed throughout a report, simply change the styles and all of the output objects within the report are automatically updated.

Non-Windows Styles

Using other Sequiter Software products and the CodeReporter API (Application Programming Interface), reports designed within CodeReporter under Windows may be outputted in other operating systems (such as DOS, Unix, OS/2, etc.). Handling the output under operating systems other than Windows presents special problems.

CodeReporter provides a solution to this problem through the use of non-Windows styles. See , below, for more information on outputting reports in other operating systems.

Creating Styles

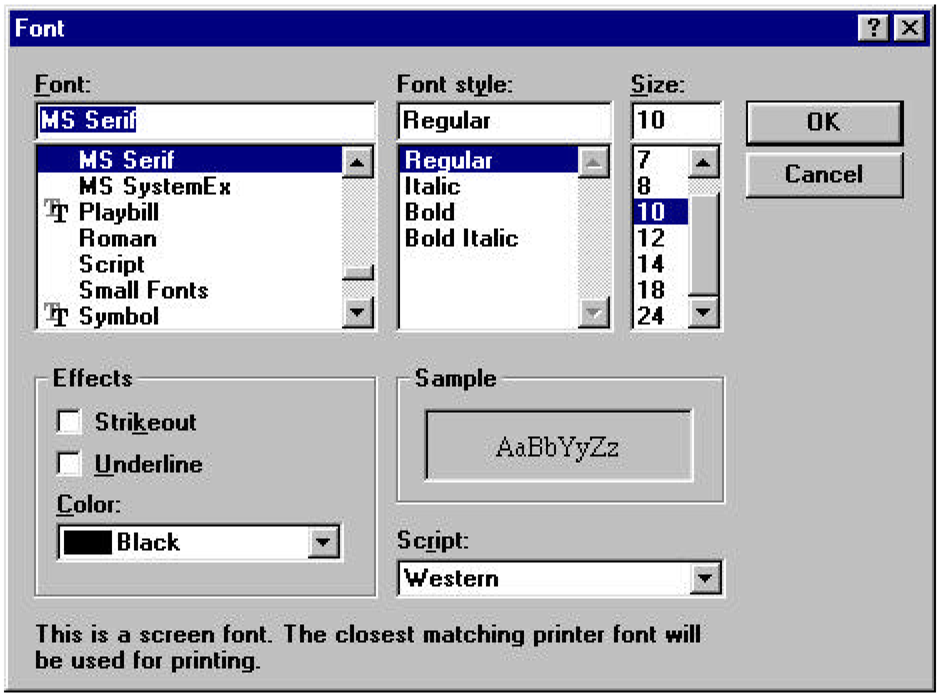
When output objects are created, the default style is used for its output. If no styles have been created, the CodeReporter style "Plain Text" is used as the default style. If other styles have been created, the last selected style is used as the default style for all new output objects.

Styles are created using the **STYLE | CREATE** menu option. Once selected, CodeReporter prompts for the name of the new style. The new style name must be unique for the report. That is, a style may not be created if the report contains a previously created style of the same name.

Once the style name is selected, the "Font" common dialog is displayed. This dialog, like the "File" dialog, is common to most Windows applications. Most Windows word processors will have this dialog, or one similar, for choosing the font information.

Choose the desired font, font size, color, etc. and click on the "OK" button. From this point on, whenever an output object has this new style selected, the specified font, font size, etc. will be used when the object is outputted.

When creating new styles, CodeReporter uses the currently selected style as the basis for the new style. To create several closely related styles, create the first one, select it, and then use it as a basis for the additional ones.



Font: MS Serif
Font style: Regular
Size: 10
Color: Black
Script: Western

This is a screen font. The closest matching printer font will be used for printing.

Figure 8.1 Font Dialog

Deleting a Style

Styles may be deleted by selecting the **STYLE | DELETE** menu option. If there are styles that may be deleted, a submenu appears listing the styles. Selecting a style causes it to be deleted.

If an attempt is made to delete a style that is currently being used by output objects, CodeReporter issues a warning.

Output objects using a deleted style are reverted to the first style in the style popup menu.

CodeReporter must have at least one style defined for each report. As a result, if there is only one style within a report, it may not be deleted.

Modifying a Style

The currently selected style may be modified by choosing the **STYLE | MODIFY** menu option. This invokes the "Font" common dialog so that the custom changes may be made.

When the currently selected style is modified, all output objects using the modified style are automatically updated.

Changing a style to a larger font size does not change the size of the output objects using the style. As a result, the output objects may not display all of the text for the object. Manually change the sizes of the output objects to accommodate the new size.

Selecting a Style

A style may be selected by using the style popup button (See Figure 1 in the Getting Started section of this manual for the location of this button). When depressed, this button creates a popup list containing all of the previously created styles for the report.

A style may be selected from this list with a single or double click. In the case of the single click, however, an additional click on the style popup button is necessary to close the style popup menu. Selecting an output object also selects the style associated with that output object.

For an Object

If there are any output objects selected while a style is selected in this manner, they are set to the new style. This is the quickest way to set a style for an output object.

A style may also be set using the "Style" drop down list box within the "Object Settings" dialog. See the for information on the "Object Settings" dialog.

Non-Windows Styles

Report output within Windows is handled by the various display and print drivers available to Windows. A report outputted on a laser printer looks about the same if outputted on a dot matrix printer. Whether or not a printer supports a particular font is irrelevant, since Windows makes approximations -- or outputs the text as graphics -- to achieve the same result. This is one of the Windows advances that non-Windows applications are not able to take advantage of.

Non-Windows styles are only necessary if a report is to be output in an operating system other than Windows. If a report is only to be outputted in Windows, non-Windows styles may be ignored.

Non-Windows applications are limited to the fonts and characteristics that are supported by the printer; since these issues are solely handled by the printer. To confuse matters even more, different printers activate the same characteristic (eg. Bold letters) differently. One may require the application send it one set of information, while another printer may need something different.

Through the use of non-Windows style definitions, printer specific control codes -- which cause a printer to output text in different fonts -- may be stored in the styles of a style sheet. The advantage in this is that different style sheets may be created for the different printers and selectively loaded at the time a report is run in the other operating system. A 'Bold' style in a style sheet for a laser printer would contain the bold printer control codes for the laser, while the 'Bold' style in a dot matrix's style sheet would contain the appropriate codes for bolding text on the dot matrix.

The CodeReporter API functions don't care what the actual codes are or which style sheet is loaded.

See the CodeReporter API for more information about outputting reports in non-Windows operating systems.

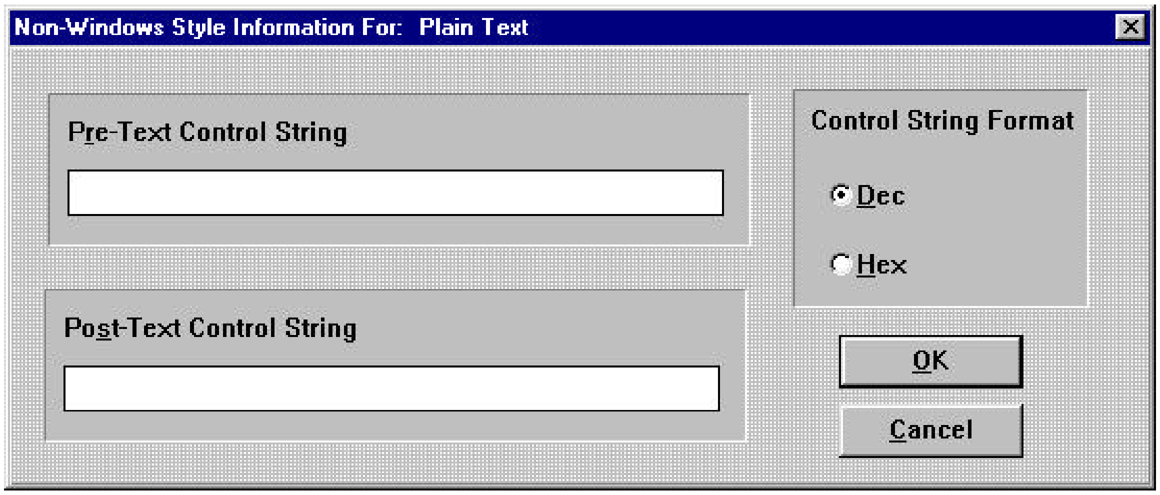
Specifying Styles

As mentioned above, there are printer specific control codes which cause the printer to output text in a different manner -- bold, for instance.

Most printers use two sets of codes to control any specific function or attribute: one set to turn "on" a function, and another to turn it "off." Turning "on" the bold typeface may be ESC "G+", while turning it "off" and resetting the printer to standard mode might be ESC "G-". The actual codes necessary for a font or characteristic is dependant upon the printer used to output the report in the non-Windows operating system. These codes should be listed in the documentation that comes with the printers.

The CodeReporter API functions send the style's "on" codes before the text of each output object, and the "off" codes once the object is outputted.

The special control codes for the currently selected style may be entered using the **STYLE | NON-WINDOWS DEFINITION** menu option. This invokes the "Non-Windows Style Information" dialog.



Control String Format: [✓ Dec] Hex

Figure 8.2 Non-Windows Style Information Dialog

The "Pre-Text Control String" edit control is used to specify the control codes to turn "on" a specific printer font, while the "Post-Text Control String" is used to turn the same attribute "off".

The format of the control string is determined by the "Control String Format" radio buttons. When the "Dec" radio button is selected, the text entered in the dialog's edit controls are interpreted as decimal values -- when the "Hex" radio button is selected, the text entered must be the equivalent hexadecimal values. Each control code entered must be separated by a space. The CodeReporter API does not send the space character -- it is only used as a delimiter between the control codes.

Multiple non-Windows Codes

Multiple attributes for non-Windows output objects, such as emphasized and italic, must be entered within one non-Windows style definition. Simply separate the last code of the previous control setting by a space and begin the new control code sequence. Some printers may require a specific order when combining codes. Refer to the printer's documentation on combining control codes.

As mentioned above, the exact values necessary for any one typeface or attribute (such as bold) vary from printer to printer. See , for a conversion between ASCII, Hexadecimal, and Decimal values.

Example

The following example illustrates the steps necessary to create an italic font (for both a Windows and a non-Windows operating system) and how to specify the new style for an existing output object. It is assumed that the non-Windows printer is an Epson compatible printer.

Start a New Report

Begin a new report by selecting the **FILE | NEW** menu option. Since this example report does not require a specific data file, choose any data file from the .\EXAMPLES directory as the top master data file.

CodeReporter automatically creates a default "Body" group (with a header area) and a default style ("Plain Text").

Create Text Objects

Using the **OBJECT | TEXT** menu option to put CodeReporter into Text Object insertion mode, place two text output objects in the group header area with the following text:

- This is NOT italic
- This IS italic

At this point in the report, both output objects use the default "Plain Text" style which is not italic.

Create the Windows Style

Create a new italic style by selecting the **STYLE | CREATE** menu option. When prompted for the style name, enter a descriptive name for the italic style, such as "Italic," and select the "OK" button.

CodeReporter invokes the "Font" dialog (Figure 8.1) using the currently selected font as the basis for the new font. To set this new style as italic, change the "Font Style" list box so that the "Italic" entry is highlighted and select the "OK" button.

Setting an Object

Creating a new style does not alter any output objects within the report. It is necessary to select the desired output objects and set their new style.

Single click on the text object that contains the text "This IS italic" to select it. Notice that the Styles Popup button indicates that the output object uses the "Plain Text" style. Single click on the Styles Popup button to display the Styles list, and double click on the newly-created "Italic" style to select the style for the selected output object.

Display the report (using the **FILE | PRINT PREVIEW** menu option) to verify that the italic style is being used with the "This IS italic" text output object.

Setting the Non-Windows Codes

The report just created uses the Windows display driver to output the italic and non-italic text of the report to the screen. This example report may also to be outputted in a non-Windows application where the Windows drivers would not be available. It is necessary, therefore, to enter the printer-specific control codes into the newly created "Italic" style so that the report could be outputted correctly in a non-Windows application.

With the "Italic" style selected, chose the **STYLE | NON-WINDOWS DEFINITION** menu option to invoke the "Non-Windows Style Information" dialog.

The Epson-compatible code to select italic printing (as listed in Epson printer manuals) is "ESC+4". If the actual characters "ESC+4" were entered within the "Pre-Text Control String" edit control, and the "OK" button were selected, CodeReporter would display an error. This is because the only supported formats for entering control codes are decimal and hexadecimal -- not ASCII representations of control codes.

The decimal equivalent for the escape key is 27. The four is represented as 52 (See Appendix D for a list of ASCII characters and their decimal/hexadecimal equivalents). Ensure that the "Dec" radio button is selected and enter "27 52" (without the quotation marks) within the "Pre-Text Control String" edit control.

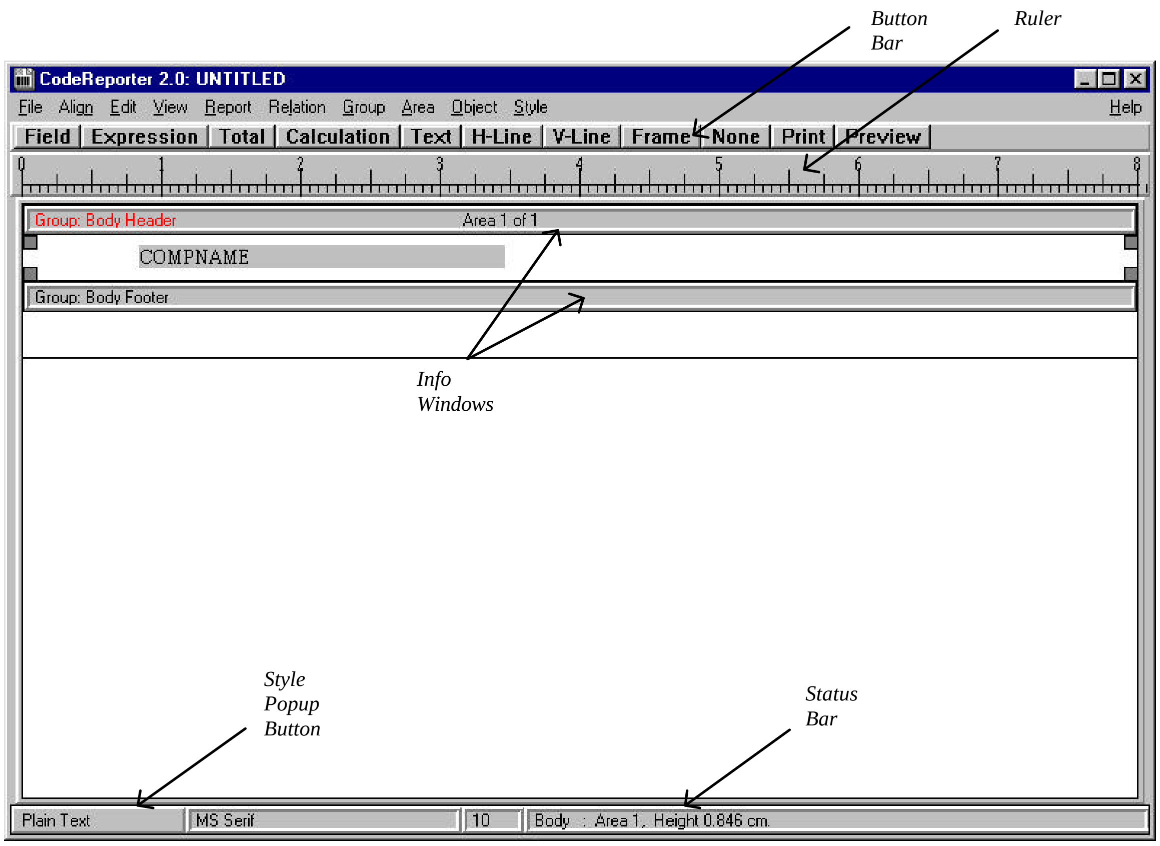
Since it is important to release the italic printing after the output object is printed (failure to do so could cause non-italic objects to be outputted in an italic typeface), specify the codes to turn "off" italic printing. Epson-compatible printers use "ESC+5" to do this. Enter "27 53" within the "Post-Text Control String" edit control and select the "OK" button to complete the non-Windows definition of the "Italic" style.

9. CodeReporter Options

This chapter deals with some of the "cosmetic" preference settings available within CodeReporter. These settings do not affect reports or report files in any way, but merely alter some of the functioning and appearance of CodeReporter.

View Options

When CodeReporter is first invoked, several CodeReporter options are defaulted. Some of the most noticeable are the view options.


Button Bar
Ruler
CodeReporter 2.0: UNTITLED
File Align Edit View Report Relation Group Area Object Style Help
Field Expression Total Calculation Text H-Line V-Line Frame None Print Preview
0 1 2 3 4 5 6 7 8
Group: Body Header
Area 1 of 1
COMPNAME
Group: Body Footer
Info Windows
Style Popup Button
Status Bar
Plain Text
MS Serif
10
Body : Area 1, Height 0.846 cm.

The **VIEW** menu controls the display or lack of display for the various elements of the report design screen: button bar, ruler, status bar, and info windows.

When the various menu options are checked (the default), the report design screen element is displayed. Some elements may take up needed visual room in which the report is designed.

Selecting a checked menu option removes the report design screen element and unchecks the menu option. Selecting an unchecked menu option adds the report design screen element and checks the menu option.

The flexibility of adding and removing report design screen elements allows the report designer the flexibility of customizing his/her working environment.

Report Preferences

Some additional operational preferences can be set within the "Report Preferences" dialog -- such as the preferred unit of measure, the unit of measure of the ruler (if enabled) and the report display height.

Display Units

CodeReporter internally represents the positions and dimensions of output objects and report areas in increments of 1/1000ths of an inch. However, since the average report designer does not need this amount of precision when placing output objects, CodeReporter uses customizable units of measurement for the design interface.

The default unit of measurement for CodeReporter is inches. This may easily be changed by using the "Report Preferences" dialog and selecting another radio button in the "Units" area. This unit is then used whenever CodeReporter requires or provides a measurement. This includes the dimensions of the report and report areas, the coordinates and dimensions of output objects, margins, etc.

CodeReporter automatically changes its display of all parts of the report to reflect the new unit of measurement. CodeReporter doesn't actually change the position/size of anything within the report when this setting changes, since everything is always internally represented in 1/000ths of an inch.

The only aspects of CodeReporter that are not affected by the "Units" setting are the common dialogs (eg. "Font") and the ruler. Since the common dialogs may not be modified by CodeReporter, they are unaffected by the "Units" setting.

Ruler

The ruler is used as a visual aid for horizontally placing and moving output objects. As an example, some reports may be printed on special company invoices that have a specific column for some figures. Instead of attempting to place output objects by "trial and error", a physical ruler may be used to measure the document and the CodeReporter ruler may be used to quickly place the object.

The units of measure for the ruler are set within the "Report Preferences" dialog. Selecting the "Inches" or "Centimeters" radio buttons in the "Ruler" section sets the units used in the ruler.

This setting may be different than the "Units" setting.

	Inches	Points	Centimeters
1 Inch		72	2.45
1 Point	0.014		0.34
1 Centimeter	0.41	29	

Table 9.1 Units Conversion Table

View Page Size

CodeReporter reports may be previewed on the screen before actually being outputted to the printer. This is done using the **FILE | PRINT PREVIEW** menu option.

The preview of reports is often done simply to see the information within a report, or to see how pages are reset. This is facilitated by CodeReporter setting the preview page size equal to the size of the display screen. In effect, the vertical page size of the report is shrunk to the size of the display screen so that the full-sized page header and page footer may both be seen simultaneously.

The report may be previewed in the same size as it is printed by unchecking the "Page size equal to screen size when viewing report" check box in the "Report Preferences" dialog.

Unchecking this option causes the report to be displayed in the same size font as is printed, and on the same size display page as printed. However, since most display screens are not as large as most pages, unchecking this check box causes vertical scroll bars to appear.

The "Page size equal to screen size when viewing report" check box has no effect on the actual printing of a report. This setting only affects the size of the page used when the File | Print Preview menu option is selected.

10. Customizing Reports

CodeReporter makes certain assumptions concerning all new reports. Aspects such as page size, report width, margins, numeric formatting, etc. are all set to default values when a new report is created.

However, it is not always the case that these default settings are appropriate for the report designer, or for the report itself. All reports are not outputted on the same size paper, nor do they use the same margins. Many reports require that special custom changes be made in order to satisfy the report's demands.

These and other settings may be customized once a report is initiated by using two dialog boxes: "Margins" and "Report Preferences"

Margins and Page Size

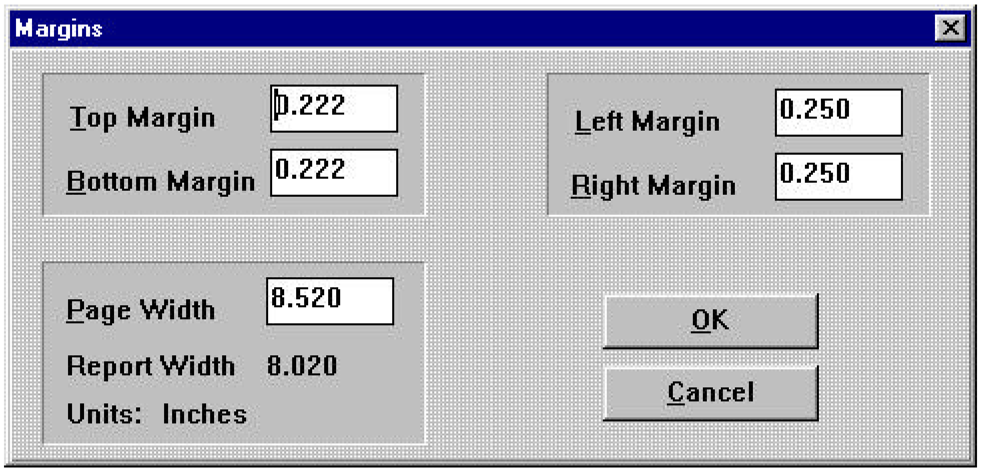
The "Margins" dialog, which is invoked using the **REPORT | MARGINS** menu option, is used to change both the margins of the report and the width of the page.

The margins and page size are automatically set according to the maximum printing area available on the Windows default printer. For most dot matrix printers, this means that the report initially is set up to cover the entire surface of the page. Some other printers, notably laser printers, have a hardware margin that prohibits the output of text beyond a certain point. CodeReporter automatically detects this and, as a safety feature, does not allow margins of a report to be less than the hardware margins for the selected printer.

Margins

The margins of the report are determined by the various margin settings within the "Margins" dialog. To alter the top, bottom, left or right margins, simply change the value within their respective edit controls.

The page header area(s) and page footer area(s) are outputted between the top and bottom margins. For example, if the top margin was set to one inch, the upper edge of the first page header area would be outputted at one inch.



Top Margin 0.222
Bottom Margin 0.222
Left Margin 0.250
Right Margin 0.250
Page Width 8.520
Report Width 8.020
Units: Inches

Figure 10.1 Margins Dialog

To make it appear that portions of the page header or page footer are outside of the top and bottom margins of a report, set the top/bottom margins to a small value, and then increase the size of the page header/footer areas.

Page Width

As mentioned above, the page width for new reports is determined by the current page setting for the Windows default printer. If the ultimate destination printer is wider or narrower than the default printer, the "Page Width" setting may be used to change the width used by CodeReporter.

The height of a page is determined from the selected printer when the report is outputted.

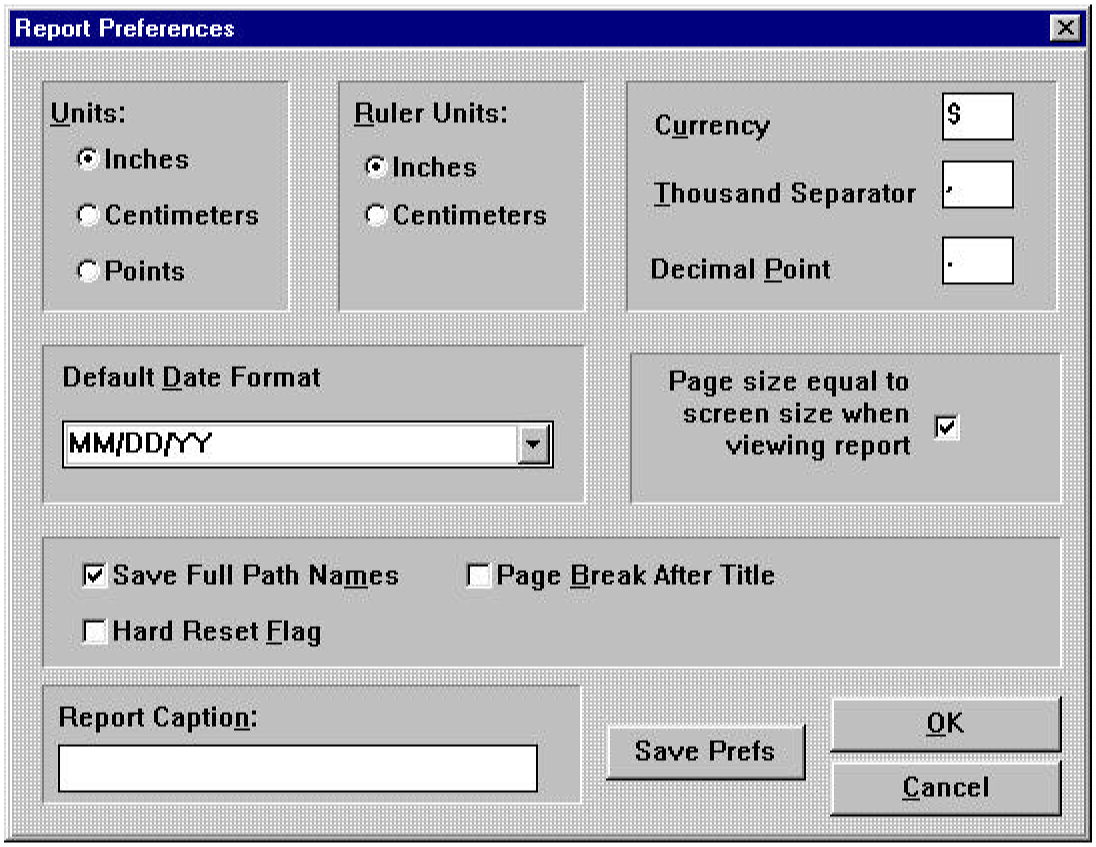
It is up to the person outputting the report to ensure that the specified printer is properly set up within Windows -- including the correct paper size and orientation.

Orientation

The orientation of a report (landscape or portrait) is determined by the set up of the printer at output time. This may be either set through Windows Control Panel, or temporarily through the **FILE | PRINTER SELECTION** menu option. See for information on setting the page orientation.

Report Preferences

The "Report Preferences" dialog can be used to change some of the report-wide default settings that affect the actual output of the report -- including the default formatting of numbers and dates.


Report Preferences

Units: [✓ Inches] Centimeters Points

Ruler Units: [✓ Inches] Centimeters

Currency: $

Thousand Separator: ,

Decimal Point: .

Default Date Format: MM/DD/YY

Page size equal to screen size when viewing report: ✓

Save Full Path Names: ✓

Page Break After Title: ☐

Hard Reset Flag: ☐

Report Caption:

Figure 10.2 Report Preferences Dialog

Numeric Format

Most of the settings that affect output objects that evaluate to numeric results are set on an object-by-object basis through the "Object Settings" dialog. Those settings include: the number of decimals displayed; whether the object is a percent, a currency value or a simple number; how a zero value is outputted, etc.

Some settings for numeric output objects are changed on a report-by-report basis. These include the use of the thousands separator, the currency symbol for the report, the character used for a decimal point, etc. The "Report Preferences" dialog is used to modify these settings for the current report.

Currency

When numeric output objects are set to be displayed as currency values the character(s) specified in the "Currency" edit control are outputted immediately before the numeric contents of the object   (See for formatting output objects as currency values). The default currency symbol is the "dollars" ($) symbol, however, up to ten characters may be specified.   Table 10.1 list some common currency symbols that are supported by the standard Windows character set.

Character	Name	Key Strokes
¢	Cent	Alt+0162
£	Pound	Alt+0163
¥	Yen	Alt+0165

Table 10.1 Common Currency Symbols

Thousands

Numeric values greater than one thousand may use a character placeholder to format the output to the thousand, million, billion, etc. place. The normal North American separator -- and the CodeReporter default -- is the comma.

The "Thousand Separator" edit control may be used to change this value to any single character (including a space) or it may be deleted completely.

If a thousand separator is specified, all numeric values use it as a placeholder. If it is deleted, the numbers are outputted in raw form.

Example:

Using a thousand separator	1,000,000
Using no separator	1000000

Decimal Point

When CodeReporter outputs fractional numbers it places the character specified in the "Decimal Point" edit control between the whole number and the fractional part. Any character may be entered in this edit control and used to separate fractional numbers from whole numbers.

In North America, the period is used to divide fractional numbers from whole numbers. In some countries, other characters such as a comma are used.

This setting has no bearing on the number of decimal places outputted for an object. (See for setting the number of decimal places)

Date Format

Output objects that evaluate to a date value (eg. date fields, date expressions) may be outputted in a variety of formats. New date output objects use the setting in the "Default Date Format" drop down combo box to determine how they are outputted initially.

A specific date format picture may be entered into the edit portion of the combo box, or one of the pre-defined date formats may be chosen from the drop down list.

See for more information on the date format picture.

Changing the value in the "Default Date Format" drop down combo box does not affect the date format for any previously created date output

objects. The "Default Date Format" setting only affects new output objects.

Path Names

CodeReporter can use several data files within a report that may be located across several different drives and directories. CodeReporter does this by saving the full drive and directory path to each data file used in the report. While this provides flexibility in regards to the possible sources of the information, it also requires that the data files must be where they were when the report was created.

The "Save Full Path Names" check box within the "Report Preferences" dialog determines whether or not the complete path to the data files in the report are saved with the report.

If this check box is not checked, CodeReporter does not save the drives and directories to the data files. The next time the report is retrieved, CodeReporter is unable to access the data files in their original paths (since they were not saved) and assumes the data files are in the current directory.

Hard Reset

When a group encounters a group reset condition and it has the Reset Page option set (see), the "Hard Reset Flag" check box is used to determine the method of resetting the page.

If "Hard Reset Flag" is not enabled (the default), a new page is not generated if the group with the Reset Page option was reset as a result of a higher level group resetting. Only if it was the group's group expression itself that caused the group reset condition, is a new page generated. When "Hard Reset Flag" is checked, a new page is generated regardless.

See Figure 10.3 in the CodeReporter manual for an illustration of both settings.

Page Break After Title

The "Page Break After Title" check box determines whether or not CodeReporter should begin the report on a new page after the title area has been outputted. If this check box has been checked, the title area, if used in the report, is outputted and the report continues on a separate page.

If the check box is not checked, the title area is outputted on the same page as the header area for the first group.

Report Caption

The CodeReporter Print Preview window, by default, displays "CodeReporter 2.0" in the window's title. This title may be customized to reflect the name of the report or any appropriate title by changing the value of the "Report Caption" edit control in the "Report Preferences" dialog.

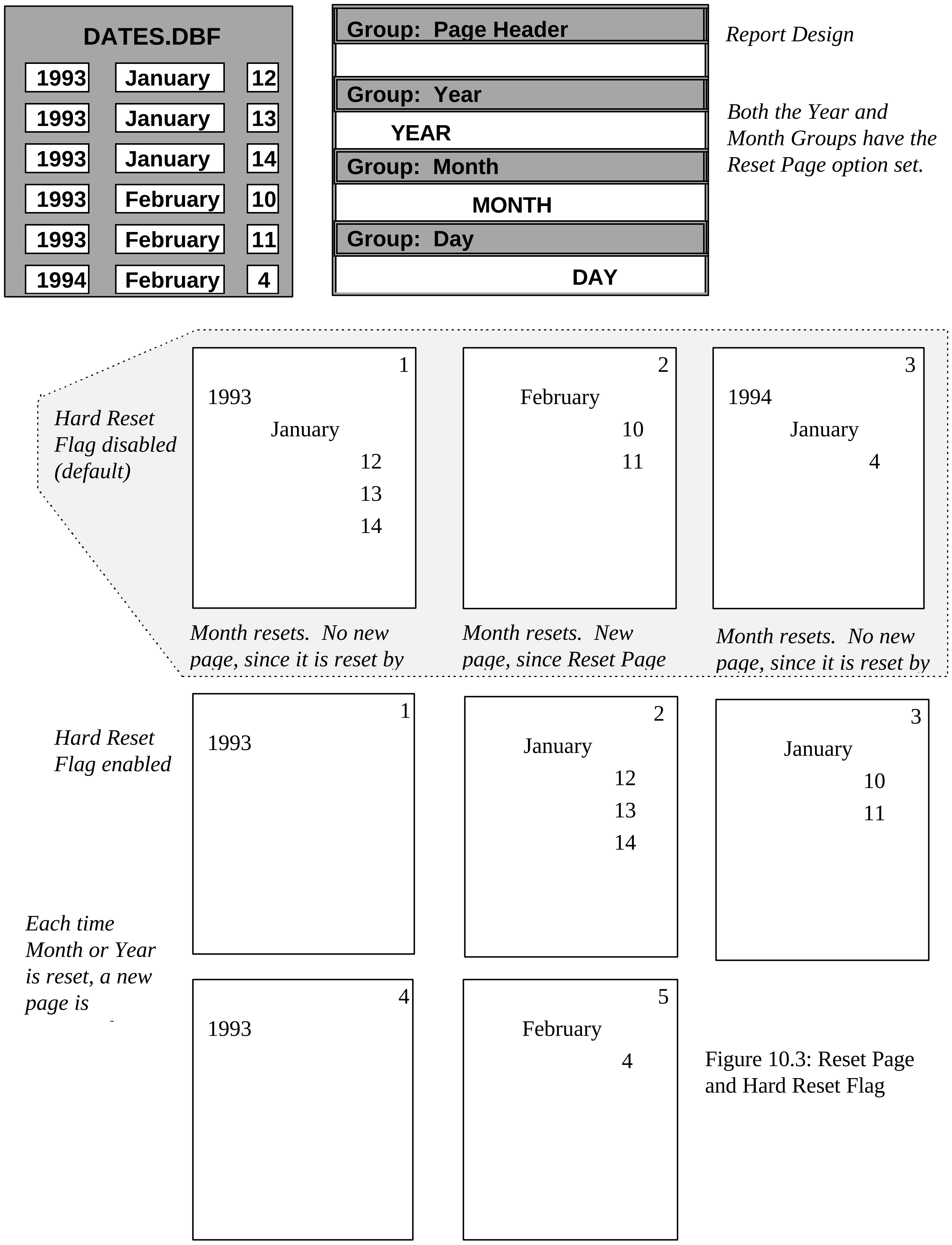


Figure 10.3: Reset Page and Hard Reset Flag

11. Printing

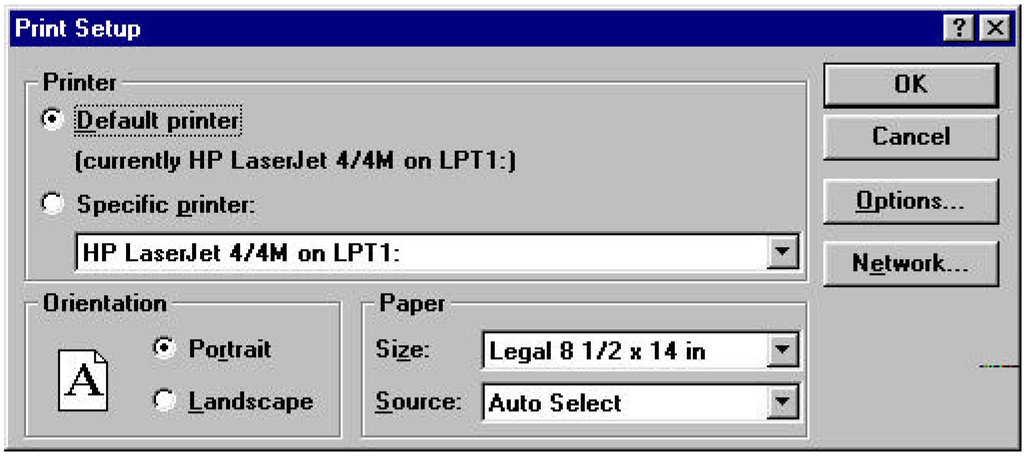
The ultimate goal of any report is to have it on paper. The usual destination of a report is a printer. However, this is not always the case. In some situations, it is desirable to output a report to the computer's screen or to a file. This chapter discusses the procedures necessary to output a report.

Selecting a Printer

Many hardware setups have a single computer hooked up to a single printer and all printed output from the computer goes to the one printer. In this case, selecting a printer is easy -- there's only one to choose from.

Other setups, however, may have a single computer hooked up to one or more printers locally and/or via a network. Unless CodeReporter is told otherwise, it uses the Windows default printer for all printed output.

If the Windows default printer is acceptable, no changes need to be made. Any other printer must explicitly be selected using the "Print Setup" common dialog. This is invoked with the **FILE | PRINTER SELECTION** menu option, or the "Setup" button in the "Print" common dialog.



Printer

[✓ Default printer]
(currently HP LaserJet 4/4M on LPT1:)

Specific printer:
HP LaserJet 4/4M on LPT1:

Orientation

[✓ Portrait]
Landscape

Paper

Size: Legal 8 1/2 x 14 in
Source: Auto Select

Figure 11.1 Print Setup Dialog

To select a non-default printer, choose the "Specific Printer" radio button and use the drop down list box to select one of the installed printers.

This dialog is also used for selecting landscape/portrait mode, different sizes of paper, etc. Select the options appropriate to the report and then choose the "OK" button.

The settings in the Print and Print Setup dialogs are not saved with the report, and so may need to be reset each time the report is loaded.

To the Screen

CodeReporter provides a way to preview a report before it is outputted to a printer. Displaying the report to the screen is a useful way of ensuring the report is correctly laid out -- without wasting several sheets of paper.

The **FILE | PRINT PREVIEW** menu option creates a new window and outputs the report, page-by-page, within it. CodeReporter creates the window maximized (covering the entire screen), however it may be minimized or resized using the window's maximize/minimize buttons and the sizing bar.

The **NEXT** menu option of this window is used to move to the top of the next page. **CLOSE** is used to exit the report preview screen and return to the CodeReporter design screen.

When a report is previewed, the vertical page size of the report may be temporarily set to the size of the screen. This is optional and may be changed to use the page size for the selected printer. See the for more information.

To a Printer

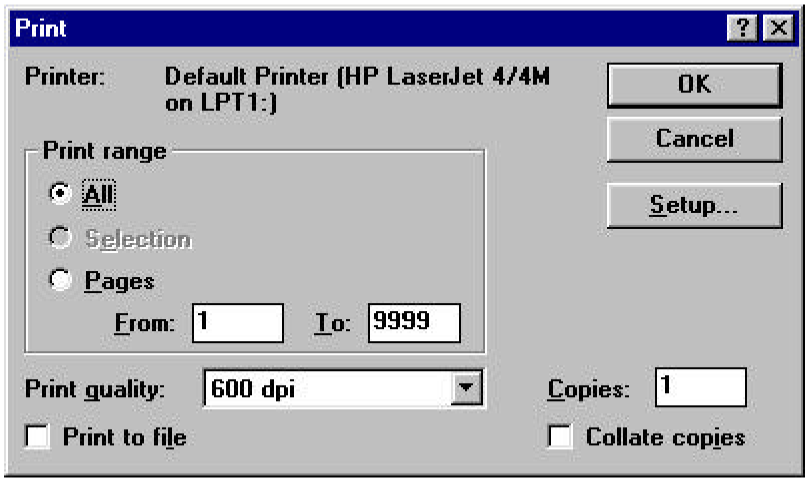
Once a report is previewed and appears to be correct, the report may be outputted to the selected printer using the **FILE | PRINT** menu option. This invokes the "Print" common dialog.

Select the number of copies, and/or use the "Setup" button to select and configure the desired printer. Use the "OK" button to begin printing, or "Cancel" to return to the CodeReporter design screen without printing.

In the context of a report, which does not have set page breaks, setting a range of pages to output doesn't make sense. The page range settings are ignored by CodeReporter.

Setting the "Print Quality" drop down combo box to 'Draft' may cause the printer driver to re-adjust the vertical and horizontal spacing of output objects to the character boundary.

This can cause reports that preview as single spaced to be outputted as double spaced. Either set the "Print Quality" to 'High', or adjust the size of the report's report areas so that they are 12 points high (or a multiple thereof).



Printer: Default Printer (HP LaserJet 4/4M on LPT1:)

Print range

[✓ All]
Selection
Pages
From: 1 To: 9999

Print quality: 600 dpi

Copies: 1

☐ Print to file

☐ Collate copies

Figure 11.1 Print Dialog

To a File

A printed version of the report may be saved in a file for a number of reasons. Either to print the report at a later time, transfer it to another program, or simply to save the report in electronic form.

The following steps are taken to print a report to a file.

1. Select the **FILE | PRINT** menu option to invoke the "Print" dialog.
2. Choose the "Print to File" check box.
3. Choose the "OK" button to begin printing.
4. When the "Print to File" dialog box appears, enter the file name (and path if desired) under which the printed report is to be saved. This file name should be different than the name of the report file (eg. if the current report is "INVOICE.REP" don't save the printed report under "INVOICE.REP").

Print vs. View

A file containing a printed report holds all the information necessary to output a report in the desired font, size, position, etc. As mentioned under , Windows uses printer drivers to print just about anything on any printer.

When printing to a file, Windows dumps all of the printer specific codes for doing lines, graphics, fonts, etc. into the file. If the file is later copied to the printer, the report appears exactly the same as if it had been printed directly from the application. There are two general disadvantages to printing a report to a file: first, the file can become quite large -- especially if the report is quite long and uses several different fonts, graphics, etc. secondly, since all of the printer codes are stored within the file, the file is usually illegible when viewed within another application (such as a text editor).

Using the Generic Printer

As an alternative to storing all of the printer specific information within the file, Windows has a special generic print driver which only outputs text. Graphic elements (lines, frames, graphic objects), and fonts are not outputted. Simply the text of the report is outputted. When this driver is used to print a report to a file, it has the advantage of reducing the size of the file, and making it legible to other applications.

This generic print driver is a part of Windows, and must be installed before CodeReporter can utilize it. Install this driver using the Windows Control Panel -- Printers application.

1. Select the "Add" button from the "Printers" dialog,
2. Select the "Generic / Text Only" printer from the "List of Printers" list box, and
3. Select the "Install" button.

This new printer can then be selected and used to output reports rapidly to text only printers or to ASCII files.

When outputting a report to an ASCII file using the generic printer driver, it is important to set the report areas to 12 points high (or a multiple thereof) in order to have the report saved with the correct spacing.

To a Database File

An important feature of CodeReporter is the ability to output a report to a data file. This means storing the results of the report, including totals, calculations, fields, etc. into a data file which may then be used as the basis for another report.

This is accomplished by specifying the output objects to be included in the final output data file. However, since most output objects are dynamic and have their value change from composite record to composite record, it is necessary to also specify when the values of the selected output objects are written to the output data file.

Both of these tasks are accomplished with the "Output File Template" dialog , which is invoked from the **REPORT | OUTPUT FILE TEMPLATE** menu option. The settings made in this dialog are saved with the report, and may be modified at any time.

Objects

The "Objects" list box lists all of the output objects within the report -- excluding memo fields, lines, frames, graphics and objects within the title/summary and page header/footer areas.

When an output object is selected the "Object Info" window displays information about the output object. Adding an output object to the output data file is a simple matter of selecting the object and using the "Add >>" and "Add All >>" buttons.

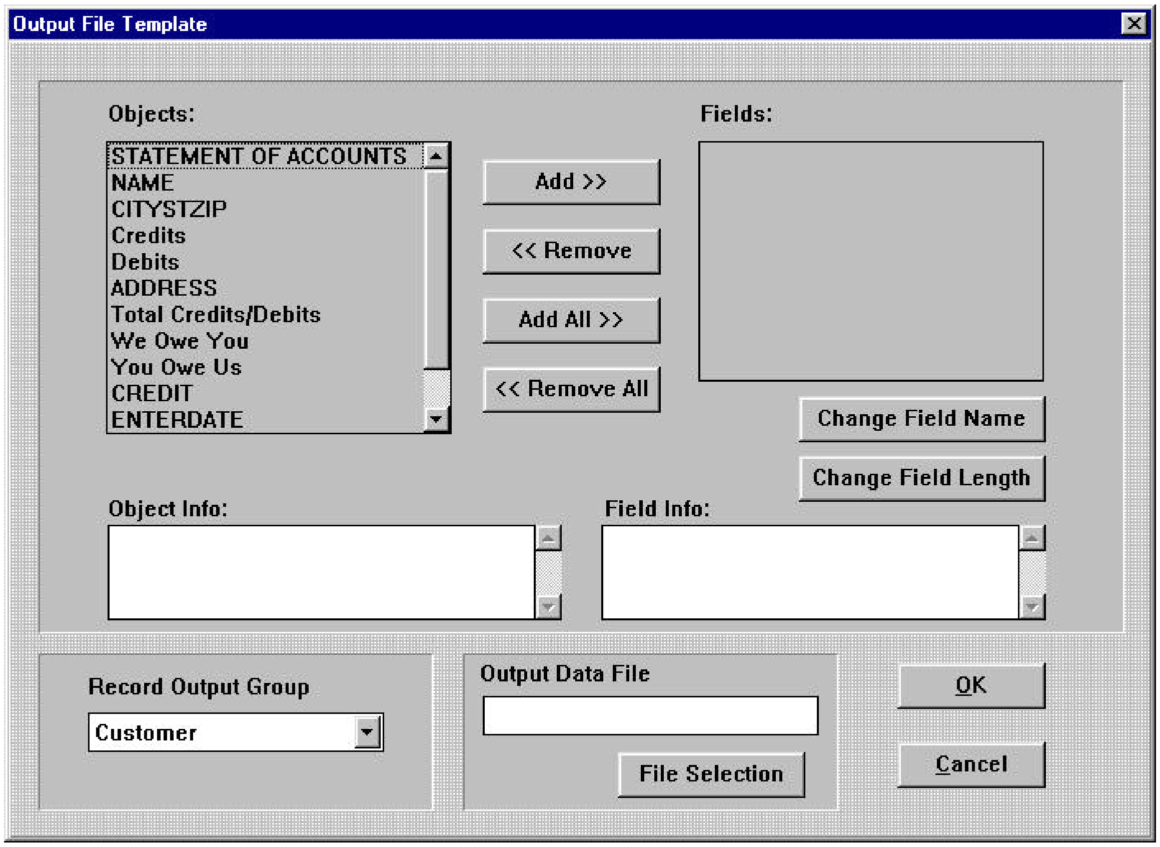


Figure 11.3 Output File Template Dialog

In cases where the output object's identification is not a valid field name (such as the case for totals and calculations), or if a field of the same name has already been added to the output data file, CodeReporter prompts for a new destination field name.

Once added, the particulars of the field may be modified using the "Change Field Name" and "Change Field Length" buttons.

Record Output Group

CodeReporter uses the reset expression of the "Record Output Group" to determine when to write the contents of the field output objects into a new record. When the specified group encounters a reset condition, a new record is created on disk and the values of the output objects are written to this record using the values obtained in last record of the previous group's subset (i.e. the group footer).

The "Record Output Group" drop down list box, by default, lists the outer most group in the report. If another group is desired, it may be selected using this control.

Output Data File

The "Output Data File" section lists the file name of the data file in which the report is saved. To set this name, use the "File Selection" button to invoke the "Select a Datafile" dialog (a common File Open dialog) to specify the drive, directory, and file name of the destination output data file.

If the specified data file exists when the report is printed to it, CodeReporter prompts before overwriting it.

Selecting the "OK" button saves the new data file template within the current report.

Print to Datafile

Once a data file template is designed, the report may be outputted to the data file using the **FILE | PRINT TO DATAFILE** menu option. When selected, this menu option creates the data file specified in the "Output File Template" dialog and fills it with information from the report. This new data file may then be used in the exact same manner as any other data file.

Example

This short example demonstrates the usefulness of printing a report to a data file. This example will take the PERSONEL.DBF data file, sort it, and transform it into the PERSONS.DBF data file.

While CodeReporter is running, select the **FILE | NEW** menu option to initiate a new report and select the PERSONEL.DBF data file as the top master.

Add the Fields

Add all of PERSONEL.DBF's fields to the report by using the "Field" button on the button bar, selecting all of the fields, and clicking within the Body group. Since this report is primarily being used as a data transformation tool, it doesn't really matter where the fields are placed within the area.

Create an expression object

PERSONEL.DBF has the people's names separated into first and last name fields (FNAME and LNAME, respectively). Suppose the PERSONS.DBF data file needed to have the name in a single field, in the format "Smith, John" and "Andrews, Peter".

Using CodeReporter to do this is simple. Merely delete the two name fields, and add an expression output object that contains both fields. For example:

```
TRIM( LNAME ) + ', ' + FNAME
```

If the report were previewed at this point, it would show all of the information in the PERSONEL.DBF data file, but with a combined name field.

Create the template

Since all the fields are now added to the report, it is time to generate the output file template. Select the **REPORT | OUTPUT FILE TEMPLATE** menu option and invoke the "Output File Template" dialog (Figure 11.3). Use the "Add All >>" button to add all of the output objects in the report to the new data file.

As this occurs, CodeReporter indicates that the expression output object doesn't have a field name that can be added to the new data file. Enter "NAME" into the "New Name" edit control and select "OK".

Select the "File Selection" button and in the resultant dialog, enter "PEOPLES.DBF" as the name for the new data file. All other settings in the "Output File Template" dialog are appropriate, so select "OK" to save the template.

Sorting the data fileIf desired, use the **REPORT | SORT EXPRESSION** menu option to enter a sort expression for the report. When the report is outputted, the records in the output data file are written in this sorted order. An appropriate sort expression may be LNAME+FNAME, EMPID, or SALARY.

Generate the data file

The new PEOPLES.DBF data file is generated by selecting the **FILE | PRINT TO DATA FILE** menu option.

Load the new PEOPLES.DBF using **FILE | NEW** (saving the current file if desired) to verify that the information was written as expected.

12. Function Reference

This chapter documents the functions and techniques for using the report module in C/C++. Visual Basic and Delphi programmers, please refer to Appendix F.

The report module functions are used to build custom reports. With the report module, you can easily summarize, format, display and print information in data, index and memo files.

All of the report module functionality can be accessed indirectly via CodeReporter. CodeReporter uses the report module to generate soft-coded report files and their C source code. The generated source code is full of calls to the report module functions.

In some cases, there may be a need to create an entire report by hand or modify a report designed with CodeReporter. The application might use the report module to load a report designed with CodeReporter, change the query and/or sort expressions, specify where to output the report, and then finally execute the report. Report functions are used directly in order to accomplish these custom tasks.

The report module contains no function for creating calculations. This is because the calculation creation function, **exp4_create**, is part of the CodeBase expression evaluation module. Refer to the CodeBase expression evaluation module and **obj4calcCreate** for more details.

Report Module Names

The functions of the report module are grouped together by functionality. Report functions designed to effect the entire report are grouped together; functions to create and manipulate groups are grouped together, etc. In a sense, the report "module" is actually a number of modules.

area4

Each group may contain one or more areas where output objects may be placed. Functions that change the size of the areas, and determine their suppression conditions begin with **area4**.

group4

Each report contains a number of groups. The functions that change the way a group acts, and is accessed, begin with **group4**.

obj4

The functions used to create, free, and modify output objects begin with **obj4**.

report4

These are the main report functions. They are used when something applies to the entire report. These function initialize the report, load and save relations and styles, change the page size, change the report's default settings, etc. The report specific functions begin with **report4.**

style4

Every output object can be assigned an output characteristic such as typeface and color. These characteristics, called styles, need only be defined once in a report, and not for every output object that uses them. The **style4** functions create and modify the styles for a report.

total4

The definitions upon which total output objects are based are created and freed are set using these functions. Creation of the actual total output object is not done with these functions, but rather with the **obj4** functions.

The following sections document all of the above "report modules".

An application can only load one report at a time. If an application is to use more than one report, **report4free** must be called between each report**.**

Saving As Code

When CodeReporter saves reports to disk with the **FILE | SAVE** menu option, it does so in soft-coded report files with the extension of ".REP". This is adequate for most uses, including end-user applications, since these report files may be loaded while the application is running. This provides the developer the flexibility of modifying a report layout without having to re-compile the application that uses the report.

In some instances, it may be desirable to save the report directly as source code. This is done using the **FILE | SAVE AS CODE** menu option. This invokes the "Specify Report File for Save" dialog which prompts for the file name and source code language for the code.

CodeReporter generates language specific source code for the currently loaded report and saves it in the specified file. The source code file contains two functions, which may be called in an application to load the report and/or relation. These two functions must be prototyped in the application prior to their use but may be renamed in the generated source file (and prototype) as desired.

buildRelate

Usage: RELATE4 *buildRelate(CODE4 *cb, int openFiles)

Description: This function creates and/or populates a **RELATE4** structure for the relation of the saved report. This structure may be used with the CodeBase 5 relation module, or with the report module function **report4init**.

This function is automatically called by **buildReport**.

Parameters:

cb This is a pointer to the application's **CODE4** structure. This is used for memory management and error handling.

openFiles This parameter determines whether **buildRelate** should automatically open the data files referenced within the report file. If *openFiles* is a true value (non-zero) the data files for the report are opened if they are not already open. If *openFiles* is a false value (zero) the data files are assumed to be open.

Returns: This function returns a valid pointer to a **RELATE4** structure if successful. NULL is returned if the data files for the relation could not be found.

buildReport

Usage: REPORT4 *buildReport(CODE4 *cb, int openFiles)

Description: This function builds the report and returns a pointer to the populated report structure. **buildReport** automatically calls buildRelate to build the relation behind the report.

Parameters:

cb This is a pointer to the application's **CODE4** structure. This is used for memory management and error handling.

openFiles This parameter determines whether **buildRelate** should automatically open the data files referenced within the report file. If *openFiles* is a true value (non-zero) the data files for the report are opened if they are not already open. If *openFiles* is a false value (zero) the data files are assumed to be open.

Returns: This function returns a valid pointer to a **REPORT4** structure if successful. NULL is returned if the data files for the relation could not be found, or if there wasn't enough memory to build the report.

Using Report Functions

The report functions provide a method of designing complex reports in and out of the Windows environment. Hand-writing all the code necessary to create a report, in most cases, is a lot of work. As a solution, CodeReporter has two options that are useful to the application developer: saving a report as a soft-coded report file (.REP extension) and saving the report as C source code. When a report is saved with CodeReporter, it is placed within a soft-coded report file (.REP) that can be directly loaded into an application. With a few function calls, a report can be loaded and quickly executed. In the majority of cases, this is sufficient.

In the cases where these report files are inappropriate, the report can be saved as C source code by CodeReporter. This code can be used without having the time consuming burden of hand-coding every function needed to create the report.

There are very few cases where a soft-coded report file, or CodeReporter-generated source code will not meet the reporting needs. The CodeReporter report module functions are provided so that a report can be modified or built from scratch to meet the application's reporting demands.

Using a Report File

The most common, and flexible, case of implementing a report in an application is by loading a soft-coded report file and executing the report.

PROGRAM

REP1.C Using a CodeReporter report file.

```
#include "d4all.h"

#ifdef __TURBOC__
   extern unsigned _stklen = 10000 ;
#endif

void main( int argc, char *argv[] )
{
   CODE4  cb ;
   REPORT4 *report ;
   if( argc < 2 )
      return ;

   code4init( &cb ) ;

   report = report4retrieve( &cb, argv[1], 1, NULL) ;
   if( report )
```

```
  {

     report4do( report ) ;
     report4free( report, 1, 1) ;
  }

  code4initUndo( &cb ) ;
  return ;
}
```

Function **report4retrieve** loads the specified report file from disk, creates an internal **REPORT4** structure, and returns a pointer to the structure. This pointer is then used to execute the report with function **report4do**.

The code in REP1.C is a generic way of displaying any report file, since a report file name is specified on the command line. The file name is all that is needed to load a report from disk. The entire report, including the names of the data files, is stored in the soft-coded report file. All that is necessary to display the report is to call function **report4do**. The internal functions take care of the rest.

If this same application were to be written for Windows, the following code might be used.

PROGRAM

WREP1.C   Using a CodeReporter report file under Windows.

```
#include <windows.h>
#include "d4all.h"
#include "r4report.h"

#define IDM_DOREPORT 101
static char *reportName ;
long FAR PASCAL WndProc (HWND, UINT, WPARAM, LPARAM) ;

int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                        LPSTR lpszCmdParam, int nCmdShow)
{
    static char szAppName[] = "WREP1" ;
    HWND        hwnd ;
    MSG         msg ;
    WNDCLASS    wndclass ;
    reportName = lpszCmdParam ;

    if (!hPrevInstance)
    {
        wndclass.style         = CS_HREDRAW | CS_VREDRAW ;
        wndclass.lpfnWndProc   = WndProc ;
        wndclass.cbClsExtra    = 0 ;
        wndclass.cbWndExtra    = 0 ;
        wndclass.hInstance     = hInstance ;
        wndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION) ;
        wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
        wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
        wndclass.lpszMenuName  = "MAINMENU" ;
        wndclass.lpszClassName = szAppName ;

        RegisterClass (&wndclass) ;
    }

    hwnd = CreateWindow (szAppName, "Application  Window",WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
CW_USEDEFAULT, NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, nCmdShow) ;
    UpdateWindow (hwnd) ;
    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }

    return msg.wParam ;
}

long FAR PASCAL WndProc (HWND hWnd, UINT message, WPARAM wParam
                                                            , LPARAM lParam)
{
    static CODE4 cb ;
    static REPORT4 *report ;

    switch (message)
    {
        case WM_COMMAND:
            switch( wParam )
            {
               case IDM_DOREPORT:
                  report = report4retrieve( &cb, reportName, 1, NULL ) ;
                  if( report )
                  {
                     report4parent( report, hWnd ) ;
                     report4toScreen( report, 1 ) ;
                     report4do( report ) ;
                  }
                  break ;
            }
            break ;
        case WM_CREATE:
            code4init( &cb ) ;
            break ;
        case CRM_REPORTCLOSED: /* Sent by report4do, see API */
            report4free( report, 1, 1 ) ;
            break ;
        case WM_DESTROY:
            code4initUndo( &cb ) ;
            PostQuitMessage (0) ;
            return 0 ;
    }

    return DefWindowProc (hWnd, message, wParam, lParam) ;
}
```

PROGRAM

WREP1.RC Windows resource file for the menu.

```
MAINMENU MENU
BEGIN
  MENUITEM "&Display Report", 101
END
```

PROGRAM

WREP1.DEF Windows definition file.

```
DESCRIPTION      'WREP1 CodeReporter Example'
EXETYPE          WINDOWS
STUB             'WINSTUB.EXE'
CODE             PRELOAD MOVEABLE DISCARDABLE
DATA             PRELOAD FIXED MULTIPLE
HEAPSIZE         4096
STACKSIZE        15000
```

Using Generated Code

CodeReporter can generate source code for any report file. The hard-coded source code to these reports is stored in individual files which may be compiled and linked with any application needing the report. All that is necessary, from the application's side, is a prototype of the function which creates the **REPORT4** structure, and a call to that function. Once created, the 'do' and 'free' sequence is carried out exactly as if the report was loaded using function **report4retrieve**.

PROGRAM
REP2. C Using source code generated by CodeReporter.

```
#include "d4all.h"

#ifdef __TURBOC__
   extern unsigned _stklen = 10000 ;
#endif

/* Prototype the CodeReporter-generated function */
REPORT4 *buildReport( CODE4 *, int) ;
void main( void )
{
   CODE4  cb ;
   REPORT4 *report ;

   code4init( &cb ) ;

   report = buildReport( &cb, 1 ) ;

   if( report )
   {
      report4do( report ) ;
      report4free( report, 1, 1) ;
   }
   code4initUndo( &cb ) ;
   return ;
}
```

REP2.C uses a report file saved as source code. As documented above, the **buildReport** function, generated by CodeReporter, is used to build the report. The code for the **buildReport** function must be linked with REP2.C to produce the final executable.

The prototype of function **buildReport** should be placed above *main.* This prototype is used to properly instruct the compiler on the parameters and return value for the CodeReporter-generated function. The prototype of the CodeReporter-generated report function follows this pattern:

REPORT4 *buildReport(CODE4 *cb, int openFiles);

The generated source code may also be used in a Windows application in the exact same manner. WREP2.C illustrates this point. By simply replacing the **report4retrieve** line with the function name in the generated source code report file, the report may be loaded and outputted.

PROGRAM

WREP2.C Partial code for using generated code under Windows.

```
...
long FAR PASCAL WndProc (HWND hWnd, UINT message, WPARAM wParam,
                                                      LPARAM lParam)
{
   static CODE4 cb ;
   static REPORT4 *report ;

   switch (message)
   {
      case WM_COMMAND:
         switch( wParam )
         {
           case IDM_DOREPORT:
             report = buildReport( &cb, 1 ) ;
             if( report )
             {
               report4parent( report, hWnd ) ;
               report4toScreen( report, 1 ) ;
               report4do( report ) ;
             }
             break ;
         }
         break ;
...
```

Creating a Report from Scratch

Since CodeReporter generates source code for any report, it is not recommended that reports be created from scratch. If customization of a report is necessary, it may be done using a previously generated report's code as a basis for the new custom report.

The following steps may be taken if it is necessary to create a report manually.

1. Create the relation first. This may either be done using the CodeBase 5 relation module functions, or a relation may be loaded from a relation file with **relate4retrieve**.

2. Initialize the report structure by calling **report4init**. This sets up some internal memory and sets many default settings.

3. Set up the applicable query and/or sort conditions using report4querySet and **report4sortSet**.

4. Modify any of the default settings with the following functions: **report4caption, report4currency, report4decimal, report4hardResets, report4margins, report4pageSize, report4separator,** and **report4titlePage**.
5. Create the groups of the report with **group4create**. Since the page header/footer and title/summary areas are automatically created by **report4init**, it is unnecessary to create them.

6. Create the areas for the groups using **area4create**.

7. Create any report-wide calculations (if applicable) using the CodeBase 5 function **expr4calc_create**.

8. Create and place the output objects in the appropriate areas using their respective creation functions.

Once the report has been constructed in this manner, the normal report sequence of 'load', 'do', and 'free' may be used to output the report.

Custom Output Drivers

The CodeReporter API comes with two different drivers for different platforms: Microsoft Windows and MS-DOS. If the report functions are to be used in other platforms, or if a custom display library (such as CodeScreens) is to be used, it is necessary for the developer to create his/her own equivalent to the report4do function.

The main functions used in a custom output driver are:

- **report4pageInit**
- **report4generatePage**
- **report4pageObjFirst**
- **report4pageObjNext**
- **report4pageFree**

In order to properly use these functions, the following structures must be used to properly output the data for a report.

OBJECT4

Description: This structure is used to describe an evaluated output object. A pointer to this structure is returned by the **report4pageObj** functions.

Members:

objtype: This flag is used to indicate the type of the current output object. *objtype* may have any one of the following constant values:

obj4type_field	*info* stores text representing the contents of a field.
obj4type_expr	*info* stores text representing the contents of an expression.
obj4type_total	*info* stores text representing the value of a total.
obj4type_text	*info* stores static text.
obj4type_hline	*info* is blank. The horizontal line is described by x, y, w, and h.
obj4type_vline	*info* is blank. The vertical line is described by x, y, w, and h.
obj4type_frame	*info* contains two bytes which are used to determine the state of the fill and rounded corners. If the left byte is '1', the frame is filled. If the right byte is '1', the frame has rounded corners.

alignment: Output objects that evaluate to *text (obj4type_field, obj4type_expr, obj4type_text, obj4type_total*) are justified along the left, right, or center of the object. The possible values for *alignment* are:

justify4right	Right justification.
justify4left	Left justification.
justify4center	Centered.

x: This is the horizontal position of the upper left corner of the output object in thousandths of an inch.

y: This is the vertical position from the top of the page of the upper left corner of the output object in thousandths of an inch.

w: This is the width of the output object in thousandths of an inch.

h: This is the height of the output object in thousandths of an inch.

Info_len: This is the size of the information stored in *info* for the output object.

info: This is the information to be outputted for the object.

style_index: This is an index into the report's style sheet.

See also: **report4pageObjFirst, report4pageObjNext, style4index**

STYLE4

Description:	This structure contains the information necessary to describe the font, color, and attributes of the report's styles. A pointer to the appropriate style structure may be obtained by **style4lookup** and **style4index**.
Members:	
name	This is a null terminated character array containing the name of the style as set in CodeReporter.
lfont	This is a pointer to a Windows LOGFONT structure which describes the typeface of the font used by the style.
color	This is an unsigned long value that stores the RGB colors set for the style. One byte for the red value, one byte for the green, and one byte for the blue. Each byte can contain a value from 0 to 255 indicating the shade of color. Use the following macros to retrieve the individual settings: R4GETRVALUE(rgb) R4GETGVALUE(rgb) R4GETRVALUE(rgb)
point_size	This is the size in points of the font used in the style
codes_before_len	This is the length of the codes pointed to by *codes_before*.
codes_after_len	This is the length of the codes pointed to by *codes_after*.
codes_before	This is a character array containing the printer control codes stored in the style sheet that turn 'on' a printer's attribute.
codes_after	This is a character array containing the printer control codes stored in the style sheet that turn 'off' a printer's attribute.
See also:	**style4index, style4create**

Using the Custom Driver Shell

The basic structure of a report output driver is essentially the same no matter for which environment or interface library the driver is intended.

PROGRAM

DRSHELLC Listed below is the source code and in code documentation for a sample custom interface driver shell.

```
int S4FUNCTION report4doDriverShell( REPORT4 *report )
{
   int rc, error ;
   OBJECT4 *obj ;
   STYLE4  *style ;

               if( report4pageInit( report ) < 0 ) /* initialize the page structure */
                  return -1 ;

   error = 0 ;
   while( (rc = report4generatePage( report )) >= 0 ) /* fill the page */
   {
       if( rc == 2 ) /* the last page has been reached */
            break ;
       for( obj = report4pageObjFirst( report ); obj != NULL && !error
                                                           ; obj = report4pageObjNext( report ) )
      {
         /* cycle through all the evaluated output objects within the * report */
         switch( obj->objtype )
         {
            case obj4type_field:
            case obj4type_expr:
            case obj4type_total:
            case obj4type_text:
               /* textual output routine */

                if( (style = style4index( report, obj->style_index )) == NULL)
               {
                 obj = NULL ;
                 error = 1 ;
                 break ;
               }
               /* get information about the object's style
                * use style->color, style->lfont, and style->point_size
                * to construct the appropriate output font */

               if( report->output_handle == 1 )
               {
                  /* outputting to screen */

                  /* position to coordinates obj->x, obj->y
                   * and use a screen interface function to output the
                   * text pointed to by obj->info to a length of
                   * obj->info_len, using obj->w and obj->h if
                   * necessary to handle word wrap and obj->alignment */
               }
               else
               {
                  /* outputting to a printer */
                  /* position to coordinates obj->x, obj->y
                   * and use a printer interface function to output the
                   * text pointed to by obj->info to a length of
                   * obj->info_len, using obj->w and obj->h if
                   * necessary to handle word wrap and obj->alignment */
               }
               break ;
            case obj4type_hline:
            case obj4type_vline:
              /* Line drawing routine */
               if( (style = style4index( report, obj->style_index )) == NULL)
               {
                 obj = NULL ;
                 error = 1 ;
                 break ;
               }
               /* use the style->color to set the color for the lines*/

               if( report->output_handle == 1 )
               {
                  /* position to obj->x, obj->y and draw a
                    * filled rectangle to obj->x+obj->w, obj->y+obj->h */
               }
               else
               {
                  /* position to obj->x, obj->y and print a
                    * filled rectangle to obj->x+obj->w, obj->y+obj->h */
               }
               break ;
            case obj4type_frame:
                /* box drawing routine */
               if( (style = style4index( report, obj->style_index )) == NULL)
               {
                 obj = NULL ;
                 error = 1 ;
                 break ;
               }
               /* use the style->color to set the color for the frame*/
               if( *((char *)obj->info) == 1 )
               {
                  /*  set for rounded corners */
               }
               else
               {
                  /* set for square corners */
               }

               if( *( ((char *)obj->info)+1 ) == 1 )
               {
                  /* set for filled rectangle */
               }
               else
               {
                  /* set for hollow rectangle  */
               }

               if( report->output_handle == 1 )
               {
```

```
            /* draw the appropriate rectangle
             * from obj->x, obj->y to
             * obj->x+obj->w, obj->y+obj->h */
          }
          else
          {
            /* print the appropriate rectangle
             * from obj->x, obj->y to
             * obj->x+obj->w, obj->y+obj->h */
          }
         break ;
       default:
          /* ignore all other object types */
      }
    }
    /* clear output device for new page */
  }
  report4pageFree( report ) ;
  if( report->code_base->error_code < 0 || error)
    return -1 ;
  return 0 ;
}
```

As it can be seen, the custom driver cycles through all of the evaluated output objects outputting them individually, for each page of the report

This code assumes that the output device may be written to at any point within the page. If output to a device is done on a line by line basis (such as with a dot-matrix printer), it may be necessary to perform custom storage of the output objects.

This may be done by creating a buffer in memory the size of the output device and storing the output objects within it as they are retrieved. When **report4pageObjNext** indicates that there are no more objects on the page, this buffered copy of the page could be outputted.

Another option that may be possible is to cycle through the objects in the page, creating a linked list of objects sorted by vertical and horizontal position. When there are no more objects within the page, this sorted list may be used to output the elements on a line by line basis.

It can be seen from these few examples that creating a custom report using the CodeReporter API can be as simple or as complex as desired. In almost every situation a combination of soft-coded reports and/or CodeReporter-generated source code can create complex reports with a minimum of hand coding.

area4 Functions

The **area4** functions are used for creating report output areas wherein output objects may be placed. These functions are also used to iterate through the output objects placed within a report area.

area4create

Usage: AREA4 *area4create(GROUP4 *group, long height, short isHeader, char *suppressExpr)

Description: This function creates a report header or footer area for the specified group in which output objects may be placed.

Parameters:

group	This is a pointer to the group with which the new report area is associated.
height	This **(long)** value is the height of the new report area in thousandths of an inch.
isHeader	This logical flag determines if the new report area should be associated with the group's header or footer. If *isHeader* is non-zero (true) the new area is associated with the group's header. If *isHeader* is zero (false) the new area is associated with the group's footer.
suppressExpr	This is a null terminated character array which points to a logical dBASE expression used to determine whether or not the report area is to be suppressed when a group reset condition occurs. When a group reset condition occurs, *suppressExpr* is evaluated. If it evaluates to a .TRUE. value, the report area is not outputted for that group reset condition. If it evaluates to a .FALSE. value, the report area is outputted. If *suppressExpr* is NULL or points to an array of blank spaces, the newly created area is never suppressed.

area4create creates a copy of the *suppressExpr* parameter. As a result, the memory for *suppressExpr* may be freed once this function is called.

Returns: **area4create** returns an **AREA4** pointer if successful. If the area could not be created, **area4create** returns a NULL value.

See Also: **area4free, group4create, group4titleSummary**

area4free

Usage:	void area4free(AREA4 *area)
Description:	This function removes a specified report area from the report. In addition, all memory associated with the report area is freed and all output objects within the report area are removed and freed.
Parameters:	
area	This AREA4 pointer identifies the report area to free
See Also:	**area4create, group4free**

area4numObjects

Usage:	int area4numObjects(AREA4 *area)
Description:	This function returns the current number of output objects placed within the specified area.
Parameters:	
group	This **AREA4** pointer is used to identify the area
Returns:	
0	There are no output objects for the specified area.
Not Zero	The number of output objects placed within the specified area.
See Also:	**area4objFirst, area4objNext, area4objLast, area4objPrev**

area4objFirst

Usage:	OBJ4 *area4objFirst(AREA4 *area)
Description:	This function retrieves a pointer to the first output object placed in the specified area. This function is used in conjunction with **area4objNext** to iterate through the output objects within an area.
Parameters:	
area	This is a pointer to the area containing output objects.
Returns:	
Not Zero	An **OBJ4** pointer for the first output object placed in the report area is returned.
0	Error. *area* was invalid, or there are no output objects in the specified area.
See Also:	**area4objNext, area4objLast**

area4objLast

Usage:	OBJ4 *area4objLast(AREA4 *area)
Description:	This function retrieves a pointer to the last output object placed in the specified area. This function is used in conjunction with **area4objPrev** to iterate backwards through the output objects within an area.
Parameters:	
area	This is a pointer to the area containing output objects.
Returns:	
Not Zero	An **OBJ4** pointer for the last output object placed in the report area is returned.
0	Error. *area* was invalid, or there are no output objects in the specified area.
See Also:	**area4objPrev**, area4objFirst

area4objNext

Usage:	OBJ4 *area4objNext(AREA4 *area)
Description:	This function retrieves a pointer to the next output within the specified area. This function is used in conjunction with **area4objFirst** to iterate through the output objects placed within the specified area.
Parameters:	
area	This is a pointer to the area containing output objects.
Returns:	
Not Zero	An **OBJ4** pointer for the next output object placed in the report area is returned.
0	Error. *area* was invalid, or the last output object retrieved was the last output object in the area.
See Also:	**area4objFirst, area4objPrev**

area4objPrev

Usage:	OBJ4 *area4objPrev(AREA4 *area)
Description:	This function retrieves a pointer to the previous output within the specified area. This function is used in conjunction with **area4objLast** to iterate backwards through the output objects placed within the specified area.
Parameters:	
area	This is a pointer to the area containing output objects.
Returns:	
Not Zero	An **OBJ4** pointer for the previous output object placed in the report area is returned.
0	Error. *area* was invalid, or the last output object retrieved was the first output object in the area.
See Also:	**area4objLast, area4objNext**

area4pageBreak

Usage:	void area4pageBreak(AREA4 *area, int allowBreaks)
Description:	This function is used to allow or disallow page breaks within the specified area. If this function is not called, areas will not allow page breaks to occur within them.
Parameters:	
area	This AREA4 pointer is used to identify the area.
allowBreaks	If *allowBreaks* is zero (false), a page break is not allowed to occur within the area. If a page break would occur within the group, it is then outputted on the next page. If *allowBreaks* is a positive non-zero value (true), a page break may occur within the report area. If a page break would occur within the report area, much as possible of the area is outputted on the current page, and the remainder is outputted on the following page.
Returns:	
>= 0	This function returns the previous allow page break setting.
< 0	*area* or *allowBreaks* is invalid.

group4 Functions

The **group4** functions are used to define the actions performed within a subset of the composite data file. These actions are mostly involved with outputting the report areas. These functions are also used to specify special characteristics of the output, including swapping the header/footer area(s) with those of the page, resetting the page and page number, etc.**group4** functions are also used to iterate through the header and footer areas associated with the group.

group4create

Usage:	GROUP4 *group4create(REPORT4 *report, char *name, char *resetExpr)
Description:	**group4create** creates a new group in the specified report. By default, the new group is the outermost group of the report.
Parameters:	
report	This is a pointer to the report in which the new group is added.
name	This is a null-terminated character array containing a unique descriptive name for the group. If this parameter is NULL, the name of the group defaults to "Group *n*", where *n* is the position of the group created. **group4create** creates a copy of *name*.
resetExpr	This is a null-terminated character array containing a dBASE expression used to determine when the group resets. The report module evaluates this expression for each record in the composite data file, and if its value changes the group resets and outputs the areas for the group. If *resetExpr* is NULL, the group resets for every record in the composite data file.
Returns:	
0	An error has occurred in creating the group.
Not Zero	A pointer to the successfully created group is returned.
See Also:	**area4create, group4free**

group4footerFirst

Usage: AREA4 *group4footerFirst(GROUP4 *group)

Description: This function returns an **AREA4** pointer to the first footer area created for the group.

Parameters:

group This is a **GROUP4** pointer for the group.

Returns: **group4footerFirst** returns an **AREA4** pointer for the first footer area created for the group. If the group does not have a footer area, this function returns NULL.

See Also: **group4headerfirst, group4footerNext, group4footerprev**

group4footerNext

Usage: AREA4 *group4footerNext(GROUP4 *group, AREA4 *area)

Description: This function returns a pointer to the footer area created in the specified group after the specified footer area. This function is generally used in combination with **group4footerFirst** to step through the footer areas of a group.

Parameters:

group This is a pointer to the group containing the *area* area.

area This is a pointer to the footer area used to locate the next footer area created.

Returns: This function returns an **AREA4** pointer to the footer area created after the *area* footer area. If *area* is the last footer area created for the group, or if area is NULL, **group4footerNext** returns NULL.

Unexpected results can occur if the *area* parameter is not a valid footer area for the *group* group.

See Also: **group4footerFirst, group4numFooters**

group4footerPrev

Usage: AREA4 *group4footerPrev(GROUP4 *group, AREA4 *area)

Description: This function returns a pointer to the footer area for the specified group created immediately before the specified area. This function is generally used in combination with **area4lastFooter** to step through the footer areas of a group.

Parameters:

group This is a pointer to the group containing the *area* footer area.

area This is a pointer to the footer area used to locate the previously created footer area.

Returns: This function returns an AREA4 pointer to the footer area created before the area footer area. If *area* is the last footer area created for the group, or if area is NULL, **group4footerPrev** returns NULL.

Unexpected results can occur if the *area* parameter is not a valid footer area for the *group* group.

See Also: **area4firstFooter, area4numFooters**

group4free

Usage: void group4free(GROUP4 *group)

Description: This function removes the specified group from the report and frees any associated memory. If the group has header and/or footer areas, they are removed by calls to **area4free**.

Parameters:

group This is the pointer of the group to remove from the report.

See Also: **group4create, area4free**

group4headerFirst

Usage: AREA4 *group4headerFirst(GROUP4 *group)

Description: This function returns an **AREA4** pointer to the first header area created for the group.

Parameters:

group This is a **GROUP4** pointer for the group.

Returns: **group4headerFirst** returns an **AREA4** pointer for the first header area created for the group. If the group does not have a header area, this function returns NULL.

See Also: **group4headerFirst, group4headerPrev, group4headerNext**

group4headerNext

Usage: AREA4 *group4headerNext(GROUP4 *group, AREA4 *area)

Description: This function returns a pointer to the header area created in the specified group after the specified header area.

This function is generally used in combination with **group4headerFirst** to step through the header areas of a group.

Parameters:

group This is a pointer to the group containing the *area* area.

area This is a pointer to the header area used to locate the next header area created.

Returns: This function returns an **AREA4** pointer to the header area created after the *area* header area. If *area* is the header last area created for the group, or if *area* is NULL, **group4footerNext** returns NULL.

> Unexpected results can occur if the *area* parameter is not a valid header area for the *group* group.

See Also: **group4headerFirst, group4numHeaders**

group4headerPrev

Usage: AREA4 *group4headerPrev(GROUP4 *group, AREA4 *area)

Description: This function returns a pointer to the header area for the specified group created immediately before the specified area. This function is generally used in combination with **area4lastHeader** to step through the header areas of a group.

Parameters:

group This is a pointer to the group containing the *area* header area.

area This is a pointer to the header area used to locate the previously created header area.

Returns: This function returns an **AREA4** pointer to the header area created before the *area* header area. If *area* is the last header area created for the group, or if area is NULL, **area4prevFooter** returns NULL.

> Unexpected results can occur if the *area* parameter is not a valid footer area for the *group* group.

See Also: **area4firstHeader, area4numHeaders**

group4numFooters

Usage:	int group4numFooters(GROUP4 *group)
Description:	This function returns the current number of footer areas for the specified group.
Parameters:	
group	This **GROUP4** pointer is used to identify the group.
Returns:	
0	There are no footers for the specified group.
Not Zero	The number of footer areas created for the specified group.
See Also:	**area4create, area4free**

group4numHeaders

Usage:	int group4numHeaders(GROUP4 *group)
Description:	This function returns the current number of header areas for the specified group.
Parameters:	
group	This **GROUP4** pointer is used to identify the group.
Returns:	
0	There are no headers for the specified group.
Not Zero	The number of header areas created for the specified group.
See Also:	**area4create, area4free**

group4repeatHeader

Usage:	int group4repeatHeader(GROUP4 *group, int repeatHeader)
Description:	This function is used to determine whether or not the specified group's header area(s) should be displayed at the top of a new page if inner groups span a page break.
	This function does not affect the page header. If this function is called with *repeatHeader* set to '1', the header is displayed under the page header (and any other higher positioned repeated headers) and before the first header line of a lower positioned group.
Parameters:	
group	This is the group for which the repeat header option is set.
repeatHeader	If *repeatHeader* is '1', the header is repeated on the next page. If it contains a '0' value (the default), the header area is only outputted when it encounters a reset condition.
	If this function is not called, the header is not repeated.
Returns:	
>= 0	The previous *repeatHeader* setting is returned.
< 0	Error.
See Also:	**group4swapHeader**

group4resetExprSet

Usage:	int group4resetExprSet(GROUP4 *group, char *resetExpr)
Description:	This function changes the specified group's reset expression.
Parameters:	
group	This is the group for which the reset expression is set.
resetExpr	This is a null terminated character array containing the new dBASE expression used to determine when the areas of the group are outputted. If *resetExpr* is NULL, the group reset expression is removed and the group resets for every composite record.
Returns:	
0	The group reset expression was successfully set.
< 0	Error.
See Also:	**group4create**

group4resetPage

Usage:	int group4resetPage(GROUP4 *group, int resetPage)
Description:	This function is used to determine whether a page break should be forced before the specified group's header area is outputted.
Parameters:	
group	This is the group for which the reset page option is set.
resetPage	This parameter may have two settings:
1	When *resetPage* is set to one, a page break is forced each time the specified group encounters a reset condition.
0	When *resetPage* is set to zero, the default, no special measures are taken to ensure that the group header is displayed on a new page.
	If this function is not called, the page is not reset when the group resets.
Returns:	
>= 0	The previous *resetPage* setting is returned.
< 0	Error.
See Also:	**group4resetPageNum**

group4resetPageNum

Usage:	int group4resetPageNum(GROUP4 *group, int resetPageNum)
Description:	This function is used to determine whether a page break should be forced before the specified group's header area is outputted. If the page break option is set, the page number is also reset to '1'.
Parameters:	
group	This is the group for which the reset page numberoption is set.
resetPageNum	This parameter may have two settings:
1	When *resetPageNum* is set to one, a page break is forced each time the specified group encounters a reset condition. In addition, the page number is set to '1'.
0	When *resetPage* is set to zero, the default, no special measures are taken to ensure that the group header is displayed on a new page, and the page number continues to accumulate.
Returns:	
>= 0	The previous *resetPage* setting is returned.
< 0	Error.
See Also:	PAGENO( ) dBASE Expression, **group4resetPage**

group4swapFooter

Usage:	int group4swapFooter(GROUP4 *group, int swap)
Description:	This function is used to cause a group to swap its footer area with the page footer area when the group resets. When the group encounters a reset condition, the rest of the current page is skipped and the specified group's footer area(s) are outputted instead of, and in the place of, the normal page footer. The page footer is not outputted. If the group is not reset and a page break occurs, the normal page footer is used.
Parameters:	
group	This pointer specifies the group whose footer area(s) are used in place of the page footer area.
swap	This logical flag determines whether or not the group should swap its footer area. The possible values for *swap* are:
1	The footer of the group is swapped.
0	The footer of the group is not swapped. If this function is not called, this value is assumed.
Returns:	
>= 0	The previous *swap* value is returned.
< 0	Error.
See Also:	**group4swapHeader, report4pageHeaderFooter**

group4swapHeader

Usage:	int group4swapHeader(GROUP4 *group, int swap)
Description:	This function is used to cause a group to swap its header area with the page header area when the group resets. When the group encounters a reset condition, the rest of the current page is skipped (the footers of all inner groups are outputted on the current page) and the specified group's footer area(s) are outputted at the top of the next page in the place of, and instead of, the normal page header. The page header is not Outputted. If the group is not reset and a page break occurs, the normal page header is used.
Parameters:	
group	This pointer specifies the group whose header area(s) are used in place of the page header area.
swap	This logical flag determines whether or not the group should swap its header area. The possible values for *swap* are:
1	The header of the group is swapped.
0	The header of the group is not swapped. If this function is not called, this value is assumed.

Returns:	
>= 0	The previous *swap* value is returned.
< 0	Error.
See Also:	**group4swapFooter, report4pageHeaderFooter**

obj4 Functions

The **obj4** functions are used for creating and modifying the output objects within a report area. Objects are created with their type's creation function and removed from the report with a their type's free function or **obj4delete**.

In addition, once an object is created, special formatting functions and style functions may be called to alter the way in which the output object appears.

obj4bitmapStaticCreate

Usage: OBJ4 *obj4bitmapStaticCreate(AREA4 *area, HANDLE hDIB
, long x, long y, long width, long height)

Description: This function creates a graphic object using a handle to a Windows device-independent bitmap. Once **obj4bitmapStaticCreate** creates a graphic output object, the handle to the device-independent bitmap (*hDIB)* must not be freed by the programmer. The bitmap is automatically freed by **obj4bitmapStaticFree.** The bitmap is scaled to fit within the coordinates provided. The aspect ratio of the bitmap is not necessarily maintained. The image of the static graphic object is stored within the report file.

Parameters:

area	This **AREA4** pointer specifies the report area in which the new graphic output object is placed.
hDIB	This is a handle to a Windows device-independent bitmap.
x	This is the horizontal coordinate, in 1000ths of an inch, where the left side of the graphic object is placed.
y	This is the vertical coordinate, in 1000ths of an inch (starting from the top of the report area), where the top edge of the graphic object is placed.
width	This is the horizontal width of the graphic output object, in 1000ths of an inch.
height	This is the vertical height of the graphic output object, in 1000ths of an inch.

Returns:

Not Zero	A pointer to the new graphic output object is returned if its creation was successful.
0	Error. The graphic output object could not be created.

See Also: **obj4bitmapStaticFree, obj4bitmapFileCreate**

obj4bitmapStaticFree

Usage:	void obj4bitmapStaticFree(OBJ4 *obj)
Description:	This function removes a static graphic output object (created with **obj4bitmapStaticCreate**) from the report and frees any memory associated with the bitmap and the output object.
Parameters:	
obj	This is a pointer to the static graphic output object to be freed.
See Also:	**obj4delete, obj4bitmapStaticCreat**e, report4free

obj4bitmapFileCreate

Usage:	OBJ4 *obj4bitmapFileCreate(AREA4 *area, char *fileName, long x, long y, long width, long height)
Description:	This function creates a static graphic output object by opening the provided Windows bitmap file (.BMP) and creating an internal copy of the image. The bitmap file is closed once the graphic output object is created. The bitmap is scaled to fit within the coordinates provided. The aspect ratio of the bitmap is not necessarily maintained. When a report containing this type of static graphic object is saved, only a reference to the file name is saved. The actual image for the bitmap is re-loaded from the provided bitmap file each time the report is executed.
Parameters:	
area	This **AREA4** pointer specifies the report area in which the new graphic output object is placed.
fileName	This is a null terminated character array containing the drive, directory and file name of a Windows bitmap. If a drive and/or directory is not provided, the current drive/directory is assumed.
x	This is the horizontal coordinate, in 1000ths of an inch, where the left side of the graphic object is placed.
y	This is the vertical coordinate, in 1000ths of an inch (starting from the top of the report area), where the top edge of the graphic object is placed.
width	This is the horizontal width of the graphic output object, in 1000ths of an inch.
height	This is the vertical height of the graphic output object, in 1000ths of an inch.
Returns:	
Not Zero	A pointer to the new graphic output object is returned if its creation was successful.
0	Error. The graphic output object could not be created.
See Also:	**obj4bitmapFileFree, obj4bitmapStaticCreate**

obj4bitmapFileFree

Usage: void obj4bitmapFileFree(OBJ4 *obj)

Description: This function removes a static graphic output object (created with **obj4bitmapFileCreate**) from the report and frees any memory associated with the bitmap and the output object.

Parameters:

obj This is the pointer returned by **obj4bitmapFileCreate** for the graphic object to remove from the report.

See Also: **obj4bitmapFileCreate, obj4delete, report4free**

obj4bitmapFieldCreate

Usage: OBJ4 *obj4bitmapFieldCreate(AREA4 *area, FIELD4 *field, long x,
long y, long width, long height)

Description: This function creates a dynamic graphic output object by opening the Windows bitmap file (.BMP) described in the data file field and creating an internal copy of the image. The bitmap file is closed once the graphic output object is outputted. The bitmap is scaled to fit within the coordinates provided. The aspect ratio of the bitmap is not necessarily maintained. When a report containing this type of static graphic object is saved, only a reference to the field is saved. The actual image for the bitmap is re-loaded from the bitmap file specified in the current field's value each time the graphic output object is outputted.

Parameters:

area This **AREA4** pointer specifies the report area in which the new graphic output object is placed.

field This is a **FIELD4** pointer to a data file field containing the drive, directory, and file name of a Windows bitmap file (.BMP). If the field does not contain a drive and/or directory, the current drive/directory is assumed.

x This is the horizontal coordinate, in 1000ths of an inch, where the left side of the graphic object is placed.

y This is the vertical coordinate, in 1000ths of an inch (starting from the top of the report area), where the top edge of the graphic object is placed.

width This is the horizontal width of the graphic output object, in 1000ths of an inch.

height This is the vertical height of the graphic output object, in 1000ths of an inch.

Returns:

Not Zero — A pointer to the new graphic output object is returned if its creation was successful.

0 — Error. The graphic output object could not be created.

See Also: **obj4bitmapFieldFree, obj4delete**

obj4bitmapFieldFree

Usage: void obj4bitmapFieldFree(OBJ4 *obj)

Description: This function removes a static graphic output object (created with **obj4bitmapFile**Create) from the report and frees any memory associated with the bitmap and the output object.

Parameters:

obj — This is the pointer returned by **obj4bitmapFieldCreate** for the graphic object to remove from the report.

See Also: **obj4bitmapFieldCreate, obj4delete, report4free**

obj4brackets

Usage: int obj4brackets(OBJ4 *obj, int useBrackets)

Description: This function specifies whether or not the specified numeric output object should use brackets for negative values.

Parameters:

obj — This is a pointer to the numeric object for which brackets are to be used.

useBrackets — This parameter determines whether brackets are used for negative values. *useBrackets* may be one of the following values

1 — Negative numbers are outputted within brackets. (i.e. (123))

0 — Negative numbers use the negative sign. (i.e. -123)

Returns:

>= 0 — The previous *useBrackets* setting is returned.

< 0 — Error. *obj* or *useBrackets* were invalid.

See Also: **obj4numericType, obj4displayZero**

obj4dataFieldSet

Usage: int obj4dataFieldSet(OBJ4 *obj, char *destField, char type, int length, int decimals)

Description: This function is used when outputting a report to a data file to associate a report object with a field in the destination data file. This setting is used only if **report4output** or **report4toScreen** are used to output the report to a data file.

Parameters:

obj	This is a pointer to the output object to be directed to a data file.
destField	This is a null terminated character array containing the name of the data file field in which the objects contents are placed.
type	This is an uppercase character used to describe the type of data file field within which the contents of the output object are placed. type may be one of the following: 'C' (character), 'D' (date), 'L' (logical), 'N' (numeric).
length	This is the maximum number of characters of the output object's contents to be copied into the output data file.
decimals	This is the number characters out of length to reserve as decimal places. decimals is only used if type is 'N', otherwise it is ignored.

Returns:

0	The output object was successfully associated with the destination field.
< 0	Error. obj or destField were invalid.

See Also: **report4dataFileSet, report4dataGroup, d4create from CodeBase 5.**

obj4calcCreate

Usage: OBJ4 *obj4calcCreate(AREA4 *area, EXPR4CALC *calc, long x, long y, long width, long height)

Description: This function creates a calculation output object using a calculation created with the CodeBase 6 **expr4calcCreate** function.

Parameters:

area	This **AREA4** pointer specifies the report area in which the new graphic output object is placed.
calc	This is a pointer to a calculation created with **expr4calcCreate**.
x	This is the horizontal coordinate, in 1000ths of an inch, where the left side of the calculation object is placed.
y	This is the vertical coordinate, in 1000ths of an inch (starting from the top of the report area), where the top edge of the calculation object is placed.

width This is the horizontal width of the calculation output object, in 1000ths of an inch.

height This is the vertical height of the calculation output object, in 1000ths of an inch.

Returns:

Not Zero A pointer to the new calculation output object is returned if its creation was successful.

0 Error. The calculation output object could not be created.

See Also: **expr4calc_create, obj4calcFree, obj4delete**

obj4calcFree

Usage: void obj4calcFree(OBJ4 *obj)

Description: This function removes the calculation output object from the report and frees any memory associated with the output object.

obj4calcFree does not free the memory associated with the actual calculation, nor does it remove the calculation. Use CodeBase 5 function **expr4calc_reset** or **report4free** to remove the calculation.

Parameters:

obj This is a pointer to the calculation object to be removed from the report.

See Also: **obj4calcCreate, obj4delete, report4free**, CodeBase 5 function **expr4calc_reset**

obj4dateFormat

Usage: int obj4dateFormat(OBJ4 *obj, char *dateFormat)

Description: This function sets the format (also known as a picture or mask) with which the date output object uses during output. If the output object does not evaluate to a date value, this function has no effect. If this function is not called and the output object evaluates to a date value, the default date format for the report is used (see **report4dateFormat**)

Parameters:

obj This is a pointer to the output object for which the date format is set.

dateFormat This null terminated character array contains the new date format used for the output object. If *dateFormat* is NULL, the current date format is ignored, and the report's default date format is used. **obj4dateFormat** makes a copy of dateFormat.

Returns:

0	The date format for the output object was successfully set.
< 0	Error.

See Also: **report4dateFormat**

obj4decimals

Usage: int obj4decimals(OBJ4 *obj, int numDecimals)

Description: This function sets the number of decimals used in the output of numeric output objects. Any unused decimal places are filled with zeros. If the output object does not evaluate to a numeric value, this function has no effect. All numeric output objects, with the exception of output objects created with **obj4fieldCreate**, have no decimals set by default. Numeric field output objects, by default, use the number of decimals specified by the field.

Parameters:

obj	This pointer specifies the object for which the decimals are set.
numDecimals	This is the number of decimals used in the output object.

Returns:

0	Success.
< 0	Error. *obj* is invalid.

See Also: **report4decimal**

obj4delete

Usage: void obj4delete(OBJ4 *obj)

Description: This is a generic function to delete any type of output object. **obj4delete** determines the type of the *obj* output object and calls the appropriate "free" function.

Parameters:

obj	This is a pointer to the object to remove from the report.

See Also: **report4free**

obj4displayOnce

Usage: int obj4displayOnce(OBJ4 *obj, char *supprExpr)

Description: This function causes the specified output object only to be outputted when the value of the provided expression changes.

Parameters:

obj	This is a pointer to the object for which the display once option is set.
supprExpr	This is a null terminated character array containing a dBASE expression which is used to determine when the output object is outputted. When the group the object is in is reset, this expression is evaluated. If the expressions value has changed since the last time it was evaluated, the output object is outputted. If the value is the same, the output object is ignored.

Returns:

0	Success.
< 0	Error. *obj* is invalid.

obj4displayZero

Usage: int obj4displayZero(OBJ4 *obj, int displayZero)

Description: This function specifies whether or not to output a zero value for a numeric output object. If the specified output object does not evaluate to a numeric value, this function has no effect.

Parameters:

obj	This is a pointer to the output object for which the display zero option is set.
displayZero	This true/false flag may have two values:
1	When *displayZero* is one (true), zero values are outputted. If this function is not called, this value is assumed.
0	If displayZero is zero (false), zero values are not outputted for the specified output object.

Returns:

0	Success.
< 0	Error. obj was invalid.

See Also: **obj4brackets, obj4numericType**

obj4exprCreate

Usage:	OBJ4 *obj4exprCreate(AREA4 *area, EXPR4 *expr, long x, long y, long width, long height)
Description:	This function creates an expression output object using an expression created with the CodeBase 6 function **expr4parse**.
Parameters:	
area	This **AREA4** pointer specifies the report area in which the new expression output object is placed.
expr	This is an **EXPR4** pointer to a parsed expression.
x	This is the horizontal coordinate, in 1000ths of an inch, where the left side of the expression object is placed.
y	This is the vertical coordinate, in 1000ths of an inch (from the top of the report area), where the top edge of the expression object is placed.
width	This is the horizontal width of the expression output object, in 1000ths of an inch.
height	This is the vertical height of the expression output object, in 1000ths of an inch.
Returns:	
Not Zero	A pointer to the new expression output object is returned if its creation was successful.
0	Error. The expression output object could not be created.
See Also:	CodeBase function **expr4parse**, **obj4exprFree**

obj4exprFree

Usage:	void obj4exprFree(OBJ4 *obj)
Description:	This function removes the specified expression output object from the report and frees any memory associated with the object. This function automatically frees the expression upon which the expression output object is based by calling CodeBase 6 function **expr4free**.
Parameters:	
obj	This is a pointer to the expression output object to be removed from the report.
See Also:	CodeBase 6 function expr4free, obj4exprCreate, obj4delete

obj4fieldCreate

Usage:	OBJ4 *obj4fieldCreate(AREA4 *area, FIELD4 *field, long x, long y, long width, long height)
Description:	This function creates a field output object for the specified data file field. Field output objects are automatically trimmed, so they may be centered, left justified, or right justified.
Parameters:	
area	This AREA4 pointer specifies the report area in which the new field output object is placed.
field	This is a **FIELD**4 pointer to a data file field. This may be obtained by using CodeBase 5 function **d4field**.
x	This is the horizontal coordinate, in 1000ths of an inch, where the left side of the field object is placed.
y	This is the vertical coordinate, in 1000ths of an inch (starting from the top of the report area), where the top edge of the field object is placed.
width	This is the horizontal width of the field output object, in 1000ths of an inch.
height	This is the vertical height of the field output object, in 1000ths of an inch.
Returns:	
Not Zero	A pointer to the new field output object is returned if its creation was successful.
0	Error. The field output object could not be created.
See Also:	**obj4fieldFree, obj4delete,** CodeBase 5 function **d4field**

obj4fieldFree

Usage:	void obj4fieldFree(OBJ4 *obj)
Description:	This function removes a field output object from the report and frees any memory associated with it. This function does not affect the data file field referenced by the output object.
Parameters:	
obj	This is a pointer to the field output object (created with **obj4fieldCreate**) to remove from the report.
See Also:	**obj4fieldCreate, obj4delete, report4free**

obj4frameCorners

Usage:	int obj4frameCorners(OBJ4 *obj, int rounded)

Description:	This function is used with frame output objects to determine what type of corners (rounded or square) should be used when the object is outputted.
Parameters:	
obj	This is a pointer to the frame output object for which the corner type is set.
rounded	This parameter determines whether or not the corners of the frame object are rounded. *rounded* may have one of the two values below:
1	The frame object is created with rounded corners.
0	The frame object is created with square (90 degree) corners. If this function is not called, this value is assumed.
Returns:	
>= 0	The previous rounded setting is returned.
< 0	Error. obj is invalid.
See Also:	**obj4frameCreate, obj4frameFree**

obj4frameCreate

Usage:	OBJ4 *obj4frameCreate(AREA4 *area, long x, long y, long width, long height)
Description:	This function creates a frame output object within the specified report area. By default, the new frame has square corners and is hollow.
Parameters:	
area	This **AREA4** pointer specifies the report area in which the new field output object is placed.
x	This is the horizontal coordinate, in 1000ths of an inch, where the left side of the frame object is placed.
y	This is the vertical coordinate, in 1000ths of an inch (starting from the top of the report area), where the top edge of the frame object is placed.
width	This is the horizontal width of the frame output object, in 1000ths of an inch.
height	This is the vertical height of the frame output object, in 1000ths of an inch.
Returns:	
Not Zero	A pointer to the new frame output object is returned if its creation was successful.
0	Error. The frame output object could not be created.
See Also:	**obj4frameCorners, obj4frameFill,** obj4frameFree**, obj4lineWidth**

obj4frameFill

Usage: int obj4frameFill(OBJ4 *obj, int fill)

Description: This function is used to set the fill status for the specified frame output object. If the frame is filled, the report module output functions fill the frame with the color of the frame's selected style. If the frame is not filled (the default), only the frame outline is outputted.

Parameters:

obj This is a pointer to the frame output object for which the fill status is set.

fill This parameter determines whether or not the frame is filled when outputted. fill may have the following values:

1 If fill is set to one (true), the frame output object is filled.

0 If fill is set to zero (false), the frame output object is not filled.

Returns:

0 The previous value of fill is returned.

< 0 Error. obj or fill were invalid.

See Also: **obj4frameCreate, obj4frameCorners, obj4lineWidth**

obj4frameFree

Usage: void obj4frameFree(OBJ4 *obj)

Description: This function removes a frame object from the report and frees any memory associated with the output object.

Parameters:

obj This is a pointer to the frame output object to be freed.

See Also: **obj4frameCreate, obj4delete, report4free**

obj4justify

Usage: int obj4justify(OBJ4 *obj, int justification)

Description: This function specifies whether objects that contain textual output should be centered, left justified, or right justified within the bounds of the object. All objects are left justified by default.

Parameters:

obj This specifies the object which is to be justified.

justification The justification parameter may be one of the following pre-defined constants:

justify4left	Text for the output object is outputted beginning at the leftmost bounds of the object.
justify4right	Text for the output object is outputted beginning at the rightmost bounds of the object with the last character of the object and proceeds to towards the left.
justify4center	Text for the output object is centered within the bounds of the object.

Returns:

>= 0	The previous justification setting is returned.
< 0	Error.

See Also: **obj4exprCreate, obj4fieldCreate, obj4totalCreate, obj4textCreate**

obj4leadingZero

Usage: int obj4leadingZero(OBJ4 *obj, int leadingZero)

Description: This function is used to set the leading zero option for the specified output object. The leading zero option is used when the output object evaluates to a numeric value between 1 and -1 to determine whether or not a zero should be placed in the units position of the fractional number.

Leading Zero	No Leading Zero
0.33	.33
-0.33	-.33
3.33	3

Parameters:

obj	This is a pointer to the numeric output for which the leading zero option is set
leadingZero	This parameter determines whether or not a leading zero is used for the numeric output object. *leadingZero* may have the following values:
1	A leading zero is used for fractional numbers.
0	A leading zero is not used for fractional numbers. If this function is not called, this value is assumed.

Returns:

>= 0	The previous value of leadingZero is returned.
< 0	Error. obj and/or leadingZero were invalid.

See Also: **obj4displayZero**

obj4lineCreate

Usage:	OBJ4 *obj4lineCreate(AREA4 *area, int vertical, long x, long y, long length)
Description:	This function is used to create vertical and horizontal line output objects of a specified length. The line output object has default thickness of one thousandth of an inch (1/1000th inch).
Parameters:	
area	This is a pointer to the area in which the line output object is placed.
vertical	This parameter determines whether the line output object is vertical or horizontal. *vertical* may have one of the following values.
1	The line output object is a vertical line.
0	The line output object is a horizontal line.
x	This is the horizontal coordinate, in 1000ths of an inch, of the beginning point of the line.
y	This is the vertical coordinate, in 1000ths of an inch (starting from the top of the report area), that specifies where the line output object begins.
length	This is the length of the line object in 1000ths of an inch.
Returns:	
Not Zero	A pointer to the new line output object is returned if its creation was successful.
0	Error. The line output object could not be created.
See Also:	**obj4lineFree, obj4lineWidth**

obj4lineFree

Usage:	void obj4lineFree(OBJ4 *obj)
Description:	This function removes a line output object from the report and frees any memory associated with the output object.
Parameters:	
obj	This is a pointer to the line object to be removed from the report.
See Also:	**obj4lineCreate, obj4delete, report4free**

obj4lineWidth

Usage:	int obj4lineFree(OBJ4 *obj, long width)
Description:	This function changes the width of the lines used to draw line and frame output objects.
Parameters:	
obj	This is a pointer to the line output object for which the length is set.
width	This is the new width, in thousandths of an inch, of the line object.
Returns:	
0	The width was successfully set.
-1	Error. *obj* was invalid or *width* was a negative number.
See Also:	**obj4lineCreate**

obj4lookAhead

Usage:	int obj4lookAhead(OBJ4 *obj, int lookAhead)
Description:	This function sets the specified output object as a look ahead object. When the object is outputted, it contains the value it would have in the group's footer.
	A look ahead total output object contains the value it would have when its total reset expression changes and not necessarily the value it would have in the group's footer.
Parameters:	
obj	This is a pointer to the output object for which the look ahead option is to be set.
lookAhead	This parameter determines whether or not an output object is a look ahead object. *lookAhead* may be one of the following values:
1	The specified object is set to be a look ahead object.
0	The object is not set to be a look ahead object. If this function is not called, this value is assumed.
Returns:	
>= 0	The previous *lookAhead* value is returned.
< 0	Error. *obj* or *lookAhead* were invalid.

obj4numericType

Usage:	int obj4numericType(OBJ4 *obj, int numericType)
Description:	This function determines how the specified numeric output object is to be formatted.
Parameters:	
obj	This is a pointer to an output object that evaluates to a numeric value.
numericType	This parameter is used to set the formatting of the numeric output object. *numericType* may be any one of the following defined constants:
obj4numNumber	The output object is not formatted in any way.
obj4numExponent	The output object is formatted in scientific notation (ie. *n.nnnnn e xx* where *n* is the numeric value and *x* is the exponential value)
obj4numCurrency	The currency symbol (set with **report4currency**) is outputted before the numeric value.
obj4numPercent	When outputted, the percentage symbol ('%') immediately follows the numeric value (multiplied by 100).
Returns:	
>= 0	The previous setting of *numericType* is returned.
< 0	Error. *obj* or *numericType* was invalid.
See Also:	**report4currency, report4decimal, obj4decimals**

obj4style

Usage:	int obj4style(OBJ4 *obj, STYLE4 *style)
Description:	By default, all output objects are created with the currently selected style. If a style has not been selected, the report module default style "Plain Text" is used. This function sets a specific style for the output object.
Parameters:	
obj	This is a pointer to the output object for which the style is set
style	This is a pointer to the style used for the output object.
Returns:	
0	Success.
< 0	Error.
See Also:	**style4create, style4lookup, style4next**

obj4textCreate

Usage:	OBJ4 *obj4textCreate(AREA4 *area, char *text, long x, long y, long width, long height)
Description:	This function creates a static text object.
Parameters:	
area	This **AREA4** pointer specifies the report area in which the new text output object is placed.
text	This is a null terminated character array containing the text to be outputted. **obj4textCreate** makes a copy of *text*.
x	This is the horizontal coordinate, in 1000ths of an inch, where the left side of the text object is placed.
y	This is the vertical coordinate, in 1000ths of an inch (starting from the top of the report area), where the top edge of the text object is placed.
width	This is the horizontal width of the text output object, in 1000ths of an inch.
height	This is the vertical height of the text output object, in 1000ths of an inch.
Returns:	
Not Zero	A pointer to the new text output object is returned if its creation was successful.
0	Error. The text output object could not be created.
See Also:	**obj4textFree, obj4delete**

obj4textFree

Usage:	void obj4textFree(OBJ4 *obj)
Description:	This function removes the specified static text output object from the report and frees any memory associated with the output object.
Parameters:	
obj	This is a pointer to the static text object to be removed from the report
See Also:	**obj4textCreate, obj4delete**, report4free

obj4totalCreate

Usage:	OBJ4 *obj4totalCreate(AREA4 *area, TOTAL4 *total, long x, long y, long width, long height)
Description:	This function creates a total output object using a total created with **total4create**.
Parameters:	
area	This **AREA4** pointer specifies the report area in which the new total output object is placed.
total	This is a **TOTAL4** pointer to a total created with **total4create**.
x	This is the horizontal coordinate, in 1000ths of an inch, where the left side of the total object is placed.
y	This is the vertical coordinate, in 1000ths of an inch (starting from the top of the report area), where the top edge of the total object is placed.
width	This is the horizontal width of the total output object, in 1000ths of an inch.
height	This is the vertical height of the total output object, in 1000ths of an inch.
Returns:	
Not Zero	A pointer to the new total output object is returned if its creation was successful.
0	Error. The total output object could not be created.
See Also:	**obj4totalFree, total4create, total4free, obj4lookAhead**

obj4totalFree

Usage:	void obj4totalFree(OBJ4 *obj)
Description:	This function removes the specified total output object from the report and frees any memory associated with the object. In addition, any other output object or expression that uses the total is also removed from the report. Finally, the total upon which the object is based is also freed.
	This function can cause a chain-reaction of object deletions that can quickly destroy a report.
Parameters:	
obj	This is the total output object to be removed from the report.
See Also:	**obj4totalCreate, total4create, total4free, obj4delete, report4free**

relate4 Functions

The two **relate4** functions listed herein provide the ability to save and retrieve a relation from disk. These functions are not CodeReporter-specific and may be used in any CodeBase 5 application to retrieve and save relations.

relate4retrieve

Usage:
```
RELATE4 *relate4retrieve( CODE4 *cb, char *fileName, int openFiles,
                                                char *dataPathName )
```

Description: This function retrieves a relation file and constructs the relation that was saved with **relate4save**. In the process of loading the relation file, **relate4retrieve** may also open the relation's data files.

Parameters:

cb	This is a pointer to the application's **CODE4** structure. This is used for memory management and error handling.
fileName	This is a null terminated character array which contains the file name (including drive and directory) of the relation file. A file extension need not be provided since the .REL extension is always used.
openFiles	If *openFiles* is a true value (non-zero), **relate4retrieve** attempts to open the data, index, and memo files referenced in the saved relation file if they are not already opened. If **relate4retrieve** cannot find a certain data file referenced in the relation file, that file and all lower level slave data files of that file are omitted from the relation and an attempt is made to locate the next data file. If *openFiles* is a false value (zero), **relate4retrieve** assumes that all of the data, index, and memo files are already opened. If a data file referenced in the relation file is not opened, that file and all lower level slave data files in the relation are omitted from the relation and the **relate4retrieve** continues to build the relation.
dataPathName	This parameter is a null terminated character array containing a new drive and path for the data, index, and memo files stored in the relation file. If *dataPathName* is NULL, the paths stored in the relation file are used. If no paths were stored in the file, **relate4retrieve** attempts to open the files in the current directory. If *dataPathName* is specified, it is used to override the paths saved within the relation file.

Returns:	
Not Zero	The relation was successfully retrieved from the specified relation file.
Zero	An error occurred while reading the relation file or opening the relation's top master data file. See the **CODE4.error_code** member variable for the specific error setting.
See Also:	**relate4save, relate4init, relate4free**

relate4save

Usage:	int relate4save(RELATE4 *relate, char *fileName, int savePathNames)
Description:	This function saves the specified relation in a relation file.
Parameters:	
relate	This is a pointer to the relation that is to be saved to a relation file.
fileName	This is a null terminated character array which contains the file name (including drive and directory) of the relation file. A file extension need not be provided since the .REL extension is always used.
savePathNames	If this parameter contains a true value (non-zero), **relate4save** saves the full path name of the files used in the relation. If *savePathNames* contains a false value (zero), only the actual file name is saved.
Returns:	
0	The relation file was successfully saved.
r4no_create	The relation file could not be created. This is generally caused when *fileName* conflicts with a file that already exists, or if the application does not have read/write privileges to the desired drive.
< 0	Error.
See Also:	**relate4retrieve**

report4 Functions

The **report4** functions provide a means of specifying report-wide settings, such as page width, margins, currency symbol, whether output goes to the screen, the selected printer, etc.

report4caption

Usage:	int report4caption(REPORT4 *report, char *caption)
Description:	This function sets the text of the caption for the report output window when the report is sent to the screen.
Parameters:	
report	This is a pointer to the report for which the report output window caption is set.
caption	This is a null terminated character array containing the text to be placed in the caption portion of the output window. **report4caption** makes a copy of *caption.*
Returns:	
0	The caption was set successfully.
< 0	Error.

report4currency

Usage:	int report4currency(REPORT4 *report, char *currency)
Description:	This function sets the text to be displayed immediately to the left of numeric output objects that are formatted as currency values.
Parameters:	
report	This is a pointer to the report for which the currency characters are set.
currency	This is a null terminated character array containing the currency symbol(s). *currency* may contain up to ten (10) characters. **report4currency** makes a copy of *currency.* If this function is not called, the dollar symbol ($) is assumed.
Returns:	
0	The currency character(s) were set successfully.
< 0	Error.
See Also:	**obj4numericType**

report4dataDo

Usage:	int report4dataDo(REPORT4 *report)
Description:	This function outputs the report to a data file as specified in the report's data file template, or with functions **obj4dataFieldSet, report4dataFileSet,** and report4dataGroup.
Parameters:	
report	This is a pointer to the report to be outputted to a data file.
Returns:	
0	Success.
< 0	Error. *report* was invalid, or did not contain a data file template.
See Also:	**obj4dataFieldSet, report4dataFileSet, report4dataGroup**

report4dataFileSet

Usage:	int report4dataFileSet(REPORT4 *report, char *destFile)
Description:	This function sets the file name used to create the output data file when report output is directed to a data file by **report4dataDo**.
Parameters:	
report	This is a pointer to the report for which the data file name is set.
destFile	This is a null terminated character array containing the drive, directory and file name of the data file where report output is stored. If the drive and/or directory is not provided, the current directory is assumed.
Returns:	
0	Success.
< 0	Error. *report* or *destFile* were invalid.
See Also:	**obj4dataFieldSet, report4dataGroup**

report4dataGroup

Usage:	int report4dataGroup(REPORT4 *report, GROUP4 *group)
Description:	This function identifies the group whose reset condition generates a new record in the output data file. The output objects are stored in the resultant record containing the values they would have if they were outputted in the *group's* group footer area.
Parameters:	
report	This is a pointer to the report with which the group and data file is associated.
group	This is a GROUP4 pointer for the group which generates records in the output data file.
Returns:	
0	Success.
< 0	Error. *report* or *group* were invalid.
See Also:	**report4groupLookup, obj4dataFieldSet, report4dataFileSet**

report4dateFormat

Usage:	int report4dateFormat(REPORT4 *report, char *format)
Description:	This function sets the default date format for the specified report. All new output objects that evaluate to a date value, by default, use this format for output. When the report is initially created, the value of the **CODE4.date_format** member variable is stored within the report's default date format.
Parameters:	
report	This is a pointer to the report for which the date format is set.
format	This is a null terminated character array which contains the default date format. This string should contain the picture formatting characters ('D', 'M', 'C', 'Y'). **report4dateFormat** creates a copy of *format*, so *format* may point to temporary memory.
Returns:	
0	Success.
< 0	Error. invalid *report*, or not enough memory to copy *format*.
See Also:	**obj4dateFormat**

report4decimal

Usage:	int report4decimal(REPORT4 *report, char decimalChar)
Description:	This function sets the character to be outputted to separate whole numbers from fractional numbers within a numeric output object.
Parameters:	
report	This is a pointer to the report for which the decimal character is used.
decimalChar	This is the character used as the decimal. If this function is not called, the decimal point ('.') is assumed.
Returns:	
0	Success.
< 0	Error. report was invalid.
See Also:	**obj4decimals**

report4do

Usage: int report4do(REPORT4 *report)

Description: This high-level function causes the specified report to be outputted to the selected device.

When outputting the report under Windows, **report4do** disables the report's parent window (specified by **report4parent**) and creates an output window that processes the report. The output window sends the report's parent window a **CRM_REPORTCLOSED** message once the report is completed.

If this function is to be used, it is necessary to first call **report4parent**. Failure to do so can cause unpredictable results.

If the report is outputted in a non-Windows application, **report4do** sends the report to the device specified by **report4output** and returns once the report is completed.

Parameters: *report* specifies the report to be outputted.

Returns:

0	Success. The report was successfully outputted. Under Windows, this value is returned immediately, even though the output window may not have completed the output of the report.
r4terminate	A relation was unable to be made and the error action specified with **relate4error_action** was **relate4terminate**.
< 0	Error.

See Also: **report4toScreen, report4parent, report4printerSelect, report4printerSet, report4output**

report4free

Usage: void report4free(REPORT4 *report, int freeRelate, int closeFiles)

Description: This function frees all memory associated with the report, including all output objects, all groups, and all areas.

> In a Windows application, this function should only be called after the report's parent window has received a CRM_REPORTCLOSED message. Calling **report4free** immediately after **report4do** under Windows, can cause unpredictable results.

Parameters:

report	This is the report to be freed from memory.
freeRelate	If this parameter contains a true value (non-zero), the memory associated with the report's relation is automatically freed. If a false value (zero) is passed, the relation is unaffected.
closeFiles	This parameter, when it contains a true value (non-zero), causes **report4free** to automatically close the data, index, and memo files referenced in the report. If *closeFiles* is false (zero), or if *freeRelate* is false, this setting is ignored.

See Also: **report4pageFree,** and **relate4free** in the CodeBase 5 manual

report4generatePage

Usage: int report4generatePage(REPORT4 *report, HDC hDC)

Description: This low-level function is used to store the next page of the report in a Windows device context such as a bitmap device context or a printer device context. This function does not clear the device context prior to storing the page. If this function is used to output the report to a printer, it is the programmer's responsibility to send the following codes to the device with the Windows Escape function: SETABORTPROC (if desired), STARTDOC, NEWFRAME, ENDDOC. If this function is used to output the report to a bitmap (to display in a window, save to disk, etc.), it is the programmer's responsibility to create a memory device context containing a bitmap large enough to store a report page.

Parameters:

report	This is a pointer to the report from which a page is retrieved.
hDC	This is a handle to a valid Microsoft Windows device context in which the next page of the report is placed. It is the programmer's responsibility to free *hDC* once it is no longer needed.

Returns:

0	The new page was successfully stored in *hDC.*
2	There are no more pages in the report. *hDC* is unaltered.
< 0	Error. *report and/or hDC* were invalid.

See Also: **report4init**

report4generatePage

Usage:	int report4generatePage(REPORT4 *report)
Description:	This low-level function is used to generate an internal structure containing the information for the next page of a report. The values of the evaluated output objects within the report page may be retrieved from this internal buffer using **report4pageObjFirst** and **report4pageObjNext**.
Parameters:	
report	This is a pointer to the report for which the next page is retrieved.
Returns:	
0	The new page was successfully stored in the internal buffer.
2	There are no more pages in the report.
< 0	Error. report was invalid.
See Also:	**report4pageObjFirst, report4pageObjNext, report4init**

report4groupFirst

Usage:	GROUP4 *report4groupFirst(REPORT4 *report)
Description:	report4groupFirst returns a pointer to the innermost group, which is the first group created in the report. This function, in conjunction with **report4groupNext** is used to iterate through the groups within a report.
Parameters:	
report	This is a pointer to the report from which the first group is retrieved.
Returns:	
0	The specified report does not have a group created.
Not Zero	This is a pointer to the innermost group of the report.
See Also:	**report4groupNext, report4numGroups**

report4groupLast

Usage:	GROUP4 *report4groupLast(REPORT4 *report)
Description:	**report4groupLast** returns a pointer to the outermost group, which is the last group created in the report. This function, in conjunction with **report4groupPrev** is used to iterate through the groups within a report.
Parameters:	
report	This is a pointer to the report from which the last group is retrieved.
Returns:	
0	The specified report does not have a group created.
Not Zero	This is a pointer to the outermost group of the report.
See Also:	**report4groupPrev**, report4numGroups

report4groupLookup

Usage:	GROUP4 *report4groupLookup(REPORT4 *report, char *name)
Description:	This function is used to obtain a **GROUP4** pointer to specified named group.
Parameters:	
report	This is a pointer to the report for which the groups belong.
name	This is a null terminated character array which contains the name of the group for which a **GROUP4** pointer is desired.
Returns:	
Not Zero	A pointer to the specified group is returned.
0	*name* did not match with any named groups in the specified report.
See Also:	**group4create**

report4groupNext

Usage:	GROUP4 *report4groupNext(REPORT4 *report, GROUP4 *group)
Description:	This function is used to obtain a **GROUP4** pointer to the group created immediately after the specified group. This is used in conjunction with **group4first** to iterate through the groups in a report.
Parameters:	
report	This is a pointer to the report for which the groups belong.
group	This is a pointer to an inner group which is used to obtain the next outer group.
Returns:	
0	There were no more groups in the report. *group* is the outermost *group*.
Not Zero	A pointer to the next group is returned.
See Also:	**group4first, report4numGroups**

report4groupPrev

Usage:	GROUP4 *report4groupPrev(REPORT4 *report, GROUP4 *group)
Description:	This function is used to obtain a **GROUP4** pointer to the group created immediately before the specified group. This is used in conjunction with **report4groupLast** to iterate through the groups in a report.
Parameters:	
report	This is a pointer to the report for which the groups belong.
group	This is a pointer to an outer group which is used to obtain the previously created inner group.
Returns:	
0	There were no more groups in the report. *group* is the innermost group.
Not Zero	A pointer to the next group is returned.
See Also:	**report4groupLast**, **report4numGroups**

report4hardResets

Usage:	int report4hardResets(REPORT4 *report, int hardResets)
Description:	This function is used to specify the method in which groups with the reset page option generate new pages.
Parameters:	
report	This is a pointer to the report for which the hard reset flag is set.
hardResets	This parameter is used to determine how page resets are handled. *hardResets* may be one of the following values:
1	Always generate a new page before a group with the reset page flag is outputted.
0	Only generate a new page for a group with a reset page flag set if the group is being reset as a result of its own reset condition being changed. If the group is outputted as a result of a higher level group being reset, a new page is not generated. If this function is not called, this value is assumed.
Returns:	
>= 0	The previous *hardResets* value is returned.
< 0	Error. *report* and/or *hardResets* was invalid.
See Also:	**group4resetPage**

report4init

Usage: REPORT4 *report4init(RELATE4 *relate)

Description: This function initializes a report structure with default values, and returns a report pointer which may be used with the rest of the report module functions. This function is automatically called by **report4retrieve**.Once the report is completed, call **report4free** to free up the memory associated with the report structure.

report4init sets the following default values:

Margins:	Left: 1/4 inch, Right: 1/4 inch, Top 0, Bottom 0
Page Size:	8 1/2 x 11 inches
Decimal Point:	'.'
Thousands Separator	','
Currency Symbol	"$"
Default Style:	"Plain Text" (Windows: MS Serif 10pt., Non-Windows: No control codes)
Title/Summary Group	Size of zero

Page Header/Footer Group   Size of zero

Parameters:

relate	This is the relation upon which the report is based.

Returns:

Not Zero	The report was successfully initialized and a pointer to the report structure is returned.
Zero	Error. The report could not be initialized. See the **CODE4.error_code** setting for more information.

See Also: **report4retrieve, relate4retrieve, report4do, report4free**

report4margins

Usage: int report4margins(REPORT4 *report, long left, long right, long top, long bottom, int unitType)

Description: This function is used to change the default margins of the report.

Some output devices, such as laser printers, have a hardware margin which is not under software control. **report4margins** checks for this condition and will not allow the margins to violate the physical margins of the device.

Parameters:

report	This is a pointer to the report for which the margins are set.
left	This is the size of the left margin in the provided increments.
right	This is the size of the right margin in the provided increments.
top	This is the size of the top margin in the provided increments.
bottom	This is the size of the bottom margin in the provided increments.
unitType	This is the unit of measure for the above margin settings. In graphical user interfaces, 1000ths of an inch may conveniently be used. In character-based interfaces, it is often more convenient to use characters. *unitType* may be one of the following values:

1	The units listed are in characters.
0	The units listed are in 1000ths of an inch.

If *unitType* is set for characters, this function assumes ten (10) characters per inch and six (6) lines per inch.

Returns:

0	The margins were successfully set.
< 0	Error.

See Also: **report4pageSize**

report4numGroups

Usage:	int report4numGroups(REPORT4 *report)
Description:	This function returns the current number of groups within the report. This is useful when iterating through the groups in the report.
Parameters:	
report	This **REPORT4** pointer indicates the report for which the number of groups is desired.
Returns:	
0	There are no groups in the specified report.
> 0	This is the number of groups added to the report.
< 0	An error has occurred.
See Also:	**report4groupNext, report4groupPrev, group4first, report4groupLast**

report4numStyles

Usage:	int report4numStyles(REPORT4 *report)
Description:	This function returns the current number of styles within the report. This is useful when iterating through the styles of the report.
Parameters:	
report desired.	This **REPORT4** pointer indicates the report for which the number of styles is
Returns:	
> 0	This is the number of styles within the report. There will always be at least one style within the report.
< 0	An error has occurred.
See Also:	**report4styleFirst, report4styleNext, report4styleLast, report4stylePrev**

report4output

Usage:	int report4output(REPORT4 *report, int outputHandle, int useStyles)
Description:	This function is called prior to report generation and is used to instruct **report4do** to send the report to a system handle such as 'standard out', 'standard print', or an open file. If this function is not called, **report4do** sends report output to the 'standard out' monitor.
	report4do outputs reports by using the standard C library function write() which uses a system handle to output text. **report4do** passes *outputHandle* to write().
	This function is used when outputting a report to a file.
Parameters:	
report	This is the report for which a destination is set.
outputHandle	This is a standard C system handle as returned by C functions such as open() and sopen(). Other common handles that are pre-defined by the C language are:
	1 This is 'standard out' which is by default the monitor.
	4 This is 'standard print'. On IBM computers this is usually the printer on the LPT1 port.
useStyles	This parameter is used to determine whether or not the information outputted by **report4do** should contain the printer control codes as defined within the style sheets. *useStyles* must be one of the following values:
	1 The printer pre- and post-control codes are outputted to the specified handle.
	0 The printer control codes are ignored and only the text for the output objects are outputted.
Returns:	
0	The settings were successfully set.
< 0	Error. report was invalid.
See Also:	**report4do, report4toScreen**

report4pageFree

Usage: int report4pageFree(REPORT4 *report)

Description: This is a low-level function which is used internally to free the memory associated with the internal representation of an output page. This function is automatically called at the end of the report by **report4do**.

Parameters:

report This is a pointer to the report for which the output page is freed.

Returns:

0 The page was successfully freed.

< 0 Error.

See Also: **report4free, report4generatePage**

report4pageHeaderFooter

Usage: GROUP4 *group4pageHeaderFooter(REPORT4 *report)

Description: This function returns a **GROUP4** pointer to the report's page header and footer "group." The returned group, which is automatically created by **report4init**, cannot be deleted.

Parameters:

report This is a pointer to the report that contains the desired page header/footer "group."

Returns: A **GROUP4** pointer to the page header/footer group is returned.

See Also: **area4create, report4init**

report4pageInit

Usage: int report4pageInit(REPORT4 *report)

Description: This low-level function is used to create an internal page buffer for report output. This function is automatically called by **report4do**.

Parameters:

report This is a pointer to the report for which the internal page buffer is created.

Returns:

0 The page buffer was successfully created.

< 0 Error.

See Also:	**report4pageFree, report4do**

report4pageMarginsGet

Usage:	int report4marginsGet(REPORT4 *report, long *left, long *right, long *top, long *bottom)
Description:	This function is used to retrieve the margins set for the report. All margins are retrieved in increments of a thousandths of an inch.
Parameters:	
report	This is a pointer to the report for which the margins are retrieved.
left	This is a pointer to a **(long)** variable where the size of the left margin is stored.
right	This is a pointer to a **(long)** variable where the size of the right margin is stored.
top	This is a pointer to a **(long)** variable where the size of the top margin is stored.
bottom	This is a pointer to a **(long)** variable where the size of the bottom margin is stored.
Returns:	
0	The margins were successfully retrieved.
< 0	Error. *report* was invalid.
See Also:	**report4margins**

report4pageObjFirst

Usage:	OBJECT4 *report4pageObjFirst(REPORT4 *report)
Description:	This low level function is used to retrieve an internal representation of the evaluated first output object from the current page of the report.
Parameters:	
report	This is a pointer to the report for which the first object of the current output page is retrieved.
Returns:	
>= 0 is returned.	An **OBJECT4** pointer for the evaluated first object of the current output page
0	There are no objects on the current page.
See Also:	**report4pageObjNext**

report4pageObjNext

Usage: OBJECT4 *report4pageObjNext(REPORT4 *report)

Description: This low-level function retrieves the next evaluated output object from the current page of the report.

Parameters:

report This is a pointer to the report for which the next object of the current output page is retrieved.

Returns:

>= 0 An **OBJECT4** pointer for the evaluated next object of the current output page is returned.

0 There are no objects on the current page.

See Also: **report4pageObjFirst**

report4pageSize

Usage: int report4pageSize(REPORT4 *report, long height, long width, int unitType)

Description: This function is used to set the vertical and horizontal page size for the report. If this function is not called within a Windows application, the current page size of the selected printer is used. If this function is not called in a non-Windows application, the default page size of 25 x 80 characters is used.

Parameters:

report This is a pointer to the report for which the page size is set.

height This is the vertical size of the output page in the specified units.

width This is the horizontal size of the output page in the specified units.

unitType This parameter is used to determine the unit of measure used by height and width. unitType may be one of the following values:

1 The height and width are in characters.

0 The ***height* and** *width* are in 1000ths of an inch.

Returns:

0 The page size was successfully set.

< 0 Error.

See Also: **report4margins, report4printerSelect**

report4pageSizeGet

Usage:	int report4pageSizeGet(REPORT4 *report, long *width, long *height)
Description:	This function retrieves the current size of the report page, in thousandths of an inch.
Parameters:	
report	This is a pointer to the report for which the page size is retrieved.
width	This is a pointer to a long variable which is to receive the horizontal width of the page.
height	This is a pointer to a long variable which is to receive the vertical height of the page.
Returns:	
0	The page size was successfully retrieved.
< 0	Error. *report* was invalid.
See Also:	**report4pageSize, report4marginsGet**

report4parent

Usage:	int report4parent(REPORT4 *report, HWND parent)
Description:	This function specifies the parent window used for report output. **report4do** disables the *parent* window while the report is being output and sends it the CRM_REPORTDONE message once the report output window has been closed.
Parameters:	
report	This is a pointer to the report for which the parent window is set.
parent	This is a Microsoft Windows window handle which is used for report output.
	It is necessary to call this function before **report4do**. Failure to do so can cause unpredictable results.
See Also:	**report4do, report4free**

report4printerSelect

Usage: void report4printerSelect(REPORT4 *report)

Description: This function invokes the "Printer Setup" common dialog to specify a printer for the report. In order for this function to work correctly, the application must be running under Microsoft Windows NT/95/98 (or higher) or have the Microsoft Windows common dialog dynamic link library COMMDLG.DLL in the system path.

Parameters:

report This is a pointer to the report that is configured for the selected printer.

report4printerDC

Usage: HDC report4printerDC(REPORT4 *report, HDC hDC)

Description: This function is used to specify the handle to a printer device context to which report output is sent.

This low-level function is useful if the handle to the printer device context is obtained using standard Windows function calls. For an interactive selection of the report output device, use **report4printerSelect**.

The specified printer device context is not used by **report4do** if **report4toScreen** is called to send output to the screen.

Parameters:

report This is a pointer to the report that is configured for the specified printer device context.

hDC This is a handle to the printer device context in which the report output is to be placed.

See Also: **report4toScreen, report4printerSelect**

report4querySet

Usage: int report4querySet(REPORT4 *report, char *queryExpr)

Description: This function sets a query for the relation set. The *queryExpr* expression is evaluated for each composite record. If the expression evaluates to a .TRUE. value, the record is used in the report. If *queryExpr* evaluates to a .FALSE. value, the record is ignored.

This function overwrites any query expression set with **relate4query_set**.

Parameters:

report	This is a pointer to the report for which a query is set.
queryExpr	This is a logical dBASE expression that is used to place a limit on the composite data file. If *queryExpr* is NULL, all the records of the composite data file are used within the report.

Field names in the query expression must use the data file qualifier. eg. **"DBF->NAME='SMITH'** "

Returns:

0	The query was successfully set.
< 0	Error.

See Also: **relate4query_set, relate4sort_set, report4sortSet**

report4retrieve

Usage:
```
REPORT4 *report4retrieve( CODE4 *cb, char *fileName, int openFiles,
                                             char *dataPath )
```

Description: This function retrieves a report file from disk and constructs the appropriate **REPORT4** structure. Implicitly, a relation set is also created along with a corresponding RELATE4 structure.

Report files created with CodeReporter and/or **report4save** are not necessarily portable from one operating system to another. If a report is needed on another platform, it may be necessary to link the report with CodeReporter generated source code.

Parameters:

cb	This is a pointer to the application's **CODE4** structure. This is used for memory management and error handling.
fileName	This is a null terminated character array which contains the drive, directory and file name of the report file. If no file name extension is provided, **report4retrieve** assumes a .REP extension.
openFiles	If *openFiles* contains a true value (non-zero), **report4retrieve** attempts to open the data files referenced in the report if they are not already open. If a referenced data file cannot be located, it and any dependant slave data files are not included in the report. Any output objects and/or expressions that use the missing data files are automatically removed from the report. If *openFiles* contains a false value (zero), all files are assumed to be open.
dataPath	If *dataPath* is NULL, **report4retrieve** uses the paths stored in the report file to locate the report's data files. If the report file does not include path names to the data files, **report4retrieve** assumes the data files are in the current directory. If *dataPath* is not NULL, it is assumed to be a null terminated character array containing the drive and/or directory where all of the report's

data files may be located. The dataPath directory overrides any paths stored within the report.

Returns:

Not Zero	The report was successfully loaded. The returned **REPORT4** pointer may be used with other report module functions.
0	Error. The report could not be loaded. This may result from an inability to locate the top master data file, or allocate enough memory for the report.

See Also: **relate4retrieve**

report4save

Usage: int report4save(REPORT4 *report, char *fileName, int savePaths)

Description: This function saves a report into a soft-coded report file which may be retrieved either by the CodeReporter application or **report4retrieve**.

report4save does not alter the report in memory in any way. It may be called before or after **report4do** with no ill effects.

If a report with graphic output objects is loaded in a non-Windows application and saved with **report4save**, the graphic output objects are not saved in the new report file.

Parameters:

report	This is a pointer to the report to be saved to disk.
fileName	This is a null terminated character array containing the drive, directory, and file name of the file in which the report is saved. If an extension is provided, it is used; otherwise the default extension of .REP is appended to the file name. If a drive and/or path is not provided, the current directory is assumed.
savePaths	If *savePaths* contains a true value (non-zero), **report4save** includes the drive and path for each file referenced in the report within the report file. If *savePaths* contains a false value (zero), only the file names are saved within the report.

Returns:

0	The report was successfully saved to the specified file.
< 0	Error. report was invalid, or the specified file could not be created.

See Also: **report4retrieve**

report4separator

Usage:	int report4separator(REPORT4 *report, char separator)
Description:	This function specifies the character to be used as the separator between hundreds and thousands, between thousands and millions, etc.
Parameters:	
report	This is a pointer to the report for which the numeric separator is specified.
separator	This is the character used as a numeric separator. If no numeric separator is desired, pass a zero (0) for *separator.* If this function is not called, a comma (,) is used as the default numeric separator.
Returns:	
0	The numeric separator was successfully set.
< 0	Error. *report* was invalid.
See Also:	**obj4numericType**

report4sortSet

Usage:	int report4sortSet(REPORT4 *report, char *sortExpr)
Description:	This function specifies the sorted order in which the composite records of the report are retrieved. This function overwrites any sort expression set with **relate4sort_set**.
Parameters:	
report	This is a pointer to the report for which the sorted order applies.
sortExpr	This is a null terminated character array which contains the dBASE expression used to sort the composite data file. This expression may evaluate to a Character, Date, or Numeric value. Field names in the query expression must use the data file qualifier. "DBF->NAME='SMITH' " is an example of a valid query expression.
Returns:	
0	Success.
< 0	Error or report was invalid.
See Also:	**relate4sort_set, report4querySet**

report4styleFirst

Usage:	STYLE4 *report4styleFirst(REPORT4 *report)
Description:	This function returns a pointer to the first style created for the report. This is useful, in conjunction with **report4styleNext**, for iterating through the styles created for a report.
Parameters:	
report	This is a pointer to the report which contains styles.
Returns:	A pointer to the first style created for the report is returned.
See Also:	**report4styleNext, report4styleLast**

report4styleLast

Usage:	STYLE4 *report4styleLast(REPORT4 *report)
Description:	This function returns a pointer to the last style created for a report. This function is useful, in conjunction with **report4stylePrev**, for iterating backwards through the styles of the report.
Parameters:	
report	This is a pointer to the report containing some styles.
Returns:	**report4styleLast** returns a pointer to the last style created. If there is only one style in the report, this function also points to the first style in the report.
See Also:	**report4stylePrev, report4styleFirst**

report4styleNext

Usage:	STYLE4 *report4styleNext
Description:	This function returns a pointer to the next style created for a report. This function is useful in conjunction with **report4styleFirst** to iterate forwards through the styles within a report.
Parameters:	
report	This is a pointer to the report containing some styles.
Returns:	
Not Zero	A **STYLE4** pointer to the next style in the report is returned.
0	There are no more styles within the report.
See Also:	**report4styleFirst, report4stylePrev**

report4styleSelect

Usage: int report4styleSelect(REPORT4 *report, STYLE4 *style)

Description: This function sets the specified style as the "selected" style. All new output objects are created with this style.

Parameters:

report This is a pointer to the report in which the style is selected.

style This is a **STYLE4** pointer to a previously created style that is set as the selected style.

Returns:

0 The style was successfully set.

< 0 Error. *report* or *style* were invalid.

report4styleSelected

Usage: STYLE4 *report4styleSelected(REPORT4 *report)

Description: This function returns a pointer to the reports "selected" style.

Parameters:

report This is a pointer to the report in which the style is selected.

Returns: **report4styleSelected** returns a **STYLE4** pointer to the previously created selected style. By default, the last style created is the selected style (unless **report4styleSelect** is used). If no styles have been created, this function returns a pointer to the report module default "Plain Text" style.

report4styleSheetLoad

Usage: int report4styleSheetLoad(REPORT4 *report, char *fileName, int overRide)

Description: This function adds the styles from a CodeReporter style sheet to the specified report.

Parameters:

report This is a pointer to the report in which the new styles are added.

fileName This is a null terminated character array containing the drive, directory, and file name of the CodeReporter style sheet. All CodeReporter style sheets have a .CRS extension. If *fileName* does not contain a drive and/or directory, the current directory is assumed.

overRide This parameter is used to resolve conflicts that occur when styles in the report and styles in the style sheet have the same name. If *overRide* contains a true value (non-zero) **report4styleSheetLoad** uses the styles in the style

	sheet when conflicts occur. If *overRide* contains a false value (zero), the original styles in the report are maintained.
Returns:	
1	The style sheet was successfully loaded.
0	Error. The file could not be found, it was corrupted, or it was out of date.

report4styleSheetSave

Usage:	int report4styleSheetSave(REPORT4 *report, char *fileName)
Description:	This function saves the styles within the specified report to a CodeReporter style sheet.
Parameters:	
report	This is a pointer to the report from which the styles are saved.
fileName	This is a null terminated character array containing the drive, directory, and file name of the file in which the styles are saved. If a drive and/or directory are not provided, the style sheet is saved in the current directory. All CodeReporter style sheets have a .CRS file extension.
Returns:	
1	The style sheet was successfully saved.
0	The style sheet could not be saved. This is usually do the attempting to save over the top of an existing file.

report4titlePage

Usage:	int report4titlePage(REPORT4 *report, int titlePage)
Description:	This function is used to force a page break after the title **REPORT** area(s) are outputted.
Parameters:	
report	This is a pointer to the report for which the title page setting applies
titlePage	This parameter determines whether or not a page break follows the title area. *titlePage* may have one of the following settings:
1	A page break is generated after the title area(s) are outputted.
0	A page break is not generated after the title area(s) are outputted. If this function is not called, this value is assumed.
Returns:	
>= 0	The previous *titlePage* setting is returned.
< 0	Error. report or *titlePage* were invalid.

report4titleSummary

Usage:	GROUP4 *report4titleSummary(REPORT4 *report)
Description:	This function is used to return a **GROUP4** pointer for the report's title/summary group.
Parameters:	
report	This is a pointer to the report from which the title/summary group is retrieved.
Returns:	
Not Zero	A **GROUP4** pointer to the group's title/summary group is returned.
0	Error. *report* is invalid.

report4toScreen

Usage:	int report4toScreen(REPORT4 *report, int toScreen)
Description:	This function is used to indicate that **report4do** should create a window and send the report output to it, or instead send the report to selected printer. By default, **report4do** sends report output to a window.
Parameters:	
report	This is a pointer to the report that is to be outputted to the screen.
toScreen	This parameter is used in Windows applications to determine where the report should be sent. *toScreen* may have one of the following values:

1 A window is created and report output is handled by the window procedure.

0 Report output is sent to the selected printer.

Returns:	
>= 0	The previous toScreen setting is returned.
< 0	Error. *report* or *toScreen* is invalid.
See Also:	**report4output, report4do**

style4 Functions

The **style4** functions are used to group a set of font attributes under a common name which may be associated with output objects using **obj4style**.

style4color

Usage: int style4color(STYLE4 *style, R4COLORREF color)

Description: This function changes the Windows RGB color for the specified style.

Parameters:

style	This is a pointer to the style for which the color is set.
color	This parameter is a Windows COLORREF value that describes the color for the specified style.

Returns:

0	The color was successfully set.
< 0	Error. *style* was invalid.

style4create

Usage: STYLE4 *style4create(REPORT4 *report, R4LOGFONT *font, char *name, R4COLORREF color, int pointSize)

Description: This function adds a new style to the report using the specified Windows font.

Parameters:

report	This is a pointer to the report with which the new style is associated.
font	This is a pointer to a Windows LOGFONT structure that describes the font used for the new style.
name	This is a null terminated character array containing the name of the new style. *name* may point to up to 19 characters.
color	This parameter is a Windows COLORREF value that describes the color for the new style.
pointSize	This is the size of the font, in points, used for the new style.

Returns:

Not Zero	The new style was successfully created. The returned pointer may be considered valid and may be used with other report module functions.

0	Error. The style was not created.

style4create

Usage:	STYLE4 *style4create(REPORT4 *report, char *name, int beforeLen, char *beforeCodes, int afterLen, char *afterCodes)
Description:	This function creates a non-Windows style which is used to store the printer-specific control codes that describe a particular printer typeface.
Parameters:	
report	This is a pointer to the report with which the new style is associated.
name	This is a null terminated character array containing the name of the new style. *name* may point to up to 19 characters.
beforeLen	This is the length of the printer control code character array that is sent before the text of the output object.
beforeCodes	This is a character array containing the printer control codes sent before the text of the output object. These codes should turn 'on' a specific printer attribute or typeface.
afterLen	This is the length of the printer control code character array that is sent after the text of the output object.
afterCodes	This is a character array containing the printer control codes sent after the text of the output object. These codes should turn 'off' a specific printer attribute or typeface.
Returns:	
Not Zero	The new style was successfully created. The returned pointer may be considered valid and may be used with other report module functions.
0	Error. The style was not created.

style4delete

Usage:	int style4delete(REPORT4 *report, char *styleName)
Description:	This function removes a named style from the report and frees any memory associated with the style. If the deleted style was the selected style, the first style in the report becomes the selected style.
Parameters:	
report	This is a pointer to the report that contains the style to be deleted.
styleName	This is a null terminated character array that contains the name of the style to be deleted. **style4delete** iterates through the styles in the report, comparing the stored names to *styleName*. If a match is found, the style is removed from the report.
Returns:	
1	The style was successfully located and removed.

0	A style with the *styleName* name was not found within the report.

style4free

Usage:	int style4free(REPORT4 *report, STYLE4 *style)
Description:	This function removes the specified style from the report and frees any memory associated with the style. If the deleted style was the selected style, the first style in the report becomes the selected style.
Parameters:	
report	This is a pointer to the report that contains the style to be deleted.
style	This is a pointer to the style to be deleted.
Returns:	
1	The style was successfully removed.
0	Error. *report* and/or *style* were invalid.

style4index

Usage:	STYLE4 *style4index(REPORT4 *report, int styleIndex)
Description: position.	This function returns a **STYLE4** pointer to the style in the *styleIndexth*
Parameters:	
report	This is a pointer to the report which contains the desired style.
styleIndex	This is an index into the reports internal style sheet. This function is used to quickly retrieve a pointer to the *styleIndexth* style in the report. The first style in the report is style 1 (one). This is used with the **report4pageObj** functions.
Returns:	
Not Zero	This is a **STYLE4** pointer to the specified style
0	*styleIndex* was greater than the number of styles within the report or zero.
See Also:	**style4lookup, report4pageObjFirst**

style4lookup

Usage:	STYLE4 *style4lookup(REPORT4 *report, char *styleName)
Description:	This function returns a **STYLE4** pointer to the style with the specified name.
Parameters:	
report	This is a pointer to the report which contains the desired style.
styleName	This is a null terminated character array which contains the name of the style that is looked up.
Returns:	
Not Zero	This is a **STYLE4** pointer to the named style.
0	A style with the styleName name could not be located in the specified report.
See Also:	**style4index**

total4 Functions

The **total4** functions are used to specify the information necessary for a total. Once created the **TOTAL4** pointer may be used with **obj4totalCreate** to add a total output object to the report.

total4addCondition

Usage:	int total4create(TOTAL4 *total, char *addConditionSrc, int logical)
Description:	This function is used to specify a conditional accumulation for the total output object. If *addConditionSrc* is a logical dBASE expression (and logical is non-zero), the total is accumulated whenever the condition evaluates to a .TRUE. value . If *addConditionSrc* is any other type of expression (and logical is zero), the total is accumulated when the value of the evaluated condition changes. If this function is not called, the total is accumulated for every record in the composite data file.
Parameters:	
total	This is a pointer to the total for which the conditional accumulation is applied.
addConditionSrc	This is a null terminated character array containing dBASE expression which is used to determine when the total is accumulated. This expression may evaluate to any type, excluding memo. When this evaluated expression evaluates to a .TRUE. value (and *logical* is non-zero) or if the evaluated expression changes (and logical is zero), the total is accumulated.
logical	This flag is used to determine whether the total is accumulated on a logical condition or a change of value. When *logical* contains a true value (non-zero) *addConditionSrc* is assumed to evaluate to a logical value. When logical contains a false value (zero), the total is accumulated only when the evaluated expression changes. It is possible to have *addConditionSrc* evaluate to a logical value and have logical be false (zero). In this case, the total would be accumulated when the evaluation of the expression changes from `.TRUE.` to `.FALSE.` and from `.FALSE.` to `.TRUE.`
Returns:	
0	The condition was successfully added to the total output object.
< 0	Error. An invalid parameter was passed, *addConditionSrc* did not evaluate to a logical value when logical was set to true (non-zero), or *addConditionSrc* could not be evaluated.
See Also:	**obj4totalCreate, expr4calc_create**

total4create

Usage:	TOTAL4 *total4create(REPORT4 *report, char *totalName, char *totalExpr, int type, char *resetExpr)
Description:	This function defines a total which is used in **obj4totalCreate**.
Parameters:	
report	This is a pointer to the report for which the total is to be added.
totalName	This is a null terminated character array containing the descriptive name used in other dBASE expressions to reference the total. This name may not contain spaces.
totalExpr	This is a null terminated character array containing a numeric dBASE expression upon which the total is created. This may simply be a data file field, or a calculation created with CodeBase 5 function **expr4calc_create**.
	All data file fields referenced within the *totalExpr* expression must have a field qualifier.
type	This flag is used to determine how the total is to maintain its value when the total output object is evaluated. *type* may have one of the following constant
values:	
total4average	This constant creates a total that maintains the arithmetic mean (average) value of the *totalExpr* expression.
total4highest	This constant creates a total that stores the highest value encountered for the *totalExpr* expression.
total4lowest	This constant creates a total that stores the lowest value encountered for the *totalExpr* expression.
total4sum	This constant creates a total that maintains an arithmetic sum of all values encountered for the *totalExpr* expression.
resetExpr	This is a null terminated character array that contains a dBASE expression which is used to determine when the value of the total output object is reset to its initial value.
Returns:	
Not Zero	A pointer to a successfully created total is returned.
0	There were problems parsing the dBASE expressions.
See Also:	**obj4totalCreate**, **expr4calc_create**

total4free

Usage:	void total4free(TOTAL4 *total)
Description:	This low-level function frees any memory associated with the total definition. This function is automatically called by **obj4totalFree**.
Parameters:	
total	This is a pointer to the total definition to be created.
See Also:	**obj4totalFree, total4create**

Appendix A: dBASE Functions

dBASE Expression Functions

The functions listed below can be used as a dBASE expression or as part of an dBASE expression. Like dBASE operators, constants, and fields, functions return a value. Functions always have a function name and are followed by a left and right bracket. Values (parameters) may be inside the brackets.

Function List

CTOD(Char_Value)

The character to date function converts a character value into a date value:

eg. " CTOD("11/30/88") "

The character representation is always in the format specified by the **Code4::dateFormat** member variable which is by default "MM/DD/YY".

DATE()

The system date is returned.

DAY(Date_Value)

Returns the day of the date parameter as a numeric value from "1" to "31".

eg. "DAY(DATE())"

Returns "30" if it is the thirtieth of the month.

DESCEND()

(Clipper Compatibility Only) Returns a complemented version of an expression.

DEL()

Returns "*" if the current record is marked for deletion. Otherwise " " is returned.

DELETED()

Returns .TRUE. if the current record is marked for deletion.

DTOC(Date_Value)

The date to character function converts a date value into a character value.

The format of the resulting character value is specified by the Code4::dateFormat member variable which is by default "MM/DD/YY".

eg. " DTOC(DATE()) "

Returns the character value "05/30/87" if the date is May 30, 1987.

DTOS(Date_Value)

The date to string function converts a date value into a character value. The format of the resulting character value is "CCYYMMDD".

e.g. ." DTOS(DATE()) "

Returns the character value "19870530" if the date is May 30, 1987.

IIF(Log_Value, True_Result, False_Result)

If 'Log_Value' is .TRUE. then IIF returns the 'True_Result' value. Otherwise, IIF returns the 'False_Result' value. Both True_Result and False_Result must be the same length and type. Otherwise, an error results.

eg. "IIF(VALUE << 0, "Less than zero   ", "Greater than zero")"

e.g. ."IIF(NAME = "John", "The name is John", "Not John   ")"

LTRIM(Char_Value)

This function trims any blanks from the beginning of the expression.

MONTH(Date_Value)

Returns the month of the date parameter as a numeric.

eg. " MONTH(DT_FIELD) "

Returns 12 if the date field's month is December.

PAGENO()

When using the report module or CodeReporter, this function returns the current report page number.

RECCOUNT()

The record count function returns the total number of records in the database:

eg. " RECCOUNT() "

Returns 10 if there are ten records in the database.

RECNO()

The record number function returns the record number of the current record.

STOD(Char_Value)

The string to date function converts a character value into a date value:

eg. " STOD("19881130") "

The character representation is in the format "CCYYMMDD".

STR(Number, Length, Decimals)

The string function converts a numeric value into a character value. "Length" is the number of characters in the new string, including the decimal point. "Decimals" is the number of decimal places desired. If the number is too big for the allotted space, *'s will be returned.

eg. " STR(5.7, 4, 2) " returns " '5.70' "

The number 5.7 is converted to a string of length 4. In addition, there will be 2 decimal places.

eg. " STR(5.7, 3, 2) " returns " '***' "

The number 5.7 cannot fit into a string of length 3 if it is to have 2 decimal places. Consequently, *'s are filled in.

SUBSTR(Char_Value, Start_Position, Num_Chars)

A substring of the Character value is returned. The substring will be 'Num_Chars' long, and will start at the 'Start_Position' character of 'Char_Value'.

eg. " SUBSTR("ABCDE", 2, 3)" returns " 'BCD' "

eg. "SUBSTR("Mr. Smith", 5, 1)" returns " 'S' "

TIME()

The time function returns the system time as a character representation. It uses the following format: HH:MM:SS.

e.g. " TIME() " returns " 12:00:00 " if it is noon.

e.g. " TIME() " returns " 13:30:00 " if it is one thirty PM.

TRIM()

This function trims any blanks off the end of the expression.

UPPER(Char_Value)

A character string is converted to uppercase and the result is returned.

VAL(Char_Value)

The value function converts a character value to a numeric value.

eg. " VAL('10') " returns " 10 "

eg. " VAL("-8.7") " returns " -8.7 "

YEAR(Date_Value)

Returns the year of the date parameter as a numeric:

eg. "YEAR(STOD('19920830')) " returns " 1992 "

Appendix B: Keyboard Interface

CodeReporter requires a Microsoft compatible mouse. There are certain actions, such as placing output objects, which may only be done with a mouse.

Most other actions may be performed both with the mouse and with the keyboard. This appendix systematically lists the keyboard controls and the actions they perform.

Menu Accelerators

Menu accelerators are a single keystroke that perform a menu action, thus saving the time and keystrokes necessary to activate many of the common CodeReporter tasks.

Ctrl-A

AREA | NEW HEADER AREA. This accelerator creates a new header area for the selected group.

Ctrl-C

EDIT | COPY. A copy of the currently selected output objects are placed within the Windows clipboard, where they may be retrieved in CodeReporter using **EDIT | PASTE**.

Ctrl-Insert

EDIT | COPY. A copy of the currently selected output objects are placed within the Windows clipboard, where they may be retrieved in CodeReporter using **EDIT | PASTE**.

Ctrl-E

ALIGN | CENTER. This accelerator moves the selected object so that its center is at the horizontal center of the report. If multiple objects are selected, this accelerator aligns the centers of all of the selected output objects with the center of the first object selected.

Ctrl-F

AREA | NEW FOOTER AREA. This accelerator creates a new footer area for the selected group.

Ctrl-G

GROUPS | NEW. This accelerator invokes the "Group Settings" dialog and creates a new group, including a header and footer area.

Ctrl-H

SENSITIVITY | SPACE HORIZONTAL. This accelerator moves all selected output objects so that the horizontal distance between them is equal. The first and the last selected output objects are not moved.

Ctrl-L

ALIGN | LEFT. This accelerator moves all selected output objects so that their left edges are aligned with the left edge of the first selected object.

Ctrl-M

GROUPS | MODIFY. This accelerator invokes the "Group Settings" dialog for the currently selected group.

Ctrl-O

AREA | MODIFY AREA. This accelerator invokes the "Modify Area" dialog for the selected area.

Ctrl-P

FILE | PRINT. This accelerator invokes the "Print" dialog which may be used to output the current report to the specified printer.

Ctrl-R

ALIGN | RIGHT. This accelerator moves all selected output objects so that their right edges are aligned with the right edge of the first selected object.

Ctrl-S

FILE | SAVE. This accelerator saves the current report to disk. If the report has not previously been saved, CodeReporter prompts the designer for a file name.

Ctrl-T

SENSITIVITY | SPACE VERTICAL. This accelerator moves all selected output objects so that the vertical distance between them is equal. The first and the last selected output objects are not moved.

Ctrl-V

EDIT | PASTE. CodeReporter is put into insertion mode for output objects within the Windows clipboard. If the Windows clipboard contains simple text from another application, CodeReporter creates a text output object. If the Windows clipboard contains a bitmap image, it is placed within the report as a static graphic object.

Shift-Insert

EDIT | PASTE. CodeReporter is put into insertion mode for output objects within the Windows clipboard. If the Windows clipboard contains simple text from another application, CodeReporter creates a text output object. If the Windows clipboard contains a bitmap image, it is placed within the report as a static graphic object.

Ctrl-W

FILE | PRINT PREVIEW. This accelerator creates a full screen window and outputs the report within the window.

Ctrl-X

EDIT | CUT. The currently selected output objects are removed from the report and placed within the Windows clipboard, where they may be retrieved in CodeReporter using **EDIT | PASTE**.

Shift-Delete

EDIT | CUT. The currently selected output objects are removed from the report and placed within the Windows clipboard, where they may be retrieved in CodeReporter using **EDIT | PASTE**.

Delete

OBJECT | DELETE. This accelerator deletes the currently selected output object. If no object is selected, nothing happens.

Esc

OBJECT | NONE. This accelerator moves CodeReporter out of insertion mode.

Report Design Screen

The report design screen accepts the following keystrokes as equivalencies for many mouse generated actions.

Tab

The Tab key sets the next object in an area as the "selected" output object. If the Tab key is pressed while the last object in an area is selected, the first object is selected. If no object is selected, pressing the Tab key selects the first object in the selected area.

Shift-Tab

The Shift-Tab key combination performs the same selection process as the Tab key. The only difference is that the Shift-Tab key combination cycles backwards through the objects in the selected area.

Ctrl-Tab

The Ctrl-Tab key combination is used to multiply select the output objects in the selected area. This key combination selects the next object in the area while keeping all previously selected output objects selected.

Shift-Ctrl-Tab

The Shift-Ctrl-Tab key combination performs the same selection process as the Ctrl-Tab combination. The only difference is that this combination cycles backwards through the objects in the selected area.

Page Up

These keystrokes scroll the report design screen up one screen full. If all of the report design elements are visible on the current screen, these keys do nothing.

Shift-Up Arrow

These keystrokes scroll the report design screen up one screen full. If all of the report design elements are visible on the current screen, these keys do nothing.

Page Down

These keystrokes scroll the report design screen down one screen full. If all of the report design elements are visible on the current screen, these keys do nothing.

Shift-Down Arrow

These keystrokes scroll the report design screen down one screen full. If all of the report design elements are visible on the current screen, these keys do nothing.

Shift-Left Arrow

This keystroke scrolls the report design screen to the left a little bit. If all of the report design elements are visible on the current screen, this key does nothing.

Shift-Right Arrow

This keystroke scrolls the report design screen to the right a little bit. If all of the report design elements are visible on the current screen, this key does nothing.

Ctrl-Left Arrow

This keystroke scrolls the report design screen all the way to the right edge of the window.   If all of the report design elements are visible on the current screen, this key does nothing.

Ctrl-Right Arrow

This keystroke scrolls the report design screen all the way to the right edge of the window.   If all of the report design elements are visible on the current screen, this key does nothing.

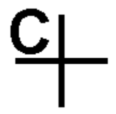
Return

This keystroke invokes the Object Menu for the currently selected output object. If no objects are selected, this key does nothing.

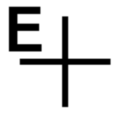
Appendix C: Cursors

When the report designer has put CodeReporter into insertion mode for a particular output object type, the mouse cursor alters its shape to indicate the insertion mode as well as the type of object being inserted. Listed below are the different types of cursors and the type of output object they insert. Each cursor has a "cross-hairs". The intersection of the two lines indicates the position of the upper left corner the new output object occupies.

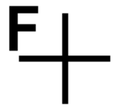
CodeReporter may be moved out of insertion mode by selecting the "None" button on the button bar, or by pressing the ESC key.



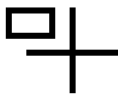
This cursor is used to indicate that CodeReporter is in insertion mode for calculation output objects.



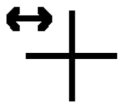
This cursor is used to indicate that CodeReporter is in insertion mode for expression output objects.



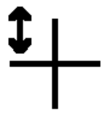
This cursor is used to indicate that CodeReporter is in insertion mode for field output objects. If multiple fields were selected, they are placed horizontally or vertically (as set in the "Field Layout" dialog") from this point.



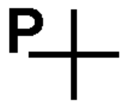
This cursor is used to indicate that CodeReporter is in insertion mode for frame output objects.



This cursor is used to indicate that CodeReporter is in insertion mode for horizontal line output objects. Lines are created with a default length.



This cursor is used to indicate that CodeReporter is in insertion mode for vertical line output objects. Lines are created with a default length.



This cursor indicates that CodeReporter is in insertion mode for objects placed in the Windows clipboard. If multiple output objects were cut or copied to the clipboard, they are pasted in relation to the first object pasted.

This cursor is used to indicate that CodeReporter is in insertion mode for text output objects.

This cursor is used to indicate that CodeReporter is in insertion mode for total output objects.

Appendix D: ASCII Chart - Partial

Listed below are the most commonly used characters in printer control codes, and their hexadecimal equivalent. Most printer manuals list the hexadecimal values for the control codes. This is provided merely as an additional reference.

ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex
ESC	27	1B	D	68	44	h	104	68
!	33	21	E	69	45	i	105	69
"	34	22	F	70	46	j	106	6A
#	35	23	G	71	47	k	107	6B
$	36	24	H	72	48	l	108	6C
%	37	25	I	73	49	m	109	6D
&	38	26	J	74	4A	n	110	6E
'	39	27	K	75	4B	o	111	6F
(	40	28	L	76	4C	p	112	70
)	41	29	M	77	4D	q	113	71
*	42	2A	N	78	4E	r	114	72
+	43	2B	O	79	4F	s	115	73
,	44	2C	P	80	50	t	116	74
-	45	2D	Q	81	51	u	117	75
.	46	2E	R	82	52	v	118	76
/	47	2F	S	83	53	w	119	77
0	48	30	T	84	54	x	120	78
1	49	31	U	85	55	y	121	79
2	50	32	V	86	56	z	122	7A
3	51	33	W	87	57	{	123	7B
4	52	34	X	88	58	|	124	7C
5	53	35	Y	89	59	}	125	7D
6	54	36	Z	90	5A	~	126	7E
7	55	37	[	91	5B			
8	56	38	\	92	5C			
9	57	39	]	93	5D			
:	58	3A	^	94	5E			
;	59	3B	_	95	5F			
<	60	3C	`	96	60			

=	61	3D	a	97	61			
>	62	3E	b	98	62			
?	63	3F	c	99	63			
@	64	40	d	100	64			
A	65	41	e	101	65			
B	66	42	f	102	66			
C	67	43	g	103	67			

Appendix E: Error Codes

This appendix documents the error codes that are returned by the CodeReporter API functions when an error occurs. Other CodeBase error codes, which may be returned as well, are documented in the respective CodeBase reference manual.

Constant Name	Value	Meaning
e4report	-810	**Report Error** General reporting error. Any report error not covered below is an e4report error.
e4style_create	-811	**Error Creating Style** The style could not be created due to a duplicate style name existing, or a memory shortage.
e4style_select	-812	**Error Selecting Style** An invalid **STYLE4** pointer was provided to a style selection function.
e4style_index	-813	**Error Finding Style** The specified style could not be located. This may result from either passing an invalid **STYLE4** pointer, or from passing a non-existant style name to a style look up function.
e4area_create	-814	**Error Creating Area** The area could not be created due to a failure to allocate memory for the area or parse the suppression expression.
e4group_create	-815	**Error Creating Group** The group could not be created due to a lack of memory, or invalid **GROUP4** parameter.
e4group_expr	-816	**Error Setting Group Reset Expression** The group reset expression could not be parsed correctly.
e4total_create	-817	**Error Creating Total** The total output object could not be created due to a lack of memory, due to an invalid numeric calculation, or an invalid

		reset expression.
e4obj_create	-818	**Error Creating Object** The output object could not be created due to a lack of memory, or to an invalid **AREA4** pointer.
e4rep_win	-819	**Error In Windows Output** An error occurred registering the Windows output window class, or CreateWindow failure.
e4rep_out	-820	**Error In Report Output** The evaluation of an output object caused an error. This is usually due to a lack of memory.
e4rep_save	-821	**Error Saving Report** An error occurred while saving a report file. This may be due to the file already existing, a lack of disk space, or a file creation problem.
e4rep_ret	-822	**Error Retrieving Report** The report file could not be retrieved from disk. This may be a result of not being able to locate the top master data file, or from the report file being corrupt.
e4rep_data	-823	**Error in creating output data file** The report could not be sent to an output data file due to a file creation error, or a data storage error.

Appendix F: Basic/Pascal API

This section documents the available functions for running and modifying reports when using the CodeBase Basic or Pascal API. Using these functions you can perform such operations as retrieving reports from disk, displaying and modifying reports on the fly, and saving any changes back to disk.

All the functions listed in this section, except for three, are **report4** functions. The three exceptions are **relate4** functions. These functions can be used in conjunction with CodeReporter's **FILE | SAVE RELATION** option to visually build sophisticated relations that can be saved to disk. These relations can then be retrieved for use in your application.

PROGRAM REPTEST.BAS Visual Basic test application

```
Sub ReportTest (cb As Long, Report As Form)

  Dim lReport As Long

    'Retrieve report file TUT1.REP
    lReport = report4retrieve(cb, App.Path + "TUT1", 1, App.Path)

    'Check for Error
    If  lReport = 0 Then
       rc = code4errorCode(cb, 0)
      Exit Sub
    End If

    'Change some of the report's attributes
    rc = report4caption(lReport, "New Caption")
    rc = report4currency(lReport, "£")

    'Change report's relation set
    rc = report4querySet(lReport, "STUDENT->L_NAME > 'R'")
    rc = report4sortSet(lReport, "STUDENT->L_NAME")

    'Save changes to a different file
    rc = report4save(lReport, App.Path + "\STUDENT2", 1)

    'Set the report's Parent handle before displaying report
    rc = report4parent(lReport, Report.hWnd)

    'Send report to screen   rc = report4do(lReport)
    'Set the output printer   Call report4printerSelect(lReport)

    'Now send to printer
    rc = report4toScreen(lReport, 0)
    rc = report4do(lReport)

    'Free memory and close files
   Call report4free(lReport, 1, 1)

End Sub
```

relate4retrieve

VB Usage: RELATE4& = relate4retrieve(CODE4&, fileName$, openFiles%, dataPathName$)

Delphi Usage: Function relate4retrieve (c4 : CODE4 ; fileName : PChar; openFiles : Integer; dataPathName : PChar) : RELATE4;

Description: This function retrieves a relation file and constructs the relation that was saved with **relate4save**. In the process of loading the relation file, **relate4retrieve** may also open the relation's data files.

For complex relations, use CodeReporter and it's **RELATION/SAVE RELATION** menu option to visually build the relation set and save it to disk. You can then retrieve this relation into your application with **relate4retrieve.**

Parameters: (vb/delphi)

CODE4/c4 This is a pointer to the application's **CODE4** structure. This is used for memory management and error handling.

fileName This is a null terminated character array which contains the file name (including drive and directory) of the relation file. A file extension need not be provided since the .REL extension is always used.

openFiles If *openFiles* is a true value (non-zero), **relate4retrieve** attempts to open the data, index, and memo files referenced in the saved relation file if they are not already opened. If **relate4retrieve** cannot find a certain data file referenced in the relation file, that file and all lower level slave data files of that file are omitted from the relation and an attempt is made to locate the next data file.

If *openFiles* is a false value (zero), **relate4retrieve** assumes that all of the data, index, and memo files are already opened. If a data file referenced in the relation file is not opened, that file and all lower level slave data files in the relation are omitted from the relation and the **relate4retrieve** continues to build the relation.

dataPathName	This parameter is a null terminated character array containing a new drive and path for the data, index, and memo files stored in the relation file. If *dataPathName* is NULL, the paths stored in the relation file are used. If no paths were stored in the file, **relate4retrieve** attempts to open the files in the current directory. If *dataPathName* is specified, it is used to override the paths saved within the relation file.
Returns:	
Not Zero	The relation was successfully retrieved from the specified relation file.
Zero	An error occurred while reading the relation file or opening the relation's top master data file. See the **CODE4.error_code** member variable for the specific error setting.
See Also:	**relate4save, relate4init, relate4free**

relate4save

VB Usage:	rc% = relate4save(RELATE4&, fileName$, savePathNames%)
Delphi Usage:	Function relate4save (r4 : RELATE4 ; fileName : PChar; savePathNames : Integer) : Integer ;
Description:	This function saves the specified relation in a relation file.
Parameters: (vb/delphi)	
RELATE4/r4	This is a pointer to the relation that is to be saved to a relation file.
fileName	This is a null terminated character array which contains the file name (including drive and directory) of the relation file. A file extension need not be provided since the .REL extension is always used.
savePathNames	If this parameter contains a true value (non-zero), **relate4save** saves the full path name of the files used in the relation. If *savePathNames* contains a false value (zero), only the actual file name is saved.
Returns:	
0	The relation file was successfully saved.
r4no_create	The relation file could not be created. This is generally caused when *fileName* conflicts with a file that already exists, or if the application does not have read/write privileges to the desired drive.
< 0	Error.
See Also:	**relate4retrieve**

relate4topMaster

VB Usage:	RELATE4& = relate4topMaster(RELATE4&)
Delphi Usage:	Not available.
Description:	This function returns a pointer to the **RELATE4** structure of the top master in the relation set. This function is only valid from Visual Basic.
Parameters: (vb/delphi)	
RELATE/r4	This is a pointer to any **RELATE4** structure in the relation set.
Returns:	
Not Zero	A pointer to the **RELATE4** structure of the top master relation.
0	Error.

report4caption

VB Usage:	rc% = report4caption(REPORT4&, caption$)
Delphi Usage:	Function report4caption (r4 : REPORT4; caption : PChar) : Integer;
Description:	This function sets the text of the caption for the report output window when the report is sent to the screen.
Parameters: (vb/delphi)	
REPORT4/r4	This is a **REPORT4** pointer to the report for which the window caption is set.
caption	This is a string containing the text to be placed in the caption portion of the output window. **report4caption** makes a copy of *caption.*
Returns:	
0	The caption was set successfully.
< 0	Error.

report4currency

VB Usage:	rc% = report4currency(REPORT4&, currency$)
Delphi Usage:	Function report4currency (r4 : REPORT4; currency : Char) : Integer;
Description:	This function sets the text to be displayed immediately to the left of numeric output objects that are formatted as currency values.
Parameters: (vb/delphi)	
REPORT4/r4	This is a **REPORT4** pointer to the report for which the currency characters are set.
currency	This is a string containing the currency symbol(s). *currency* may contain up to ten (10) characters. **report4currency** makes a copy of *currency*. If this function is not called, the dollar symbol ($) is assumed.
Returns:	
0	The currency character(s) were set successfully.
< 0	Error.

report4dateFormat

VB Usage:	rc% = report4dateFormat(REPORT4&, format$)
Delphi Usage:	Function report4dateFormat (r4 : REPORT4; format : PChar) : Integer;
Description:	This function sets the default date format for the specified report. All new output objects that evaluate to a date value, by default, use this format for output. When the report is initially created, the value of the **CODE4.date_format** member variable is stored within the report's default date format.
Parameters: (vb/delphi)	
REPORT/r4	This is a pointer to the report for which the date format is set.
format	This is a string which contains the date format to be used. This string should contain the picture formatting characters ('D', 'M', 'C', 'Y'). **report4dateFormat** creates a copy of *format*.
Returns:	
0	Success.
< 0	Error. *REPORT4* was invalid.

report4decimal

VB Usage:	rc% = report4decimal(REPORT4&, decimalChar$)
Delphi Usage:	Function report4decimal (r4 : REPORT4; decimalChar : Char) : Integer;
Description:	This function specifies the character to be used as the decimal separator between the whole a fractional portion of a number in a numeric output object.
Parameters: (vb/delphi)	
REPORT4/r4	This is a **REPORT4** pointer to the report for which the decimal character is used.
decimalChar	This is the character used as the decimal separator. The default character is the decimal point ('.').
Returns:	
0	Success.
< 0	Error. *REPORT4* was invalid.

report4do

VB Usage:	rc% = report4do(REPORT4&)
Delphi Usage:	Function report4do (r4 : REPORT4) : Integer;
Description:	This function causes the specified report to be outputted to the selected device.

When outputting the report under Windows, **report4do** disables the report's parent window (specified by **report4parent**) until the report window has been closed. This prevents the application from possibly updating any report-specific database information while the report is executing.

For Windows programs, you must call **report4parent** before calling this function. Failure to do can cause unpredictable results.

Parameters: (vb/delphi)	*REPORT4/r4* specifies the report to be outputted.
Returns:	
0	Success. The report was successfully outputted.
r4terminate	A relation was unable to be made and the error action specified with **relate4errorAction** was **relate4terminate**.
< 0	Error.
See Also:	**report4toScreen, report4printerSelect, report4output**

report4free

VB Usage:	Call report4free(REPORT4&, freeRelate%, closeFiles%)
Delphi Usage:	Procedure report4free (r4 : REPORT4; freeRelate : Integer; closeFiles : Integer);
Description:	This function frees all memory associated with the report.
Parameters: (vb/delphi)	
REPORT4/r4	This is a **REPORT4** pointer which specifies the report to be freed from memory.
freeRelate	If this parameter contains a true value (1), the memory associated with the report's relation is automatically freed. If a false value (0) is passed, the relation is unaffected.
closeFiles	Setting this parameter to a true value (1), causes **REPORT4FREE** to automatically close the data, index, and memo files referenced in the report. If *closeFiles* is false, or if *freeRelate* is false, this setting is ignored.

report4margins

VB Usage:	rc% = report4margins(REPORT4&, left&, right&, top&, bottom&, unitType%)
Delphi Usage:	Function report4margins (r4 : REPORT4; left, right, top, bottom : Longint; unitType : Integer) : Integer;
Description:	This function is used to change the default margins of the report.

Some output devices, such as laser printers, have a hardware margin which is not under software control. **report4margins** checks for this condition and will not allow the margins to violate the physical margins of the device.

Parameters: (vb/delphi)	
REPORT4/r4	This is a **REPORT4** pointer to the report for which the margins are set.
left	This is the size of the left margin in the provided increments.
right	This is the size of the right margin in the provided increments.
top	This is the size of the top margin in the provided increments.
bottom	This is the size of the bottom margin in the provided increments.

unitType This is the unit of measure for the above margin settings. In graphical user interfaces, 1000ths of an inch may conveniently be used. In character-based interfaces, it is often more convenient to use characters. *unitType* may be one of the following values:

1	The units listed are in characters.
0	The units listed are in 1000ths of an inch.

Returns:

0	The margins were successfully set.
< 0	Error.

See Also: **report4pageSize**

report4pageSize

VB Usage: rc% = report4pageSize(REPORT4&, height&, width&, unitType%)

Delphi Usage:
```
Function  report4pageSize ( r4 : REPORT4; height : Longint;
                     width : Longint; unitType : Integer ) : Integer;
```

Description: This function is used to set the vertical and horizontal page size for the report. For a Windows application, the default setting is the current page size of the selected printer. For a non-Windows application, the default setting is 25x80 characters.

Parameters: (vb/delphi)

REPORT/r4 This is a pointer to the report for which the page size is set.

height This is the vertical size of the output page in the specified units.

width This is the horizontal size of the output page in the specified units.

unitType This parameter is used to determine the unit of measure used by *height* and *width*. unitType may be one of the following values:

1	The *height* and *width* are in characters.
0	The ***height* and** *width* are in 1000ths of an inch.

Returns:

0	The page size was successfully set.
< 0	Error.

See Also: **report4margins, report4printerSelect**

report4parent

VB Usage:	rc% = report4parent(REPORT4&, Form.hWnd%)
Delphi Usage:	Function report4parent (r4 : REPORT4; hW : HWND) : Integer;
Description:	This function designates the parent window handle of the window created for report output. *Form.hWnd* should be the **.hWnd** property of the form in your Visual Basic application where focus will be returned to when the report window is closed.
Parameters: (vb/delphi)	
REPORT4/r4	This is a **REPORT4** pointer to the report for which the parent window is set.
Form.hWnd/hW	This is a Microsoft Windows window handle to the form whose focus will be set to when the report is finished. If output is begin sent to a window, this form will be disabled until the report window is closed.
Returns:	
0	Success
< 0	Error. *REPORT4* is invalid.

For Windows applications, this function must be called before calling **report4do**. Failure to do so can cause unpredictable results.

report4printerSelect

VB Usage:	Call report4printerSelect(REPORT4&)
Delphi Usage:	Procedure report4printerSelect (r4 : REPORT4);
Description:	This function invokes the "Printer Setup" common dialog to specify a printer for the report. This function requires the presence of the Microsoft Windows 3.1 Dynamic Link Library, COMMDLG.DLL. Normally the Windows Setup application installs this file in your \WINDOWS\SYSTEM sub-directory. If your application runs under Windows 3.0, or you don’t have this file installed in an appropriate directory, this function will not succeed.
Parameters: (vb/delphi)	
REPORT4/r4	This is a **REPORT4** pointer to the report that is configured for the selected printer.

report4querySet

VB Usage:	rc% = report4querySet(REPORT4&, queryExpr$)
Delphi Usage:	Function report4querySet (r4 : REPORT4; queryExpr : PChar) : Integer;
Description:	This function sets a query for the report's relation set. The *queryExpr* expression is evaluated for each composite record. If the expression evaluates to a .TRUE. value, the record is used within the report. If *queryExpr* evaluates to a .FALSE. value, the record is ignored.
Parameters: (vb/delphi)	
REPORT/r4	This is a **REPORT4** pointer to the report for which a query is set.
queryExpr	This is a logical dBASE expression that is used to place a limit on the composite data file. If *queryExpr* is a null string (“”), all the records of the composite data file are used within the report. Field names in the query expression must use the data file qualifier. eg. "DBF->NAME='SMITH' " is an example of a valid query expression.
Returns:	
0	The query was successfully set.
< 0	Error.
See Also:	**relate4querySet, relate4sortSet, report4sortSet**

report4relate

VB Usage:	RELATE4& = report4relate(REPORT4&)
Delphi Usage:	Function report4relate (r4 : REPORT4) : RELATE4;
Description:	This function returns a pointer to the **RELATE4** structure associated with the report.
Parameters: (vb/delphi)	
REPORT4/r4	This is a **REPORT4** pointer to the report for which the associated **RELATE4** pointer is returned.
Returns:	
Not Zero	The **RELATE4** pointer associated with the report.
0	Error.
See Also:	**report4querySet, report4sortSet**

report4retrieve

VB Usage: REPORT4& = report4retrieve(CODE4&, fileName$, openFiles%, dataPath $)

Delphi Usage:
```
Function  report4retrieve ( c4 : CODE4; fileName : PChar;
                  openFiles : Integer; datapath : PChar ) : REPORT4;
```

Description: This function retrieves a report file from disk and constructs the appropriate **REPORT4** structure.

Implicitly, a relation set is also created along with a corresponding RELATE4 structure.

Parameters: (vb/delphi)

CODE4/c4 This is a pointer to the application's **CODE4** structure. This is used for memory management and error handling.

fileName This is a string which contains the drive, directory and file name of the report file. If no file name extension is provided, **report4retrieve** assumes a .REP extension.

openFiles If *openFiles* contains a true value (1), **report4retrieve** attempts to open the data files referenced in the report if they are not already open. If a referenced data file cannot be located, it and any dependant slave data files are not included in the report. Any output objects and/or expressions that use the missing data files are automatically removed from the report.

If *openFiles* contains a false value (0), all files are assumed to be open.

dataPath If *dataPath* is a null string (“”), **report4retrieve** uses the paths stored in the report file to locate the report's data files. If the report file does not include path names to the data files, **report4retrieve** assumes the data files are in the current directory.

If *dataPath* is not null, it is assumed to be a string containing the drive and/or directory where all of the report's data files may be located. The *dataPath* directory overrides any paths stored within the report.

Returns:

Not Zero The report was successfully loaded. The returned **REPORT4** pointer may be used with other report module functions.

0 Error. The report could not be loaded. This may result from an inability to locate the top master data file, or allocate enough memory for the report.

See Also: **relate4retrieve**

report4save

VB Usage:	rc% = report4save(REPORT4&, fileName$, savePaths%)
Delphi Usage:	Function report4save (r4 : REPORT4; fileName : PChar; savePath : Integer) : Integer;
Description:	This function saves a report into a soft-coded report file which may be retrieved either through CodeReporter or by calling **report4retrieve**.

> **report4save** does not alter the report in memory in any way. It may be called before or after **report4do** with no ill effects.

> If a report with graphic output objects is loaded in a non-Windows application, and saved with **report4save**, the graphic output objects are not saved in the new report file.

Parameters:
(vb/delphi)

REPORT4/r4	This is a **REPORT4** pointer to the report to be saved to disk.
fileName	This is a string containing the drive, directory, and file name of the file in which the report is saved. If an extension is provided, it is used; otherwise the default extension of .REP is appended to the file name. If a drive and/or path is not provided, the current directory is assumed.
savePaths	If *savePaths* contains a true value (1), **report4save** includes the drive and path for each file referenced in the report within the report file. If *savePaths* contains a false value (0), only the file names are saved within the report.

Returns:

0	The report was successfully saved to the specified file.
< 0	Error. *REPORT4* report was invalid, or the specified file could not be created.

See Also: **report4retrieve**

report4separator

VB Usage:	rc% = report4separator(REPORT4&, separator$)
Delphi Usage:	Function report4separator (r4 : REPORT4; separator : Char) : Integer;
Description:	This function specifies the character to be used as the separator between hundreds and thousands, between thousands and millions, etc.
Parameters: (vb/delphi)	
REPORT/r4	This is a **REPORT4** pointer to the report for which the numeric separator is specified.
separator	This is the character used as a numeric separator. If no numeric separator is desired, pass a null string (“”) for *separator.*
	If this function is not called, a comma (‘,’) is used as the default numeric separator.
Returns:	
0	The numeric separator was successfully set.
< 0	Error. *REPORT4* was invalid.

report4sortSet

VB Usage:	rc% = report4sortSet(REPORT4&, sortExpr$)
Delphi Usage:	Function report4sortSet (r4 : REPORT4; sortExpr : PChar) : Integer;
Description:	This function specifies the sorted order in which the composite records of the report are retrieved.
	This function overwrites any sort expression set with **relate4sort_set**.
Parameters: (vb/delphi)	
REPORT4/r4	This is a **REPORT4** pointer to the report for which the sorted order applies.
sortExpr	This is a string which contains the dBASE expression used to sort the composite data file. This expression may evaluate to a Character, Date, or Numeric value.
	Field names in the query expression must use the data file qualifier. "DBF->NAME='SMITH' " is an example of a valid query expression.
Returns:	
0	Success.
< 0	Error or *REPORT4* was invalid.
See Also:	**relate4sortSet, report4querySet**

report4toScreen

VB Usage:	rc% = report4toScreen(REPORT4&, toScreen%)
Delphi Usage:	Function report4toScreen (r4 : REPORT4; toScreen : Integer) : Integer;
Description:	This function is used to indicate whether **report4do** should create a window and send the report output to it, or instead send the report to selected printer. The default action for **report4do** is to send the report output to a window.
Parameters: (vb/delphi)	
REPORT4/r4	This is a **REPORT4** pointer to the report that is to be outputted.
toScreen	This parameter specifies where the report output should go. *toScreen* may have one of the following values: 1 A window is created and report output is sent to the window. 0 Report output is sent to the selected printer.
Returns:	
>= 0	The previous *toScreen* setting is returned.
< 0	Error. *REPORT4* or *toScreen* is invalid.
See Also:	**report4output, report4do**

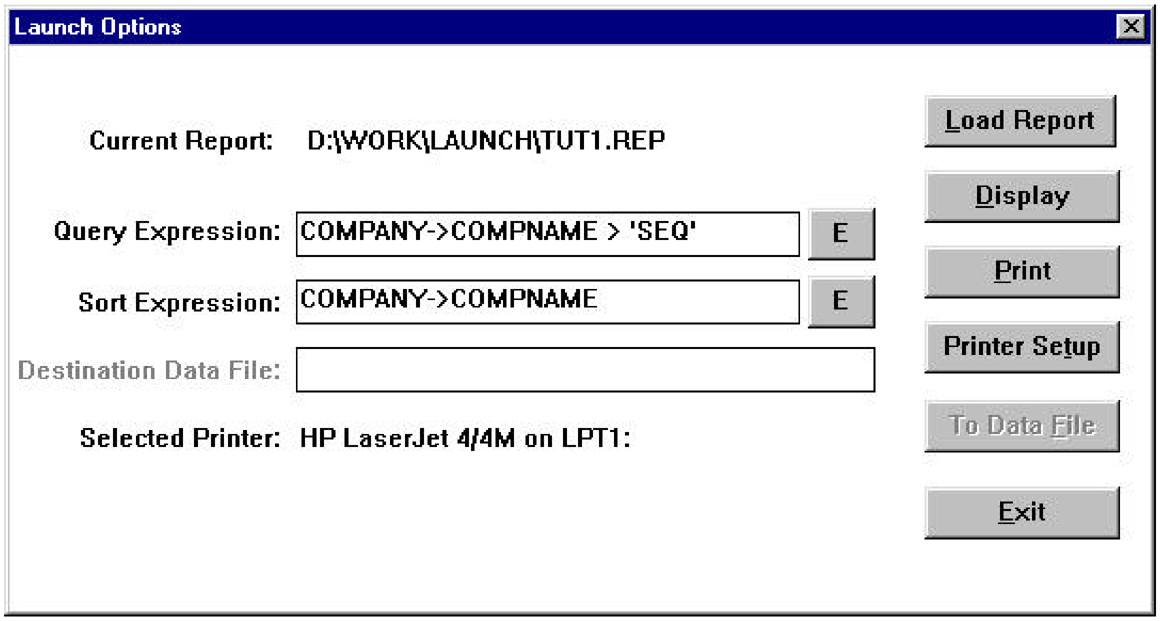
Appendix G: Launch Utilities

CodeReporter ships with utility programs which may be used to output CodeReporter report files outside of CodeReporter. These utilities are described below.

Windows

CodeReporter provides a Windows utility program which may be used to output CodeReporter report files. This program, called LAUNCH_W.EXE, is a Microsoft Windows executable that can be used to quickly view or print reports from Windows without loading CodeReporter. LAUNCH_W has an interactive interface as well as a command line interface for easy use in icons.

The source code, LAUNCH_W.C, is provided as an added example of using CodeReporter report files and the report functions under Windows.


Launch Options

Current Report: D:\WORK\LAUNCH\TUT1.REP

Query Expression: COMPANY->COMPNAME > 'SEQ'

Sort Expression: COMPANY->COMPNAME

Destination Data File:

Selected Printer: HP LaserJet 4/4M on LPT1:

When LAUNCH_W is executed without command line parameters, the "Launch Options" dialog is invoked with all controls but the "Load Report" and "Exit" buttons disabled. Use "Load Report" to invoke the "Specify Report" dialog and select a report file, or choose "Exit" to end the application. The "Launch Options" dialog is used to determine the destination of the report, and set the sort and/or query expressions.

Load Report	If the current report was loaded in error, or if another report is to be outputted, use the "Load Report" button to invoke the "Select Report" dialog. Use this dialog to locate and open a new report file. When the "Launch Options" dialog returns, the new report is loaded.
Display	Clicking on the "Display" button outputs the report to a window. The page size of the window is set to that of the currently selected printer, and so it may be necessary to use the scroll bars to view the entire page of the report.
Print	Clicking on the "Print" button causes the launch utility to print the report to the currently selected printer. During the printing of the report, a dialog is displayed to indicate the report is being printed. The "Cancel" button of this dialog may be used to stop the printing of the report.
Printer Setup	If the report is not to be outputted to the Windows default printer, or if the printer's settings must be modified, the "Printer Setup" button may be used to invoke the "Print Setup" common dialog. This is used to specify a printer to which the report is outputted, and to configure it.
To Data File	If the report file includes an output data file template definition (made with **REPORT | OUTPUT FILE TEMPLATE**), the "To Data File" button will be enabled. Selecting this button causes the report to be directed to the data file specified in the "Destination Data File" edit control. If the report does not include a data file template definition, the "To Data File" button is disabled and the report can not be outputted to a data file.
Sort Expression	The report launch utility also provides the ability to modify the sort and query expressions that are saved in the report. A change in the contents of the "Sort Query Expression" and/or "Query Expression" edit controls is reflected in the output of the report.

For more information on sorting and querying a report, see the "Relational Reporting" chapter.

Once the report is outputted, the launch utility returns to this dialog.

Command Line

LAUNCH_W accepts command line parameters which may be used to automate the load and display process. These parameters may be added to the LAUNCH_W Properties or specified in the Program Manager's File | Run option. See the *Windows User's Guide* for more information about using command line parameters with Windows applications.

Usage: LAUNCH_W [{*name*} [-q{*expr*}] [-s{*expr*}] [[-v | -p]

| [{ -d{*name*}} [-q{*expr*}][-s{*expr*}]] [. . .]]]

name	This is the name of the first report to be outputted or loaded.
-*d{name}*	If subsequent reports are to be outputted, they may be specified using the -d option. This option may be considered as a separator between reports.

-q{expr}	Change the default query expression. Query expressions must be entered in double quotes ("). For example*:*

```
LAUNCH_W SAMPLE.REP -q “DBF->FIELDNAME > 'LINCOLN'"
```

String literals, such as 'Lincoln', above, must be entered in single quotes (').

-s{expr}	Change the default sort expression. Sort expressions must be entered in double quotes ("). For example:

```
LAUNCH_W SAMPLE.REP -s “DBF->FIELDNAME"
```

String literals, such as 'Lincoln', above, must be entered in single quotes (').

-v	The -v option automatically displays the specified report to the monitor.
-p	The *-p* option automatically prints the report to the Windows default printer. When either of these options is specified, LAUNCH_W is invoked minimized.

If neither option is specified, the specified report is loaded, and the "Launch Options" dialog is invoked -- only the first report is loaded.

Both -p and -v may not be specified. This setting affects all loaded reports.

Non-Windows

CodeReporter also provides a set of non-Windows versions of the launch utility. In the stand-alone configuration of CodeBase, three different version are installed—one to support each supported file formats (FoxPro, dBASE IV and Clipper).

In the client/server configuration—where index format is determined on the server-side—there are two utilities provided, one for IPX/SPX communications and the other for TCP/IP communications. These utility can be used for accessing whichever file format the server is presently supporting. When launching reports, keep in mind that the report’s data files must be accessible by the database server.

All of these programs are DOS executables that can be used to quickly view or print reports from a DOS command line or batch file.

The source code for these various configurations of the launch utility is supplied in LAUNCH_D.C, and is provided as an added example of using the report functions, and as a way to use LAUNCH_D under other operating systems, or with different configuration switches.

The non-Windows launch utility cannot access Windows-specific styles, so reports outputted with CodeReporter will be different than those outputted with this utility.

Sizes of group headers and footers -- as well as the title, summary, page header and page footer -- are rounded to the nearest 1/6th of an inch (12 points), since a line on most printers is 1/6th of an inch high.

File Names

The pre-compiled versions of the launch utility (installed in the .\LAUNCH directory) are named as follows:

Name	Compatibility
LNCH_FOX*	FoxPro 2.0 (or higher) (.CDX)
LNCH_MDX*	dBASE IV (.MDX only)
LNCH_CLI*	Clipper (.NTX)
LNCH_SPX†	Server's File Format - IPX/SPX 16-Bit
LNCH_SK†	Server's File Format - TCP/IP 32-Bit

* stand-alone
† client/server

As a convention, this manual lists the launch utility name as LAUNCH_D

Usage: LAUNCH_D {name} [-q{*expr}] [-s{expr}] [-x{nn}] [-y{nn}]* [-p[*dest]* | -*f[dataName] [-t]*

Options:

-q{*expr}* Change the default query expression. Query expressions must be entered in double quotes ("). For example:

```
LAUNCH_D SAMPLE.REP -q "DBF->FIELDNAME > 'LINCOLN'"
```

String literals, such as 'Lincoln', above, must be entered in single quotes (').

-s{*expr}* Change the default sort expression. Sort expressions must be entered in double quotes ("). For example:

```
LAUNCH_D SAMPLE.REP -s "DBF->FIELDNAME"
```

String literals, such as 'Lincoln', above, must be entered in single quotes (').

-x{*nn}* Change the default horizontal size of the page (in characters). The default number of characters per line is 80. For example:

```
LAUNCH_D SAMPLE.REP -y70
```

-y{*nn}* Change the default vertical size of the page (in lines). The default number of lines is 25. The most common printer page size is 66. For example:

```
LAUNCH_D SAMPLE.REP -y66
```

-p[*dest]*	Change the default print destination. Reports by default go to the monitor. Specifying the -p option alone sends output to 'standard print'. If a character string is specified, output is directed to a file with the provided name. For example:

```
LAUNCH_D SAMPLE.REP -p              <== Output goes to 'standard print'
LAUNCH_D SAMPLE.REP -pSAMPLE.OUT <== Output goes to file
LAUNCH_D SAMPLE.REP -pLPT2       <== Output goes to LPT2 port
```

This option may not be used with the -f option.

-f[*dataName]*	Specify that report output should be sent to a data file. This option only applies if the report has a data file template saved within it. If dataName is not specified, the report is outputted to the data file name saved in the report. If dataName is specified, a data file is created with its name and the report output is stored within it.
-*t*	Use the non-Windows printer codes stored in the report file's styles. By default, LAUNCH_D does not send the printer codes.