Fitrix

Visual Development Tool (VDT)
Report Tools
Course Workbook
4.12

Restricted Rights Legend

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS252 227-7013 Fourth Generation Software Solutions, 2814 Spring Rd , Suite 300, Atlanta, GA 30039

Copyright

Software License Notice

Your license agreement with Fourth Generation Software Solutions, which is included with the product, specifies the permitted and prohibited uses of the product Any unauthorized duplication or use of Fitrix, in whole or in part, in print, or in any other storage and retrieval system is forbidden

Licenses and Trademarks

Fitrix is a registered trademark of Fourth Generation Software Solutions
Informix is a registered trademark of Informix Software, Inc
UNIX is a registered trademark of AT&T

FITRIX MANUALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, FURTHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE FITRIX MANUALS IS WITH YOU SHOULD THE FITRIX MANUALS PROVE DEFECTIVE, YOU (AND NOT FOURTH GENERATION SOFTWARE OR ANY AUTHORIZED REPRESENTATIVE OF FOURTH GENERATION SOFTWARE) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR, OR CORRECTION IN NO EVENT WILL FOURTH GENERATION BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF OR INABILITY TO USE SUCH FITRIX MANUALS, EVEN IF FOURTH GENERATION OR AN AUTHORIZED REPRESENTATIVE OF FOURTH GENERATION HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY IN ADDITION, FOURTH GENERATION SHALL NOT BE LIABLE FOR ANY CLAIM ARISING OUT OF THE USE OF OR INABILITY TO USE SUCH FOURTH GENERATION SOFTWARE OR MANUALS BASED UPON STRICT LIABILITY OR FOURTH GENERATION'S NEGLIGENCE SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS AND YOU MAY ALSO HAVE OTHER RIGHTS, WHICH VARY FROM STATE TO STATE

Fourth Generation Software Solutions
2814 Spring Road, Suite 300
Atlanta, GA 30339

Corporate: (770) 432-7623
Fax: (770) 432-3448
E-mail: info@fitrix com

Copyright

Welcome to the Fitrix , Visual Development Tool (VDT) Report Tools Course Workbook. This manual is designed for use in the Fitrix VDT Training class. We hope that you find all of this information clear and useful.

The screen images for the Report Writer product are all 'Character Screens'. The Fitrix 'Report Writer' is a character based end user report writer that can be used either for end users to create simple reports, or as a front-end for the Fitrix 'Report Code Generator'. The Fitrix Report Writer is not available in Windows Graphical Mode, but can be run from a Windows desktop in character mode.

The Fitrix Report Code Generator creates report and posting programs in 4GL. Any program created by the Report Code Generator offers the option of being viewed in a graphic based Windows screen.

If you have any questions about how to view your products in graphical mode, please consult your Installation Instructions or contact the Fitrix help desk at 1(800)374-6157. You can also contact us by email: support@fitrix.com. Please be prepared to offer your name, your company, telephone number, the product you are using, your Fitrix serial number and your exact question. We hope you enjoy using our products and look forward to serving you in the future.

Table of Contents

Section 1: Introduction to FourGen *Report*

Section 2: Creating Reports

Section 3: Editing Reports

Section 4: Managing the Data Dictionary

Section 5: *Report* Code Generator

Section 6: Using the Featurizer

Section 7: Creating Posting Programs

Section 8: New Features

Preface

The Report Training Course Workbook is an extensive aid for training on the products that make up the *Report*.

General Workbook Information

The *Report* Training Course Workbook is an aide for training on the products that make up *Report*. This workbook serves as a supplement to classroom training offered by . This workbook is not designed to be used by itself as a self study tutorial.

Coverage

This workbook covers the following *CASE* Tools products:

- *Report* (*Report* Writer, *Report* Code Generator, and Featurizer)
- 132 Column Pager

This workbook does *not* cover the following *CASE* Tools products:

- *Screen* (Form Painter, User Control Libraries, *Screen* Code Generator)
- *Menus*

Audience

This workbook is divided into three parts; each part addresses a separate audience:

- Sections 2 through 4 address end-users with no technical knowledge, who use *Report* Writer to create reports.
- Sections 2 through 5 address system implementors with some technical knowledge, who set up and support end-users.
- Sections 1 through 8 address programmers with extensive technical knowledge, who modify report code for end-users.

1

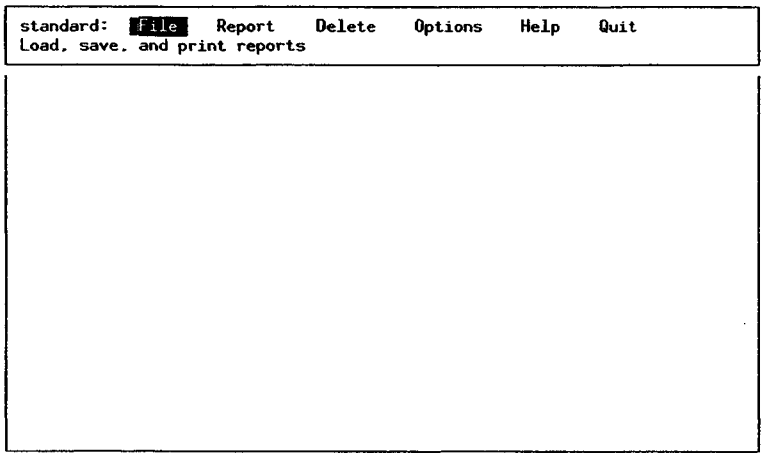
Introduction to Report

Major topics in this chapter include the following:

- Starting the *Report* Writer
- Using *Report* Writer's menus
- Creating a simple report

Starting the *Report* Writer

Start *Report* Writer by typing **`fg.writer`** at the UNIX prompt. If DBNAME is set in your .profile, there is no need to specify the name of the database prior to entering the program. If you need to, this is done by passing a "-d" flag to fg.writer, e.g., `fg.writer -d stores{}`.



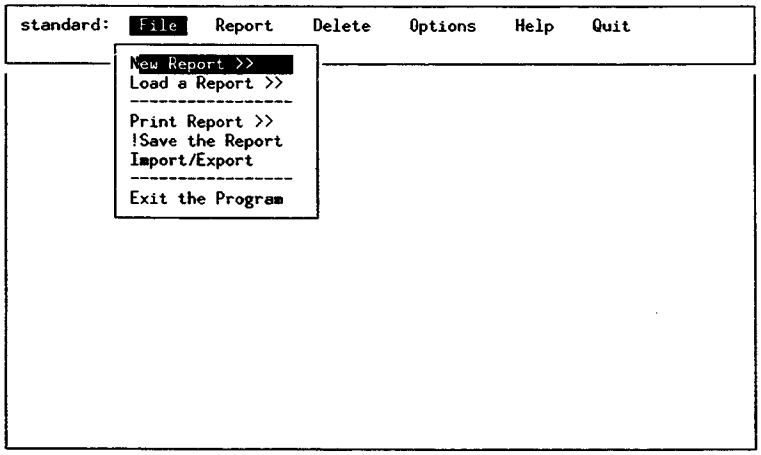
Report Writer's Menus

Report Writer ring menu is the user interface for reporting information stored in the database without entering structured query language code. Notice that the database name, in this case "report," is identified to the left of the ring menu options.

Each option on the main ring menu identifies a pull-down menu with additional options for activities in *Report* Writer. To move between the menu options, type the first letter of the menu option or press the arrow keys and press [ENTER] to display the selected option's pull-down menu.

File Pull-Down Menu

The File menu lets you create and maintain report files.



To select an option on the File pull-down menu, press the first letter of the option or use the arrow key to move to the option and press [ENTER]. The pull-down menus contain two characters which either proceed or follow the menu option. When an option is proceeded

with a !, the menu option is not available at the current time. When an option is followed by >>, selecting this option causes additional options to appear.

The File pull-down menu options include the following:

New Report leads you through the Report Definition and creates a new report file.

Load a Report provides a list of existing report files and allows the user to load an existing report.

Print a Report lists existing report files and prints the selected report.

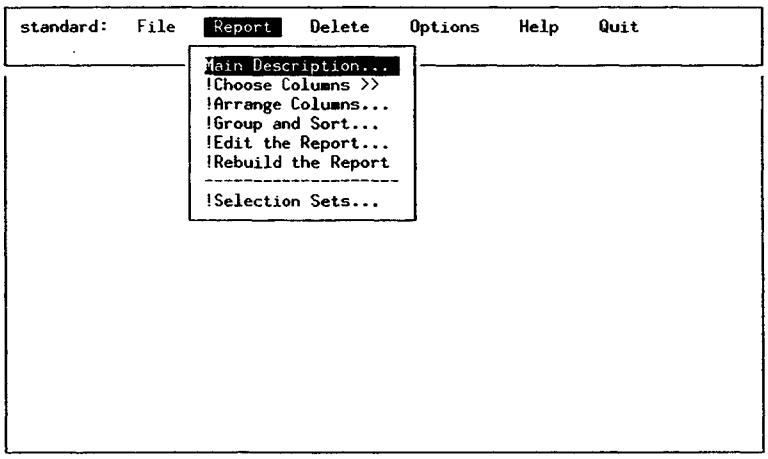
Save the Report saves a newly created or edited report file.

Import/Export allows the user to store report definitions in a text format. The format is saved in a file, which can be imported into another database.

Exit the Program confirms that the user is ready to quit and exits to the UNIX prompt.

Report Pull-Down Menu

The Report menu option lets you build a new report or modify a current report.



You must be working with a current report or have begun a new report description before these options are available.

The Report menu options include the following:

Main Description brings up the Report Description Form, where you define the report parameters. Parameters include database information, report format, who created the report, and when it was created.

Choose Columns lists the available columns for your report based upon the database information you enter into the Report Description form.

Arrange Columns lets you define the order that the columns will appear on the report and mark which columns are totaled.

Group and Sort allows you to specify a column or sequence of columns by which to sort the report.

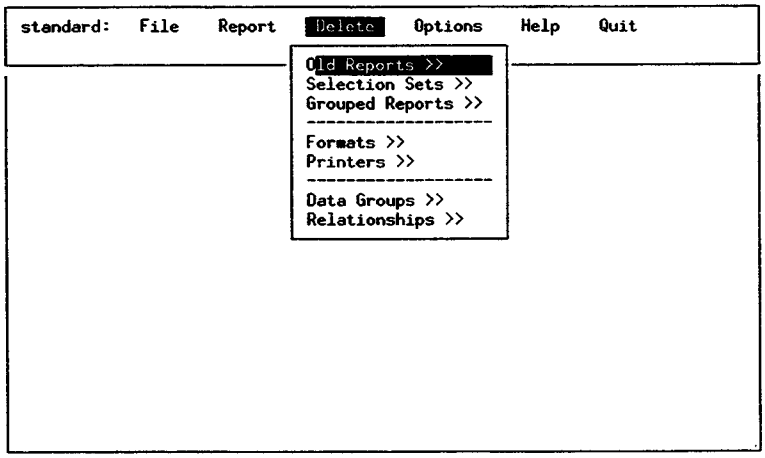
Edit the Report allows you to make modifications to the report template that is created by the *Report* Writer.

Rebuild the Report removes any modifications performed during the current work session by reverting to the most recently saved copy of the report.

Selection Sets allows you to define selection criteria to define the data displayed in the report.

Delete Pull-Down Menu

The Delete menu lets you remove old report files and other data files used by the *Report* Writer.



The Delete pull-down menu options include the following:

Old Reports lets you delete a saved report.

Selection Sets lets you delete selection sets.

Grouped Reports lets you delete selected groups of reports defined under the Options menu.

Formats lets you delete selected formats defined under Options.

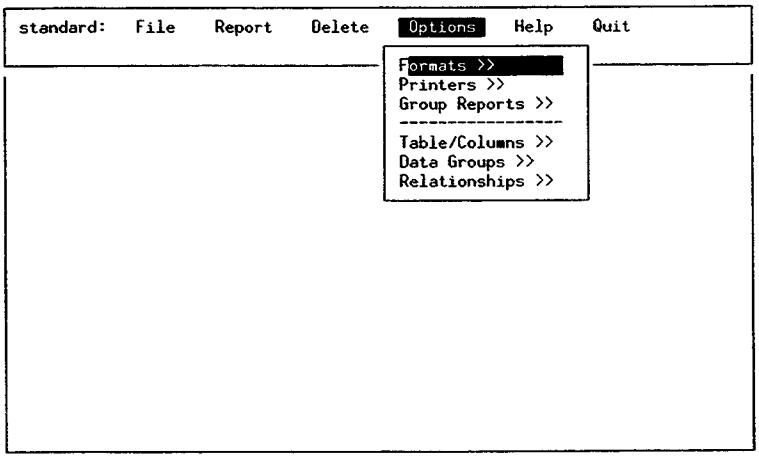
Printers lets you delete printer definitions defined under Options.

Data Groups lets you delete selected data groups defined under Options.

Relationships lets you delete selected table relationships defined under Options.

Options Pull-Down Menu

The Options menu lets you define and maintain report and database files.



The first three options let you set up reference files.

Formats allows you to define multiple page templates for the report's output. Formats define the column width, length, and margins.

Printers is the form on which you set up the printers to be used to print reports.

Group Reports allows you to organize reports into like groups.

The last three options let you perform data administration tasks.

Table/Columns allows you to assign descriptive names to the tables and columns in your database.

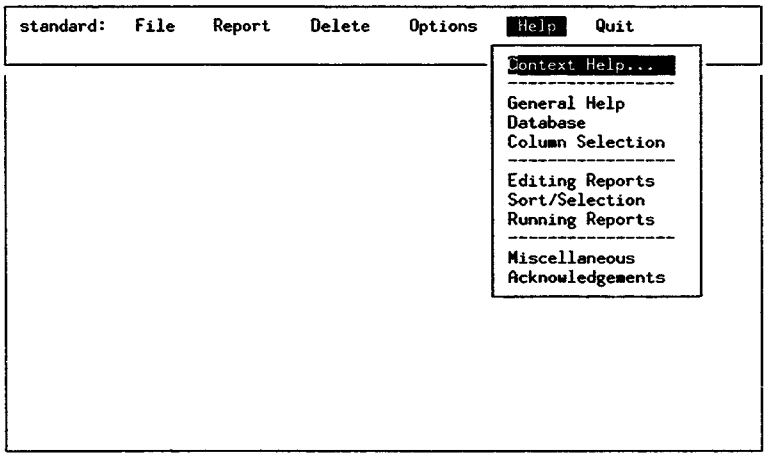
Data Groups allows you to organize tables into logical groups.

Relationships allows you to define relationships between tables in the database. These relationships, or "joins," should be defined by the data dictionary manager.

Help Pull-Down Menu

The Help menu lets you view detailed help information.

This includes context-sensitive help and general discussions on database administration, creating reports, and editing reports.



Quit Pull-Down Menu

The **Quit** menu lets you quit *Report* Writer.

After selecting the Quit option, you are asked to confirm your request. Select YES if you want to quit, NO or CANCEL if you want to continue using *Report* Writer.

Creating a Simple Report

You can create several types of reports using the *Report* Writer.

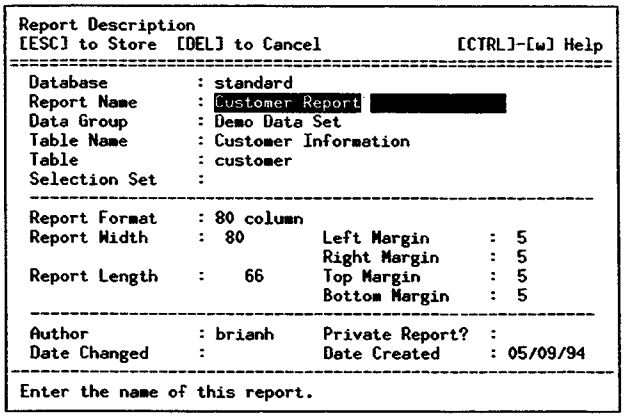
When creating a report, follow these basic steps:

1. Identify the data to be used in the report.
2. Define any totaling, or data manipulation, that results in a display element.
3. Format how the data appears on the report.
4. Print or save the report.

Report Description Window

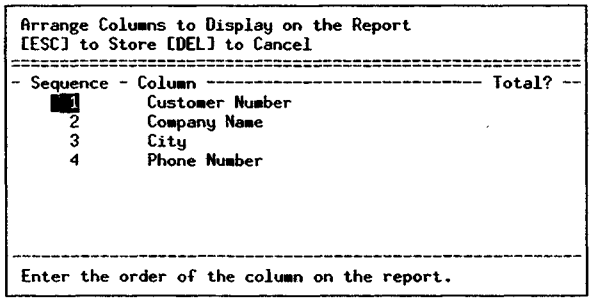
The *Report* Writer automatically leads you through the steps of creating a new report. The first steps to creating a new report include creating the report definition:

1. Select the Data Group
2. Choose a Table
3. Choose a Report Format
4. Name the Report



R*eport* Writer then leads you through the steps to creating a specific report.

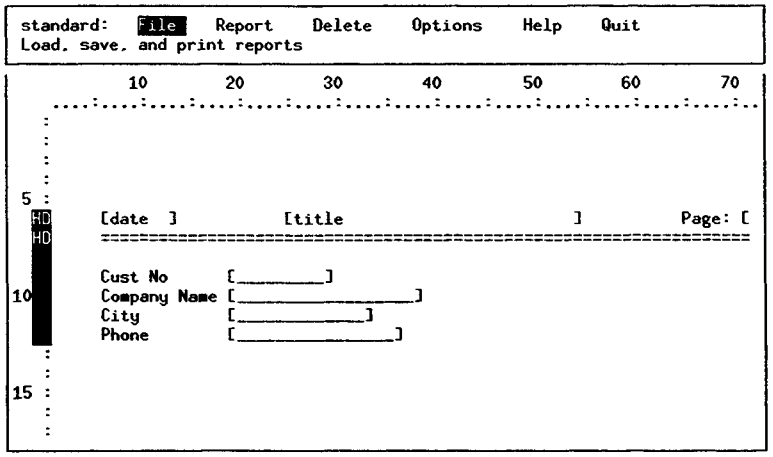
5. Select the columns in the report
6. Arrange the columns in the report.



Once the base report description information is defined, *Report* Writer automatically creates a report template.

Automatic Report Template

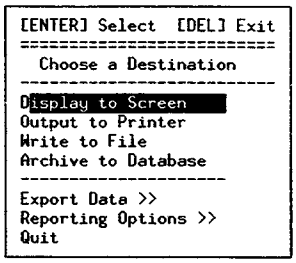
The report template appears on the screen in the *Report* Editor.



This template displays a preview of your report layout and how your report will appear when it is printed.

Print the Report

To print the report, choose a destination to which to send the report. You have the following options:



Display to Screen displays the report to the screen. You can move through the report using the options on the Pager ring menu.

Output to Printer prints a hardcopy of your report to the selected printer.

Write to File saves the report to a file.

Archive to Database stores the text of your report in tables in the database.

You can also select one of the following:

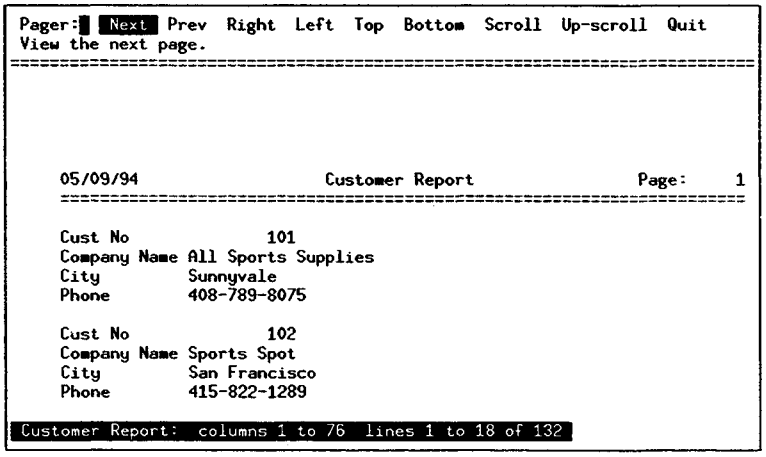
Export Data prints the report to an ASCII file.

Reporting Options allows you to define several options to control the detail of the report. These options include Summary Only, Test Length, Form Alignment, Data Selection, and Store Raw Data.

Quit returns you to the File pull-down menu.

Displaying a Report to the Screen

If you choose to display the report to the screen, the report appears in the *Report* Pager.



The Pager ring menu options help you to view different sections of the report. Move the cursor to the self-explanatory ring menu selections using the arrow keys or by keying the first letter of the selection, which is capitalized. Press [ENTER] to execute the command.

Section Summary

- From the *Report* Writer ring menu, you perform most reporting activities using the File and Report menu options.
- You manage your data dictionary management and perform delete activities from the Delete and Options selections on the ring menu.
- To create a simple report, identify the data to be used in the report, define manipulations to the data, format how the data appears on the report, and print or save the report.
- *Report* Writer gives you the ability to set up and maintain a data dictionary using the options for data administration under the Options menu.

Lab Exercise 1

Objective: Create a simple report and display it to the screen.

Start *Report* Writer

1. **Execute *Report* Writer using the `stores{}` database by typing:**

   ```
   fg.writer -d stores{}
   ```

   Substitute the name of your database for `stores{}` or set `$DBNAME` to the name of your database and leave off the `-d` flag.

Create a Simple Report

The first step to creating a new report is to define the report description. *Report* Writer will lead you through this process.

1. **From the File pull-down menu, select "New Report."**
2. **Choose "Demo Data Set" as the Data Group.**
3. **Choose the "Customer Information" table.**

   If the Report Format box appears, press [DEL].

4. **Name the report "Customer Phone Numbers."**
5. **Press [ESC] to store the Report Description.**
6. **Use the arrow keys to move to the following columns and press [ENTER] to choose the columns to appear on the report.**

   Customer Number

   Company Name

   Phone Number

Note	When you choose a column, an asterisk will appear next to the column name indicating this column is selected. To remove a column, press [ENTER] again to toggle off the column selection.

7. **Press [ESC] to store the columns in their default order.**
8. **Accept the sequence shown by pressing the [ESC] key.**

   Notice how the report template is displayed on the screen behind the menus.

9. **Press [DEL] to avoid choosing a Selection Set.**
10. **Press [DEL] again to clear any open pull-down menus and view the entire report template.**

    At this time a preview of the report template is visible on the screen. As you are working with more detailed reports, you can view the effects of any changes made in your report definitions as you are making them.

Display the Report to the Screen

1. **From the File pull-down menu, select "Print Report."**
2. **Press [ENTER] to save the report, with the default name: "Customer Phone Numbers."**
3. **If prompted for a Selection Set, press [DEL], as you have yet to define any selection criteria.**
4. **Select "Display to Screen" to display the "Customer Phone Numbers" report to the screen.**
5. **Use the Pager ring menu options to move through the report.**
6. **Select Quit to return to the Report Destination window.**
7. **Press [DEL] or Quit to return to the Main *Report* Writer menu and exit the program.**

2

Creating Reports

Major topics of this chapter include the following:

- Manipulating Report Data
- Printing Reports
- Grouping Data in a Report
- Creating Selection Sets

Creating a New Report

The *Report* Writer leads you through the basic steps of creating a report. As you begin creating a report, you define the report description, choose the columns to appear on the report, and specify the order in which they should be arranged.

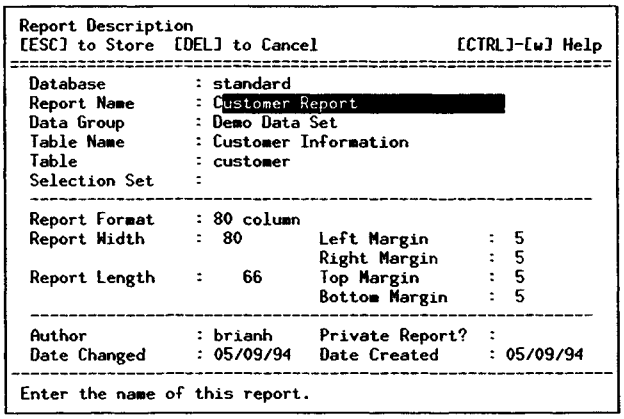
Defining the Report Description

To define a report description:

1. Choose a data group. A data group is a predefined group of related tables in your database. Defining data groups allows you to organize related tables in your database from which you can create your report. Data groups are defined by the database administrator using the Options selection from the main ring menu.

2. Choose a Table. The list of tables displayed are familiar names describing all the tables in the selected data group.

3. Complete the Report Description. Once you have selected the data group and main table, you are led to the Report Description window. This form is divided into three sections.



- The top section contains report file information. You assign a name to the report here. This name prints at the top of your reports.
- The middle section contains the report format information. Report formats are predefined under Options. A format is a predefined page size for your report. The type of information set in the report format is the report column width, report length in number of lines, and margin spacing.
- The bottom section identifies who created the report, when it was created, and when it was last changed. You also have the ability to define the report as a private report available only to the author.

Choosing the Columns to Appear on the Report

You are automatically led to the next logical step in creating a report: selecting which columns the report will use. The columns listed are the columns found in the main table defined in the report description. You are not limited to reporting from these columns. You can also select columns from tables that are defined as being related to or joined to the main table. The columns are listed by their familiar, or descriptive, names.

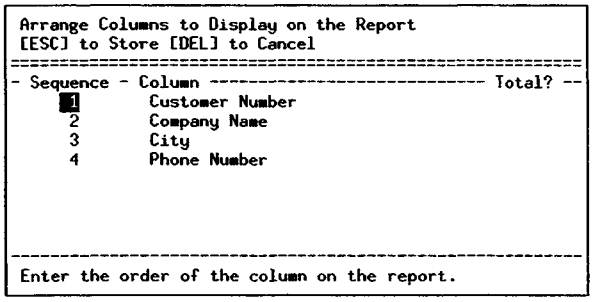
From this screen, there are two ways to select columns to appear on the report.

1. Move to the column name using your arrow keys and press [ENTER] to select the column. Note that an asterisk (*) appears in front of the column name indicating that this column has been selected.
2. Press `[CTRL]-[z]` and you will be prompted to select a data file. A list of columns is displayed for you to select from. The data files are other tables that have a relationship to the main table defined in the report description. In addition to selecting columns from the main table, you have the option of selecting columns from related tables. Table relationships are defined by the database administrator from the Options pull-down menu.

Arranging the Columns to Display on the Report

The next step is to arrange the columns in the order you want them to appear on your report. In addition to arranging the order of the columns for the report, you can add a column and total numeric column values.

The columns appear in the order they were selected. To change their order on the reports, alter the number in the Sequence field. For example, entering "1" (for "Item Description") in the Sequence field indicates that it will be the first column of the report. It will appear either as the column on the far left of the report or as the first column if the report data is stacked.

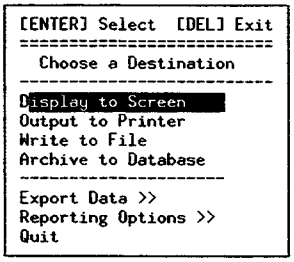


You can add another column to the report from this entry screen by typing a column name under the Column field. You also have the option of totaling a column by entering "Y" in the Total field.

Once you have stored your column arrangements, the *Report* Writer displays the template in the background of the pull-down menus, allowing you to preview the report layout as you create it.

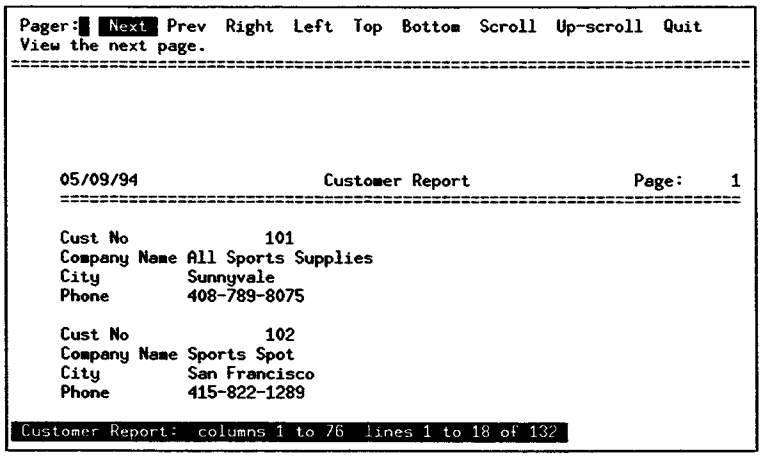
Printing Reports

When you select the Print Report option, you will be prompted for a name. You will be asked to specify a selection set. Once you have named the report and specified the selection set, you can choose a destination for the report. You have the following options:



Display to the Screen

This option sends the output of your report to the screen. Use your arrow keys to highlight an option and press [ENTER] to select it. You can also type the first letter in the option's name, such as "p" for Prev (Previous) to highlight it.



Report Pager Ring Menu Options

- **Next** allows you to page down and view the next page.
- **Prev** allows you to move back up to the previous page.
- **Right** allows you to move the screen display to the right, should you have more columns across the report than are displayed to the screen.
- **Left** allows you to move the screen display of the report back to the left.
- **Top** allows you to move to the top of the report.
- **Bottom** allows you to move to the end of the last page of the report.
- **Scroll** allows you to move down through the report one line at a time.
- **Up-scroll** allows you to move up towards the beginning of the report one line at a time.
- **Quit** exits from the report display and returns to the Select Destination menu.

Output to Printer

Prompts you to choose from the printers set up to print reports, and routes your report to the printer.

Write to File

Prompts you to enter a file name for the report's destination, and saves your report to the file.

Archive to Database

Stores the text of your report in two tables in the *Report* Writer database.

Export Data

Outputs the detail of your report to an ASCII file, which can be imported by other software programs.

Reporting Options

Allow you to define several options to control the report output.

- **Detail or Summary** toggles between printing detail only or summary only. Detail prints out every row of the report. Summary prints only group headings and their corresponding subtotals.
- **Test Length** allows you to define how many rows to print. Use this to preview a sample before printing the entire report.
- **Form Alignment** allows you to send positioning marks to the printer so you can see if the forms are aligned properly.
- **Data Selection** allows you to select multiple selections sets for a report.
- **Create Data Table** stores the raw data of your report to a table, allowing subsequent reports from this table to run faster.

Manipulating Report Data

Report data can be manipulated by the following methods: changing the Arrange column sequence, grouping data and calculating subtotals, and sorting the data.

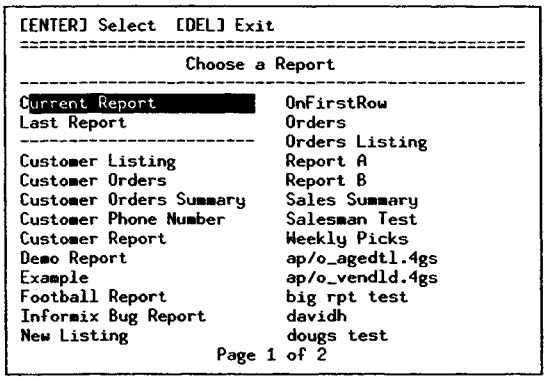
Once you have created a report you may want to make editing changes to that report or to another saved report.

Loading a Saved Report

To load an existing report, from the File pull-down menu select Load a Report. A screen is displayed prompting you to Choose a Report. The options are divided into two parts:

1. The first contains Current Report or Last Report. Selecting Current Report will return you to the report on which you are working. Selecting Last Report loads the last report you were working with.

2. The second portion of this screen displays a list of all reports created and saved for this database:



Editing the Report Data

Editing options can be accessed from the Report pull-down menu. The first three options on the Report pull-down menu access the input programs that the *Report* Writer automatically leads you through when creating a new report. Changes to column information can be entered by selecting one of the first three options.

- **Main Description** allows you to modify the report file definition.
- **Choose Columns** allows you to change the selection of columns for the report. If changes are made to the columns selected for the report, the program automatically brings up the Arrange Column screen.
- **Arrange Columns** allows you to change the column display on the report or define a column for totalling.

The four selections following allow you to create more complex reports.

- **Group and Sort** allows you to define criteria to group data, and to sort the display order of the data.
- **Edit the Report** allows you to access the report template viewed on the screen and make changes to the report through the template.
- **Rebuild the Report** recalculates the math functions and redraws the report template display.
- **Selection Sets** allows you to create a new selection set or choose from defined selection sets. A selection set specifies the selection criteria for data included on the report.

Grouping and Sorting

More functionality can be added to your report by defining a column to group data by, and to sort the data in ascending or descending order. When Group and Sort is selected, the list of columns chosen for this report is displayed. When a column is chosen as a group, that column will appear as a subheading for the group. If any columns in the report are marked for totaling, a subtotal will be calculated for each group defined.

Once a column is selected, the Sort the Report by these Columns screen is displayed.

```
Sort the Report by these Columns
[ESC] to Store [DEL] to Cancel
================================================================================
- Seq - Sort? - Group? - Page Bk.- Column ------------------------
   1            Y         N         Customer Number
   2            Y         N         Order Number
   3            Y         N         Item Number



--------------------------------------------------------------------------------
Enter order of the column for sorting.
```

- **Seq** is an abbreviation for sequence. If you have data grouped by more than one column or criteria, define the sequence by which to sort the columns. In the above example, data will first be grouped by the Customer Number then by Order Number.
- **Sort?** allows you to sort a column in "A" (ascending) or "D" (descending) order.
- **Group?** allows you to mark columns that you want subtotaled. A "Y" indicates the column will be subtotaled.
- **Page Bk.** is an abbreviation for page break. You can define a page break before or after a grouping, or select "N" for no page break at all. The above example has an "A" defined for Customer Number. This will print a new page for each customer number group.
- **Column** displays the name of the column which is to be grouped. By using [CTRL]-[z], you Zoom to additional columns that can be added from this input screen.

The following report template has two groups defined. The first group appears with the subheading Customer Number. The second group, indented under the Customer Number subheading, is Order No. Each grouping has a corresponding subtotal calculated for the Extension column, which is marked for totaling.

```
standard:  File   Report   Delete   Options   Help   Quit
Build and modify a report

          10        20        30        40        50        60        70
    ....:....:....:....:....:....:....:....:....:....:....:....:....:....:..
   :
   :
   :
   :
 5 :
   HD     [date  ]           [title                          ]          Page: [
   HD     ================================================================================
   B1
   B1     Cust No: [_________]
10 B2
   B2       Order No: [_________]
   B2         Company Name          City             Extension
   B2         -------------------- ---------------- -----------
              [__________________] [______________] [_________]
15 A2      Subtotals for [_________]                 -----------
   A2                                               [_________]
   A1     Subtotals for [_________]                 -----------
```

Below is the report displayed to the screen:

```
Pager:  Next  Prev  Right  Left  Top  Bottom  Scroll  Up-scroll  Quit
View the next page.
================================================================================



    05/09/94                    Customer Report                    Page:     1
    ============================================================================

    Cust No: 101

      Order No: 1002
        Company Name          City             Extension
        -------------------- ---------------- -----------
        All Sports Supplies   Sunnyvale          $960.00
        All Sports Supplies   Sunnyvale          $240.00
      Subtotals for 1002                       -----------
                                                 1200.00
    Subtotals for 101                          -----------
Customer Report:  columns 1 to 76  lines 1 to 18 of 264
```

Selection Sets

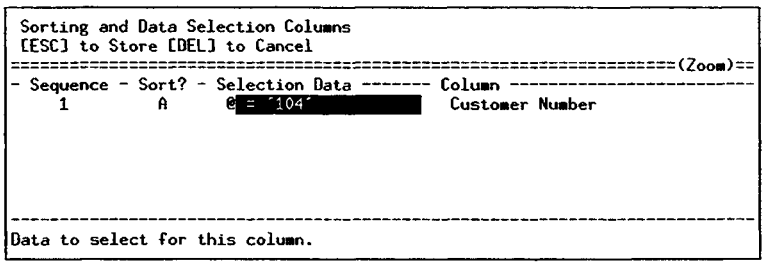
A selection set is a group of columns and their selection criteria that determine what data is displayed on the report. Selection sets are not tied directly to any given report. A report and a selection set are two independent elements. All reports that share the same main table (as defined in the report definition) can use any selection set that uses the same table as the main table.

Defining Selection Sets

From the Report pull-down menu, Selection Sets can be chosen to define a new selection set or modify an existing set. If no previous selection sets have been defined, the only option on the Choose Selection Set menu is New Selection Set.

The first step to defining a selection set is to choose the column by which to select or sort the data.

Once a column has been chosen, the Sorting and Data Selection Columns window appears.



Note that this screen looks similar to the Arrange Columns screen and the Group and Sort screen, but there is a new column, Selection Data.

- **Sequence** lets you prioritize the columns for sorting.
- **Sort?** allows you to sort in "A" (ascending) or "D" (descending) order.

- **Selection Data** lets you define your selection criteria.
- **Column** displays, adds, or updates columns for selecting and sorting.

Selection Data

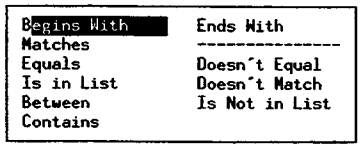
The Selection Data field allows you to build simple or complex selection clauses to limit the data displayed on your report. There are two ways to enter a selection clause.

1. If you are familiar with SQL "select" statements, you may manually enter your SQL selection clause by typing it directly into the Selection Data field.

   The Selection Data field can store an SQL statement up to 100 characters long. If your statement requires more than 100 characters, then you can continue on the next row of the Data Selection field. The following key strokes can be used for editing your select statement:

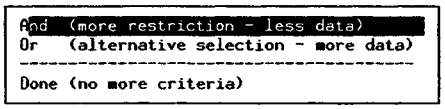
   - [CTRL] - [a] inserts a space
   - [CTRL] - [x] deletes a character
   - [CTRL] - [d] delete to end of line
   - [CTRL] - [u] undo an edit

2. If you are not familiar with SQL selection statements, there is a Zoom on the Data Selection field which helps you build your selection statement.



A list of operators is displayed for you to choose from. Once you have selected an operator, you are prompted for more specific selection criteria. Comparison operators differ depending upon the type of column, that is, whether it is character, numeric, or date column.

Once you have entered the criteria, another window appears allowing you to define an "And" clause, which further restricts the selection. You can also choose an "Or" clause, which expands the criteria.



Section Summary

- When creating a new report, the *Report* Writer automates the steps to create a basic report. These steps include the following:
  - Defining the report file description.
  - Selecting the columns to appear on the report.
  - Arranging the columns to appear on the report.
  - Viewing the report template.
- Column templates can be modified and edited using the options under the Report pull-down menu.
- The report template, which appears in the background of the pull-down menus, allows you to view the changes made to the report definition while you are making them. You do not have to print or display the report to view your changes.
- Reports can be:
  - displayed to the screen,
  - sent to a defined printer,
  - written to a file,
  - archived to the database,
  - exported to an ASCII File.
- Report data can be grouped by a specified column or by multiple columns. Subtotals are printed for grouped data columns.
- Selection sets are defined to limit which data appears on the report. Selection sets are independent of individual reports. They are tied to the main table defined in the report description and can be used for any report with the same main table definition.
- Selection data can be entered in the following two ways:
  - Typing an SQL select statement in the Selection Data field,
  - Zooming to define selection criteria.

Lab Exercise 2a

Objective: To create a new report that will display all sales orders and their items. The total of all orders will appear at the bottom of the report.

Defining the Report Description

1. **Create a new report using the Demo Data Set.**
2. **Select the "Sales Orders" table.**

   If the Report Format box appears, press [DEL].
3. **Name the report "Orders."**

Choosing the Columns to Appear on the Report

1. **Choose the following columns to appear on the report and arrange them in the following order:**
   - Order Date
   - Quantity (From the Line Items table)
   - Line Extension (From the Line Items table)
2. **Total the amounts in Line Extension column.**
3. **Save and display the report to the screen.**

Adding Columns and Rearranging the Column Order

1. **Return to the Choose Columns window and add the following columns to the report. (These columns are in the Items in Inventory table.)**
   - Item Number
   - Item Description
   - Unit Price

2. **Arrange the columns in the following sequence:**
   - Order Date
   - Item Number
   - Item Description
   - Unit Price
   - Quantity
   - Line Extension
3. **Display the report to the screen.**

Lab Exercise 2b

Objective: To expand the Orders report by adding subtotal logic to the Order Number and Customer Number columns; and by calculating a total for all Customer Orders.

Grouping Report Data

1. **From the Report pull-down menu, select the Group and Sort option.**
2. **Group your report by the Order Number column and sort the orders in ascending order.**

   Notice the change in your report by viewing the report template behind the pull-down menu.

Sorting Data

1. **Add an additional group, Customer Number, with a grouping sequence of 1.**

   Orders on the report will be grouped within customer numbers, so customer numbers will be your first group.
2. **Save this report under the new name "Customer Orders."**
3. **Add a page break so a different page is printed for each customer.**
4. **Add a third column, Item Number, for sorting only.**

   It is not necessary to group orders by the Item Numbers as they are already grouped by order number.
5. **Save this report under a new name: "Customer Orders Summary."**
6. **Display the report to the screen and page down through each customer order.**

Lab Exercise 2c

Objectives: You will define two Selection Sets, first, to select only customer orders for footballs, then you will be entering an SQL filter statement directly in the Selection Data field.

Selection Criteria

1. **Load the "Orders" report.**
2. **From the Report pull-down menu choose "Selection Sets."**
3. **Add a new selection set.**
4. **Zoom to the Inventory Items table.**
5. **Select Item Description as the column for selection criteria.**
6. **For Item Description, define the Sequence as 1 and sort in ascending order.**
7. **Zoom on the Selection Data column and select Equals.**
8. **Enter "football" as the data criteria.**
9. **Save the selection set as "Footballs," but do not define this as the default for the Orders report.**
10. **Print the Orders report with Footballs as the selection set.**

Entering an SQL Filter Statement

1. **Load the "Orders" report and choose "Selection Sets" from the Report pull-down menu.**
2. **Choose a new selection set.**
3. **Choose the Payment Date column as the column by which to select data.**
4. **In the Selection Data column, type the following SQL select statement:** `@ is null`
5. **Save the Selection Set under the following name: "Unpaid Orders." Do *not* define this as the default Selection Set for the "Orders" report.**
6. **Print the "Orders" report so that it shows unpaid orders only.**
7. **Print the "Orders" report so that it shows all orders.**
8. **Print the "Customer Orders Summary" report of all unpaid orders.**
9. **Print the "Customer Orders Summary" so that it shows all orders.**

3

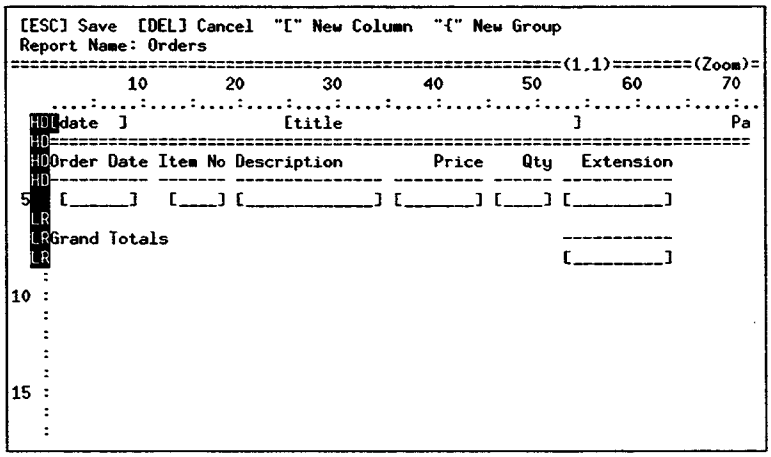
Editing Reports

Major topics in this chapter include the following:

- Using the *Report* Editor
- The Column Detail Zoom
- Modifying Report Blocks
- Using the *Report* Runner

Using the *Report* Editor

The Edit the Report option from the Report pull-down menu allows you to make changes directly to the report template generated by the automatic column formatter. Once you modify your report using the Edit the Report option, you lose the ability to make changes using the pull-down menus.



In the *Report* Editor, you have four options displayed at the top of the screen:

- [ESC] saves your report
- [DEL] returns you to the pull-down menu
- "[" allows you to add a column
- "{" allows you to add new groups of data.

You can also scroll up and down and to the left and right using the arrow keys. Rulers along the top and left edges of the report indicate the column and line numbers. Toward the upper right-hand corner on the double dotted line, the cursor location is displayed in parentheses. For example, as displayed in the example above, (1 , 1) means that the cursor is located at line 1 and column 1.

Editing the Report Template

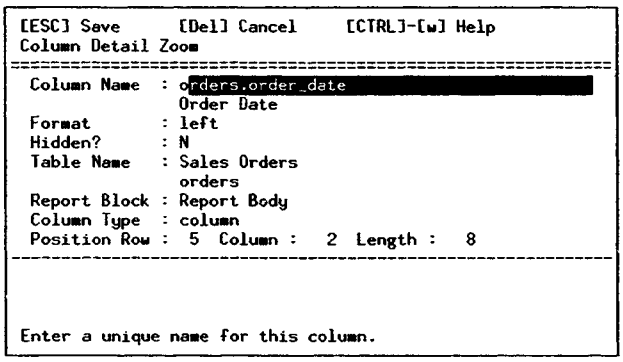
Once you are in the *Report* Editor, you can add, delete, or change columns, groups, and text by typing in the report template. The following Informix-defined editing keys are also available to use while editing your report.

Key(s)	Action
[CTRL]-[a]	insert space
[CTRL]-[x]	delete character
[CTRL]-[d]	deletes to end of line
[CTRL]-[p]	pastes a line
[CTRL]-[u]	undo an edit
[F1]	insert a blank line
[F2]	delete a line
[	define a new column group
{	define a new group
[ESC]	save
[DEL]	cancel

You can also cut and paste a line in the report by selecting [F2] to delete a line, which actually puts it to a temporary buffer. Then paste the line by pressing [CTRL]-[p].

Column Detail Zoom

The Column Detail Zoom can be accessed directly from the *Report* Editor by pressing [CTRL]-[z]. When your cursor is placed on a column in the report template, (Zoom) appears in the upper right-hand corner on the double dotted line, indicating that the Zoom option is available for accessing the Column Detail Zoom screen.



The first portion of the screen contains detailed information about the column:

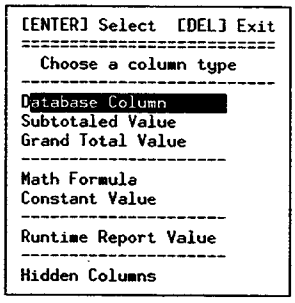
- **Column Name** displays the Informix table name and column name on the first line and the familiar name on the second line.
- **Format** allows you to define the column justification or numerical formatting.
- **Hidden?** allows you to mark "Y" or "N" to print or hide a column.
- **Table Name** displays first the *Report* Writer descriptive name, and below it the Informix table name.
- **Report Block** displays where this column is in the report block definition. This is not an entry field. It is governed by the location of your new column and where you pressed " [".
- **Column Type** displays the type of column. Column types are Database, Subtotaled, Grand Total, Math Formula, Constant Value, Runtime Report Value, and Hidden.

- **Position Row/Column/Length** gives the physical location and size of the column as it appears on the report.

The bottom portion of the screen displays the math formula if the column is a formula column. To edit an existing math formula, press [CTRL]-[z].

Adding a New Column

To add a new column press "[", which brings up the Choose a Column Type menu. You will be prompted to select a column type.



There are several different types of columns including the following:

- **Database Column** displays a list of columns in the main database. Zoom is available to list the columns in related tables. If a column is selected, you will be prompted with the Column Detail Zoom to make any changes to the column detail information.
- **Subtotaled Value** and **Grand Total Value** are columns that can be added to the report when you have added report groups through the *Report* Editor.
- **Math Formula** lets you add a field to the screen to display the value of a math formula.
- **Constant Value** allows you to define a constant value to be displayed in that column.

- **Runtime Report Value** allows you to place special values on the report that are specific to the time the report is printed. These special values include system-generated page number, time and date stamping, etc.
- **Hidden Columns** displays a list of all columns marked to not display on the report.

Once the column type is selected and the column detail is saved, the column is activated on the report.

Creating "Hidden" Columns

The *Report* Editor allows you to hide a column that you do not want to print on your report. This is commonly used in creating math formulas with column values you do not want displayed on the report.

To hide a column:

1. **Place the cursor on the column.**
2. **Press [CTRL]-[z] to modify the column detail.**
3. **Enter "Y" in the Hidden? field.**
4. **Press [ESC] to save.**

Once you have saved the column detail information, the column is removed from the *Report* Editor screen.

To list all hidden columns on your report:

1. **Press "[" to display the Choose a Column Type menu.**
2. **Press "H" to display a list of all hidden columns on your report.**

If you wish to reveal a column that has been hidden on your report or to modify the hidden column detail:

1. **List all hidden columns on your report.**
2. **Select the column from the window of the column you wish to reveal.**
3. **The Column Detail Zoom appears for you to change the Hidden? field to "N."**
4. **Press [ESC] to return to the *Report* Editor and view the column.**

Math Calculations

The *Report* Writer allows you to create math columns. You must be in the *Report* Editor to define math formulas.

To create math formulas:

1. **Press "[" to add a new column.**
2. **Select Math Formula from the menu.**
3. **Enter the math formula in the scrolling window.**

   Columns that are available to perform calculations within the report block will be listed in a window below the main formula window. The columns available to be used in the math formula are numeric columns in that Report Block only.

   Once the math formula is saved, the Column Detail Zoom screen appears for you to define the numeric format of the new math column.

To edit an existing math formula:

1. **Press [CTRL]-[z] to display the math column detail.**
2. **Press [ENTER] to move to and highlight the block at the bottom of the window.**
3. **Press [CTRL]-[z] to display the scrolling math formula definition window.**
4. **Edit the math formula.**

If you want to use a subtotal, grand total, or math formula amounts in your math formula, you must place the new math formula after (either to the right on the same row or on a row below) the calculated value. The *Report* Writer reads values from left to right. For a calculated value to be included in a formula, its value must first be calculated.

Formatting Data in the Column

The text justification or numbered formatting can be defined in the Column Detail Zoom screen in the Format field.

Justification choices include the following:

- **right** moves the text to the right edge of the column display.
- **center** centers the text within the column.
- **left** moves the text to the left edge of the column display.
- **pushleft** moves the data in the field to the left and inserts a leading space. This is helpful when making address labels. It leaves only one space between the city and the state even though the length of the city column varies.
- **flushleft** moves the data in the column to the left leaving no spaces between columns. This is useful when two columns are displayed as one item with no spaces between them.
- **flushright** moves the data in the column to the right leaving no spaces between columns. This is useful when two columns are displayed as one item with no spaces between them.

Numeric formatting choices include any Informix numeric format for formatting numeric data. Refer to the Informix Reference Manual for more information on numeric formats. Examples appear below.

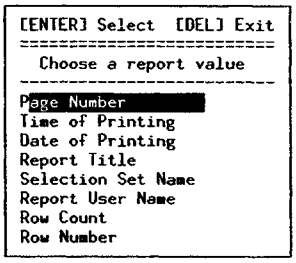
- $$$,$$$.$$
- ###,###.##
- <<,<<<

Runtime Report Values

Runtime report values are special column types that display system defined values at the time the report is printed. You can specify the location of these columns on your report when you add the column.

To add a runtime report value:

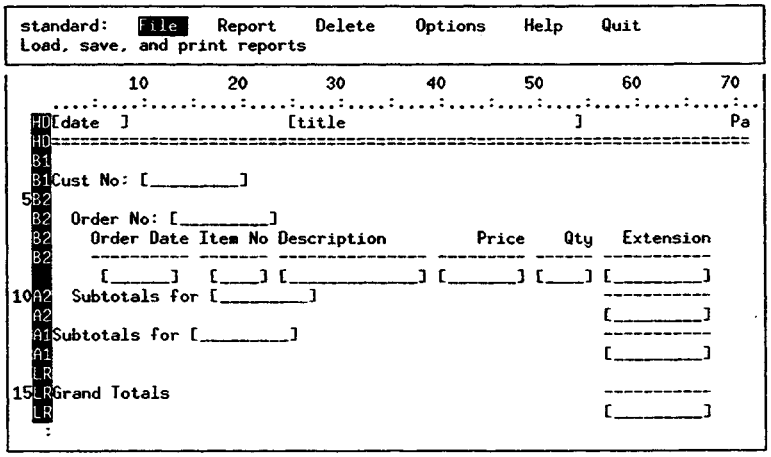
1. **Move the cursor to where you want to add the column.**
2. **Press "[" to display the Select Column Type window.**
3. **Select Runtime Report Values. The available columns or values you can add include the following:**



- **Page Number** prints the page number on the report. It only prints the number; if you want "Page 2" printed you must add "Page" before the special column in the *Report* Editor.
- **Time of Printing** prints the time the report was printed.
- **Date of Printing** prints the day the report was printed according to the Informix DBDATE definitions variable.
- **Report Title** displays the name of the report file on the report.
- **Selection Set Name** prints the name of the chosen selection set.
- **Report User Name** prints the login name of the report author.
- **Row Count** counts the total number of rows within the report. This does not count rows broken down by subgroups.
- **Row Number** assigns a number to the current row of the report.

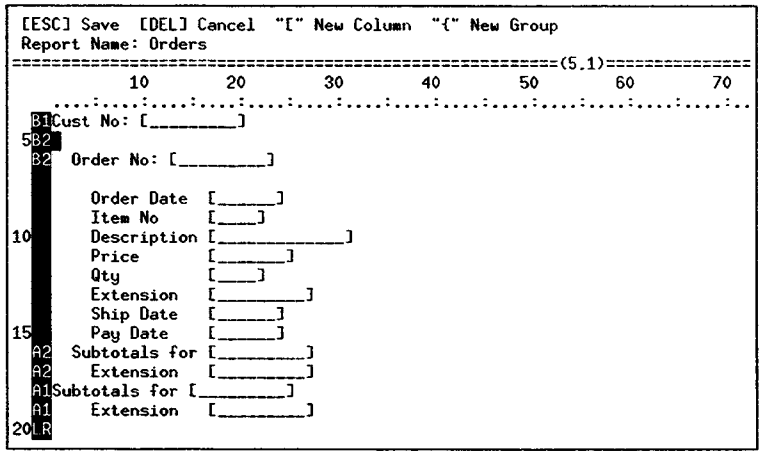
Report Blocks

The *Report* Writer places block tags on the left edge of the *Report* Editor to identify the roles that particular blocks of text play in the report. The block tags are generated automatically based on the report definitions. The blocks can be modified using the *Report* Editor. You can insert or delete rows and add new report blocks.



If the columns in the report extend beyond the width of the report format, the columns are automatically stacked rather than displayed across the report. The following report shows stacked columns and

provides a better illustration of the relation of the report block tags and the corresponding columns and column groupings than the screen above.



Type of Report Blocks

- **FR** is the First Row Block Tag. This report block will print once at the very beginning of the report.
- **HD** is the Header Block Tag. The header information is printed once at the top of every page of the report.
- **B1, B2...** are Before Block Tags. The Before block tags identify blocks of text which are printed on the report before the detailed data rows. Examples of before blocks of data are Grouped information. In the example above there is a B1 and a B2.
- **B1, or Before 1**, indicates the block of information will print before the group by Company Name information. **B2, or Before 2**, identifies a subgrouping of information based upon a group definition by the column Order No. Before groupings are numbered according to the sequence specified in the Group and Sort definition. These are group headings, which will print at the beginning of each group break.

- **Blank Tag** The row directly following the Before Tag which has no identifier identifies where the report detail data rows will print.
- **A2, A1...** are After Block Tags. The After block tag identifies information that is printed after the report detail data rows. Subtotal groupings are numbered according to the corresponding Before block tag. The sequencing is relative to the sequence defined in the Group and Sort definition.
- **LR** is the Last Row Block Tag. It is printed on the final row of the report. It contains column information such as Grand Totals.
- **TR is the Trailer Block Tag.** It is printed at the bottom of every page and can be defined to contain footer information.

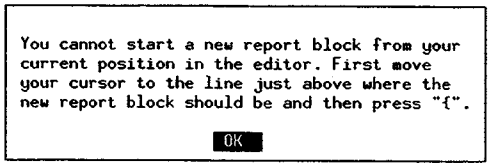
Note When defining a math formula, the columns that can be used in the formula are displayed in a window below the formula entry field. The columns that are available are numeric columns located in that Report Block.

Adding a New Report Block

The steps to adding a Report Block to your report are dependent upon the type of report block you are adding. In the *Report* Editor you can add the following types of report blocks:

- B# or Before Group
- A# or After Group
- LR or Last Group
- TR or Trailer

The type of block you can add is sensitive to the line the cursor is on in relation to the existing report blocks and predefined report groups. If you are not on a row to which a report block can be added, you will get the following message:



Adding a Before or After Group Report Block

Once you make changes to your report in the *Report* Editor, the automatic column formatter no longer updates the report template. Any additional data groupings defined in the Group and Sort will not appear on the report; adding new groups must be done through the *Report* Editor.

The steps for adding a new B or A group are as follows:

1. **Select Group and Sort from the Report ring menu option.**
2. **Enter and save the column grouping information.**

   When a new report group is defined there are two types of report blocks that can be added to the report, a Before Group or an After Group. The number associated with the B or A block tag will be dependent upon the grouping sequence defined for the column.

Since Before and After Blocks refer to a specific Group, the reference is Group specific. The following template and sequence definition displays one block:

```
Sort the Report by these Columns
[ESC] to Store [DEL] to Cancel
================================================================================
- Seq - Sort? - Group? - Page Bk.- Column ----------------------------
    1            Y         N        Customer Number
```

Enter order of the column for sorting.

The Customer Number field defines the only grouping, and can have items placed before (B1) or after (A1) the area where the data representing one customer number is presented.

```
standard:  File  Report  Delete  Options  Help  Quit
Build and modify a report

          10        20        30        40        50        60        70
    ....:....:....:....:....:....:....:....:....:....:....:....:....:....:..
 HD[date  ]                [title                                ]          Pa
 HD================================================================================
 B1
 B1Cust No: [_________]
 5B1  Order Date Item No Description          Price    Qty   Extension
 B1   ---------- ------- ---------------- ---------- ------ -----------
       [______]  [____] [______________] [________] [____] [_________]
 A1Subtotals for [_________]                                -----------
 A1                                                         [_________]
10LR
 LRGrand Totals                                            -----------
 LR                                                         [_________]
    :
    :
15  :
    :
    :
```

The following report template has more than one group defined:

```
Sort the Report by these Columns
[ESC] to Store [DEL] to Cancel
================================================================
- Seq - Sort? - Group? - Page Bk.- Column ----------------------
   1            Y         N        Customer Number
   2     A      Y         N        Order Number
   3            N         N        Item Number



----------------------------------------------------------------
Enter order of the column for sorting.
```

Even though there are three fields in the Sort consideration, only two are specified to have data grouped. This instructs the *Report* Writer to create blocks for each, B1, B2, A1, and A2:

```
standard:  File   Report   Delete   Options   Help   Quit
Build and modify a report

          10        20        30        40        50        60        70
    ....:....:....:....:....:....:....:....:....:....:....:....:....:..
 HD [date ]                  [title                              ]     Pa
 HD ================================================================
 B1
 B1 Cust No: [_________]
5B2
 B2  Order No: [_________]
 B2    Order Date Item No Description         Price    Qty   Extension
 B2    ---------- ------- --------------- ---------- ------ ----------
        [______]   [____] [_____________] [_______] [____] [_________]
10A2   Subtotals for [_________]                              ----------
 A2                                                          [_________]
 A1 Subtotals for [_________]                                ----------
 A1                                                          [_________]
 LR
15LR Grand Totals                                            ----------
 LR                                                          [_________]
  :
```

The Order Number Block (the second group) can have other information placed before it (between B1 and B2) and after it (between A2 and A1).

The Customer Number Block (first group) can have information before (between HD and B1) and after it (between A1 and LR).

In order to add information to a report template:

1. **Select Edit the Report from the Report ring menu option.**
2. **Place your cursor on the row above where you wish to insert the group.**
3. **Press "{". A window appears with the report blocks that are available to add to the screen.**
4. **Once the report block is added, you will have a blank row to add further text and column information.**

Adding Last Groups and Trailers

The steps for adding LR or TR groups are as follows:

1. **Place the cursor on the line above where you wish to add the new LR or TR group.**
2. **Press "{".**
3. **Confirm you wish to add the report block.**
4. **A new line is created for the new report block.**
5. **Add text or columns on that line.**

The *Report* Runner

The *Report* Runner allows you to print, from a UNIX prompt, reports created with the *Report* Writer. This shortcut eliminates the need to execute the *Report* Writer program for the sole purpose of printing, thus saving time. By using the *Report* Runner, the first request for input is at the **Choose a** (print) **Destination** window.

In order to run reports with the *Report* Runner, you must know the name of the report and the name of the database. If you are going to print the report with a selection set, you must know the name of that selection set.

To print a report using the *Report* Runner the syntax is:

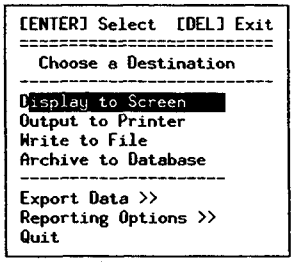
fg.runner -d *database* -r *report_name* -s *selection_set_name*

To print a report using the "stores" database called "Orders" with the selection set "Unpaid Orders" you would type:

```
fg.runner -d stores -r Orders -s "Unpaid Orders"
```

Note To specify a report or selection set that has more than one word, you must put quotes around the report or selection set's name.

Once you enter the `fg.runner` command, the Choose a Destination menu appears



From this window, the Reporting Options selection lets you further define your reporting options.

If you make an error typing the name of the database, report, or selection set, the *Report* Runner will display an error message notifying you which part of the statement it was not able to understand.

Section Summary

- The *Report* Editor allows you to make modifications directly to the report template displayed on the screen. Once you edit your report in the *Report* Editor, the *Report* Writer will not automatically format the report template.
- Column detail information is accessed by editing within the *Report* Editor, allowing you to change the column format, hide columns, and perform math calculations on column values.
- Report Blocks are tags located along the left edge of the *Report* Editor screen which identify the role of a group of lines within a report.
- New Report Groups can be added within the *Report* Editor only after they have been defined in the Group and Sort window.
- The *Report* Runner allows you to print a report any time, anywhere, from a UNIX prompt, if you know the names of the database, report, and optional Selection Set.

Lab Exercise 3a

Objective: To add columns to the report from the *Report* Editor.

Adding Columns in the *Report* Editor

1. **Load the Customer Orders Summary report.**
2. **From the Report pull-down menu, select Edit the Report.**
3. **Using the edit keys, add the following columns below the Customer Number column:**

   Company Name

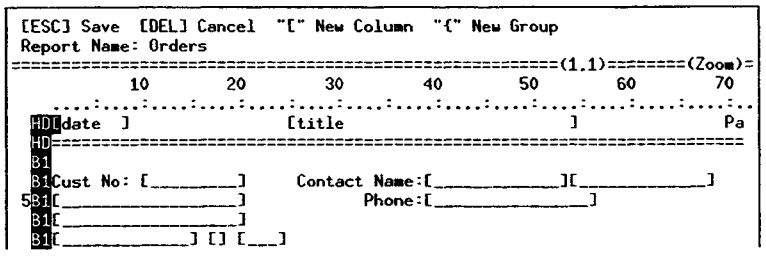
   Address

   City, State, Zip

   Contact Name (first and last)

   Phone Number

4. **Add them to report Block 1 and format the fields to "pushleft."**



5. **Display the report to the screen.**

Lab Exercise 3b

Objective: To use math columns to calculate the order sales tax and to create a new column called Order Total.

Math Formulas

1. **Load the Customer Orders Summary report and, from the Report pull-down menu, select Edit the Report.**
2. **Insert three rows to the bottom of Report Block A2.**
3. **On the line directly below the orders subtotal amount, insert a row of "----" to create a line across the report.**

   Refer to the report image at the bottom of the page.
4. **On the row below the line, add the text "Shipping Charge" and add the database column Shipping Charge to the report.**
5. **Next to the Shipping Charge column add the text "Sales Tax" and press "[" to add a new column to the report.**
6. **Select Math Formula from the column type window and enter a math formula to calculate 8.2% sales tax on the order subtotal.**

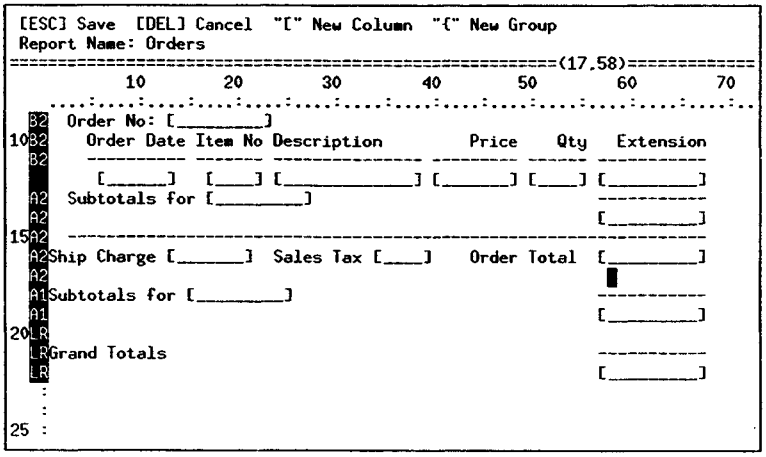
   The formula to make this calculation is (.082 * A1).

Note — On your report, the variable for the subtotal of items.total_price may be different than A1. If it is, use the variable displayed in the variables window that identifies SUBT_AF_2 _items.total_price.

7. **Enter a numeric format of $$$.$$ for this math column and reduce the field size to six characters using the [CTRL]-[x] key combination.**
8. **To the right of the sales tax column, add the text "Order Total" and press "[" to add a new column to the report.**

9. **Select Math Formula and enter a math formula to add the Orders Subtotal, Shipping Charge, and Sales Tax. Reduce the field size to nine positions.**

Your report should appear as illustrated below:



Lab Exercise 3c

Objective: To add a new report group to the Customer Orders Summary report without using the *Report* Writer to automatically format the report template.

Define Groups in the *Report* Editor

1. **Load the Customer Orders Summary report.**
2. **From the Report pull-down menu select Group and Sort.**

   Before you can add a new group in the *Report* Editor, you must first define it in the Group and Sort window from the Report pull-down menu.
3. **Change Item Number to Group as well as Sort.**
4. **Select Edit the Report.**
5. **Move to the last row of the Report Block B2 and press "{" to add a new group.**
6. **Add two columns in the new B3 report block to display the Stock # and Manufacturer's Code (located in the "Line Items" Data File).**
7. **Print the Customer Orders Summary report.**

Lab Exercise 3d

Objective: To print the Customer Orders Summary report from a UNIX prompt with the Unpaid Orders selection set.

Report Runner

1. **From a UNIX prompt, enter the command to print the Customer Orders Summary report using the Unpaid Orders selection set.**
2. **Display the report to the screen.**
3. **Without exiting back to a UNIX prompt, reprint the report with no selection set.**
4. **From the Reporting Options choose Data Selection.**
5. **Modify an existing selection set to contain no selection criteria and save it upon exiting as "No Selection Set."**
6. **Display the new report to the screen.**

4

Managing the Data Dictionary

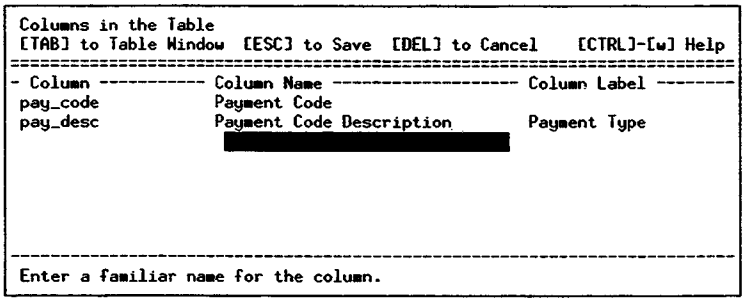
Major topics in this chapter include the following:

- Naming Tables and Columns
- Adding Tables to a Data Group
- Table Relationships

Descriptive Table and Column Names

You have the ability to give familiar names to the tables you create. Tables and columns created in INFORMIX-SQL (ISQL) are assigned default names which are often not descriptive enough for your users.

The Table/Columns option on the Options pull-down menu allows you to give tables and columns more descriptive or familiar names. Here, a table called "paymeth" is being assigned a familiar name. Each of its columns may also be assigned more descriptive names:



"Column Label" is the heading for the column as it appears on a report. The Column Name will appear as the default column heading if the Column Label field is blank. Column labels can be changed for individual reports within the *Report* Editor.

Data Groups

The Data Groups option on the Options pull-down menu allows you to put tables into a data group, or to create new data groups. Both Table Description and Table Name are Zoom-capable fields. Here is an example of the "`paymeth`" table being added to the "Demo Data Set:"

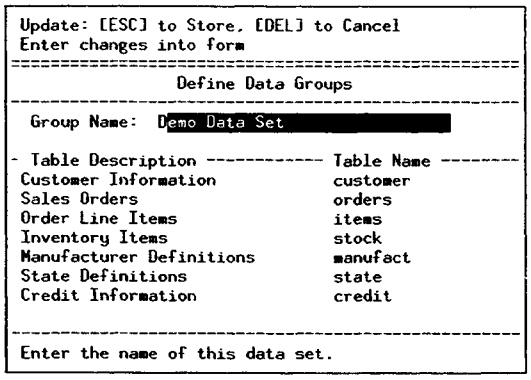
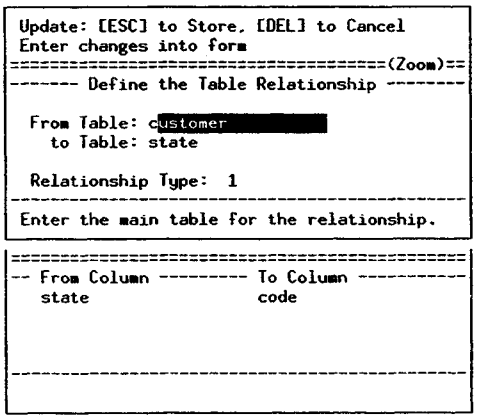


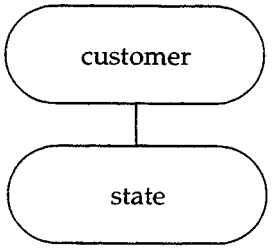
Table Relationships

After a data group has been added, you should define how the new table relates to the other tables in the database, so that the *Report* Writer knows how to incorporate this new table, and what the join column(s) are. The Table Relationships option on the Options pull-down menu allows you to establish this relationship to the other tables in your database. Here is an example of how the customer table relates to the orders table:



One-to-One Relationships

In a relational database, there are various ways that tables relate to each other. In the one-to-one relationship illustrated below, each order can have only one payment method attached to it. Therefore, a one-to-one relationship exists between the customer table and the state table.

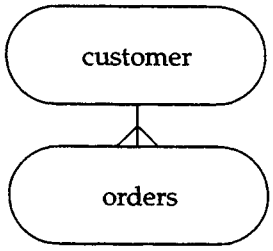


The relationship is specified as 1-1 in the Relationship Type field of the Define the Table Relationship window. The join column(s) are also specified in the Define the Table Relationship window.

One-to-Many Relationships

Another way that tables relate to each other is the one-to-many relationship.

One payment method can appear on several orders. Therefore, a one-to-many relationship exists between the customer table and the orders table.



This relationship is specified as "1-many" in the Relationship Type field of the Define the Table Relationship window. The filled-in Define the Table Relationship window would appear as follows:

```
Update: [ESC] to Store, [DEL] to Cancel
Enter changes into form
================================================
------- Define the Table Relationship --------

 From Table: customer
   to Table: orders

 Relationship Type: 2
------------------------------------------------
1.'1-1'; 2.'1-many'; 3.'1-0/1'; 4.'1-0/many'
================================================
-- From Column --------- To Column -----------
   customer_num          customer_num



------------------------------------------------
```

Notice the join column(s) are specified in the bottom portion of the Define the Table Relationship window.

Outer Joins

The purpose of defining table relationships is to present the data on a report correctly. The *Report* Writer will take into account how tables relate to each other at the time the report is printed. If table relationships are improperly defined, the data on the report will not be correct.

For "1-1" and "1-many" relationships, data will appear on the report only if a value exists in the join column of the table listed in the to Table field in the Define the Table Relationships window. If the join column's value is null, nothing will appear on the report.

Outer joins placed on a table relationship allow you to see all data on a report regardless of whether values in the join columns of the table listed in the to Table field are null or not. You specify outer joins with "1-0/1" or "1-0/many" in the Relationship Type field of the Define the Table Relationships window.

Section Summary

- Each table added to a database can be given a descriptive name. A descriptive name is familiar to the user of the table. Likewise, each column can have a familiar name.
- Each new table should be added to a data group.
- Each table's relationship to other tables in the database must be known to the *Report* Writer. Types of relationships are "1-1," "1-many," "1-0/1," and "1-0/many."
- Outer join relationships display all data on a report regardless of whether there is a match in the join columns.

Lab Exercise 4a

Objective: To introduce a new table to the *Report* Writer. The table will hold credit code information, and each customer will have an individual credit code. Give this table a familiar name, include it in a data group, and establish its relationship with the customer table.

Adding the Table Through ISQL

1. **Add a table called credit to your `stores{}` database.**

   (If you already have this table, you may proceed to Exercise 4b). Use ISQL to add your "credit" table. The "credit" table will hold the following columns:

Column Name	Type
credit_code	char(3)
credit_desc	char(10)
credit_amt	decimal(10,2)

2. **Once the table has been created, add the following rows to your "credit" table with ISQL:**

credit_code	credit_desc	credit_amt
AAA	Excellent	$10,000
BBB	Good	$5,000
CCC	Fair	$1,000
DDD	Poor	$250

Give Your Tables Descriptive Names

Next you can give your tables familiar names so that your users do not have to remember technical names.

1. **Start the *Report* Writer with:**

   fg.writer -d stores{}

2. **Select Table/Columns from the Options pull-down menu and pull up your credit table.**

3. **On the window that comes up, assign a familiar name to your credit table, such a Credit Code Information.**

4. **Give each column a descriptive, familiar name, if names have not been assigned yet.**

5. **Press [ESC] to save your work in this window.**

Lab Exercise 4b

Next you will add your new "credit" table to the "Demo Data Set" data group.

Adding Your New Table to a Data Group

1. **Select Data Groups from the Options pull-down menu. Select the "Demo Data Set" data group.**
2. **Under Table Description or Table Name, enter either the familiar name or technical name of your credit table. You may use the Zoom feature to help you.**

Note — If you enter the familiar name, the technical name is looked up for you after you press [ENTER]. If you enter the technical name, the familiar name is looked up for you after you press [ENTER].

3. **Press [ESC] when done.**
4. **Exit the *Report* Writer.**

Adding a Column to the Customer Table

Each customer can have one credit code. Therefore, you will add a column to the customer table to hold a particular customer's credit code. (If your customer table already has this column, you may proceed to "Adding the Table Relationship," below.)

1. **Using ISQL, add a Credit Code column to the customer table.**

   Make sure the Credit Code column is the same name and type as the Credit Code column in your credit table.

2. **In ISQL, enter some credit codes for some of the customers.**

   Be sure the credit code values are the same as the values for credit code in the credit table.

3. **Exit ISQL.**

Adding the Table Relationship

Next you will make known to the *Report* Writer how the `credit` table relates to other tables in the `stores{}` database.

1. **Start the *Report* Writer:**

   ```
   fg.writer -d stores{}
   ```

2. **Select Relationships from the Options pull-down menu.**

3. **Select Add a New Relationship at the Choose a Table Relationship window.**

4. **Enter "customer" in the From Table field of the Define the Table Relationship window.**

5. **Enter "credit" in the to Table field. Then determine the relationship type that exists between these two tables.**

   Ask yourself, how many credit codes can one customer have? For example, you can have many orders for one customer. Therefore, you would enter a "2" in the Relationship Type field for the `"customer" - "orders"` relationship. "2" is a "1-many" relationship.

6. **Tab to the From Column --- To Column area, and enter the join information.**

   Both fields are Zoom capable.

7. **Press [ESC] to save your work.**

8. **Again, select Add a New Relationship at the Choose a Table Relationship window and define the relationship in reverse for the two tables.**

9. **Press [ESC] to save your work.**

Add Credit Information to Your "Customer Orders Summary" Report

Finally you will see your new table and relationship in action. Add a credit description field to your "Customer Orders Summary" report.

1. **In the *Report* Writer, load the "Customer Orders Summary" report.**
2. **Select Edit the Report from the Report pull-down menu.**
3. **Add a credit description field label (and the field) to the report in the B1 block.**
4. **Save and print your report with no selection set.**

Using an Outer Join

If the relationship types you chose were "1-1" or "1-many," you will only see those customers that have credit codes in the database. The customers that do not have credit codes are not on the report.

If you would like to see all customers regardless of the existence of a credit code, change the relationship types to "1-0/1" or "1-0/many." This is how you indicate that there is an outer join relationship between tables.

1. **In the *Report* Writer, select Relationships from the Options pull-down menu.**
2. **Change the two relationships you entered above to indicate "1/0-1" or "1/0-many" for their respective relationship types.**
3. **Re-print the report.**

   What prints now?

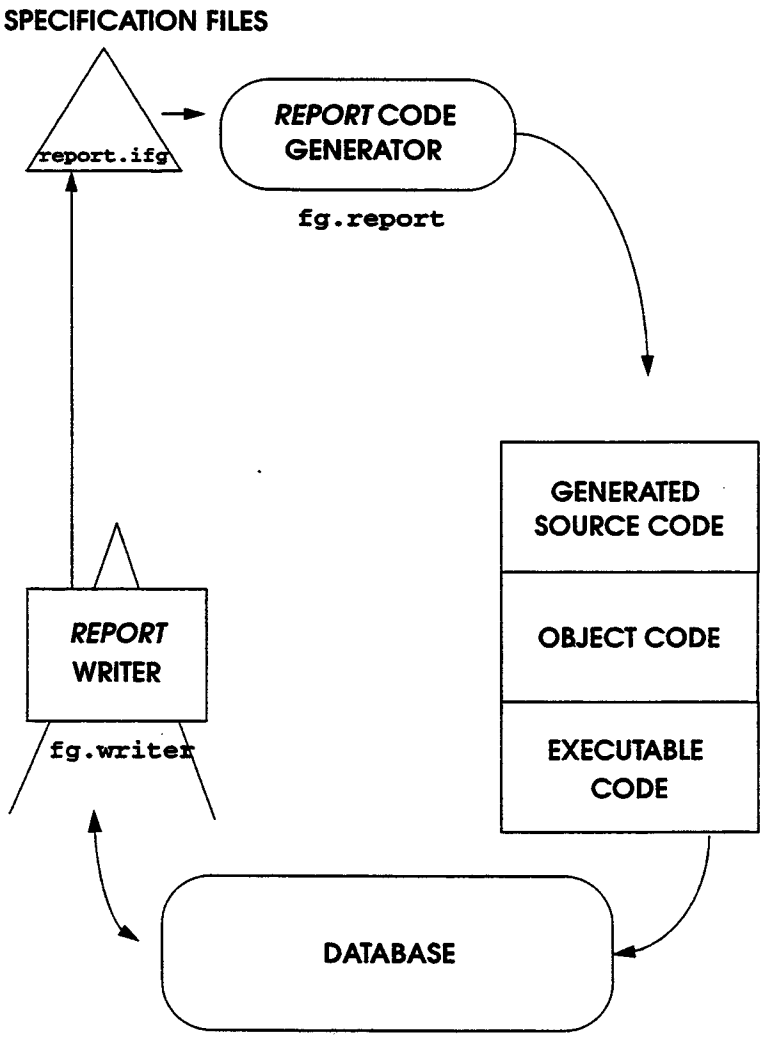
5

Report *Code Generator*

Major topics in this chapter include the following:

- Working with the *Report* Code Generator
- Specification Files for Reports
- Report Code
- Compiling and Running Report
- Editing Generated Reports
- Directory Structure

Report Code Generator Overview



Sample report.ifg File

```
database standard

output
    top margin         5
    bottom margin      5
    left margin        5
    right margin      75
    page length       66

page header
{
<A0    ]               +A1                          ]      Page: [A2 ]
================================================================================

}

before group of orders.customer_num
{
================================================================================
  Cust No: [A3                                     [A4
================================================================================

  Order No.   Date     Item No.  Price    Qty.   Extension
  ----------- -------- --------- -------- ------ ------------
}

on every row
{
  [A6       ] [A7    ] [A8  ]    [AA     ] [AB  ] [AC       ] [*
}

after group of orders.customer_num
{
                                                   -----------
                                                   [AE       ]

}

on last row
{

Grand Totals
  Extension   [AF       ]
}
attributes
    A0   = today
    A1   = constant "Orders Listing"
    A2   = pageno using "<<<<<"
    A3   = orders.customer_num using "<<<<<<<<<<"
    A4   = customer.company
    A5   = customer.zipcode
    A6   = orders.order_num
```

```
    A7    = orders.order_date
    A8    = stock.stock_num
    AA    = stock.unit_price
    AB    = items.quantity
    AC    = items.total_price
    AD    = orders.customer_num using "<<<<<<<<<<<"
    AE    = sum(items.total_price)
    AF    = sum(items.total_price)
end

select
    name    =
    tables = orders, items, stock, customer
    join    =  customer.customer_num = orders.customer_num and stock.manu_code =
items.manu_code and stock.stock_num = items.stock_num and items.order_num = orde
rs.order_num
    order   = orders.customer_num
end

defaults
    progname  = report
    prcname   = Orders Listing
    destin    = report.out
end

data group Demo Data Set
    tbls    = customer orders items stock manufact state credit
    lang    = ENG
end
```

Files Generated by the *Report* Code Generator

The *Report* Code Generator creates the following files:

```
globals.4gl        main.4gl
midlevel.4gl       lowlevel.4gl
report.4gl         Makefile
```

A Makefile is simply a list of files to compile and libraries to link. FourGen uses the UNIX make utility to compile files.

The `report.4gl` takes the WYSIWYG image in the `report.ifg` file and maps it out as follows.

Compiling Programs

The generated 4GL source code must then be compiled into object code. Also, all calls to functions in libraries must be resolved. *Report* programs are compiled with a script called `fg.make`. The `fg.make` script will compile each .4gl file and link in library functions that are called within the generated code.

Passing Criteria at Run-Time

Once you have compiled a report program, you can pass criteria to the report on the UNIX command line.

The filter flag passes an SQL "where" clause. For example:

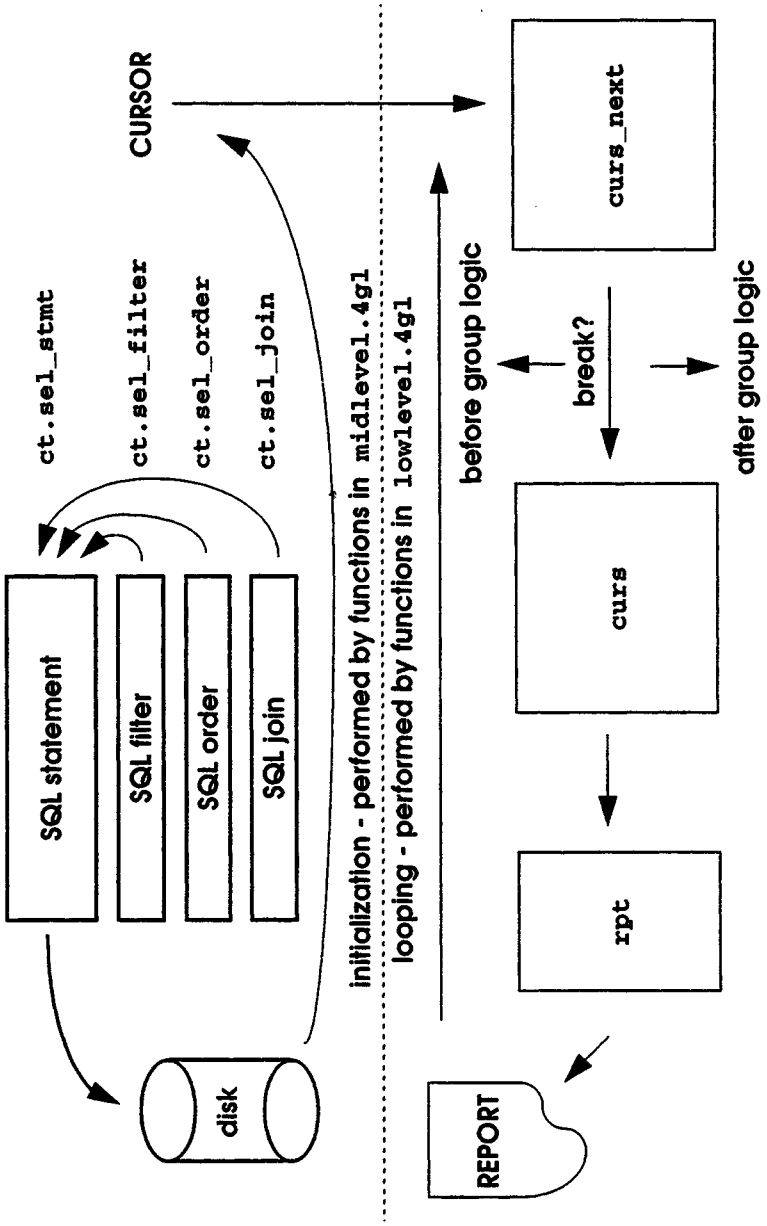
```
fglgo *.4gi -d standard filter "strtranr.doc_date > 01/01/92"
```

The order flag passes an SQL "order" clause. For example:

```
fglgo *.4gi -d standard order "strtranr.doc_date"
```

Most of the other variables in the "defaults" section of a `report.ifg` file can be overridden at run-time. Another often-used variable is `destin`. This variable is used to specify the report's destination at run-time (e.g., screen, file, or printer).

Report Data Flow



Report Functions

The following tables describe some of the key functions used by generated report programs.

Initialization Functions:

ct_defaults()	initialize all "ct" global variables with hardcoded default values
ml_defaults()	initialize "ct" variables with values from the `report.ifg` file
ct_parse()	reads command line input from report, if any
ct_display()	border info for pre-report screen called with seven different flags
ct_display()	Gathering elements message
ml_getcount()	number of records fitting selection criteria; terminates if no rows
ct_display()	Sorting ## Elements message
ml_define_cur()	run select statement with all elements required to load cursor
ct_start_rpt()	destination output initialize
ct_display()	"Processing element # of #" message
ml_fetch()	get first row of cursor into `curs_next` record
ml_curs_prep()	put data into `curs` record from `curs_next`
before_group()	"first row" initialization of report break logic; additional code can be added
ml_curs_null()	clears `curs` buffer; `curs_next` record contains first row; `curs` buffer is null

Loop on Detail Functions

ml_before_break()	check to see if curs = curs_next; if false then there is a break; break logic is also done by "Report;" custom code can be added here
ml_curs_prep()	curs_next row moves to curs record
ml_fetch()	if not last row, get next row into curs_next record
on_every_row()	moves curs data into report record
ml_after_break()	check to see if group finished
after_group()	last row of table processing
ml_output()	report line is output to "Report;" "Report" is report name generated
ct_display ()	"Processing # of #" message

End of Job Functions:

ct_finish()	"Finish Report" message
ct_display()	"Processing Complete" message
clear_screen()	clears the screen display

Report Functions Commonly Modified

There are some functions in midlevel.4gl that you will frequently modify. These functions give you control over the SQL statement that is executed to return rows from the disk. You can modify these functions to pass criteria to the SQL statement:

- ml_filter(): manages the final "where" criteria that are used in the SQL statement. The ml_filter() function manages the ct.sel_filter variable, and is typically modified to pass

selection criteria to a program. An example of how you can modify `ml_filter()` to put a selection criteria screen into a modified program is included in this section's Lab exercises.

- `ml_order()`: manages the "order" criteria that are used in the final SQL statement. It manages the `ct.sel_order` variable.
- `ml_join()`: manages the "join" criteria that are used in the SQL statement. It manages the `ct.sel_join` variable.

Referencing Fields not Printed in the Report

On occasion, you need to select data to be included on a report, or "order" report output based on a field that does not appear on the report. The optional `more` line defines the columns that are not included on the report but are needed in the select or order by statements.

If the report from the previously mentioned sample `report.ifg` file needed to be arranged by active code, payment method, or another field that does not print, a modification to the select section of the .ifg file would allow that functionality. As the following example illustrates, each field to be included is defined with its own more statement.

```
select
more = customer.paymeth
more = customer.active
name =
tables = orders, customer
join = customer.customer_num = orders.customer_num
order = orders.customer_num
```

Basic Library Philosophy

A library holds source code that is shared by other programs. Envision several lines of source code that check the form to be printed to, lines per page, carriage control, etc. Every print program will use this source code. Does it make any sense to copy the same lines of source code into every program directory that uses these lines of code? No. Therefore, the lines of source code are placed into a function, and that function is placed into a library. Any program that needs to consider these options simply calls that function in the library.

When placing a function that will be shared by programs into a library, we first ask:

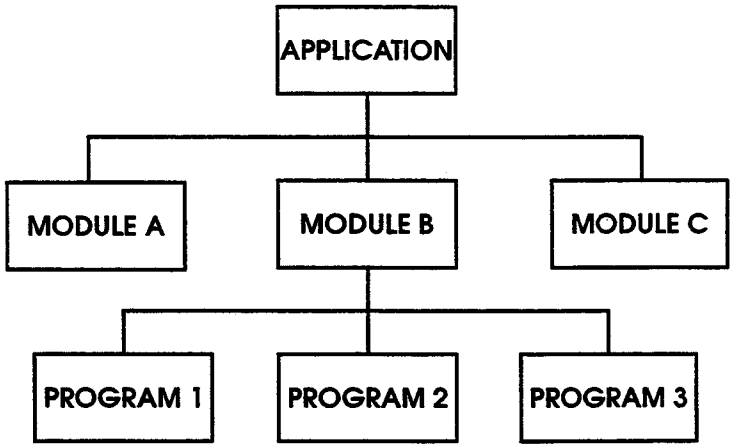
- What types of programs will use this library function?
- Will it be used only by report programs?
- Will it be used by products other than report programs, such as input programs?

Based on what types of programs will use the function,
places it into an appropriate library directory.

libraries exist as library directories. A library directory is simply a .4gs directory that contains .4gl files with code shared by multiple programs.

Directory Structure

- **Program Directories** contain all the files that make up a program. Program directories are the lowest level of directories.
- **Module Directories** contain program directories. A "module" contains a group of programs that are somehow alike.
- **Application Directories** contain module directories. An "application" contains modules that are alike.



Section Summary

- You have the ability to create INFORMIX-4GL source code on reports that you develop with the *Report* Writer. One advantage of creating code is that you can modify the code; that is, any capabilities that cannot be performed with the *Report* Writer may be coded into INFORMIX-4GL source code.
- Export a report specification file called `report.ifg`. The `report.ifg` file is read by the *Report* Code Generator, and INFORMIX-4GL source code is created based on instructions in the `report.ifg` file.
- Two special files created by the *Report* Code Generator are `report.4gl` and `Makefile`. The first, `report.4gl`, is the WYSIWYG image of the report in the `report.ifg` file mapped out in INFORMIX-4GL language. The `Makefile` contains a list of files to compile and libraries to link.
- The source code is compiled with `fg.make` script. Compiling converts each source code file into object code. The `fg.make` utility also performs linking, which means that calls to library functions in the generated source code are resolved. The script `fg.make` uses the UNIX `make` utility to perform its tasks.
- You can pass report criteria from the command line.
- Fields used to select or order report output, that are not included on the report, are declared by adding a `more` statement in the select section of the `report.ifg` file.
- The flow of a generated report is very simple. The first half of the logic handles setting up the main SQL statement and executing it. The second half of the logic handles processing each row returned by the SQL statement.
- Commonly modified functions are `ml_filter`, `ml_order`, and `ml_join`. These functions control the final filter, order, and join criteria, respectively, that are attached to the main SQL statement.
- When creating a directory to hold an exported `report.ifg` file, the same directory structure that is followed with input and edit programs applies to report programs.

Lab Exercise 5a

Objective: You will create INFORMIX-4GL source code for a report you made in the *Report* Writer.

Make a Simple Report

First, you will make a simplified "customer orders" report. It is helpful to work with a simplified report when learning how to modify the code.

1. **Start the *Report* Writer with**

   ```
   fg.writer -d stores{}
   ```

2. **Load the "Orders" report.**
3. **Under the Report pull-down menu select Group and Sort.**
4. **Group the "Orders" report by Customer Number.**
5. **Save as "Orders Listing."**
6. **Quit out of the *Report* Writer.**

Export the Report from the *Report* Writer

1. **From your HOME directory, make a directory called "labs.4gm."**
2. **cd to `$HOME/labs.4gm`**
3. **Make a directory to contain the report program.**

   Give it an o_ prefix and name it with your initials or student number. Examples:

   ```
   o_wjc.4gs, o_stu01.4gs, o_rep01.4gs
   ```

4. **cd to your newly created directory.**
5. **Start the *Report* Writer**
6. **Load your "Orders Listing" report.**

7. **Under the File pull-down menu, select Import/Export and then select Export Report.**

   You import and export reports for two purposes: To move reports from one database to another and to generate code from a report in the Writer. In this exercise, you will generate code from the "Orders Listing" report.

8. **Under "Export File Name," change `report.txt` to `report.ifg`.**

9. **Press [ESC] to export the report.**

10. **Exit the *Report* Writer when finished.**

Generate Code on the Report

1. **List the files in your report directory. Notice the `report.ifg` file.**

   The `report.ifg` file alone will not make up an entire report. It is simply a specification file. You must create and compile source code to run your program. The *Report* Code Generator will do this for you.

2. **Start the *Report* Code Generator with the following command:**

   ```
   fg.report -d stores{}
   ```

   The *Report* Code Generator reads the instructions in the `report.ifg` file and creates INFORMIX-4GL source code. This source code is the code that will drive your report.

3. **When the *Report* Code Generator is done, list the files in the directory.**

   The newly-created .4gl files contain INFORMIX-4GL source code.

Compile the Code

After generating code, you must convert this code to object code, and then link in the libraries. These processes are called "compiling" and "linking," and you use the following steps:

1. **From your report directory, compile your code by typing**

   ```
   fg.make
   ```

2. **When compiling and linking is done, list the files.**

   All the object code exists in either .4go files or .o files, depending on whether you compiled using INFORMIX-RDS or INFORMIX-4GL, respectively. The lone executable file has a .4gi or .4ge extension.

Run the Program

1. **Run the program with the following command:**

   ```
   fglgo *.4gi -d stores{}
   ```

   - or -

   ```
   *.4ge -d stores{}
   ```

   After the report runs, the results are placed in a file called `report.out`.

2. **Use vi to view the `report.out`.**

   This is your final report.

Lab Exercise 5b

Objective: To add a new field using the *Report* Writer and to generate code to see the result of adding a field.

Add the New Field with the *Report* Writer

1. **Start the *Report* Writer.**
2. **Load your "Orders Listing" report.**
3. **Under the Report pull-down menu, select Edit the Report.**
4. **Add the "Company Name" column to the right of the Customer Number column.**
5. **Under the File pull-down menu, select Save the Report to save your work.**
6. **Select Import/Export and then select Export Report.**
7. **In the Export File Name field, change `report.txt` to `report.ifg`. Overwrite the existing `report.ifg`.**
8. **Press [ENTER] and [ESC] to export your report with the new field on it.**
9. **Exit the *Report* Writer when finished.**

Generate Code on the Report

1. **Use vi to edit the `report.ifg` file and see if you can spot the changes.**
2. **Start the *Report* Code Generator.**
3. **Before the *Report* Code Generator is finished, the Overwrite menu appears. When it appears, answer choice "3" for each file.**

   If the *Report* Code Generator finds a .4gl file that already exists and its newly generated .4gl file is different from the existing .4gl file, it will ask you what to do with the existing .4gl file.

Generally you will answer choice "1" to overwrite the file. However, for this exercise you will be viewing the differences between newly generated .4gl files and old .4gl files. Choice "3" gives the existing .4gl file an ".old" extension and generates a new .4gl file. Your existing file is preserved in the .old file.

Compile the Code

The newly-generated .4gl files now must be compiled and linked.

Use the following command to compile and link your code:

```
fg.make
```

Run the Program

1. **Run the program with the following commands:**

   ```
   fglgo *.4gi -d stores{}
   ```

   -or-

   ```
   *.4ge -d stores{}
   ```

   The report will run and the results will be placed in a file called `report.out`.

2. **Use vi to view the `report.out`.**

   This is your final report. Is your new field there?

3. **Quit out of `report.out`.**

Diff the Files

Now use the UNIX diff utility to see how adding a new field affected the generated code.

1. **List the files in the directory.**

2. **For each .4gl file in your directory, run the `diff` command against the .old file.**

   The `diff` command syntax is:

`diff [first_file] [second_file]`

For example, to see the differences between `report.old` and `report.4gl`, you would use `diff` as follows:

```
diff report.old report.4gl
```

What is the difference between the old .4gl file and the newly generated .4gl file?

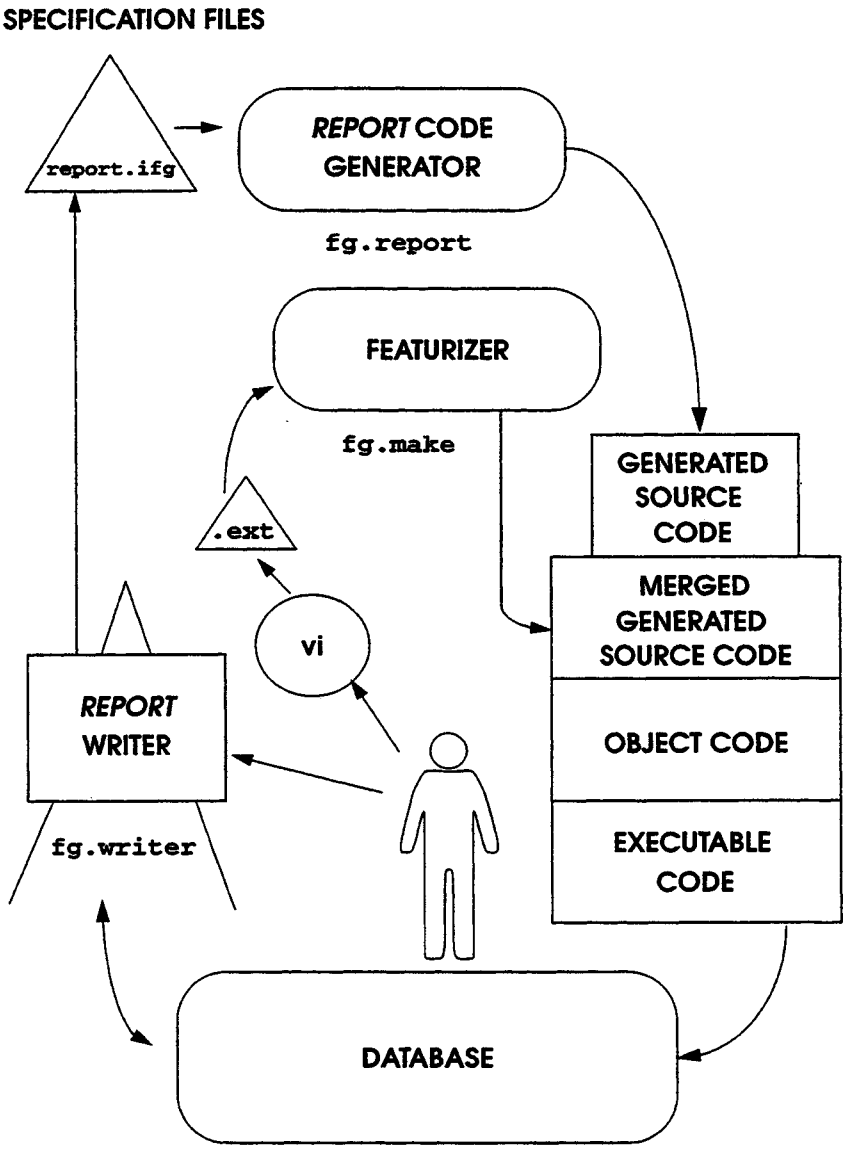
6

Using the Featurizer

Major topics included in this chapter are the following:

- Using Blocks to Modify Generated Code
- Block Command Placement
- Extension Files
- Feature Sets

Report Code Generator Overview



A Function with Blocks

Every line of code created with the *Report* Code Generator is within a block. Here is an example of blocks within a generated function:

```
########################################################################
function ml_defaults()
########################################################################
# this function sets the ct.* record values
#

#_def_prcname - insert "prcname" code here
let ct.prcname = "test 2"

#_def_destin - insert "destin" code here
let ct.destin = "report.out"

end function
# ml_defaults()
```

Block Command Examples

Block command syntax:

```
block_command [function_name] [block_ID]
```

Example 1:

```
after block ml_defaults def_destin
```

This block command places code after the `def_destin` block in the `ml_defaults` function.

Example 2:

```
before block ml_defaults def_destin
```

This block command places code before the `def_destin` block in the `ml_defaults` function.

Example 3:

```
replace block ml_defaults def_destin
```

This block command replaces the entire `def_destin` block in the `ml_defaults` function.

Where Do Block Commands Go?

Block commands for reports go into .ext files, which are specification files where you place your block commands and the custom logic you wish to add. The Featurizer then reads the .ext files and places your logic into the source code. This allows you to make further modifications to your `report.ifg` file, such as added fields, etc., and regenerate without having to manually re-insert your custom logic.

The start file Command

Any block command placed in a .ext file must be under a start file command. The start file command tells the Featurizer to which file to apply the block command.

Here is the syntax of the start file command:

```
start file "file_name"
```

For example, a block command to manipulate a block in `midlevel.4gl` would be preceded by the following line:

```
start file "midlevel.4gl"
```

Here is an example of start file with a block command under it. Note that quotes are required around a file name.

```
start file "midlevel.4gl"
  after block ml_defaults def_destin
    for n = 1 to num_args()
        if arg_val(n) = "-e"
        then
            let check_post = "CHECK"
        end if
        if arg_val(n) = "-p"
        then
            let check_post = "POST"
        end if
    end for ;
```

You can have multiple start file commands in an .ext file. Here is an .ext file with more than one start file command:

```
start file "globals.4gl"
  before block NUL define
    check_post char(5) ;

start file "midlevel.4gl"
```

```
after block ml_defaults def_destin
  for n = 1 to num_args()
      if arg_val(n) = "-e"
      then
          let check_post = "CHECK"
      end if
      if arg_val(n) = "-p"
      then
          let check_post = "POST"
      end if
  end for ;
```

The logic that you see here parses the UNIX command line when the report is run and accepts additional arguments. These arguments then set a variable.

Note

There are some reserved words for blocks. Since there are no functions in `globals.4gl`, the block command uses the reserved word "NUL" in place of the function name.

Extension Files

The logic within blocks may be specific to one feature. You may set up one .ext file for each feature that you have. This allows you to selectively plug and unplug features from the source code of a program.

For example, a feature that parses the command line for additional flags could go into a file named:

```
parse.ext
```

A feature that could put a selection criteria screen into a report could go into a file called:

```
select.ext
```

You can have as many features as you want.

Feature Set Files

Feature set (base.set) files allow you to tell the Featurizer which features to plug and unplug. The names of the features are simply the prefixes of the .ext files.

Here is an example of an entire .set file:

```
parse - looks for added command line input
select - pulls up selection criteria screen
```

This .set file instructs the Featurizer to insert the custom logic found in the `parse.ext` file and the `select.ext` file. The Featurizer reads this .set file and plugs in the parse and select features.

The Featurizer only reads the first word on a line in a .set file. Anything following the first letter space on a line can be descriptive comments, as in the above example.

Plugging in Features with a .set File

The Featurizer will, by default, find and read a .set file called `base.set`.

Section Summary

- The Featurizer handles block merging. Use a block when making a modification to report code.
- Blocks can be manipulated with block commands. Block commands go into .ext files.
- The start file command applies blocks to a specified file.
- Extension (*.ext) files are selectively merged into code.
- You specify what .ext files merge into code via a .set file.
- Feature set (base.set) files allow you to specify what .ext files get merged into code.

Lab Exercise 6a

Objective: To merge your own custom code into the generated code with the Featurizer. The code that you will merge will simply display a message logo to the user before the program runs.

Run the Program and View the Logo

The first thing a report program does is display a "Loading Program" message to the screen. You will first see this message in action by running your report program.

1. **Run the report program.**

   ```
   fglgo*.4gi -d stores
   ```

   - or -

   ```
   *.4ge -d stores
   ```

   You can place additional message information around your "Loading Program" message. For instance, you might place a copyright message here or list the programmers who created the program.

2. **Use vi to open `main.4gl` and find the function called `logo`.**

   This function is responsible for displaying a unique message to the screen just before a report program runs. It displays the "Loading Program" message, and is generated for every report program.

   FourGen's copyright message is set up to appear, but its display statements are commented out.

3. **Find the two blocks in `logo`.**

   Notice how each line of code with the `logo` function is within a block. The blocks are marked with block tags (they are preceded by a "#_").

   The first block in this function displays the "Loading Program" message to the screen.

This is the block that you will manipulate with the Featurizer to display your own logo to the screen.

4. **Quit out of `main.4gl`**

Use the Featurizer to Place Custom Logic into Code

Next you will use the Featurizer to manipulate the above block.

1. **Create a .ext file with a descriptive prefix.**

   The prefix to the .ext file should be indicative of the "feature" you are incorporating, i.e., putting a logo into a report program. For example:

   `my_logo.ext`

   Next you will use the Featurizer to place custom logic into code.

2. **Use vi to open your newly created .ext file and insert a start file command at the top.**

   When manipulating a block, you must tell the Featurizer where the block and its function are. You do this by inserting a start file command in your .ext file. All block commands placed under that `start file "file_name"` command are applied to *that file*. The syntax for the start file command is:

   `start file "file_name"`

   At the top of your newly created .ext file, add in the `start file "file_name"` command, specifying the .4gl file which contains the block you are going to manipulate. This .4gl file must be in double quotes as above.

3. **Under `start file`, place the block command that will place your custom logic into code.**

   Any of the following block commands will place code differently, but the result in the final compiled program will be the same:

   - replace block
   - after block
   - before block

Pick any of the above block commands you wish.

Once you have decided, you may insert your block command in the .ext file. Block command syntax is:

block_command function_name block_name

The function name is `logo` and the block name is what you wrote down on the previous page. Be sure your block command is under your start file command.

4. **Insert the INFORMIX-4GL logic to display your custom logo.**

   Now add the actual custom Informix logic that will be placed into the code. In your .ext file under the block command, add one or two Informix display commands to display a message of your choice. For example:

```
display "This program has been brought to you by "
display "your_name"
```

5. **End the block with a semi-colon (";"), and save and exit of the .ext file.**

   You can have multiple block commands in an extension file. Block commands are delimited with a semi-colon.

Merging with the .set File

Next you will place your modification into code.

1. **Create a .set file with the exact name `base.set`.**
2. **Use vi to create the `base.set` file.**
3. **On the first line, add the prefix to the .ext file you created.**

   Example `base.set` file contents:

```
my_logo
```

Merge and Compile with fg.make

1. **Compile and link the source code with the following command again:**

   `fg.make`

   When you type `fg.make` at the UNIX prompt, you may get a message such as:

   ```
   Compiling source files.
   main.4gl: failed.
   ERROR: /usr/       /Make/program.sh:
   Unable to compile these 4gl files:
   main.4gl
   while compiling
   ```

   This indicates that a syntax error has occurred. See Appendix A in this manual for handling syntax errors. Be sure all syntax errors are fixed before continuing.

2. **Use vi to open `main.4gl`.**

3. **Find the function you modified and inspect the change.**

   The Featurizer should have incorporated your block command into the code.

Run the Program

- **Run the program.**

Is your custom logo appearing?

Embellish Your Custom Logo

Your custom logo message should appear at the lower left hand corner of the screen. You can center it so it looks more attractive by using the Informix at command with the display command. For example, the following:

```
display "your_name" at 7,3
```

This line positions the message better on the screen.

Lab Exercise 6b

Objective: To create and incorporate a selection criteria screen into your report so that the user may limit which customers appear on the report. You will create the selection criteria .per file, code a function that handles the selection criteria screen, and place the function call into code with the Featurizer.

Adding in Report Prompts

Many reports require the end user to enter selection criteria in a prompt prior to running a report. Using the *Screen* Code Generator and Form Painter, you can use the following extension file to automatically hook in these prompts. Follow the general steps outlined below to link in your query screens:

1. Create a query screen in the *Screen* Painter.
2. Run the *Screen* Code Generator.
3. Create an image (`report.ifg`) file for your report.
4. Run the *Report* Code Generator.
5. Create the extension file (shown below) and call it in the `base.set` file.
6. Run `fg.make` to build the report program.

Report Prompt Extension File

Add the following code to an extension file. Notice that you must supply the name of your prompt screen in three locations within this extension file. The italicized word *screen_name* denotes where you should enter the name of your prompt screen less the .per extension.

```
start file "Makefile"
  libraries
    $(fg)/lib/scr.a;

start file "midlevel.4gl"
  function define ml_filter
  m smallint,
  n smallint;
```

```
replace block ml_filter sel_filter
call socketManager("screen_name", "query", "default")
    let sel_filter = null
    let n = fgStack_pop()
    if n = 0
    then
        let int_flag = true
        # set default filter if user continues
        let sel_filter = "1=1"
end for
    end if;
    else
        for m = 1 to n
            let sel_filter = sel_filter clipped, fgStack_pop()
        end for
    end if;

start file "main.4gl"
  after block main after_init
    let scr_id = "default"
    ;

  at_eof
    function switchbox(funct)
    #_define_var - define local variables
        define
            #_local_var - local variables
            funct char(20)   # Function to pass on to the screen

        #_post_scr_funct -  Post the current function
        let scr_funct = funct
    #_switchbox -  Pass flow control to appropriate screen
        case
          # put your screen in here
          when scr_id = "screen_name" call S_screen_name()
          when scr_id = "default" call lib_screen()
          #_otherwise - otherwise clause
          otherwise let scratch = "no screen"
        end case

        #_scr_funct - Reset scr_funct upon return
        let scr_funct = ""

    end function
    # switchbox()
    ;
```

7

Creating Posting Programs

Major topics in this chapter include the following:

- Types of Posting Programs
- Creating Posting Programs
- Posting Program Techniques

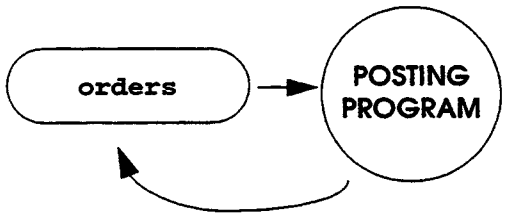
What is a Posting Program?

A posting program is a report program that performs some special processing on each row as it prints on the report. This special processing is in the form of SQL activity.

Posting programs can take data from one table and insert into another:

Posting programs can take data from more than one table and insert into more than one table:

The two "pre-posting" tables have a one-to-many relationship between each other, and the two "post-posting" tables also have a one-to-many relationship. Posting programs can also update a table:



Creating the Posting Program

There are two main steps to creating a posting program with *Report*.

1. Create a report with the *Report* Writer that will select all rows that you wish to post.
2. Use the Featurizer to insert SQL statements that act on each row as it is processed.

Creating the Report

Create a report that will pull from a table and display all rows to be posted. The report output itself is ideal because the user can see what data are to be posted.

Keep in mind when creating a report with the *Report* Writer:

- All values to be posted must be brought into the program. Therefore, you design your report so that all values to be posted are brought in by the main SQL statement in `midlevel.4gl`.
- All values to be posted do not have to be on the report itself. You may use the "Hidden Fields" capability in the *Report* Writer to pull data out of columns that do not appear on the final report.
- When posting from two tables, clarify the one-to-many relationship between the two tables. Design your report so there is a group break on the "one" table.
- Filtering criteria: How will you ensure that rows are not re-posted? Are there some rows that you never want posted?

Inserting the SQL Statements

After you have created the report with the *Report* Writer and have generated code for it, use the Featurizer to insert SQL statements into the appropriate places in code that will post your rows as they are processed. Typically you will place SQL insert and update statements into `lowlevel.4gl` to post each row as it passes through.

Items to keep in mind:

- If posting from one table to another table, place your SQL statement into the `on_every_row` function.

Here is an example of an SQL insert placed at the end of the `on_every_row` function. It is placed here via a block command in an .ext file.

```
#_unit_price - insert "on every row unit_price" code here
let rpt.unit_price = curs.unit_price
#_total_price - insert "on every row total_price" code here
    let rpt.total_price = curs.total_price
    insert into pst_items values
      (curs.item_num,
      curs.order_num,
      curs.stock_num,
      curs.manu_code,
      curs.quantity,
      curs.total_price)
end function
# on_every_row()
```

Here, as items table rows pass through the looping phase of the report, they are inserted into a table called `pst_items`.

- If posting from two tables that have a one-to-many relationship, you have two SQL statements: one to post the "one" table's data, and one to post the "many" table's data.
- The "many" table's data are processed by the `on_every_row` function, so you place one SQL statement in `on_every_row` to post.
- The "one" table's data are handled by the `after_group` function or the `a_g_[grouping column]` function, so you place your SQL statement here to post the "one" table's data.

Here is an example that places code into the `a_g_order_num` function. `a_g_order_num` holds code that is executed after the order number changes. Again, code is placed here via a block command in an .ext file.

```
#_a_order_num - insert after group code here
    insert into pst_orders values
      (curs.order_num,
      curs.order_date,
      curs.customer_num,
      curs.ship_instruct,
      curs.backlog,
```

```
    curs.po_num,
    curs.ship_date,
    curs.ship_weight,
    curs.ship_charge,
    curs.paid_date,
    curs.pay_code,
    curs.ship_method)
```

Here, as orders table rows pass through the looping phase of the report, they are inserted into a table called `pst_orders`.

Note

If you use the `curs` record as you see in the examples, make sure it contains all the variables needed for posting. It typically *will not*. You can add additional variables into your `curs` record by placing hidden fields on the report with the *Report* Writer. When you export the report, a hidden field shows up as a "`more =`" parameter in the select section of the `report.ifg` file. When you re-generate code, all supporting logic for hidden fields is created for you in code, including logic to place extra variables in the `curs` record.

Using prepare and execute

One key posting technique is to use the Informix prepare and execute commands in lieu of insert and update.

When you run an SQL insert or update, not only are these statements executed but they are also preprocessed each time they are run.

The preprocessing and execution stages can be separated, so that the insert or update is preprocessed only once, and then executed for each row of the report. You do so with the Informix prepare and execute commands, which preprocess and run respectively.

Using prepare and execute is ideal for posting programs, which are notoriously slow due to the large number of SQL statements in them. With prepare and execute, you preprocess an SQL statement only for the first row; then you simply execute the preprocessed SQL statement on all subsequent rows.

Instead of coding the Informix insert, you use a prepare and execute as follows:

```
if sql_prep is null
then
    let sql_str = "insert into pst_items values (?, ?, ?, ?, ?, ?)"
```

```
        prepare prep_stmt from sql_str
        let sql_prep = "Y"
    end if
    execute prep_stmt using
        curs.item_num,
        curs.order_num,
        curs.stock_num,
        curs.manu_code,
        curs.quantity,
        curs.total_price
```

The question marks that you see are placeholders in the preprocessed SQL statement, they are filled with values when the statement is executed. When the first row comes through the SQL insert, it is preprocessed and executed with prepare and execute.

A variable is set to indicate that the SQL statement has been prepared, and it stays prepared for the duration of the program. Then, as each subsequent row comes through, the program is invoked with execute. Executing a prepared statement takes considerably less time than executing *and* preparing an SQL statement.

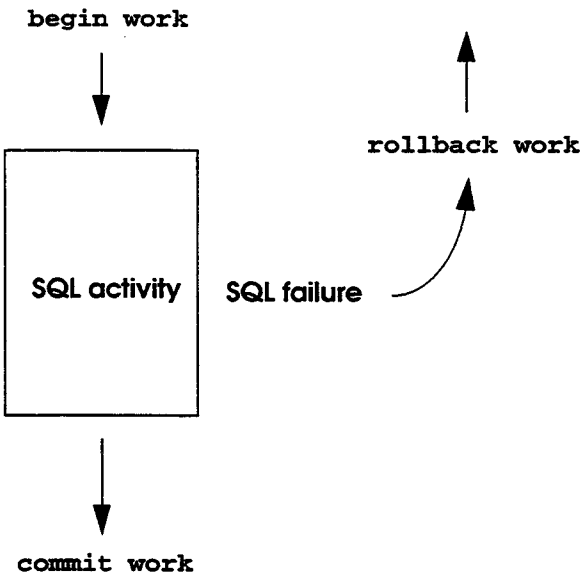
Using begin work, commit work, and rollback work

Another posting technique ensures database integrity.

When you are using more than one SQL statement in a posting program, there is always a danger that one SQL statement executes successfully while another one fails. This means that you have partially posted a record; for example, a "one" row is inserted but its "many" rows are not.

Fortunately Informix allows you to define the boundaries of a record. You can issue an Informix begin work command as the record begins processing, and then a commit work command when the record has finished processing. After each SQL statement is executed, a test for

successful execution is made. If an SQL statement fails, then a roll-back work command is issued and any SQL statements after the begin work command that have been executed are undone.



A begin work is placed in the `before_group` function or the `b_g_[grouping column]` function before any SQL statement. This is where a record starts processing.

```
#_b_order_num - insert before group code here
begin work
```

A commit work is placed in the `after_group` function or the `a_g_[grouping column]` function after any SQL statements. This is where a record completes its processing.

```
commit work
end function
# a_g_order_num()
```

Section Summary

- The following are various types of posting programs:
  - Posting programs that take data from one table and insert it into another table.
  - Posting programs that take data from multiple tables and insert the data to multiple tables.
  - Posting programs that update values in columns.
- Creating a posting program is a two-step process: use the *Report* Writer to create a report that pulls all values to be posted into the program, and then use the Featurizer to place SQL statements that do the actual posting into the code.
- It is critical to ensure that all values to post are selected into the posting program. Use of the "Hidden Fields" capability in the *Report* Writer is especially useful for accessing values that do not appear on a report.
- The performance of posting programs may be improved by using Informix prepare and execute commands in lieu of insert or update.
- When posting to multiple tables, it is important to use the begin work/commit work/rollback work commands of Informix to ensure database integrity.

Lab Exercise 7

Objective: To create an inactivation program, which inactivates customers. This type of posting program prompts the user for a customer to inactivate, and then inactivates the customer by setting a flag in the customer table.

This lab uses the Selection Criteria screen you created in the previous section's lab. The customer that the user enters into the Selection Criteria screen is the customer to be inactivated.

Create a New Program Directory

1. **cd `$HOME/labs.4gm`.**
2. **Make a new program directory prefixed with a "`p_`" and give it a .4gs extension.**

   Make sure the entire prefix does not exceed eight characters. Posting programs are prefixed with a "`p_`".
3. **cd into your new posting directory.**
4. **Copy in all files from your last lab.**

   You will convert the prior lab's report to a posting program.
5. **Remove the executable .4gi or .4ge file.**

Add a Column with isql

1. **Enter `isql` and add a column called "`inactive`" to the `customer` table.**
2. **Make its data type `char(1)` and exit out of `isql`.**
3. **Run the *Report* Writer.**
4. **Load the "Orders Listing" report.**

5. **In the *Report* Writer, change the name of the report to "Customer Inactivation Report."**
6. **Export the Report.**
7. **Exit the *Report* Writer.**

Generate Code

Generate code on your modified `report.ifg` file using the *Report* Code Generator.

Add an SQL Statement with an .ext File

Next use the Featurizer to place an SQL statement into code. The SQL statement will be an update which places a "Y" value into the `inactive` column for the customer that the user selects to inactivate.

1. **Use vi to open `lowlevel.4gl`.**
2. **Find the function that handles after group logic for the customer number field (`a_g_customer_num()`).**
3. **Find the block within the above function that allows you to place custom after group logic (`#_a_customer_num`).**

   This is the block that you will manipulate with the Featurizer.
4. **Exit `lowlevel.4gl`.**
5. **Create an .ext file with a descriptive prefix.**

Now create an .ext file to hold a block command with the modification you wish to make. Create an .ext file now (just the file). The prefix to the .ext file should be indicative of the "feature" you are incorporating, such as:

```
inact.ext
```

6. **In your newly created .ext file, insert a start file command at the top.**

   Use the following start file command:

   ```
   start file "lowlevel.4gl"
   ```

Which .4gl file contains the block you are going to manipulate?

7. **Under `start file`, place the block command that will place your update statement into code.**

   The block command should look as follows:

   ```
   after_block a_g_customer_num a_customer_num
   ```

8. **Insert an SQL update statement to set the current customer's inactive column to "Y."**

   For example, you could use the following SQL statement:

   ```
   UPDATE customer SET inactive = "N"
       WHERE customer_num = curs.customer_num
   ```

9. **End the block with a semi-colon (";") and save and quit out of the .ext file.**

Merging with the .set File

1. **vi your `base.set` file.**
2. **On the first line, add the prefix to the .ext file you created.**

   Add only the prefix. Make sure that your selection criteria screen is kept in the .set file, because your inactivation feature will depend on the selection criteria feature being in code.

Merge and Compile with fg.make

1. **Start the Featurizer:**

   ```
   fg.make -mf
   ```

   The `fg.make` command invokes the Featurizer and merges your logic into the code. The "`-mf`" flag instructs `fg.make` to only invoke the Featurizer and not compile and link.

2. **Use vi to open `lowlevel.4gl` and notice your change.**

   Is your change there?

3. **Complete the compile, and link the source code with the following command again:**

   ```
   fg.make
   ```

Run the Program

Use the fglgo runner or type the *4ge filename directly to run the report program.

Verify Through isql

Enter `isql` and select the customer you updated. Is the value of the inactive column "Y" for this customer?

8

New Features

Major topics in this chapter inlcude the following:

- Larger Selection Statement Variables
- Post Processor Flexibility
- Print Statement Block Tag Logic
- Custom Image File Block Tags
- Block Tags in `Makefile`

Larger Selection Statement Variables

The *Report* Code Generator now uses larger selection statement variables to prepare and declare selection statements. This increased size more than doubles previous capabilities.

Previous Variable Size	New Variable Size
`ct.sel_join char(200)`	`sel_join char(512)`
`ct.sel_filter char(200)`	`sel_filter char(2048)`
`ct.sel_order char(200)`	`sel_order char(512)`
`ct.sel_stmt char(1024)`	`sel_stmt char(4096)`

In addition these variables are now located in `midlevel.4gl` instead of `globals.4gl`, and the `ct.` has been dropped from their name. This move and subsequent name change gives you easier access and more control over the selection statement variables; you can increase their size whenever you near a limitation without having to change any libraries.

Backward Compatibility

An early work around to the limited size of selection statements used block commands and string replacements to increase selection limits. If you have programs that depend on this work around and you want to regenerate them, you can use an environment variable and an options file to maintain backward compatibility.

The `rpt_select` environment variable takes two options: `local` and `global`. The `global` option sets all variable values to their former size, name, and location. This is the option you should use to maintain backward compatibility. The `local` option sets all selection variables to their new size, name, and location.

You can set the `rpt_select` variable to `global` on a system, module (4gm) or program level (4gs). By default this variable is pre-set to `local`, so all selection variables start with the larger sizes.

To set on a system level, type `rpt_select="global"` in the `report.org` file. This file is located in the `$fg/codegen/options` directory.

To set on a module or program level, create a `report.opt` file in the directory that contains the programs you want to affect. In the `report.opt` file, add the `rpt_select="global"` line.

The ml_ct_sel_compat( ) Function

In addition to the new variables, a new function has also been added to `midlevel.4gl`. This function, named `ml_ct_sel_compat()`, sets local selection variables to any value passed from the command line or any other initializing method.

Post Processor Flexibility

The *Report* Code Generator lets you customize generated source code with a post processor. A post processor is useful for many tasks, such as making global changes to code, replacing or altering code blocks, and implementing bug fixes.

The `fg.report` program runs a post processor on the local application if the environment variable `$local_rpt` is set. Use this variable to point to the name of the program you wish to run on the generated 4GL code.

The same arguments that you pass to `fg.report` get passed to your post-processor program.

For example, assume you write an initialization function (say, `my_init()`) that is more relevant than the generic `init()` function that the *Report* Code Generator creates. You may want `main.4gl` to call `my_init()` rather than the `init()` function. You can set up a post processor to change the initialization call in `main.4gl` to `my_init()`.

To change `init()` in `main.4gl` to `my_init()`:

1. **Write a shell script (`chg_init`, for example) that uses the `sed` utility to remove `init()` and add a call to `my_init()`, such as:**

```
#chg_init
sed "s, call init, call my_init, " main.4gl > main.tmp &&
mv main.tmp main.4gl
```

2. **Set your `$local_rpt` environment variable to the name of the post-processor script (you might want to set this variable in your `.profile` file), for example:**

```
local_rpt=chg_init; export local_rpt
```

Once the Code Generator completes creating source (.4gl) files, your local `main.4gl` file contains the function call `my_init()` rather than `init()`.

Print Statement Block Tag Logic

In previous versions of the *Report* Code Generator, the `report.4gl` file contained a block tag for each report row. These block tags, which were numbered sequentially, allowed you to create custom code for each line in the report. In other words, you could use an extension file and block command to act on any `report.4gl` print statement. The following shows an example of the old logic:

```
format
   page header
        whenever error call error_handler
        #_print_1 - tag for the following print statement
        print
            column 1, today using "mm/dd/yy";
            let scratch = "CUSTOMER ORDER HISTORY LIST"
            let x = 40 - (length(scratch) / 2)
            print
                column x, scratch clipped;
            let scratch = pageno using "Page: <<<<"
            let x = 81 - length(scratch)
            print
                column x, scratch clipped
        #_print_2 - tag for the following print statement
        print
        #_print_3 - tag for the following print statement
        print
            column 1, "Customer"
        #_print_4 - tag for the following print statement
        print
            column 1, "Number",
            column 11, "Name",
            column 30, "Company",
            column 54, "Phone"
        #_print_5 - tag for the following print statement
        print
            column 1, "----------------------------------------",
            column 41, "-----------------------------------"

page trailer
        #_print_6 - tag for the following print statement
        print
        #_print_7 - tag for the following print statement
        print
            column 1, "----------------------------------------",
            column 41, "-----------------------------------"
        #_print_8 - tag for the following print statement
        print
            column 1, pageno using "Page: <<<<";
            let scratch = pageno using "Page: <<<<"
```

```
        let x = 81 - length(scratch)
        print
            column x, scratch clipped
```

This version of the *Report* Code Generator uses new print statement block tag logic. Instead of the entire `report.4gl` file having sequentially numbered print statement block tags, now print statement block tags are numbered according to the control block that contains them. For example:

```
format
  #_format - Format section
      #_page_header - Report block for page header.
          #_err - Trap fatal errors.
          whenever error call error_handler
          #_page_header_1 - Print statement
          call put_vararg(today using usg.today)
          call put_vararg("CUSTOMER ORDER HISTORY LIST")
          call put_vararg(pageno using usg.pageno)
          let header_image = imageManager_getLine(1)
          print header_image clipped
          #_page_header_2 - Print statement
          print
          #_page_header_3 - Print statement
          let header_image = imageManager_getLine(2)
          print header_image clipped
          #_page_header_4 - Print statement
          let header_image = imageManager_getLine(3)
          print header_image clipped
          #_page_header_5 - Print statement
          let header_image = imageManager_getLine(4)
          print header_image clipped
      #_end - End of report block.

      #_page_trailer - Report block for page trailer.
          #_page_trailer_1 - Print statement
          print
          #_page_trailer_2 - Print statement
              let header_image = imageManager_getLine(5)
              print header_image clipped
              #_page_trailer_3 - Print statement
              call put_vararg(pageno using usg.pageno)
              call put_vararg(pageno using usg.pageno)
              let header_image = imageManager_getLine(6)
              print header_image clipped
          #_end - End of report block
```

As you can see, this new scheme limits the number of lines affected when you insert a custom print statement. For instance, suppose you create a report that contains 30 print statements. When the *Report* Code Generator constructs your source code (.4gl) files, it adds 30 print statement block tags to your `report.4gl` file, one tag for each

print statement. The old method numbers these block tags from one to 30. As mentioned above, the new numbering scheme restarts at one for each control block.

Now suppose this same report program uses several extension files containing block commands that act on these print statement block tags. Using the old method, if a situation arises where you need to remove a print statement, all your extension files must be reworked. However, with the new method, you only need to alter the extension files that reference the modified control block.

Backward Compatibility

Although this new numbering scheme makes modifying reports easier, you can regenerate your existing report programs with the old method. The *Report* Code Generator uses a new variable that controls which method gets used, namely `rpttagtype`. This variable, which you set on a local basis in the `report.opt` file, accepts one of two values: `block` or `line`. By default, the *Report* Code Generator uses the new print statement block tag logic, in other words `rpttagtype` is set to `block`. If you want to use the old method, set `rpttagtype` to `line`.

As a rule, set `rpttagtype` to `line` when you are working with previously generated programs. In the `report.opt` file type:

```
rpttagtype=line
```

When developing new report programs, use the new numbering scheme to take advantage of the simplified code and regenerability.

Custom Image File Block Tags

Another new feature of the *Report* Code Generator is custom image file block tags. You can now add your custom block tags in the format section of the image file. These block tags, upon code generation, get placed into the `report.4gl` file. Once in the `report.4gl` file, you can use extension files and block statements to customize the source code. For example, the following image (`report.ifg`) file contains a custom block tag in the `before group` control block:

```
before group of customer.customer_num
{
#_custom_block - Example custom block tag
[f000     [f001              [f002                    [f008
   Activity Report:                                          [!
      Order #  PO #       Ordered        Shipped        Paid     [!
      ------------------------------------------------------------  [!
}
```

When this example image file is generated into source code, the custom block tag gets added to the appropriate line in the `report.4gl` file:

```
before group of rpt.customer_num
#_b_customer_num - Before group customer_num

   if
        rpt.order_num is not null
     or rpt.po_num is not null
     or rpt.order_date is not null
     or rpt.ship_date is not null
     or rpt.paid_date is not null
   then
        let dynamic = true
        need 4 lines
   else
        let dynamic = false
   end if


#_custom_block - Example custom block tag
call put_vararg(rpt.customer_num using usg.customer_num)
call put_vararg(rpt.fname)
call put_vararg(rpt.company)
call put_vararg(rpt.phone)
let line_image = imageManager_getLine(7)
print line_image clipped
```

Custom tags are useful, because they never change. Whereas print statement block tags change when new lines are added, custom tags are completely regenerable. When you create extension files that act on custom block tags, your modifications are preserved each time you regenerate the code.

Numbering Scheme Variable

When you create custom block tags in your `report.ifg` file, you must decide how you want the print statement block tags in the `report.4gl` file to increment. There are two methods. The first method counts your custom block tags; it is an "absolute" numbering scheme. The second method, known as "relative," does not count your custom block tags, thus the print statement block tags remain consistent no matter how many custom block tags you add to your `report.ifg` file.

In almost every case, the *Report* Code Generator defaults to the method appropriate to your situation. However, at times you might want to specify the non-default method.

To specify the non-default method, use the new `rpttagnum` variable. You can set this variable to either `absolute` or `relative`. On the local directory level, use the `report.opt` file to set this variable. For example, to set the `rpttagnum` to `absolute`, add the following line to the `report.opt` file:

```
rpttagnum=absolute
```

Note

The `rpttagnum` variable defaults to a different value depending on the value in the `rpttagtype` variable. When `rpttagtype` is set to `block`, `rpttagnum` defaults to `relative`. When `rpttagtype` is set to `line`, `rpttagnum` defaults to `absolute`.

The following lines of code show an example of both `absolute` and `relative` numbering schemes. Notice how the number of the third print statement block tag differs between the two methods:

Absolute Block Tag Numbering Scheme

```
#_page_header - Report block for page header.

    #_err - Trap fatal errors.
    whenever error call error_handler

    #_page_header_1 - Print statement
    call put_vararg(today using usg.today)
    call put_vararg("Syscolumns ")
    call put_vararg(pageno using usg.pageno)
    let header_image = imageManager_getLine(1)
    print header_image clipped

    #_custom_tag - Custom block tag
    let header_image = imageManager_getLine(2)
    print header_image clipped

    #_page_header_3 - Print statement
    let header_image = imageManager_getLine(3)
    print header_image clipped

    #_page_header_4 - Print statement
    let header_image = imageManager_getLine(4)
    print header_image clipped

    #_page_header_5 - Print statement
    print

    #_page_header_6 - Print statement
    print

#_end - End of report block.
```

Relative Block Tag Numbering Scheme

```
#_page_header - Report block for page header.

    #_err - Trap fatal errors.
    whenever error call error_handler

    #_page_header_1 - Print statement
    call put_vararg(today using usg.today)
    call put_vararg("Syscolumns ")
    call put_vararg(pageno using usg.pageno)
    let header_image = imageManager_getLine(1)
    print header_image clipped

    #_custom_tag - Custom block tag
    let header_image = imageManager_getLine(2)
    print header_image clipped

    #_page_header_2 - Print statement
    let header_image = imageManager_getLine(3)
    print header_image clipped
```

```
        #_page_header_3 - Print statement
        let header_image = imageManager_getLine(4)
        print header_image clipped

        #_page_header_4 - Print statement
        print

        #_page_header_5 - Print statement
        print

#_end - End of report block.
```

A good time to use the non-default method is when you add a custom image file block tag to a report that uses the old print statement logic (i.e., `rpttagtype=line`), and you regenerate that report. In this case, you do not want `rpttagnum` set to `absolute`, because `absolute` numbers each block tag including your custom tag, which in turn throws off every extension file that acts on the changed tag numbers. Instead, set `rpttagnum` to `relative`. This setting preserves all the existing print statement block tags. In essence this setting numbers around your custom tag.

Block Tags in Makefile

As you might have noticed already, the `Makefile` now comes with generated block tags before each `Makefile` macro. The new `Makefile` style makes including custom libraries and source files into your generated programs easier. In addition, you inherit all the flexibility and functionality of block statements and extension files, such as pluggable features and feature sets, without having to use difficult-to-maintain string replacement logic in block statements. The following example `Makefile` illustrates the new block tags:

```
########################################################################
# Copyright (C) 1993
# All rights reserved.
# Use, modification, duplication, and/or distribution of this
# software is limited by the software license agreement.
# Sccsid: %Z% %M% %I% Delta: %G%
########################################################################
# Makefile for an Informix report

#_type - Makefile type
TYPE = program

#_name - program name
NAME = tmp.4ge

#_objfiles - program files
OBJFILES = globals.o lowlevel.o main.o midlevel.o report.o

#_forms - perform files
FORMS =

#_libfiles - library list
LIBFILES = ../lib.a \
           $(fg)/lib/report.a \
           $(fg)/lib/user_ctl.a \
           $(fg)/lib/standard.a

#_globals - globals file
GLOBAL = globals.4gl

#-----------------------------------------------------------------------

#_all_rule - program compile rule
all:
        @echo "make: Cannot use make. Use fg.make -F for 4GL
compile."
```